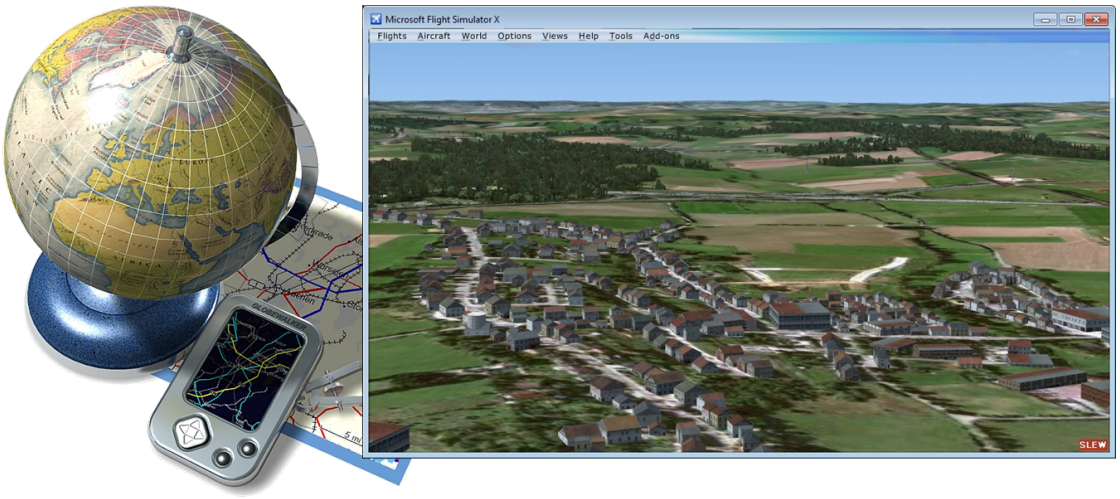



# scenProc 3.3 - User Manual

SceneryDesign.org

# Contents

**Manual conventions** **7**

**1 Introduction** **8**
1.1 The idea . . . 8
1.2 Basic scenProc workflow . . . 9
1.3 Understanding geographical data . . . 9

**2 Installation** **11**
2.1 Prerequisites . . . 11
2.2 Installation . . . 11
2.3 Configuration files . . . 11

**3 Basic workflow** **12**
3.1 The graphical user interface . . . 12
3.2 Keyboard shortcuts . . . 14
3.3 Options . . . 15
3.4 Configuration file . . . 17
3.4.1 Auto completion . . . 17
3.4.2 Tooltips . . . 18
3.4.3 Validation . . . 20
3.4.4 Conditional steps . . . 20
3.5 Filtering objects . . . 21

**4 Checking your geographical data** **24**
4.1 GIS tool . . . 24
4.2 Example: landuse attributes . . . 24

**5 Common configurations** **29**
5.1 Autogen from OpenStreetMap XML files . . . 29
5.2 Autogen from OpenStreetMap SHP files . . . 30
5.3 Autogen for FS2004 . . . 30
5.4 Identify different building shapes . . . 31
5.5 Placing autogen library objects along a road . . . 32
5.6 Placing XML library objects along a road . . . 32
5.7 Vary buildings heights in cities . . . 33
5.8 Create autogen tile grid for Google Earth . . . 35
5.9 Create SHP of QMID grid . . . 35
5.10 Create terrain vector scenery . . . 36
5.11 Create photoreal scenery . . . 37
5.12 Create Aerofly FS 2 cultivation . . . 37
5.13 Create Aerofly FS 2 buildings and use OSM levels and height attributes . . . 38
5.14 Vegetation detection from raster . . . 39

**6 Texture filter** **40**
6.1 Texture Filter Editor . . . 40
6.2 Available texture filter steps . . . 47
6.2.1 Input image . . . 47
6.2.2 Output image . . . 47
6.2.3 Add . . . 48
6.2.4 Add Weighted . . . 48
6.2.5 Add Weighted 3 . . . 48
6.2.6 Back Project . . . 48
6.2.7 Blur . . . 49
6.2.8 Brightness . . . 49
6.2.9 Burn Line . . . 49
6.2.10 Burn Point . . . 50
6.2.11 Burn Polygon . . . 50
6.2.12 Canny Edge Detection . . . 51
6.2.13 Color Correction . . . 51
6.2.14 Dilate . . . 52
6.2.15 Divide . . . 52
6.2.16 EmptyRaster . . . 52
6.2.17 Erode . . . 53
6.2.18 Intensity difference . . . 53
6.2.19 Invert . . . 53
6.2.20 KMeans . . . 53
6.2.21 Limit 3 . . . 54
6.2.22 Max value . . . 54
6.2.23 Max value (object) . . . 54
6.2.24 Mean . . . 54
6.2.25 Merge . . . 54
6.2.26 Min value . . . 54
6.2.27 Min value (object) . . . 55
6.2.28 Mlp . . . 55
6.2.29 Multiply . . . 56
6.2.30 Multi Res Segment . . . 56
6.2.31 NDVI (Vegetation) . . . 57
6.2.32 NDWI (Water) . . . 58
6.2.33 Noise . . . 58
6.2.34 Output Blendmask . . . 58
6.2.35 Output Night . . . 58
6.2.36 Output Season . . . 59
6.2.37 Output Watermask . . . 59
6.2.38 Resize . . . 59
6.2.39 Scale . . . 59
6.2.40 Sharpen . . . 59
6.2.41 Split 3 . . . 59
6.2.42 Split 4 . . . 60
6.2.43 Std Deviation . . . 60
6.2.44 Std Deviation (object) . . . 60
6.2.45 Subtract . . . 60
6.2.46 Svm . . . 60
6.2.47 TGDI (Tree/Grass) . . . 63
6.2.48 Threshold Binary . . . 63
6.2.49 Threshold Binary Inverse . . . 63
6.2.50 USI (Shadow) . . . 63
6.2.51 UWI (Water) . . . 64
6.3 Example texture filters . . . 64
6.3.1 Vegetation feature detection using back projection . . . 64

6.3.2 Vegetation feature detection using NDVI . . . 71
6.3.3 Water feature detection using NDWI . . . 74
6.3.4 Water feature detection using TSUWI . . . 78
6.3.5 Vegetation detection using SVM machine learning . . . 84
6.3.6 Photoreal texture filter . . . 90

**7 3D Buildings** **93**
7.1 Roofs . . . 93
7.2 Gables . . . 95
7.3 Features . . . 97
7.4 Building Texture Configuration Editor . . . 98

**8 Merging autogen files** **106**
8.1 Split data along autogen tiles . . . 106
8.2 Using ImportAGN step . . . 106
8.3 Using Autogen Merge Tool . . . 107

**9 Batch mode** **109**
9.1 Command line arguments . . . 109
9.2 Batch variables . . . 109
9.3 scenProc Batch Runner . . . 110
9.3.1 Worker status . . . 110
9.3.2 Jobs to process . . . 112
9.3.3 Finished jobs . . . 113
9.3.4 Options . . . 113
9.4 Workflow feeder . . . 115
9.4.1 User interface . . . 115
9.4.2 Example usage . . . 116

**10 List of available steps** **121**
10.1 Importing . . . 121
10.1.1 ImportAGN . . . 121
10.1.2 ImportBGL . . . 122
10.1.3 ImportBLN . . . 123
10.1.4 ImportGDAL . . . 124
10.1.5 ImportINF . . . 124
10.1.6 ImportOGR . . . 125
10.1.7 ImportSBuilder . . . 126
10.2 Processing . . . 126
10.2.1 AddAttribute . . . 126
10.2.2 AddAttributeIfInside . . . 127
10.2.3 AddAttributeSampleRaster . . . 128
10.2.4 AddCellAttribute . . . 128
10.2.5 AddCellAttributeFeatureCount . . . 129
10.2.6 AddCellAttributeIfInside . . . 129
10.2.7 AddCellAttributeSampleRaster . . . 130
10.2.8 BooleanFeatures . . . 130
10.2.9 BufferFeatures . . . 132
10.2.10 CalibrateFeatures . . . 134
10.2.11 CopyAttributeIfInside . . . 134
10.2.12 CreateRectangle . . . 135
10.2.13 DetectFeatures . . . 135
10.2.14 FilterDuplicateLines . . . 136
10.2.15 FilterFeatures . . . 136
10.2.16 FilterLineOnPolygon . . . 137
10.2.17 HeadingFromNearestLine . . . 137
10.2.18 LineToPoint . . . 138

10.2.19 LineToPolygon . . . 140
10.2.20 MatchLibObjectToPolygon . . . 141
10.2.21 MergeGrid . . . 142
10.2.22 MergeRasterFeatures . . . 142
10.2.23 OffsetLine . . . 143
10.2.24 PointToPolygon . . . 143
10.2.25 PolygonToPoint . . . 144
10.2.26 ProcessElevationRaster . . . 145
10.2.27 RasterToPolygon . . . 147
10.2.28 RemoveAttribute . . . 147
10.2.29 ReplacePolygonByBuildingRectangles . . . 148
10.2.30 ReplacePolygonByVegetationRectangles . . . 150
10.2.31 ScaleFeature . . . 151
10.2.32 SimplifyFeature . . . 151
10.2.33 SplitGrid . . . 152
10.2.34 UnloadFeatures . . . 153
10.2.35 UnloadXMLObjects . . . 153
10.3 Create autogen . . . 154
10.3.1 CreateAGNGenBuild . . . 154
10.3.2 CreateAGNLibObject . . . 155
10.3.3 CreateAGNPolyBuild . . . 155
10.3.4 CreateAGNPolyVeg . . . 155
10.3.5 CreateAGNRectVeg . . . 156
10.3.6 CreateAGNRowHouse . . . 156
10.3.7 SetAGNBuildingHeight . . . 157
10.3.8 SetAGNBuildingTexture . . . 158
10.3.9 SetAGNRowHouseTexture . . . 158
10.3.10 SetAGNVegetationSettings . . . 159
10.4 Create XML scenery . . . 159
10.4.1 Create3DBuilding . . . 160
10.4.2 CreateXMLBridge . . . 163
10.4.3 CreateXMLCarParking . . . 164
10.4.4 CreateXMLEffect . . . 165
10.4.5 CreateXMLExcludeRectangle . . . 166
10.4.6 CreateXMLLibObj . . . 166
10.4.7 CreateXMLRectangle . . . 168
10.4.8 CreateXMLSimObject . . . 169
10.4.9 MergeScene . . . 170
10.5 Create terrain vector scenery . . . 171
10.5.1 CreateMSFSExclude . . . 171
10.5.2 CreateTVecAirportBound . . . 172
10.5.3 CreateTVecExclusion . . . 172
10.5.4 CreateTVecFreewayTraffic . . . 172
10.5.5 CreateTVecPark . . . 173
10.5.6 CreateTVecRailroad . . . 173
10.5.7 CreateTVecRoad . . . 173
10.5.8 CreateTVecShoreline . . . 174
10.5.9 CreateTVecStream . . . 174
10.5.10 CreateTVecUtility . . . 174
10.5.11 CreateTVecWaterPoly . . . 175
10.5.12 CreateTVecWaterPolyGPS . . . 176
10.6 Create AF2 scenery . . . 176
10.6.1 CreateAF2Building . . . 176
10.6.2 CreateAF2Light . . . 177
10.6.3 CreateAF2Object . . . 178
10.6.4 CreateAF2Plant . . . 178

10.6.5 CreateAF2XRef . . . 178
10.7 Exporting . . . 179
10.7.1 ExportAGN . . . 179
10.7.2 ExportBGL . . . 179
10.7.3 ExportBLN . . . 180
10.7.4 ExportGDAL . . . 180
10.7.5 ExportGDALCombined . . . 181
10.7.6 ExportMSFS . . . 182
10.7.7 ExportOGR . . . 182
10.7.8 ExportResampleElevationBGL . . . 183
10.7.9 ExportResamplePhotorealBGL . . . 183
10.7.10 ExportResampleSeasonBGL . . . 185
10.7.11 ExportTSC . . . 186
10.7.12 ExportTVecBGL . . . 186
10.8 File management . . . 187
10.8.1 CopyAGN . . . 187
10.8.2 CreateDirectory . . . 187
10.8.3 PackageDirectory . . . 187
10.9 Debugging . . . 188
10.9.1 CreateTestGrid . . . 188
10.9.2 CreateTestPoint . . . 188
10.9.3 PrintCellAttributes . . . 188
10.9.4 PrintFeatureAttributes . . . 189

**11 Support & Consultation** **190**
11.1 Support forum . . . 190
11.2 Reporting crashes . . . 190
11.3 Consultation . . . 191

**12 User license** **192**
12.1 Third party libraries . . . 192
12.1.1 GDAL . . . 192
12.1.2 ECW JPEG 2000 . . . 193
12.1.3 MrSID . . . 195
12.1.4 OpenCV . . . 198
12.1.5 NetTopologySuite . . . 199

# Manual conventions

This is the manual for the scenProc tool. When reading this manual you should be aware of the following conventions:

## Example scripts

Examples of scenProc scripts are shown in a framed box as below. When the lines are too long in the example script a red return symbol is shown. You should not copy this symbol when copying the sample script over, but put everything on one line in your script.

```
ImportOGR|myfile.shp|*|*|+proj=sterea +lat_0=52.15616055555555 +lon_0
    ↪ =5.38763888888889 +k=0.9999079 +x_0=155000 +y_0=463000 +ellps=bessel +
    ↪ towgs84=565.417,50.3319,465.552,-0.398957,0.343988,-1.8774,4.0725 +units=m
    ↪  +no_defs <>
```

## Experimental features

Some features of scenProc are still in an experimental phase. This means they are not fully finished yet and might be a bit buggy still. To use these features you need to enable experimental steps in the options. You are not adviced to use them in a production script, as the implementation and the number of arguments can still change. But you are invited to try them and provide feedback. The description of such features is marked with an orange sidebar, like this section.

## To be deprecated features

Some features of scenProc have been replaced by new features and will be phased out in the near future. So you are advised not to use these features anymore, but they are still included in the manual until they are removed from scenProc. The description of such features is marked with an magenta sidebar, like this section.

## Deprecated features

Some features of scenProc have been replaced by new features and have been phased out. Text marked with a blueish-green sidebare provides information about such features that have been removed from scenProc.

# Chapter 1

# Introduction

## 1.1 The idea

Ever wanted to create autogen for a big Flight Simulator photoreal scenery? If so, you will know that doing so using the Annotator tool from the SDK is a lot of work. You will manually have to draw all the buildings and vegetation. For a small area that is doable, but if you want to cover an entire state or country that would be a very labour intensive task.

On the other hand geographical data of the world is becoming more and more common nowadays. On the internet you can view maps on sites like OpenStreetMap or Google Maps. And in your car you use satelite navigation to show you the way. All these applications use geographical data and there are also many sites where you can download the actual data instead of only viewing it. So how about using this available geographical data and creating your autogen from it?

That's exactly what the scenProc (Scenery Processor) tool is all about. It allows you to convert geographical data into autogen files for Flight Simulator. Figure 1.1 shows a photo scenery with autogen created with scenProc.

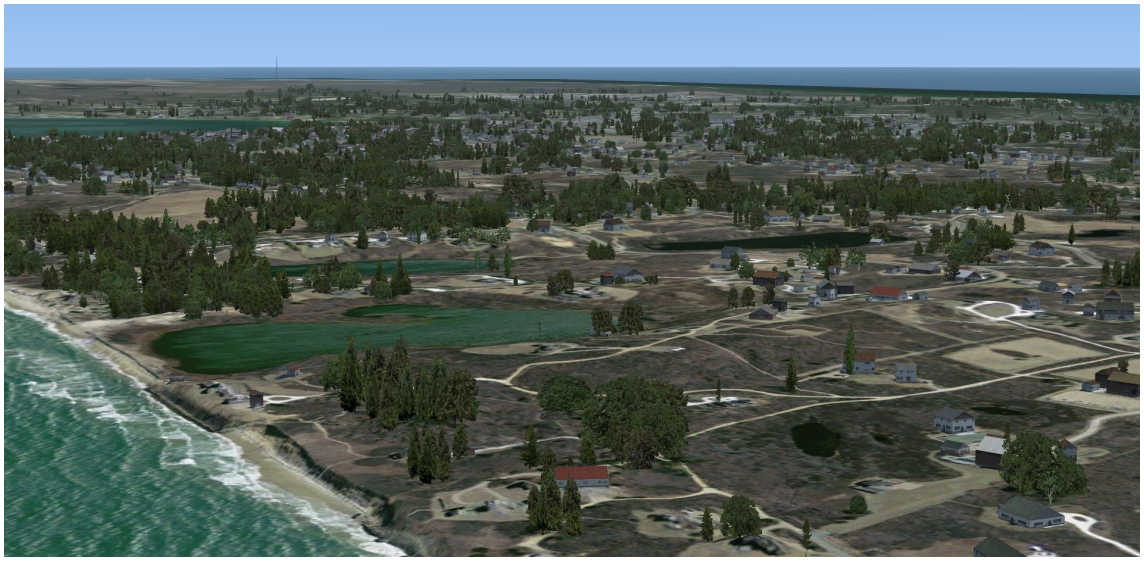


Figure 1.1: Example autogen made with scenProc

## 1.2 Basic scenProc workflow

The basic workflow of scenProc is that you make a configuration file that tells the tool which data it should use and how this data should be converted to autogen. Figure 1.2 shows the process graphically.

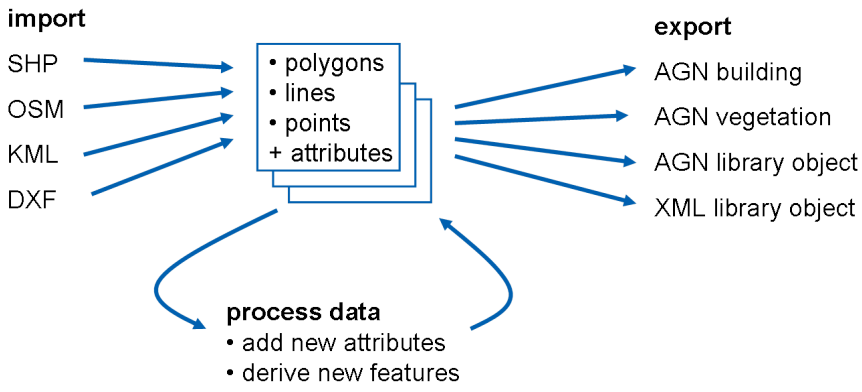



Figure 1.2: scenProc workflow

1. You start by importing the geographical data you want to use. scenProc supports most of the commonly used formats, like SHP, GML, KLM or OSM. The tool will load the features from the requested files and store the information in memory.
2. Next the features can be processed. This means that the features are prepared for export as autogen or that additional information is derived by combining information from different features. Later on in the manual more detail about the possible processing is given.
3. Finally the features are converted to autogen objects and written to AGN files so that Flight Simulator can use them. And besides autogen, it is also possible to export to BGL files using BGLComp.

## 1.3 Understanding geographical data

Since geographical data is very important when working with scenProc, this section gives a very quick overview of the different kinds of geographical data that you can use with the tool. Figure 1.3 shows examples of the different types. There are two categories of geographical data that scenProc can use:

- **Raster data** is geographical data that comes in the form of an image or a regular grid of numbers. Examples of this is an aerial imagery like you would use for your photo scenery or an elevation file specify the terrain height for a given area using a regular grid.
- **Vector data** is the kind of geographical data that is used to generate a map from, it describes the world using polygons, line and points. This is the main type of geographical data that you will use in scenProc, as most kind of autogen objects are most easily generated from vector data.

For the vector data you can encounter three types of vector features, these are:

- **Polygon features** are used to describe a enclosed area in the real world. Examples of features that are best described by a polygon are a forest or the industrial area of a town.
- **Line features** are used to describe objects in the world that are linear. The best example is probably roads, but line features can also describe rivers or power lines.
- **Point features** describe a single point in the real world. These are often used to mark the location of an object, for example the location of a church.

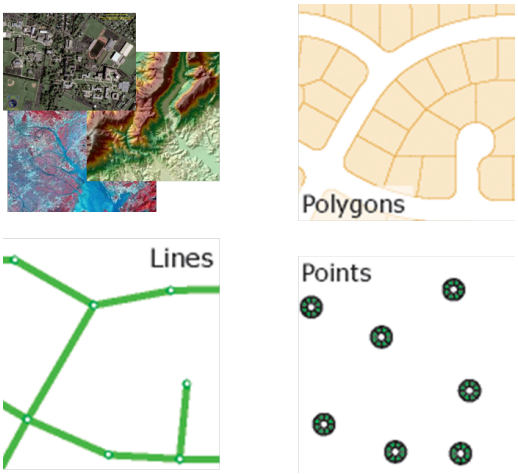



Figure 1.3: Different types of geographical data. Showing raster data and the three types of vector data (polygons, lines and points)

All types of vector features can come with additional attributes that provide additional information about the object being described by the feature. For example there can be attributes to specify what kind of trees there are in a forest or what the width of the road is. Which attributes are used is determined by the producer of the data, so you will have to examine the data you are using to find out which attributes it contains. When creating autogen these attributes are very important, as they will help you in generating more realistic autogen.

# Chapter 2

# Installation

## 2.1 Prerequisites

To be able to run scenProc you need to make sure that the following components have been installed on your computer. If you don't have them installed you can download them from the provided links.

- Visual C++ Redistributable for Visual Studio 2015. If you are running the 32 bit version of scenProc you need the 32 bit version of the runtime files, if you are running the 64 bit version of scenProc you need the 64 bit version of the runtime files.

Note: previous versions had a prerequisite on a specific .NET version as well. scenProc is now deployed as a self-contained application, which means that all required runtimes files are part of the application. Therefore it is no longer necessary to install a specific .NET version on your computer before running scenProc.

## 2.2 Installation

This manual described the development release version 3.3 of scenProc. You can also get a stable release version 3.2 without the latest changes, but that might contain less bugs.

scenProc doesn't come with an installer, so to install the tool you just have to unzip the file that you downloaded to a folder of your choice. That's all!

## 2.3 Configuration files

In case you have issues running scenProc, it can sometimes help to remove the configuration files. Sometimes they get corrupted. You can find them in the following folder:

`C:\Users\{username}\AppData\Local\SceneryDesign.org`

In that folder you will find a folder who's name starts with scenProc. There is where the configuration files are stored. Deleting the folder will revert scenProc back to the default settings.

# Chapter 3

# Basic workflow

This chapter describes the workflow of using the scenProc tool. It will cover the graphical user interface, the scenProc configuration file and how filtering of objects works. Just a warning ahead of time, scenProc is not a super easy to use tool. That's because it is a flexible tool that can process data with many different structures. But this means that you, as a user have to configure the tool to work correctly with the data you are using. But there are many features build into the tool to help you make that configuration.

## 3.1 The graphical user interface

The heart of scenProc is a configuration file that tells the tool how to process your data. This configuration file is just a simple text file. But around that configuration file a graphical user interface has been designed that helps you to write it quickly. This graphical user interface is shown in Figure 3.1. It consists of three main components:

- A toolbar with buttons for common actions. This offers the way to create, save and load configuration files, to start processing the configuration and to set the options.
- A text editor part where you edit the configuration file. The text editor has features like auto completion and tooltips showing help. When you put your mouse over a processing step, the tooltip will show the help text for that step. If you put it over a GUID the friendly name of the GUID will be shown as tooltip.
- An event log that shows information while scenProc is processing your data and also tells you when the configuration file is not correct.

In the toolbar at the top of the scenProc window you find the following buttons:

- **New** creates a new empty scenProc configuration file and will ask you for the filename and location where to save this file.
- **Open** opens an existing scenProc configuration file. If you press the arrow button next to the label you will see a list of the recently opened files.
- **Save** saves the current configuration file.
- **Save as** saves the current configuration file with a different filename.
- **Undo** button to undo changes made to the configuration file.
- **Redo** button to redo changes made to the configuration file.
- **Scan attributes** will scan the current configuration and update the auto completion data with the attributes specified in the data you are using. This makes it easier to create correct filters for your steps. But scanning the attributes can take some time for big data files. For more information about using attributes to filter objects see section 3.5.

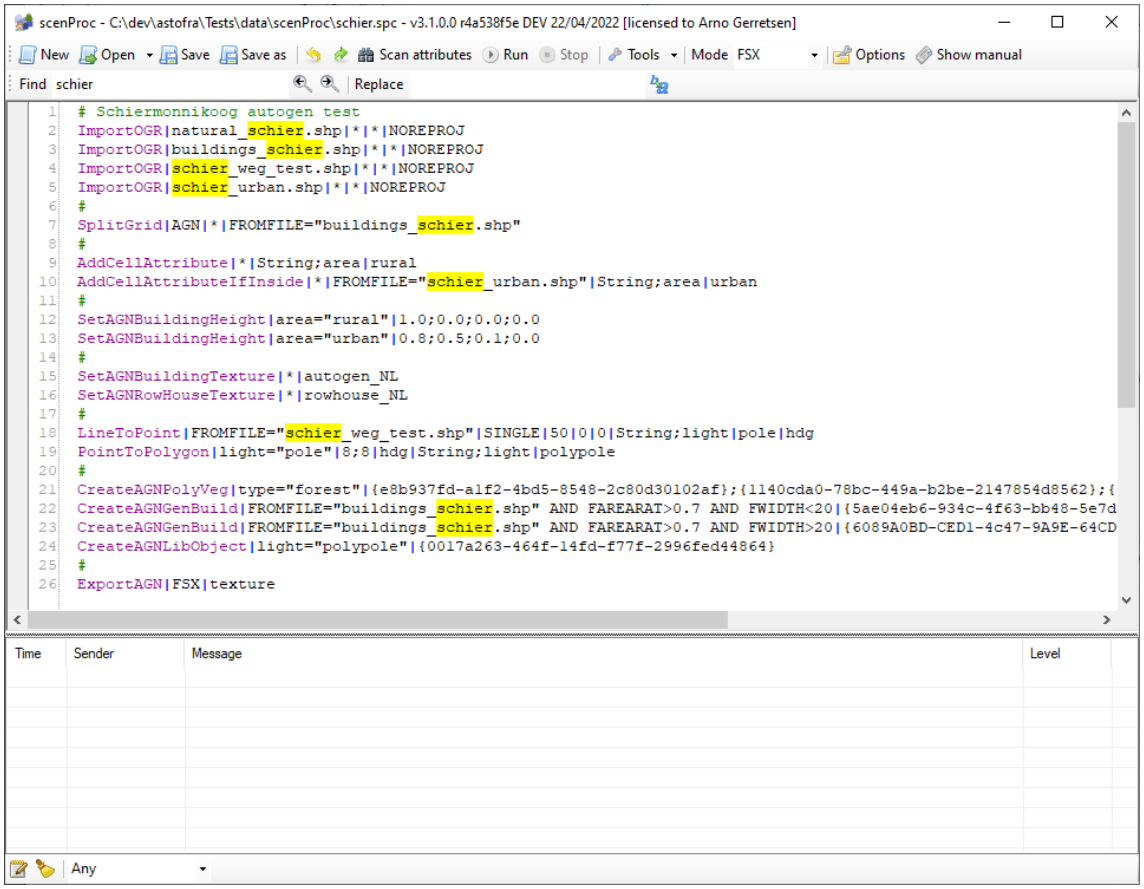



Figure 3.1: scenProc user interface

- **Run** will process the current configuration and start producing the scenery that you have specified. Depend on the amount of data you are using and the complexity of your configuration this might take some time.
- **Stop** will stop the processing of the configuration file that is currently running. This is done by restarting scenProc.
- **Tools** menu shows different tools to perform special actions. The following tools are available:
  - Texture Filter Editor, see chapter 6 for more details.
  - Autogen Merge Tool, see chapter 8 for more information about merging autogen files.
  - LOD & QMID Calculator, which allows you to determine the LOD tile name and extends based on a given location and given LOD or QMID size. See Figure 3.2 for a screenshot of this calculator.
  - Building Texture Configuration Editor, which allows you to create and edit the Building Texture Configuration files used by 3D buildings, see section 7.4 for more details.
- **Mode** is a combobox where you can select in which mode scenProc is running. The mode is the FS version that you are targeting and it determines from which FS version the autogen configuration files will be read for showing auto completion in the GUI. You can either choose FS2004, FSX, Prepar3D v1, Prepar3D v2, Prepar3D v3 or Prepar3D v4 mode. It doesn't influence how the scenProc configuration is processed.
- **Options** opens the dialog where you can set the scenProc options. For more details see section 3.3.
- **Open manual** opens this user manual.

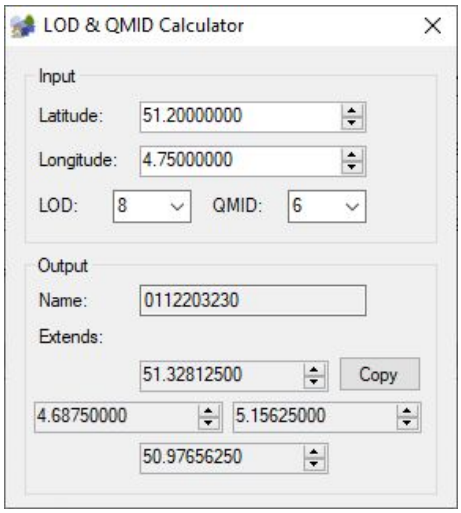



Figure 3.2: LOD & QMID Calculator

Below the toolbar you find the search and replace text boxes. You can use these to find certain words in your configuration file, to navigate to next of previous matches of the searched text and to replace the text with another text.

## 3.2 Keyboard shortcuts

The following keyboard shortcuts are available in scenProc to help you perform tasks quicker:

| | |
|---|---|
| Ctrl-N | New configuration file |
| Ctrl-O | Load configuration file |
| Ctrl-S | Save configuration file |
| Ctrl-F | Focus to search textbox |
| Ctrl-H | Focus to replace textbox |
| Ctrl-R | Run configuration file |
| Ctrl-T | Run scan attributes on configuration file |
| F3 | Next search result |
| Shift-F3 | Previous search result |

Table 3.1: Keyboard shortcuts

## 3.3 Options

When you press the Options button in the toolbar the options form will show where you can set different options for the scenProc tool. Figure 3.3 shows the options form.

The following options can be set from this form:

**Auto completion supported FS versions**

For the auto completion in the GUI scenProc needs to know where the supported versions of Flight Simulator are installed. In the top section of the options form you can specify these locations for FS2004, FSX, Prepar3D v1, Prepar3D v2, Prepar3D v3, Prepar3D v4, Prepar3D v5 and Prepar3D v6. For the versions you are using you can enter the installation path. With the checkboxes in front you can indicate if this version is used or not (when disabled, it will not be shown in the mode combo box where you select which version is used for the auto completion). The radio button can be used to select the default version when you have multiple enabled.

**Tool paths**

The second group of options is to specify the path to various tools used by scenProc. This includes the BGLComp tool used in the ExportBGL step, the MakeMDL and XtoMDL tool used when compiling MDL files, the shp2vec compiler used by the ExportTVecBGL step, the resample compiler used by the ExportResampleElevationBGL and ExportResamplePhotorealBGL steps and the MSFS fspackagetool compile is used by the ExportMSFS step.

For some of these tools you can specify the path for different FS versions, so make sure to set the path for the FS versions that you will be selecting in the ExportBGL step.

**AF2 path**

The location where the AeroFly 2 (AF2) simulator is installed.

**Other options**

There are also other options you can specify:

- **Check if options are valid** determines if scenProc will check if the entered paths in the options are valid or not at startup of the tool.
- **Check if a newer version of scenProc is available** will check online if updates have been released compared to the version you are running.
- **Include unstable releases** means that not only stable scenProc releases, but also the development releases are reported when checking for updates. This only applies when running a stable release of scenProc.
- **Use experimental steps** enables some steps that are not fully developed and still in experimental stage. These are not documented in general and should be used at own risk.

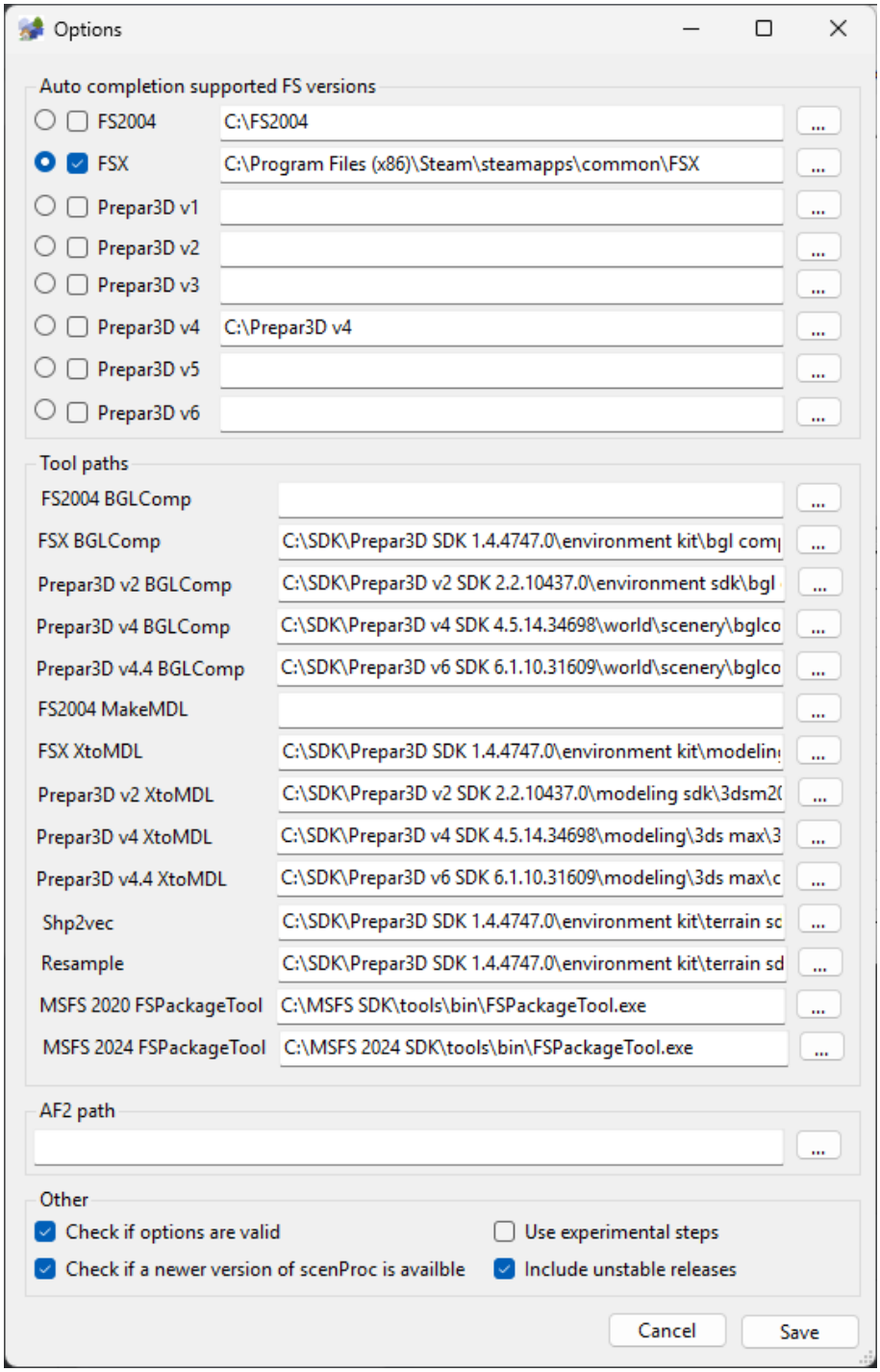



Figure 3.3: scenProc options

## 3.4 Configuration file

The configuration file is the most important part of scenProc, since it tells the tool what it should do. So most of your work as a user will be to define this configuration file correctly. This section explains the syntax used for the configuration file and other things you need to know.

The configuration file is a simple plain text file, so this means you can edit it with any text editor. Although it is advised to use the scenProc graphical user interface, since that will give you all kinds of tools and helper functions to make editing the configuration file easier. By convention it the configuration file has the SPC extension (**s**cen**P**roc **c**onfiguration), but you can use another extension if you like.

All lines starting with a hash (#) are comments and are ignored when processing your configuration file. Each lines starts with the name of the step you want to perform, followed by the arguments required for that step. The different arguments are separated with a pipe (|) symbol.

See chapter 10 for an overview of all available steps.

### 3.4.1 Auto completion

One of the features build into the scenProc tool to help you make your configuration file is auto completion. This means that while you type the tool will show you possible steps or arguments that you can enter. If you press enter after selecting the right entry from the menu shown the text will be automatically completed for you.

Figure 3.4 shows how the auto completion works while you add a new step to the configuration file. While you are typing the name of the step, the auto completion will show you all possible steps that match the name you have typed. You can select the step you want from the popup menu and once you press enter the entire step, plus a template of all arguments is added to your configuration file. This result is shown in Figure 3.5.

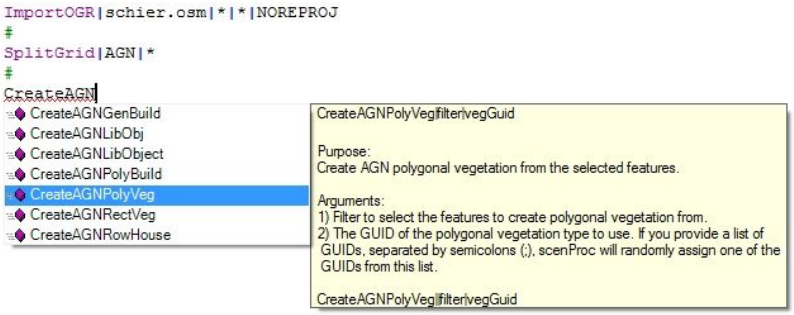



Figure 3.4: Auto completion while adding a new step to a configuration

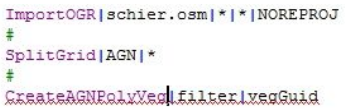



Figure 3.5: The completed line of the select step

The auto completion not only works when entering new steps, it also helps to the enter the right arguments for a step. Some examples of this are:

- When you are using geographical data that is not using geodetic coordinates you need to tell scenProc which coordinate system (projection) is used by the data. scenProc uses Proj4 style definition strings for this. But these can be quite complex to type. So therefore you can also start typing their name (for example WGS 84 / UTM Zone 31 or Amersfoort / RD). The

auto completion will show you the available projections, as shown in Figure 3.6. After you select the projection you want the correct Proj4 string will be inserted in your configuration. This is shown in Figure 3.7.




Figure 3.6: Auto completion while selecting a projection argument

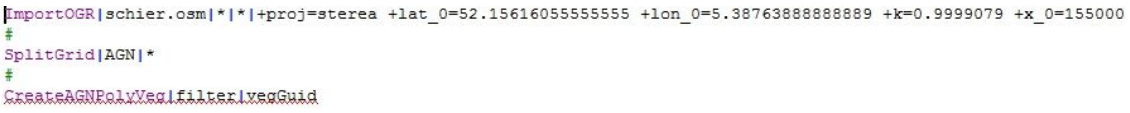



Figure 3.7: The complete projection string filled in as argument

- When you are typing a filter to select your features (for more details see section 3.5) the auto completion can help you by showing the possible attributes and values. You need to press the Scan attributes button to make scenProc aware of all attributes. Else the auto completion will not show all attributes and possible values present in your data. Figure 3.8 shows an example of auto completion while typing a filter.

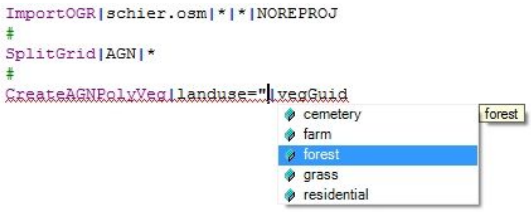



Figure 3.8: Auto completion while typing a filter

- When selecting a GUID for vegetation, roofs or objects the auto completion can also help you. Instead of typing in the hexadecimal GUID, you can just start to type the name that describes the vegetation, roof or object, see Figure 3.9 After you have selected the right entry from the list, the GUID will be inserted automatically in your configuration file, see Figure 3.10.

### 3.4.2 Tooltips

When you hover with your mouse over certain elements in the graphical user interface you will see tooltips that provide you useful information. Examples of the tooltips are:

- A detailed description about what a step does and which arguments it takes if you put your mouse over the name of the step in the configuration file, see Figure 3.11
- The friendly name or description of a GUID, when you put your mouse over the GUID, see Figure 3.12
- Details about an attribute, including the attribute type and possible values, when you put your mouse over an attribute name. You must first press the Scan attributes button for scenProc to have all the data to show in this tooltip. See Figure 3.13.

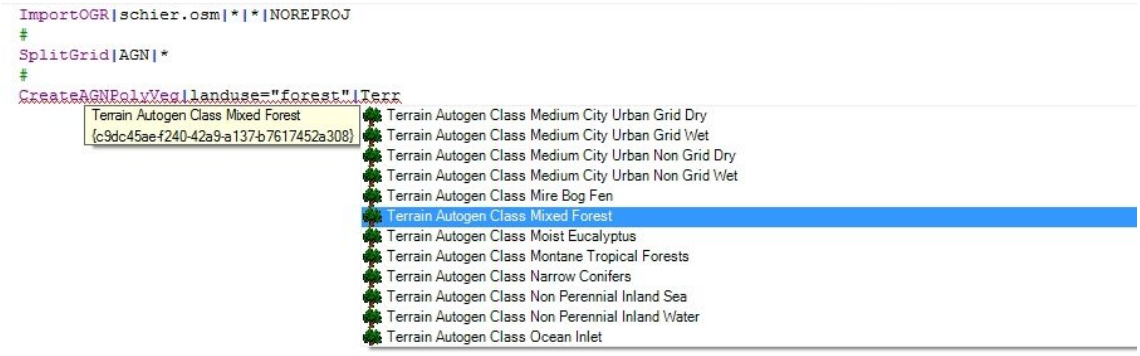



Figure 3.9: Auto complation while selecting a vegetation GUID

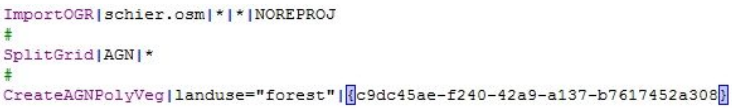



Figure 3.10: The selected GUID in the configuration file

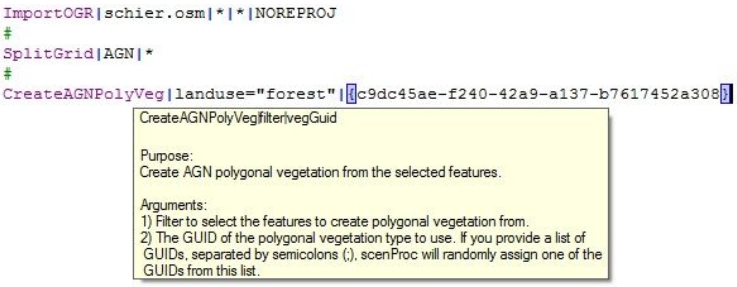



Figure 3.11: Tooltip showing help about a step

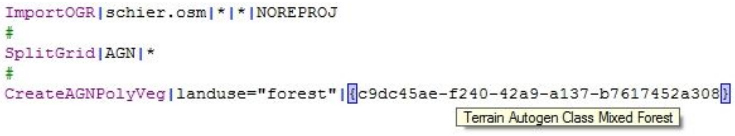



Figure 3.12: Tooltip showing the description of a GUID

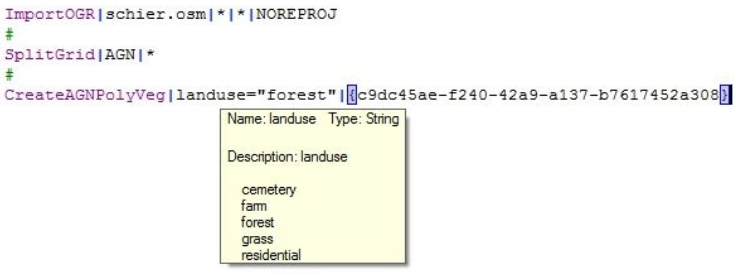



Figure 3.13: Tooltip showing attribute details

### 3.4.3 Validation

While you are typing your configuration file in the text editor, scenProc will automatically validate the configuration file. This means that if there is a mistake in the configuration, you will see a message in the event log at the bottom of the screen. See Figure 3.14 for a screenshot of how the validation errors can look. Examples of mistakes that are checked by the validation are:

- A step doesn't have the right amount of arguments
- A file or folder specified does not exist
- A step is only allowed after another step
- A GUID is not specified in the right format
- A filter is not valid

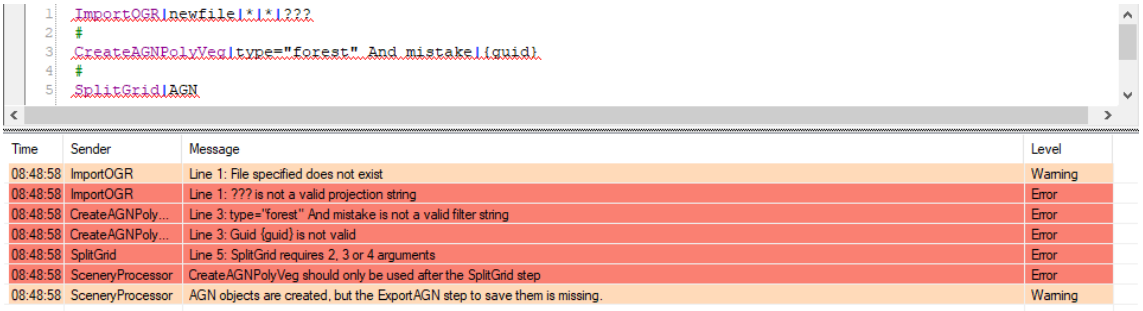



Figure 3.14: Errors while validating the scenProc configuration

As long as there are validation errors in your configuration file it is not possible to run the configuration through scenProc. You first have to fix those errors. With the information given in the event log it shouldn't be too hard to find them though.

### 3.4.4 Conditional steps

You can also make lines execute conditionally in the configuration file. This is done by placing them between `\%if` and `\%ifend` statements. The `\%if` statement can check the value of a user defined variable and if it is equal to the given value. Below is an example that will only load the second file when the first user variable has a value of 2.

```
# @0@=1
#
ImportOGR|file1.shp|*|*|NOREPROJ
#
%if(@0@=2)
ImportOGR|file2.shp|*|*|NOREPROJ
%ifend
```

If you separate the condition by a `|` symbol you can also check if one of these conditions is true (so it acts as an OR condition). The example below checks if the user variable is either 3, 5 or 7.

```
# @0@=1
#
ImportOGR|file1.shp|*|*|NOREPROJ
#
%if(@0@=3|@0@=5|@0@=7)
ImportOGR|file2.shp|*|*|NOREPROJ
%ifend
```

## 3.5 Filtering objects

In many cases you will have imported a lot of features into scenProc, but in most cases you don't want them all to become the same type of autogen. So you need to tell scenProc which features should be used to generate vegetation and which to generate buildings. That's why many processing steps have a argument that specifies the filter that should be applied to select the correct objects to process. This section explains how the filtering works.

If you don't want to filter the objects at all and would like to use all available objects, you can just enter a asterisk (*) to select all features in the filter argument. This will select all applicable objects. Some steps have build in logic to select only polygons or lines, if they require some features as input. There are also some steps where you might want to select no features at all, in that case you can enter NONE.

The most common way to filter is by checking if an attribute of the feature has the required value. For example when making forest vegetation your filter might be:

```
landuse="forest"
```

Here `landuse` is the name of the attribute and we will only use features where this attribute has the value `forest`. Be aware that you need to put quotes around string values, else they will be interpreted as attribute names. Also be aware that attribute and value names are case sensitive in scenProc, so `type` and `Type` are not the same attributes.

If you want to select all features that contain a specific attribute, with any value for that attribute you can use the following filter. The `"*"` value can be used for string, integer and double attribute types.

```
landuse="*"
```

If you want to do the opposite and select all attributes that don't have a specific attribute you can use the following filter:

```
NOT landuse="*"
```

Besides the is equal operator you can also list the other operators given in Tabel 3.2.

| | |
|---|---|
| Equal | = |
| Not equal | <> |
| Greater | > |
| Greater than | >= |
| Smaller | < |
| Smaller than | <= |

Table 3.2: Filter operators

Often you want a more complex filter by combining different attributes. You can do this by using the `And`, `Or` or `Not` statement to combine different conditions. So a more complex forest filter might be:

```
landuse="forest" And height>3 And tree_type="oak"
```

Or if you want to select all oak and pine trees you could write:

```
tree_type="oak" Or tree_type="pine"
```

Besides the attributes that are present in the geographical data you imported, there are some special attributes that scenProc adds. Tabel 3.3 shows which special attributes are available. So for example to select all building polygons that are longer than 20 meters you could use this filter:

```
type="building" And FTYPE="POLYGON" And FLENGTH>20
```

| Attribute | Type | Description |
|---|---|---|
| FAREA | Double | Area of the minimum area polygon around the objects (in square meters) |
| FCIRCUM | Double | The circumference of the feature (in meters), for line features this is implemented as the length of the line |
| FCONVEX | Integer | 1 if polygon is convex, 0 if polygon is concave |
| FHEADING | Double | Heading of the minimum area polygon around the object |
| FLAYER | String | Layer from which the feature was read (not all file formats support layers) |
| FLENGTH | Double | Length of the minimum area polygon around the object (in meters) |
| FMAXSIDE | Double | The length of the longest edge of the feature (in meters) |
| FMINSIDE | Double | The length of the shortest edge of the feature (in meters) |
| FNUMNOTPAR | Integer | The number of edges that do not have another edge that is parallel |
| FNUMPERPANG | Integer | The number of 90 degree angles in the feature |
| FNUMVERT | Integer | The number of vertices in the feature |
| FRAND | Double | Random number between 0.0 and 1.0 |
| FROMFILE | String | Name of the source file that the feature comes from (without path) |
| FROMPATH | String | Full path of the file that the feature comes from |
| FSHAPE | String | Shape of the feature (SQUARE, RECTANGLE, LSHAPE, CIRCLE or OTHER) |
| FTYPE | String | Type of feature (POINT, LINE, POLYGON or RASTER) |
| FWIDTH | Double | Width of the minimum area polygon around the object (in meters) |

Table 3.3: Available special attributes

If you want to do really complex filter you can also perform math within the filter. The basic mathematical operators and functions are supported. So if you want to select all buildings that are taller than the given vegetation height in an area you could use:

```
3.0 * numfloors > veg_height
```

The whole filtering processing explained above focuses on selecting which features to use. But there are a number of scenProc steps where a filter can be used to select from the gridcells instead of from the features. The same logic as above applies in this case, except for the special attributes that don't exist for gridcells.

# Chapter 4

# Checking your geographical data

A common problem for scenProc users is that they take one of the sample configurations found here in the manual (see Chapter 5) or found on the forum and that it doesn't work for their area. In my cases this is caused by the fact that the data for their area uses different attributes than the sample assumes and therefore the filters don't match the features and no output is created. Therefore this chapter will discuss how the check your geographical data to see which attributes it uses. With this knowledge you can then adjust your configuration file accordingly.

Different attributes are usually caused by different sources of your data. For example each national or regional geographical information office typically has its own attribute schema's for their datasets. So if I download data for the Netherlands it will have different attributes than data I download for the United States. But once I know what attributes are there, I can make sure the autogen is created as desired.

You might think that OpenStreetMap data does not have this problem, as it all comes from the same source. But unfortunately that's only partly true. The OSM XML datasets always use the same attributes, as found on the OSM wiki. But if you are using a Shapefile export of the OSM data, then you will encounter different attributes depending on the source of the Shapefile dump. And it's always good to check the OSM data you are working with, as the attributes you found on the wiki might not be used in your area.

## 4.1 GIS tool

To check the geographical data you will need to use a tool that can load and display this kind of data. Such tools are called GIS tools in general. A very commonly used tool for professional GIS users is ArcGIS from ESRI, but since this is a very expensive commercial tool I would not recommend it for your flight simulator developments.

My personal favorite tool is the open source Quantum GIS (QGIS) tool. It's a very powerful tool to load, view and edit geographical data. Another GIS tool popular for flight simulator developers is GlobalMapper, which is a commercial tool, but with a much more affordable price.

In the rest of this chapter I will use QGIS to example how you can check your geographical data. But the principles I show here will also apply to the alternative GIS tools.

## 4.2 Example: landuse attributes

In this section I will show how you can check geographical data using an example. Since many example configurations use OSM data, I will use a very different source in this example so you can

see the variation possible. In this example we will be looking at some landuse data for the island of Nantucket in the state of Massachusettsts. I downloaded this data from the GIS department of the state.

The first place to look for information on the attributes that the data uses is of course at the source. So try to find some information on the website where you found the data or inside the file that you downloaded. In this case I was lucky, as the website contained information about the attributes used. See Figure 4.1. So at least I know which attribute name I should be looking for in my data. Even better is that the website also shows a list of possible values for the attributes and their description, part of this information is shown in Figure 4.2.

**Attributes**

The layer's polygon attribute table contains the following items:

| | |
|---|---|
| **LU05_DESC** | Land Use Description (text) |
| **LUCODE** | Land Use Code (short integer) |

Figure 4.1: Information about the attributes of the landuse data as found on the website

**Land Use Code Definitions**

| Land Use Code | Land Use Description | Detailed Definition |
|---|---|---|
| 1 | **Cropland** | Generally tilled land used to grow row crops. Boundaries follow the shape of the fields and include associated buildings (e.g., barns). This category also includes turf farms that grow sod. |
| 2 | **Pasture** | Fields and associated facilities (barns and other outbuildings) used for animal grazing and for the growing of grasses for hay. |
| 3 | **Forest** | Areas where tree canopy covers at least 50% of the land. Both coniferous and deciduous forests belong to this class. |
| 4 | **Non-Forested Wetland** | DEP Wetlands (1:12,000) WETCODEs 4, 7, 8, 12, 23, 18, 20, and 21. |
| 5 | **Mining** | Includes sand and gravel pits, mines and quarries. The boundaries extend to the edges of the site's activities, including on-site machinery, parking lots, roads and buildings. |
| 6 | **Open Land** | Vacant land, idle agriculture, rock outcrops, and barren areas. Vacant land is not maintained for any evident purpose and it does not support large plant growth. |
| 7 | **Participation Recreation** | Facilities used by the public for active recreation. Includes ball fields, tennis courts, basketball courts, athletic tracks, ski areas, playgrounds, and bike paths plus associated parking lots. Primary and secondary school recreational facilities are in this category, but university stadiums and arenas are considered Spectator Recreation. Recreation facilities not open to the public such as those belonging to private residences are mostly labeled with the associated residential land use class not participation recreation. However, some private facilities may also be mapped. |
| 8 | **Spectator Recreation** | University and professional stadiums designed for spectators as well as zoos, amusement parks, drive in theaters, fairgrounds, race tracks and |

Figure 4.2: Information about possible values of the landuse attribute as found on the website

With just this information from the website I could already construct my scenProc configuration file. But in general it is better to have a look at the data for your specific area, since the attributes might be used differently than you expect. For example when different vegetation categories are

available, different areas can have different amount of detail in the amount of categories used. So I always suggest that you look at the data in a GIS tool as well.

So let's go ahead and load the data in QGIS. You will see something like shown in Figure 4.3. This will just show you the data and gives you can idea of the resolution. If you see many small polygons the data will likely have a good granularity, if you mainly see big chunky polygons the data might not fit with your photo scenery that well.

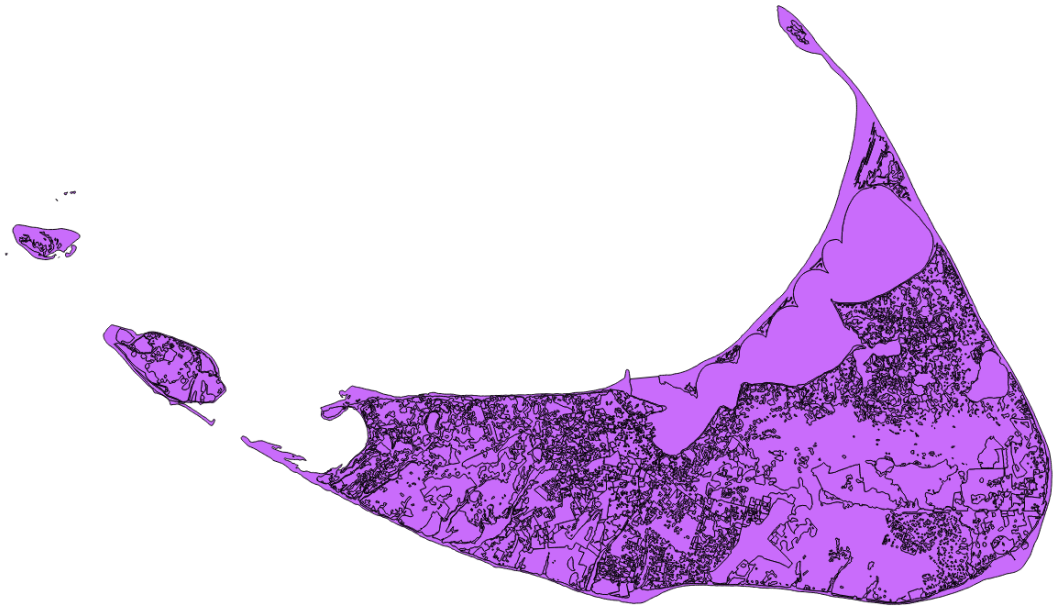


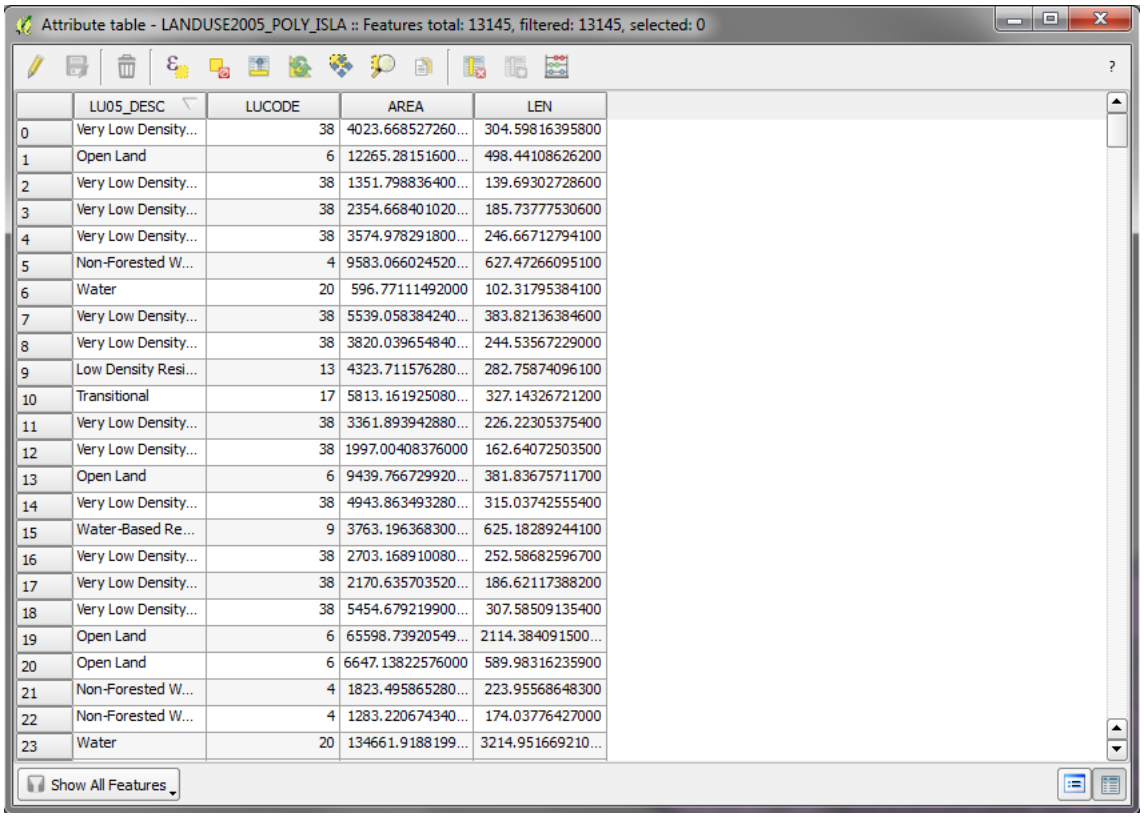
Figure 4.3: The landuse data loaded into QGIS

But to get an idea of the attributes used you need to dive a bit deeper. First right click on the data layer and select the attribute table option. This will show a table with all features and their attributes. Figure 4.4 shows part of the table for this landuse data. As you can see the LUCODE attribute as described in the documentation is there and the LU05_DESC attribute also gives a description of the type of landuse. So makes it quite easy to see what each polygon is. You won't find such a description in every dataset unfortunately.

So by going over the attribute table you will get a good idea of which attributes from the documentation are actually used in your area and which are not. But there is a much better way to check this and that is to let the GIS tool render the data with different colours for each attribute. In QGIS you can do this by double clicking on the layer and then adjust the style. You can category based on attribute values, Figure 4.5 shows the settings I use in this example to assign a random colour to each attribute type.

In Figure 4.6 you see the resulting map, which gives a good idea of the attributes that are used in the area. For example it's easy to spot where the forest polygons are that you could use for the autogen. And if you need more detail you can always click on the features to see all their detail.

After looking at the data you will have a much better idea of the attributes that you need to use in your scenProc configuration file when you create your autogen. So that should make it easier to make the autogen as you want.




| | LU05_DESC | LUCODE | AREA | LEN |
|---|---|---|---|---|
| 0 | Very Low Density... | 38 | 4023.668527260... | 304.59816395800 |
| 1 | Open Land | 6 | 12265.28151600... | 498.44108626200 |
| 2 | Very Low Density... | 38 | 1351.798836400... | 139.69302728600 |
| 3 | Very Low Density... | 38 | 2354.668401020... | 185.73777530600 |
| 4 | Very Low Density... | 38 | 3574.978291800... | 246.66712794100 |
| 5 | Non-Forested W... | 4 | 9583.066024520... | 627.47266095100 |
| 6 | Water | 20 | 596.77111492000 | 102.31795384100 |
| 7 | Very Low Density... | 38 | 5539.058384240... | 383.82136384600 |
| 8 | Very Low Density... | 38 | 3820.039654840... | 244.53567229000 |
| 9 | Low Density Resi... | 13 | 4323.711576280... | 282.75874096100 |
| 10 | Transitional | 17 | 5813.161925080... | 327.14326721200 |
| 11 | Very Low Density... | 38 | 3361.893942880... | 226.22305375400 |
| 12 | Very Low Density... | 38 | 1997.00408376000 | 162.64072503500 |
| 13 | Open Land | 6 | 9439.766729920... | 381.83675711700 |
| 14 | Very Low Density... | 38 | 4943.863493280... | 315.03742555400 |
| 15 | Water-Based Re... | 9 | 3763.196368300... | 625.18289244100 |
| 16 | Very Low Density... | 38 | 2703.168910080... | 252.58682596700 |
| 17 | Very Low Density... | 38 | 2170.635703520... | 186.62117388200 |
| 18 | Very Low Density... | 38 | 5454.679219900... | 307.58509135400 |
| 19 | Open Land | 6 | 65598.73920549... | 2114.384091500... |
| 20 | Open Land | 6 | 6647.13822576000 | 589.98316235900 |
| 21 | Non-Forested W... | 4 | 1823.495865280... | 223.95568648300 |
| 22 | Non-Forested W... | 4 | 1283.220674340... | 174.03776427000 |
| 23 | Water | 20 | 134661.9188199... | 3214.951669210... |

Figure 4.4: The attribute table of the data in QGIS

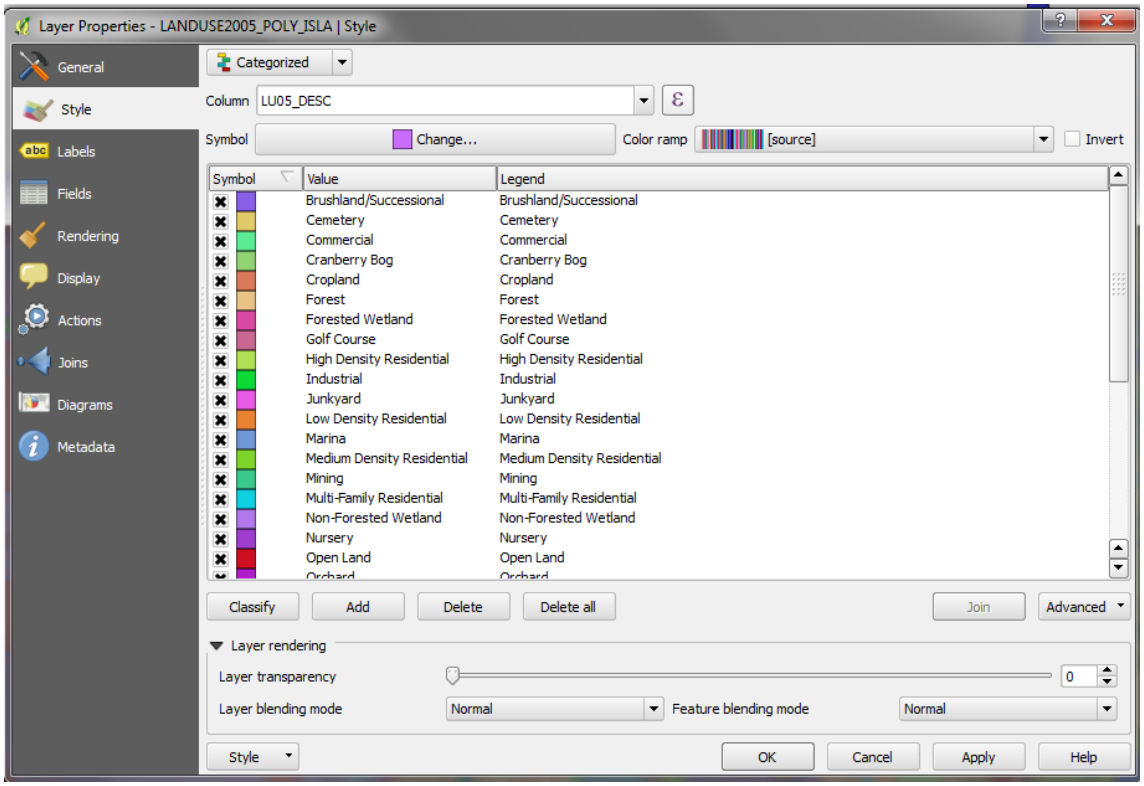



Figure 4.5: The style of the landuse layer used to render the attributes with different colours in QGIS

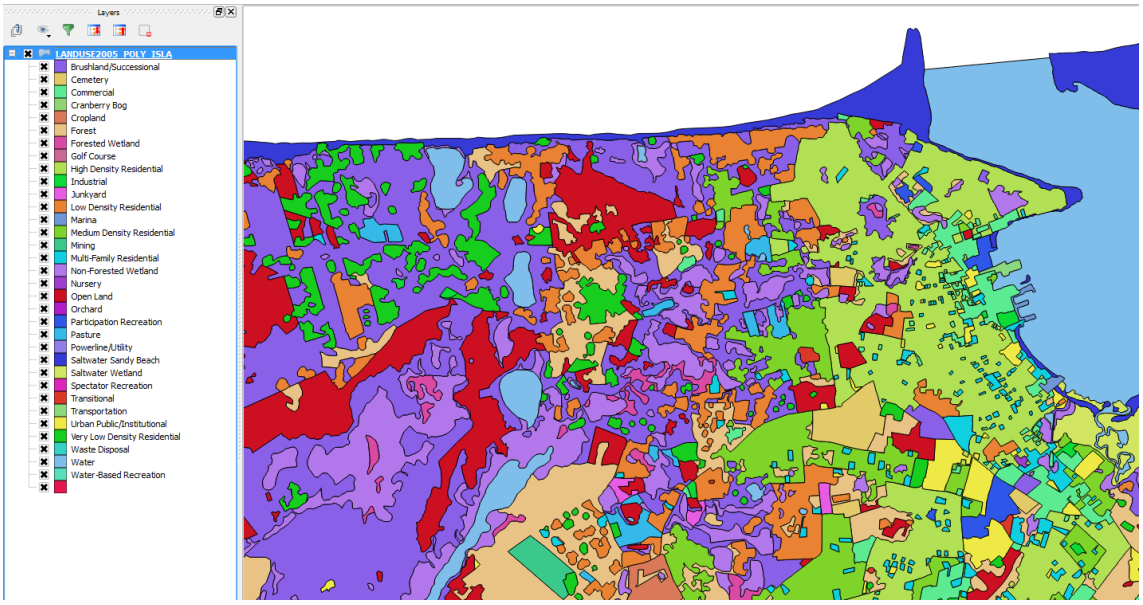



Figure 4.6: The attribute used to render the polygons with different colours in QGIS

# Chapter 5

# Common configurations

In this chapter some common configurations are given as example. You can use these as a starting point for your own configuration file. Be aware that you usually need to adapt the configuration file to the data you are using, so just copying one of the provided examples does not guarantee that you will get working autogen.

## 5.1 Autogen from OpenStreetMap XML files

In this section a sample configuration file is given that creates polygonal vegetation and buildings based on a OpenStreetMap XML file (these usually have the OSM extension).

```
ImportOGR|schier.osm|*|building;landuse|NOREPROJ
#
SplitGrid|AGN|*|building="*"
#
CreateAGNPolyVeg|FTYPE="POLYGON" And landuse="forest"|{e8b937fd-a1f2-4bd5-8548-2
    ↪ c80d30102af}
#
CreateAGNGenBuild|building="*" And FWIDTH<20|{5ae04eb6-934c-4f63-bb48-5
    ↪ e7dee601212}|MAXRATIO=2
CreateAGNGenBuild|building="*" And FWIDTH>20|{6089A0BD-CED1-4c47-9A9E-64
    ↪ CDD0E16983}
#
ExportAGN|FSX|texture
```

In the `ImportOGR` step it is specified that only features that have an attribute of the category building or landuse are imported. Since these are the only attributes we will need to create the autogen. To speed up things and save memory all other features are not imported (for example roads, points of interest, etc).

In the `SplitGrid` step the last attribute specifies that we don't want to split features of the category building. This is done because else you would get very small building segments. So instead building features are put in the grid cell where their center falls into.

Based on the width of the buildings either a gabled roof or a flat roof is applied in the `CreateAGNGenBuild` step. For the gabled roofs the extra `MAXRATIO` option is provided. This means that long, but narrow buidlings are split into multiple individual buildings. For rows of houses that usually looks better.

## 5.2 Autogen from OpenStreetMap SHP files

In this section a sample configuration file is given that creates polygonal vegetation and buildings based on a OpenStreetMap data downloaded in the Shapefile format from the GeoFabrik website. Part of the configuration file is similar to the one given in section 5.1, so please check the explanation there as well. Only the specific changes will be explained here in more detail.

```
ImportOGR|natural.shp|*|*|NOREPROJ
ImportOGR|buildings.shp|*|*|NOREPROJ
#
SplitGrid|AGN|*|FROMFILE="buildings.shp"
#
CreateAGNPolyVeg|FTYPE="POLYGON" And type="forest"|{e8b937fd-a1f2-4bd5-8548-2
    ↪ c80d30102af}
#
CreateAGNGenBuild|FROMFILE="buildings.shp" And FWIDTH<20|{5ae04eb6-934c-4f63-bb48
    ↪ -5e7dee601212}|MAXRATIO=2
CreateAGNGenBuild|FROMFILE="buildings.shp" And FWIDTH>20|{6089A0BD-CED1-4c47-9A9E
    ↪ -64CDD0E16983}
#
ExportAGN|FSX|texture
```

The main difference is that different attributes are used to select the forest and building features. There are also other sources than GeoFabrik for OpenStreetMap shapefiles, but each of these sources has slightly different attributes. So be sure to always check your data structure, so that you can create the correct filter.

Another difference is that in this example the buildings don't have an attribute that they can be identified with. Therefore in the feature a special attribute is used that checks from which file the feature has been loaded.

## 5.3 Autogen for FS2004

This example shows how to create FS2004 autogen from a set of shapefiles that each contain a specific feature type.

```
ImportOGR|vegetation.shp|*|*|NOREPROJ
ImportOGR|libobj.shp|*|*|NOREPROJ
ImportOGR|building.shp|*|*|NOREPROJ
#
SplitGrid|AGN|*|FROMFILE="building.shp"
#
SetAGNVegetattionSettings|*|6|5|0.5|1.0|6;12|5;20
#
ReplacePolygonByVegetationRectangles|FROMFILE="vegetation.shp"|0.00025|-1.0|0.25
#
CreateAGNRectVeg|FROMFILE="vegetation.shp"
CreateAGNGenBuild|FROMFILE="building.shp"|{00000000-0000-0000-0000-000000000000}
CreateAGNLibObject|FROMFILE="libobj.shp"|{82be3f0e-40dc-6365-679d-45ac6fde6872}
#
ExportAGN|FS2004|texture
```

The main differences with the configurations for FSX shown above is that you need to use the `SetAGNVegetattionSettings` step to define the type of vegetation you want, that's not done with a GUID anymore. Therefore in the `CreateAGNRectVeg` step no GUID is used. When creating buildings with the `CreateAGNGenBuild` step the roof type is also fixed already, so the GUID there has no influence.

Another difference is that you can't use polygonal vegetation, they need to be rectangular. That's why the `ReplacePolygonByVegetationRectangles` step is used to make multiple rectangles that match the vegetation polygon.

## 5.4 Identify different building shapes

One of the challenges when processing building footprints into autogen buildings is how to deal with footprints that are not perfectly rectangular. The configuration below shows how more complex buildings can be setup with different roof types.

```
ImportOGR|..\geo\osm\nantucket.osm|*|building;highway|NOREPROJ
#
SplitGrid|AGN|building="*"
#
# Add attribute to indicate the type of building
# 1 = ALMOST RECTANGLE (BASED ON AREA RATIO)
# 3 = REGULAR SHAPED (MANY PARALLEL EDGES)
# 4 = CONVEX POLYGONS
# 5 = CONCAVE POLYGONS
AddAttribute|FTYPE="POLYGON" And building="*"|Integer;BUILDTYPE|5
AddAttribute|FTYPE="POLYGON" And building="*" And BUILDTYPE=5 And FAREARAT>0.80|
    ↪ Integer;BUILDTYPE|1
AddAttribute|FTYPE="POLYGON" And building="*" And BUILDTYPE=5 And FNUMVERT<10 And
    ↪  FNUMPERPANG>3 And FNUMNOTPAR<2|Integer;BUILDTYPE|3
AddAttribute|FTYPE="POLYGON" And building="*" And BUILDTYPE=5 And FCONVEX=1|
    ↪ Integer;BUILDTYPE|4
#
# Classify industrial/commercial buildings
AddAttributeIfInside|FTYPE="POLYGON" And building="*"|LUCODE=16|String;BUILDCAT|
    ↪ INDUSTRIAL
AddAttributeIfInside|FTYPE="POLYGON" And building="*"|LUCODE=15|String;BUILDCAT|
    ↪ COMMERCIAL
#
# Add attribute for roof type
AddAttribute|FTYPE="POLYGON" And building="*" And FWIDTH>5|String;ROOFTYPE|
    ↪ PEAKED_ALL
AddAttribute|FTYPE="POLYGON" And building="*" And FWIDTH>5 And FLENGTH<12|String;
    ↪ ROOFTYPE|PEAKED_SIMPLE
AddAttribute|FTYPE="POLYGON" And building="*" And FWIDTH>20|String;ROOFTYPE|
    ↪ PEAKED_LOW_PITCH
AddAttribute|BUILDCAT="INDUSTRIAL"|String;ROOFTYPE|PEAKED_LOW_PITCH
AddAttribute|BUILDCAT="COMMERCIAL"|String;ROOFTYPE|PEAKED_LOW_PITCH
#
# Remove complex buildings
ReplacePolygonByBuildingRectangles|BUILDTYPE=3|0.8;4;4|0.25;2.0;0.5|Integer;
    ↪ BUILDTYPE|2
#
# Create buildings autogen
CreateAGNGenBuild|BUILDTYPE<3 And ROOFTYPE="PEAKED_ALL"|{c05c5106-d562-4c23-89b8-
    ↪ a4be7495b57c}|MAXRATIO=2
CreateAGNGenBuild|BUILDTYPE<3 And ROOFTYPE="PEAKED_SIMPLE"|{d4ee02a2-ed47-4f10-
    ↪ b98c-502516983383}
CreateAGNGenBuild|BUILDTYPE<3 And ROOFTYPE="PEAKED_LOW_PITCH"|{a9b0e686
    ↪ -0758-4294-a760-9bb4fd341621}
#
ExportAGN|FSX|..\..\texture
```

After loading the data first the footprints are categorized based on the complexity of the footprint shape. They are classified as (almost rectangular), regular shaped (with parallel edges), convex and concave polygons.

Next it is also checked if they are residential or industrial, by checking them against landuse polygons. This allows to select different roofs for industrial buildings for example. Also based on the size of the footprint and the building category an attribute is added to define the roof type we want to use.

With the `ReplacePolygonByBuildingRectangles` step the footprints that were classified as regular shaped are next replaced by a number of rectangles. In the final step autogen buildings are made for all buildings that are almost rectangular or have been processed by the `ReplacePolygonByBuildingRectangles` step. All other footprints, that are more complex in shape are ignored in this configuration.

## 5.5 Placing autogen library objects along a road

This script shows how to place point along a line (road) and then use these points to place autogen library objects of light poles.

```
ImportOGR|texel.osm|*|highway|NOREPROJ
#
LineToPoint|FTYPE="LINE" And highway="secondary"|CONTINUOUS|150|5|0|String;type|
    ↪ light|hdg
#
PointToPolygon|type="light"|8;8|hdg|String;type|lightpoly
#
CreateAGNLibObject|type="lightpoly"|{82be3f0e-40dc-6365-679d-45ac6fde6872}
#
ExportAGN|FSX|texture
```

The point features are created at 150 meter intervals for all roads classified as secondary. They are placed 5 meters out of the center of the road and the heading of the points is stored in an attribute named hdg.

Since autogen library objects are created from polygons, the points are next converted into polygons with a length and width of 8 meters (that's the size of the library class we want to use). In the next step autogen objects are created for all of the polygons. And finally they are exported to AGN files.

## 5.6 Placing XML library objects along a road

This script shows how to place point along a line (road) and then use these points to place XML library objects of light poles.

```
ImportOGR|texel.osm|*|highway|NOREPROJ
#
LineToPoint|FTYPE="LINE" And highway="secondary"|CONTINUOUS|150|5|0|String;type|
    ↪ licht|hdg
#
CreateXMLLibObj|type="licht"|{47c97ced-c4e6-4e4b-8e9f-b110989eea62}|hdg|0|0
#
ExportBGL|FSX|texel_light|C:\flightsim\FSX\Addon scenery\scenery
```

The point features are created at 150 meter intervals for all roads classified as secondary. They are placed 5 meters out of the center of the road and the heading of the points is stored in an attribute named hdg.

In the next step XML objects are created for all of the points. And finally they are exported to a BGL file.

## 5.7 Vary buildings heights in cities

This script shows how you can vary the height of autogen buildings based on a polygon that defines urban areas.

```
ImportOGR|buildings.shp|*|*|NOREPROJ
ImportOGR|urban_area.shp|*|*|NOREPROJ
#
SplitGrid|AGN|*|FROMFILE="buildings.shp"
#
AddCellAttribute|*|String;area|rural
AddCellAttributeIfInside|*|FROMFILE="urban_area.shp"|String;area|urban
#
SetAGNBuildingHeight|area="rural"|1.0;0.0;0.0;0.0
SetAGNBuildingHeight|area="urban"|0.8;0.5;0.1;0.0
#
CreateAGNGenBuild|FROMFILE="buildings.shp" AND FAREARAT>0.7 AND FWIDTH<20|{5
    ↪ ae04eb6-934c-4f63-bb48-5e7dee601212}|MAXRATIO=2
CreateAGNGenBuild|FROMFILE="buildings.shp" AND FAREARAT>0.7 AND FWIDTH>20|{6089
    ↪ A0BD-CED1-4c47-9A9E-64CDD0E16983}
#
ExportAGN|FSX|texture
```

The script first loads the building footprint data and the file with polygons that define urban areas. See Figure 5.1 for a visual representation of the data. As you can see some of the buildings are inside the urban area polygon, while others are not.

After splitting the data into the FS autogen grid, each cell is first given an attribute named `area` and a value of `rural`. So by default all our cells are rural. Next it is test for each cell if the center falls within the urban area polygon. If that is the case it is given an attribute named `area`, but with a value of `urban`.

Now these new cell attributes are used to assign different building heights to the cells. Those with the `rural` attribute will only get low buildings, while those with an `urban` attribute will get a mix of low and taller buildings.

The building heights are set per grid cell, so the variation in building height will not exactly follow the polygon you used as input. Figure 5.2 shows the different cell settings visually. The green cells are rural, the pink ones are the urban areas.

The remainder of the script creates the autogen buildings and exports everything to autogen files. But that is similar to the example shown in Section 5.2.

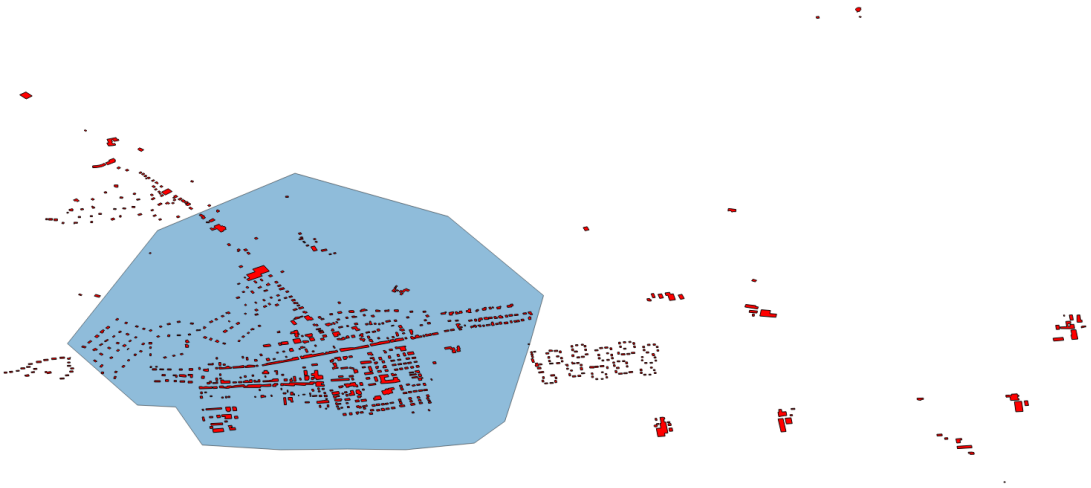


Figure 5.1: Building footprints and urban area polygon

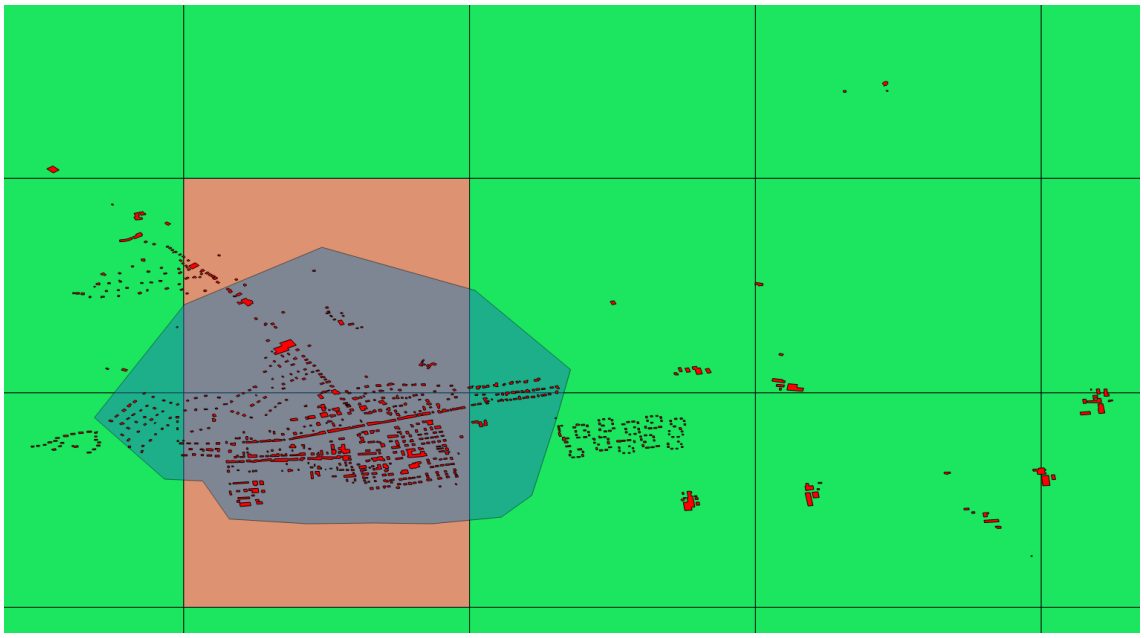


Figure 5.2: Different autogen building heights per cell

## 5.8 Create autogen tile grid for Google Earth

This script creates a KML file that can be loaded in Google Earth and that shows the location and tile names of the FS autogen grid.

```
CreateRectangle|Texel|area|texel
#
SplitGrid|AGN|*
PolygonToPoint|*|CENTER|0.0;0.0|0.0;0.0|String;obj|center|hdg
AddAttribute|FTYPE="POINT"|String;name|TILELOD
#
MergeGrid
#
ExportOGR|*|KML|grid.kml|grid
```

The first step creates a rectangle for the area that should be covered by the grid. OpenStreetMap is used to find the bounding box of the geographic area based on the name that was entered.

Next the rectangle is split into the autogen grid and a point is placed at the center of each grid cell. This point is used to display the name in Google Earth. Next the `AddAttribute` step is used to add the name of the grid tile as a name attribute. The name of the attribute must be name for Google Earth to show a label.

Finally all features are combined into one cell and the content is exported to a KML file that can be loaded in Google Earth. The result is as shown in Figure 5.3.

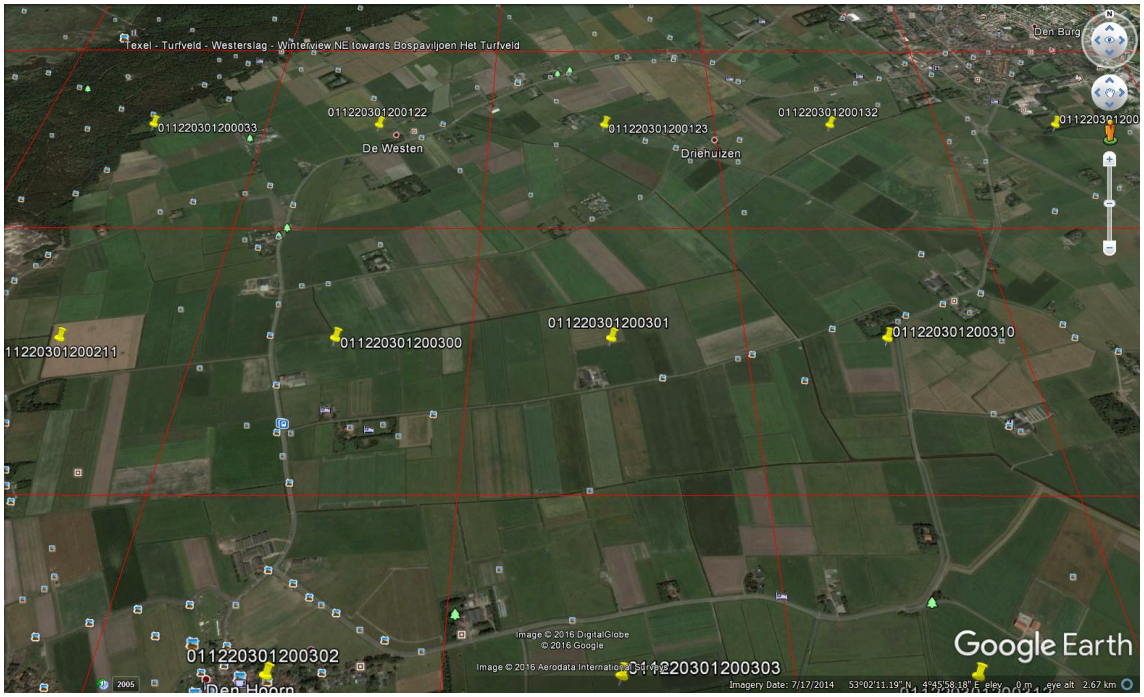



Figure 5.3: Autogen tile grid shown in Google Earth

## 5.9 Create SHP of QMID grid

This script creates a SHP file with a FS QMID grid, so that you can use it as a reference of the QMID grid tiles in your GIS tool.

```
CreateRectangle|-180;180;-90;90|String;type|poly
#
```

```
SplitGrid|QMID7|*|NONE
#
MergeGrid
#
ExportOGR|*|ESRI Shapefile|qmid_grid.shp|qmid_grid
```

The first step creates a rectangle for the area that should be covered by the grid. In this case a worldwide grid is made, but you can select a smaller area as well if you are working on a specific area.

Next the rectangle is split into the tiles of the QMID grid. You can specify the QMID size you want, in the example level 7 is used.

Finally all features are combined into one cell and the content is exported to a SHP file that can be loaded in your GIS tool.

## 5.10 Create terrain vector scenery

This script creates terrain vector scenery from OpenStreetMapData.

```
ImportOGR|andorra.osm.pbf|*|*|NOREPROJ
#
ImportGDAL|andorra_elev.tif|*|NOREPROJ
#
SplitGrid|LOD5|*
#
CreateTVecRoad|highway="primary"|{325dd470-b342-4d15-ac54-f67ed9f5914f}
CreateTVecRailroad|railway="miniature"|{dde116bf-0e97-4eb6-a9ec-7ef93d9f2d0e}
CreateTVecFreewayTraffic|highway="primary"|1|F
CreateTVecUtility|power="line"|{c942aadb-c351-4088-925e-a732c2093b63}
CreateTVecStream|waterway="river"|{2d3fc985-a72b-473d-b23b-d78e72e63b53}
CreateTVecPark|landuse="forest"|{6cea593e-64bc-4eb8-a6ca-806a03901115}
CreateTVecWaterPoly|natural="water"|{bcd5c182-9c8b-4c57-97bd-272cf492cbff}|
    ↪ RASTER_AVERAGE|123.45|*|waterway="*"
CreateTVecWaterPolyGPS|natural="water"
#
ExportTVecBGL|out|andorra
```

First the vector data and elevation data are read. The elevation data is needed for the water polygons only, since these also need to have a Z value defined. This value is sampled from the elevation data.

Next all the different types of terrain vectors are created from the loaded vector data.

Finally the terrain vectors are exported to a BGL file in the folder out. Figure 5.4 shows the resulting scenery in the TMFViewer tool of the SDK.

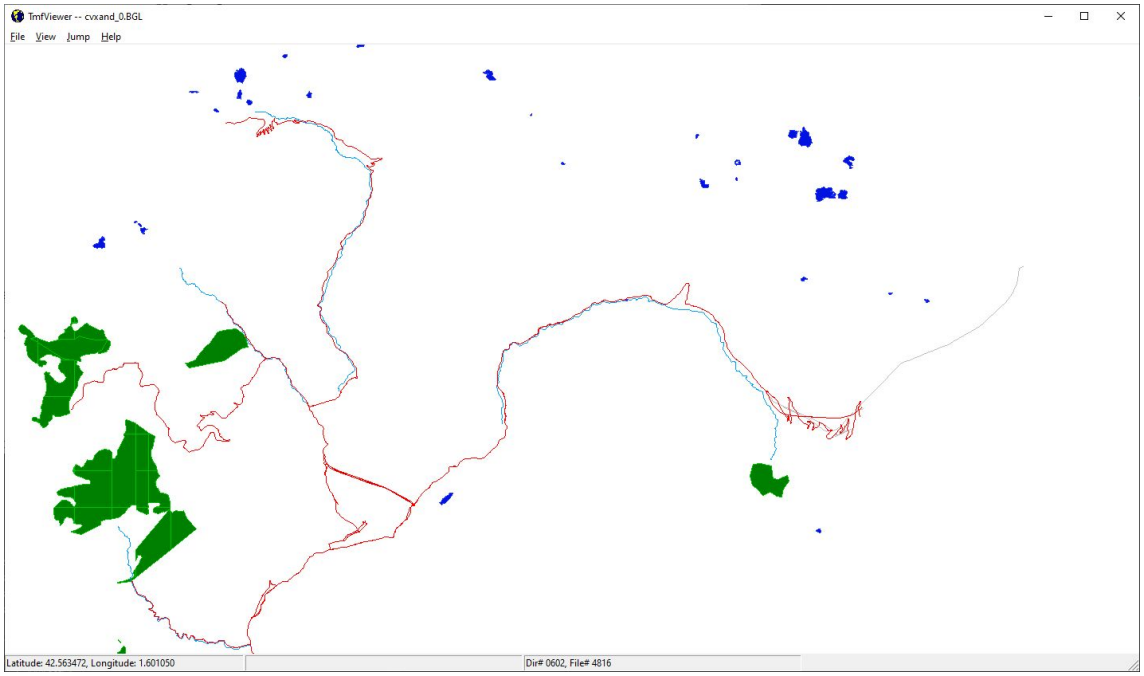



Figure 5.4: Terrain vector scenery shown in the TMFViewer tool

## 5.11 Create photoreal scenery

This script creates a photoreal scenery with watermask.

```
ImportGDAL|image.tif|*|NOREPROJ
#
ImportOGR|watermask.shp|*|*|NOREPROJ
#
ExportResamplePhotorealBGL|PR_watermask.tf2|*|NONE|85|-1;-1;-1|-1;-1|out|
    ↪ PR_watermask
```

First the raster data of the imagery and the vector data for the watermask are imported. In the last step that exports the BGL file most of the work is done. The used texture filter configuration specifies how the watermask should be created from the vector data and which manipulations need to be done to the imagery. This is explained in more detail in Chapter 6.

## 5.12 Create Aerofly FS 2 cultivation

This script creates a cultivation file for Aerofly FS 2 from OpenStreetMap data. It contains plants, buildings and lights. And it shows how to remove objects that are on an airport.

```
# Example AeroFly FS 2 scenProc script
# with function to filter out objects on airport
# ------------------------------------------------------
#
# Load OpenStreetMap data of the area to work on
# Only load highway, landuse and builidng to save memory
ImportOGR|annemasse.osm|*|highway;landuse;building|NOREPROJ
#
# Load Shapefile that contains the area of aiports that
# should be filtered out
ImportOGR|airport.shp|*|*|NOREPROJ
```

```
#
# Split the features into a grid of 0.25 x 0.25 degrees
# Do not split buildings, but filter them into thee
# right grid cells
SplitGrid|0.25|*|building="*"
#
# Place point features for the lights along roads of
# type primary at 50 meter interval
LineToPoint|highway="primary"|SINGLE|50|0|25|String;point|light|hdg
#
# Place point features for the plants in forest polygons
# using spacing of 0.00025 degrees and no randomness
PolygonToPoint|landuse="forest"|INSIDE|0.00025;0.00025|0.0;0.0|hdg|
    ↪ INHERITPARENTATTR
#
# Filter out the buldings, lights and plants that are within
# the airport polygons
AddAttributeIfInside|building="*"|FROMFILE="airport.shp"|String;skip|yes
AddAttributeIfInside|point="light"|FROMFILE="airport.shp"|String;skip|yes
AddAttributeIfInside|FTYPE="POINT" And landuse="forest"|FROMFILE="airport.shp"|
    ↪ String;skip|yes
UnloadFeatures|skip="yes"
#
# Create AF2 plants with height between 10 and 20 meter
CreateAF2Plant|landuse="forest"|10;20|0|broadleaf|T02
#
# Create AF2 lights
CreateAF2Light|point="light"|0.8;0.8;0.8|3|0;0;1|10
#
# Create AF2 buildings for polygons that are almost rectanguar
# Make longer buildings industrial with a flat roof
CreateAF2Building|building="*" And FAREARAT>0.7 And FLENGTH < 12|NONE|NONE|1|2;2|
    ↪ gable|residential|0
CreateAF2Building|building="*" And FAREARAT>0.7 And FLENGTH >= 12 And FLENGTH <
    ↪ 25|NONE|NONE|1|3;3|gable|residential|0
CreateAF2Building|building="*" And FAREARAT>0.7 And FLENGTH >= 25|NONE|NONE
    ↪ |1|2;2|flat|industrial|-1
#
# Export the AF2 TSC file
ExportTSC|out|annemasse
```

## 5.13 Create Aerofly FS 2 buildings and use OSM levels and height attributes

This script creates a cultivation file for Aerofly FS 2 from OpenStreetMap data. It demonstrates how to use the building:levels and height attributes that are present for certain areas when creating the buildings.

```
# Example AeroFly FS 2 scenProc script
# that uses the building:levels and height attributes
# for buildings
# ------------------------------------------------------
#
# Load OpenStreetMap data of the area to work on
# Only load builidng to save memory
```

```
ImportOGR|hagen.osm|*|building|NOREPROJ
#
# Split the features into a grid of 0.25 x 0.25 degrees
# Do not split buildings, but filter them into thee
# right grid cells
SplitGrid|0.25|*|building="*"
#
# Create AF2 buildings for polygons that are almost rectangular
# - for buildings with levels attribute use that
# - for buildings with height attribute use that and scale it with a factor of
    ↪ 0.3
# - for buildings without those attribute use random between 1 and 3 floors
# Note: in OSM the attribute is named building:levels,
# but after import by OGR this becomes building_levels
CreateAF2Building|building="*" And FAREARAT>0.7 And FLENGTH < 20|building_levels|
    ↪ height|0.3|1;3|gable|residential|0
# Make longer buildings industrial with a flat roof
CreateAF2Building|building="*" And FAREARAT>0.7 And FLENGTH >= 20|NONE|NONE
    ↪ |1|2;2|flat|industrial|0
#
# Export the AF2 TSC file
ExportTSC|out|hagen
```

## 5.14 Vegetation detection from raster

In this script vegetation vector data is detected from a raster image.

```
ImportGDAL|..\geo\imagery\gtif_wgs84_4band\*.tif|NOREPROJ
#
SplitGrid|AGN|*
#
DetectFeatures|FTYPE="RASTER"|filter_ndvi_trees.tf2|String;veg|trees|NONE
#
MergeGrid
#
ExportOGR|FTYPE="POLYGON"|ESRI Shapefile|veg_trees.shp|veg_trees
```

In the first step the GeoTIFF files with the imagery are loaded. In this case it is 4 band imagery, since we want to use a NDVI filter. In the second step we split the imagery into smaller pieces, this is done to optimize the processing, processing big images can result in high memory usage. The next step is running the actual feature detection. Here we load the texture filter configuration that has been made with the texture filter editor. All the vector data that has been created will get an attribute with the name veg and a value of trees. Then I merge all the data into one gridcell again, so that when saving the data we only get one shapefile, instead of hundreds. The last step is to actually save the data to Shapefile, so that I can use it later on.

# Chapter 6

# Texture filter

Many work in scenProc is done using vector data, but sometimes you need to work with raster data as well. The texture filter is a way to perform manipulations on the raster data. This can be useful for:

- Detect features that you do not have vector data for, for example detect vegetation for areas that don't provide good enough vector data.
- Perform colour corrections when making photoreal BGL files.
- Create seasonal or night variations of your imagery for the photoreal BGL files.

A texture filter is a way to configure scenProc to do such work on raster data. In this chapter it is explained how you can make such a filter and how you can use it.

## 6.1 Texture Filter Editor

The texture filter editor is the graphical user interface that you use to create and test your texture filter configuration. Once you are happy with how the filter works you can apply it in your script. You can find the texture filter editor in the Tools menu at the top of the scenProc user interface. Once you load the editor you will get a screen like shown in Figure 6.1.

You will find the following components in the texture filter editor user interface:

- A toolbar at the top with the following buttons:
  - **New** to create a new empty texture filter configuration.
  - **Open** to load a texture filter configuration (TF2) file from disk. With the arrow next to the button you can open a drop-down list of the recently opened TF2 files[1].
  - **Save** to save the current texture filter configuration to a file.
  - **Add image** to add a test image to your texture filter configuration.
  - **Remove image** to remove the selected test image from your texture filter configuration.
  - **Remove all images** to remove all test images from your texture filter configuration.
  - **Merge Results** checkbox that specifies if the output of a texture filter step should be shown merged with the input image or not.

[1] Versions of scenProc before scenProc 3.1 use the TFC format for the texture filter configuration, instead of the TF2 format. If you still have old texture filter configurations in that format you can use the tfc_to_tf2 tool to convert them to the new format. You can find this command line tool in the tools folder of your scenProc installation. To convert an old TFC file you use the following command on the command prompt:
tfc_to_tf2.exe c:\path\to\old.tfc c:\path\to\new.tf2

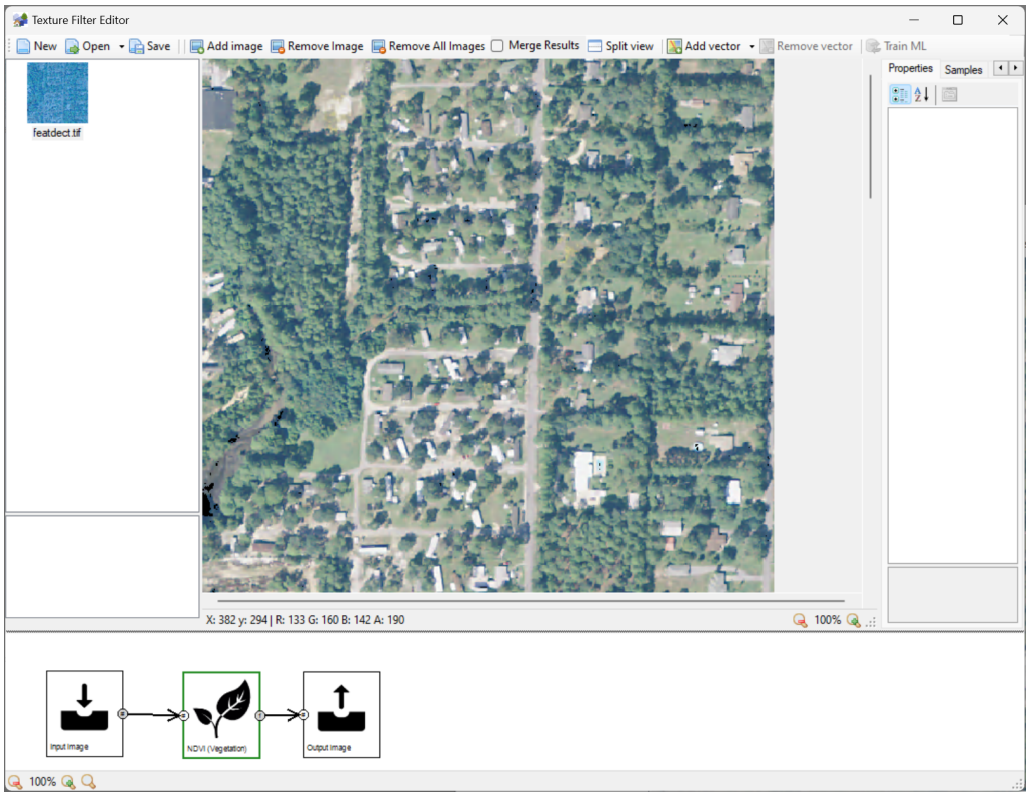



Figure 6.1: The texture filter editor

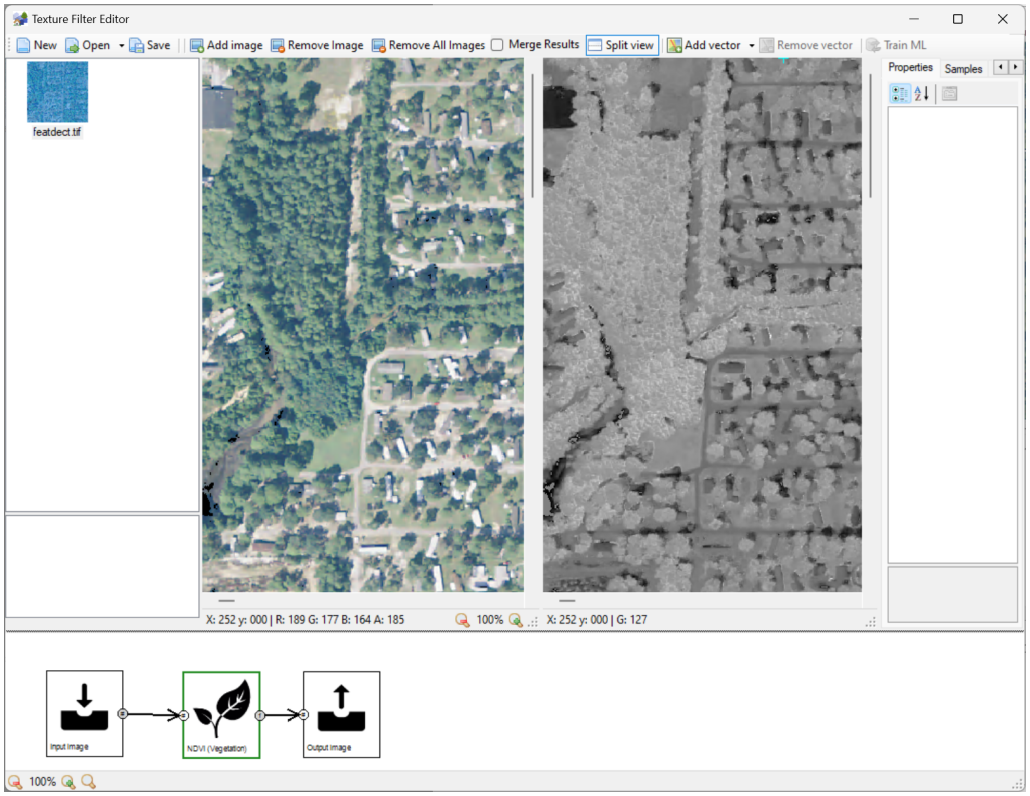



Figure 6.2: Split view showing the output of two steps at the same time

- **Split view** to select if the split view option is used. In that case the output of two steps can be shown next to each other, see Figure 6.2.
- **Add vector** to add a test vector data to your texture filter configuration.
- **Remove vector** to remove the selected test vector data from your texture filter configuration.
- **Train ML** to train machine learning steps in the texture filter using the training data.

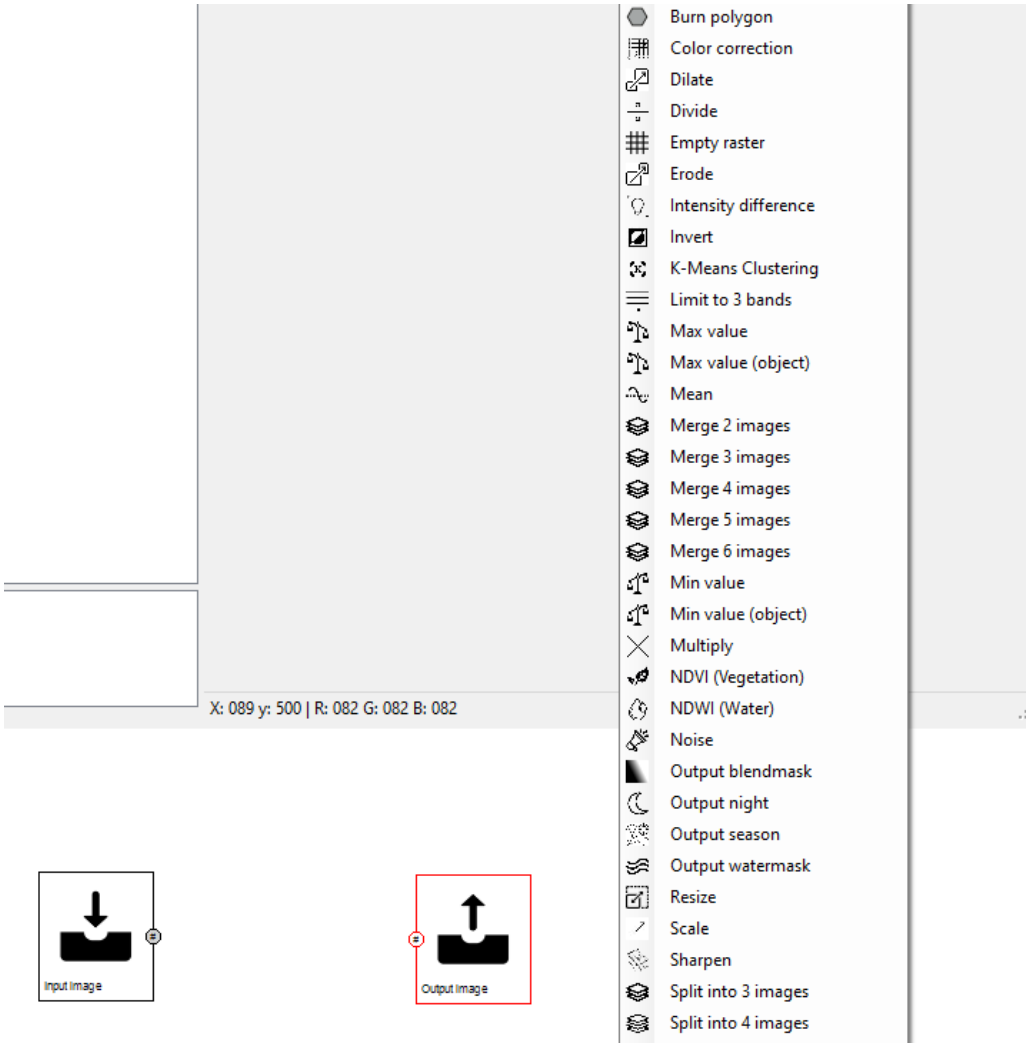



Figure 6.3: Adding a new step to the texture filter, the menu appears on a right click

- A picture box at the bottom where the texture filter is defined. In this box you can define the steps that need to be performed and how these are connected. The following mouse actions are supported in the picture box:
  - If you right click in the picture box you get a menu that allows you to add new steps to the texture filter flow (see Figure 6.3).
  - If you left click on a step you can select it for display in the preview window. If you hold the Control while clicking on the step you can set that step for display in the second preview of the split view. The selected step is shown with a blue border in the texture filter, while the selected step for the second preview is shown with a green border.
  - If you left click and hold and drag a step you can reposition it.

– If you left click and hold and drag on a connector of a step you can make a connection with the connector of another step. That way you can link the output from one step to the input of another (see Figure 6.4).

– If you left lick and hold outside of a step or connector you can pan the texture filter around in the picture box.

– With the mouse wheel or the buttons at the bottom you can zoom the texture filter in or out.

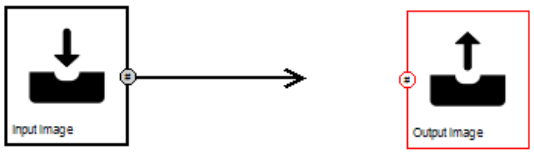



Figure 6.4: Connecting two connectors, left click on one connector and then hold and drag to the other connector

If the texture filter is not valid yet the steps that have invalid connections are drawn with a red outline. As long as some steps are invalid you will not be able to test the texture filter. If certain input or output connectors are optional, and thus do not have to be connected, they are shown with a green circle. Leaving these unconnected will not cause the texture filter to be invalid. See the description of the different steps to see how these optional connectors are handled. Machine learning steps in the texture filter can be filled with a red color, that means that their parameters have been changed since the last training (or they have never been trained) and thus that you need to retrain them before you can see the result.

Inside the the circles of the connectors a character is written that indicates what kind of output this connector gives or what kind of input it expects. Possible values are:

– # indicates that this connector can work with any image as input. For an output connector this means that the number of bands will be equal to the number of bands of the input image.

– **1** indicates that the connector provides or expects a 1 band image.

– **3** indicates that the connector requires at least 3 bands as input.

– **4** indicates that the connector requires at least 4 bands as input.

– **L** indicates that this connector provides or expects a labeled segment image, this kind of image provides the objects that certain steps use.

When a step has multiple output connectors you can determine which one is shown in the preview by double clicking on that connector. The connector that is being shown in the preview is shown in grey in the step picture box, see Figure 6.5.

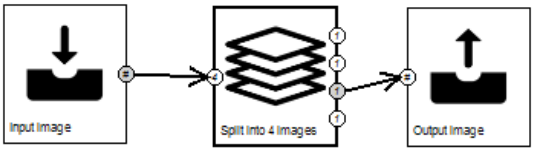



Figure 6.5: The connector rendered in gray is the one that is being displayed in the preview for this step

- In the middle there is a preview area, here you can see the effect of your texture filter on the loaded test images. If you click on a step in the texture filter you will see the output of the texture filter until that step, see Figure 6.6. That way you can easily inspect what the output is and if the texture filter gives the expected results. If you have the merge results checkbox selected the input image and the step result are shown merged, see Figure 6.7.

  If you hold and drag with the right mouse button you can pan the image around in the preview. With the mouse wheel or with the buttons at the bottom of the preview you can change the zoom level of the image. If you keep tne control key pressed while changing the zoom level a fine zoom mode is used and the zoom is changed in smaller steps.

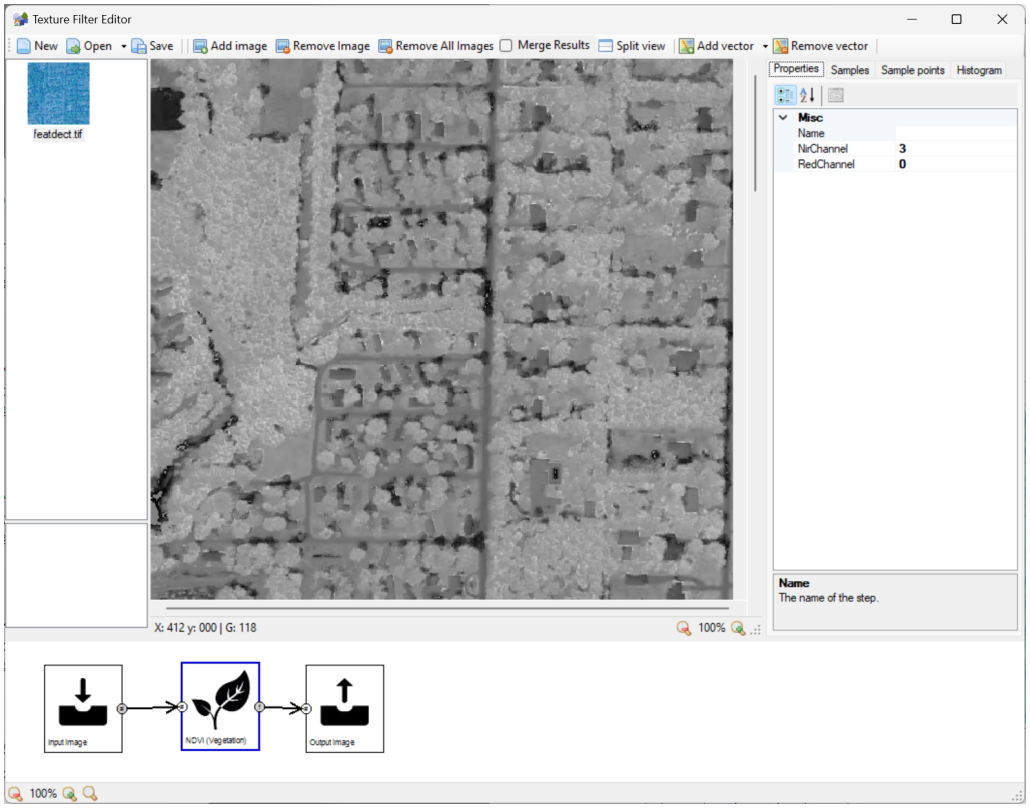



Figure 6.6: When a step is selected the result of this step is shown in the preview

- On the left of the preview area you see a list of all the loaded test images. When you click on an image in this list it will be used in the preview to see the texture filter effects. By having multiple test images you can test if the filter works well in different situations. In general it is advised to use test images that are not so big, around 1000 x 1000 pixels. That way you can see the image clearly in the texture filter editor. Test images can best be GeoTIFF or PNG files, when they are saved as normal BMP files the 4th channel is not always processed correctly.
- On the right of the preview area there are four tabs available:
  - **Properties** shows the properties of the selected step in the filter. By changing these properties you can tune the behaviour of the filter.
  - **Samples** shows the sample images that are used with the Back Project step. When you have this tab active you can also select on area in the preview by dragging over it to add a new sample image to the list, see Figure 6.8.
  - **Sample points** shows the sample points that are used with the SVM step. When you have this tab active you can also add new sample points by left clicking in the preview. Clicking with the left mouse will give a positive sample point (rendered as a green circle),

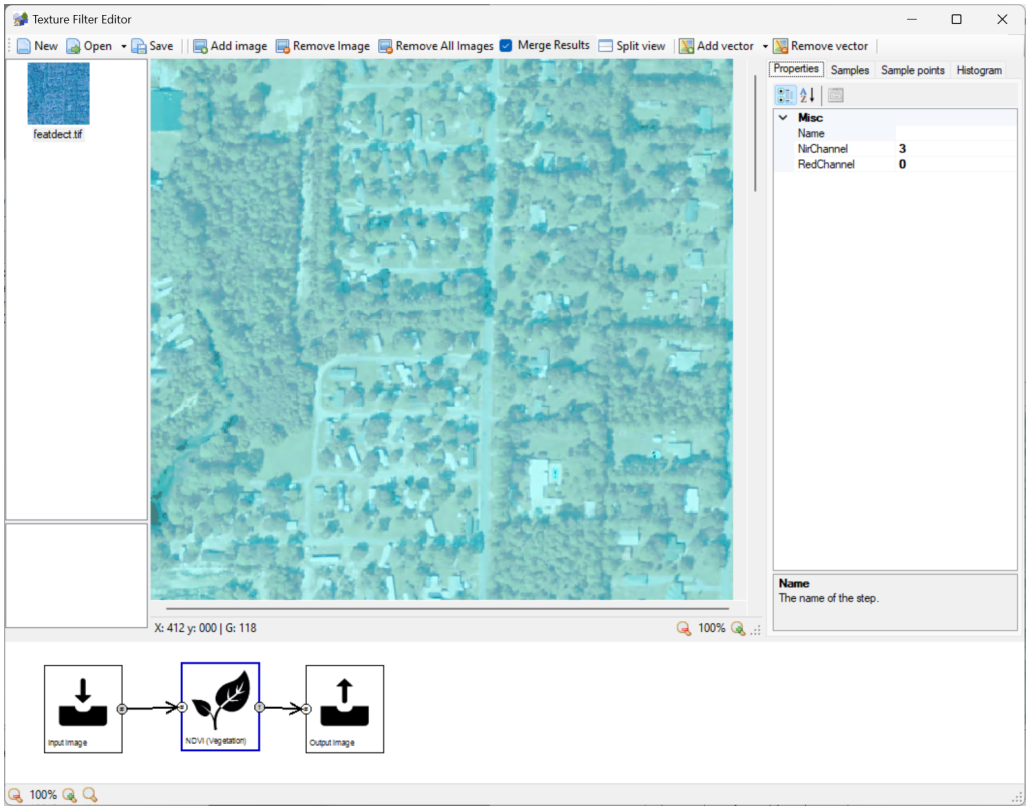



Figure 6.7: The step output merged with the input image

indicating this is a feature you want to detect. If you keep the Control key pressed while clicking you will get a negative sample (rendered as a red circle), indicating this is a feature you don't want to detect. See Figure 6.9 for an example of how the sample points are rendered.

– **Histogram** shows the histogram of the currently selected image, see Figure 6.10. By looking at the histogram you can identify the characteristics of the image and that can help you to determine properties like threshold values.

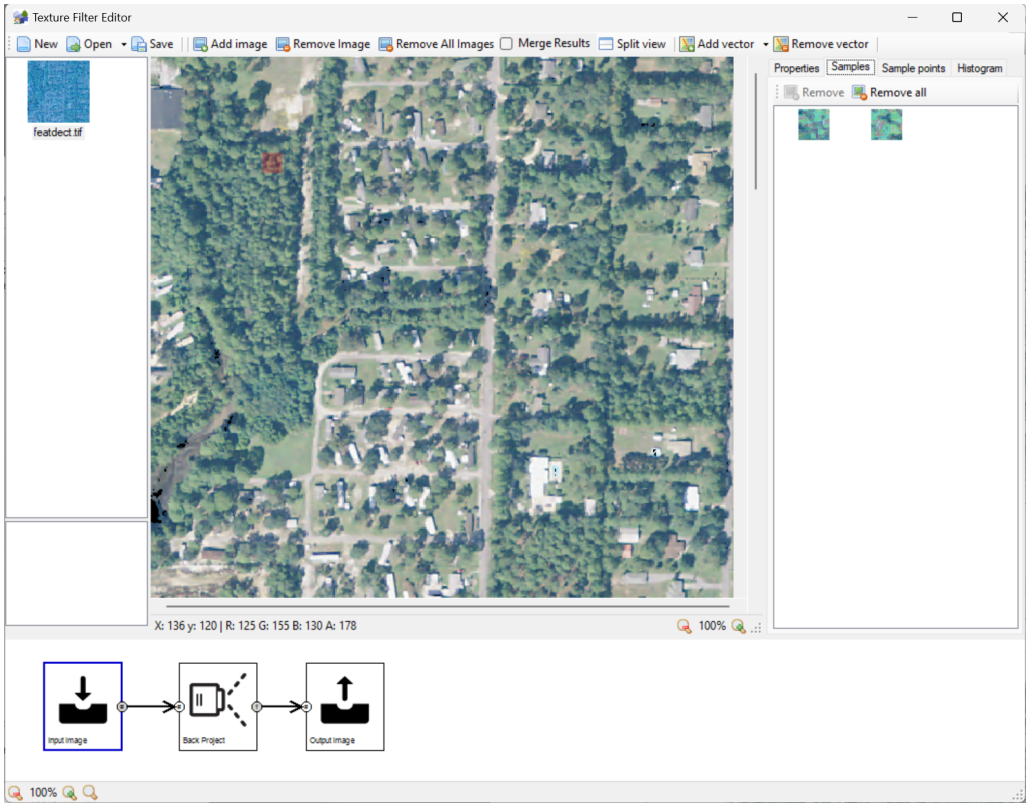



Figure 6.8: Selecting sample images

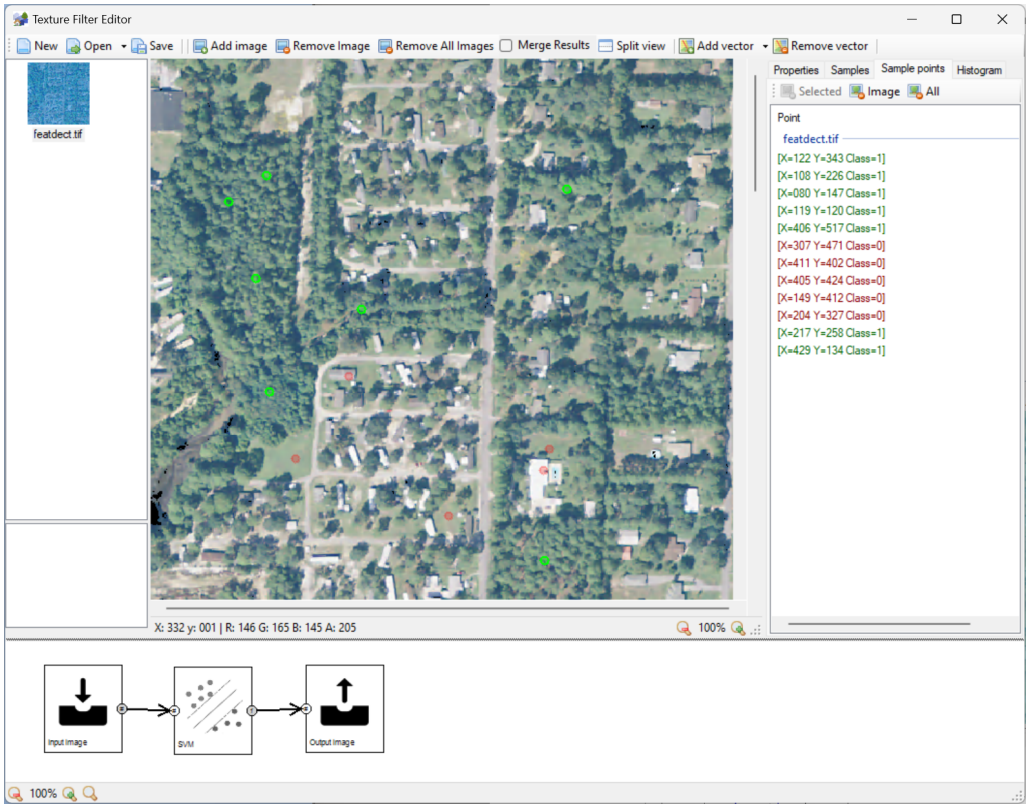



Figure 6.9: Selecting sample points

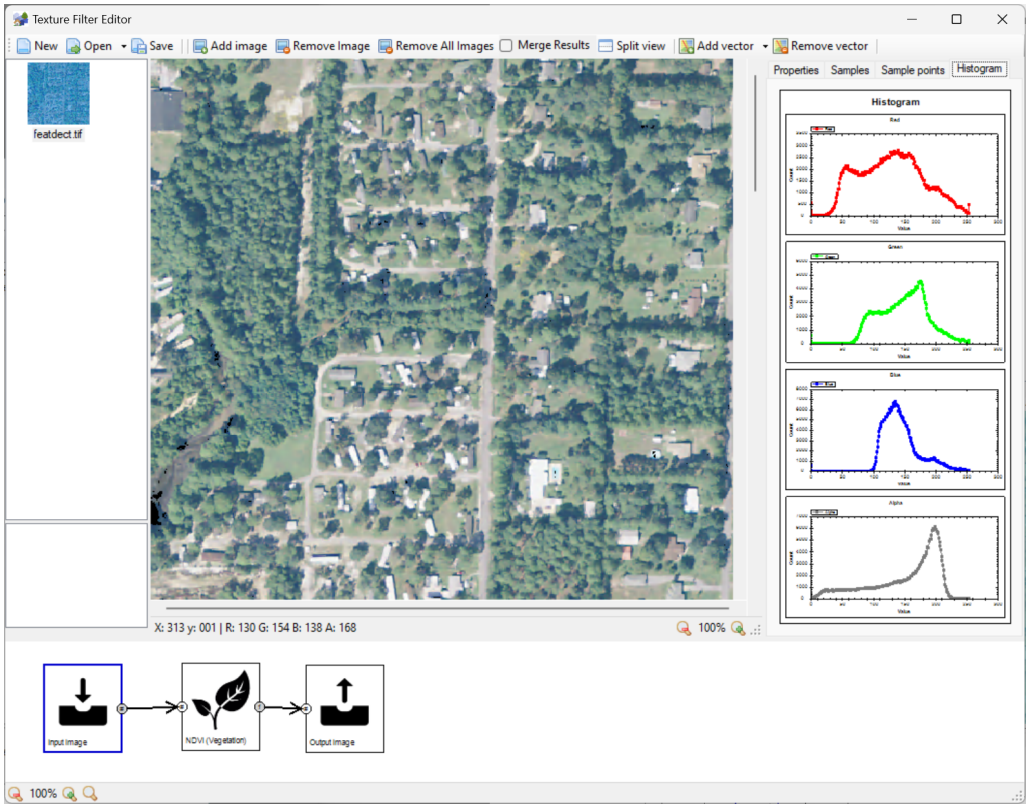



Figure 6.10: Histogram of the selected image

## 6.2 Available texture filter steps

In this section all the steps that you can use in a texture filter are explained.

### 6.2.1 Input image

#### Purpose

This is the input of the texture filter. The script using the filter will pass the input raster data to this step or when using the texture filter editor the test image is passed as input. Therefore each texture filter should always contain one instance of this step.

#### Parameters

None.

### 6.2.2 Output image

#### Purpose

This is the output of the texture filter. This result will be passed back to the script using the texture filter and is then used to vectorize feature or in the resample output. Each texture filter should contain one instance of this step.

#### Parameters

None.

### 6.2.3 Add

**Purpose**

Add the two input images together. When the input image has multiple bands this operation is performed per band, therefore the two input images need to have the same number of bands. The formula below is used to determine the output value.

$$output = input1 * \alpha + input2 * \beta + \gamma$$

**Parameters**

- **Alpha**: The multiplication factor of the first input image of the addition (the $\alpha$ parameter in the formula above).
- **Beta**: The multiplication factor of the second input image of the addition (the $\beta$ parameter in the formula above).
- **Gamma**: The offset value of the addition (the $\gamma$ parameter in the formula above).

### 6.2.4 Add Weighted

**Purpose**

Add two input images together, but instead of using a constant weight as in the Add step, the weight is provided per pixel as an additional third input image of the step. The weight input image is scaled so that the value is between 0.0 and 1.0. So the formula for this one is:

$$output = input1 * input3 + input2 * (1.0 - input3)$$

**Parameters**

None.

### 6.2.5 Add Weighted 3

**Purpose**

Add three input images together, while using two weight input images for the per pixel weight factors. The weight input images are scaled so that the value is between 0.0 and 1.0. The following formula is used:

$$output = input1 * input4 + input2 * input5 + input3 * (1.0 - input4 - input5)$$

**Parameters**

None.

### 6.2.6 Back Project

**Purpose**

This step detects how similar the image is to a set of sample images. The more similar the area of the image is to one of the sample, the higher the value of that pixel in the output becomes. The sample images are shown in the Samples tab at the right of the preview area. You can add new sample images by dragging with the mouse over the input images in the preview area.

**Parameters**

- **Bin size**: This is the size of the bins used when calculating the histogram that is used for the similarity detection.
- **Number of bands**: This is the number of bands of the image that should be considered for the histogram. If the image only has RGB values use 3 bands. Or if there is a NIR band you can use 4 bands.

### 6.2.7 Blur

**Purpose**

Blur the image.

**Parameters**

- **Blur type**: Determines if normal blur or gaussian blur is used.
- **Blur size**: The amount of blur to apply.
- **Gaussian sigma**: The sigma value to use for Gaussian blur.

### 6.2.8 Brightness

**Purpose**

Create a gray scale brightness image by taking the average value of the pixels of all bands.

**Parameters**

None.

### 6.2.9 Burn Line

**Purpose**

Burn line vector features into the raster image. This step can be used to for example burn roads or other lines into your image.

To be able to test this step in the texture filter editor you need to load vector data besides the sample images as well. And you need to make sure that your sample images are geo-referenced so that the editor know the position of the image relative to the vector data.

**Parameters**

- **Filter**: The filter to select the line features that should be burned into the image.
- **Color**: The color of the line that is drawn.
- **Exclude Filter**: The filter to select features that should be excluded from the line. This can be used for example to make a dashed line by letting certain features exclude the line being drawn.
- **Exclude Size**: The size to use around the features selected with the exclude filter.
- **Thickness**: The thickness of the line to draw.
- **Add Noise**: True if random noise should be added to the drawn line.
- **Noise Min**: The minimum noise value to use (range 0.0 and 1.0).
- **Noise Max**: The maximum noise value to use (range 0.0 and 1.0).

- **Blur Type**: The type of blurring to apply to the burned line. You can choose Blur or Gaussian.
- **Blur Size**: The amount of blurring to apply. For Gaussian type the size should be an odd number.
- **Blur Iterations**: The number of times the blurring should be applied. Sometimes applying the smoothing multiple times can give a better effect.
- **Gaussian Sigma**: The Gaussian sigma value to use for the gaussian blurring.

### 6.2.10 Burn Point

#### Purpose

Burn point vector features into the raster. This step can be used to for example burn light points into your image.

To be able to test this step in the texture filter editor you need to load vector data besides the sample images as well. And you need to make sure that your sample images are geo-referenced so that the editor know the position of the image relative to the vector data.

#### Parameters

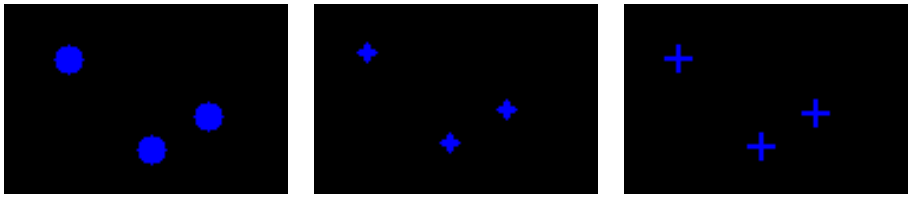
- **Filter**: The filter to select the point features that should be burned into the image.
- **Color**: The color of the point that is drawn.
- **Shape Type**: The shape to draw, either Circle, Cross or LineCross. See Figure 6.11 for a representation of the different shapes.
- **Size**: The size of the shape to draw.
- **Thickness**: The thickness of the lines when drawing a Cross or LineCross.
- **Draw Border**: If true, a black border is drawn around the shape.
- **Border Thickness**: The thickness of the black border that is drawn around the shape.
- **Add Noise**: True if random noise should be added to the drawn line.
- **Noise Min**: The minimum noise value to use (range 0.0 and 1.0).
- **Noise Max**: The maximum noise value to use (range 0.0 and 1.0).
- **Blur Type**: The type of blurring to apply to the burned line. You can choose Blur or Gaussian.
- **Blur Size**: The amount of blurring to apply. For Gaussian type the size should be an odd number.
- **Blur Iterations**: The number of times the blurring should be applied. Sometimes applying the smoothing multiple times can give a better effect.
- **Gaussian Sigma**: The gaussian sigma value to use for the gaussian blurring.

### 6.2.11 Burn Polygon

#### Purpose

Burn polygon vector features into the raster. This step can be used to for example burn water polygons into your image.

To be able to test this step in the texture filter editor you need to load vector data besides the sample images as well. And you need to make sure that your sample images are geo-referenced so that the editor know the position of the image relative to the vector data.



(a) (b) (c)

Figure 6.11: Available shapes to burn (all at Size=6, Thickness=2): (a) Circle (b) Cross (c)LineCross

**Parameters**

- **Filter**: The filter to select the polygon features that should be burned into the image.
- **Color**: The color of the point that is drawn.
- **Hole Color**: The color to use to fill the holes of the polygon.
- **Add Noise**: True if random noise should be added to the drawn line.
- **Noise Min**: The minimum noise value to use (range 0.0 and 1.0).
- **Noise Max**: The maximum noise value to use (range 0.0 and 1.0).
- **Blur Type**: The type of blurring to apply to the burned line. You can choose Blur or Gaussian.
- **Blur Size**: The amount of blurring to apply. For Gaussian type the size should be an odd number.
- **Blur Iterations**: The number of times the blurring should be applied. Sometimes applying the smoothing multiple times can give a better effect.
- **Gaussian Sigma**: The gaussian sigma value to use for the gaussian blurring.

### 6.2.12 Canny Edge Detection

**Purpose**

Use the Canny algorithm to detect edges in the image.

**Parameters**

- **ApertureSize**: The aperture size of the kernel used for the edge detection.
- **L2Gradient**: Flag to indicate if the more accurate L2 gradient is used.
- **Threshold1**: The minimum threshold value, any value below this is not considered an edge.
- **Threshold2**: The maximum threshold value, any value above this is always considered an edge.

### 6.2.13 Color Correction

**Purpose**

Apply a color correction to the image. This is done by defining a curve that alters the colors. The curve is a parabolic line through the specified X and Y location. The the X and Y stretch values are the same no color correction is applied. See Figure 6.12 for some examples. You specify the X and Y location of dots in this image and that results in curves as shown.

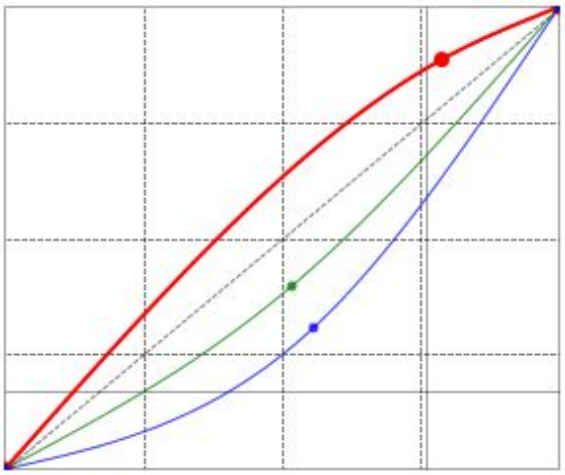


Figure 6.12: Color correction curves

**Parameters**

- **X Stretch**: The X stretch point for each channel of the image.
- **Y Stretch**: The Y stretch point for each channel of the image.
- **Scale**: The scale value for each channel of the image.

### 6.2.14 Dilate

**Purpose**

Grow the elements in the raster, this can be used to fill small holes for example.

**Parameters**

- **Iterations**: The number of dilate iterations to apply.

### 6.2.15 Divide

**Purpose**

Divide the two input images by each other. Since dividing two byte images will give small values in general a scaling is applied to ensure that the output is visible as a byte image again.

$$output = \frac{input1 * scaling}{input2}$$

**Parameters**

- **Scaling**: The scaling to apply to the image after division.

### 6.2.16 EmptyRaster

**Purpose**

Create an empty raster with a specific value, instead of using the input image. This can for example be useful when you want to create an initial watermask that has the same size as the input image.

**Parameters**

- **Number of Channels**: The number of channels in the image to create.
- **Value**: The initial value that the pixels should get.
- **Scale**: Scale factor to apply to the width and height compared to the input image. A value of 1.0 gives the same size as the input image, while 0.5 gives you half of the size.

### 6.2.17 Erode

**Purpose**

Shrink the elements in the raster, this can be used to remove small noise for example.

**Parameters**

- **Iterations**: The number of erode iterations to apply.

### 6.2.18 Intensity difference

**Purpose**

Calculate the difference in intensity between the different channels of the image and return that as a grayscale image. For example when the R channel has a value of 100, the G channel has a value of 70 and the B channel has a value of 120, the maximum intensity difference between the channel is 50 and that value is set as output for the pixel.

**Parameters**

None.

### 6.2.19 Invert

**Purpose**

Invert all channels of the image.

**Parameters**

None.

### 6.2.20 KMeans

**Purpose**

Segment the image using the K-means clustering algorithm. This algorithm assigns each pixel to the cluster of which the mean value is nearest to the value of the pixel. So therefore similar pixels are grouped in the same cluster. The output of this step is a label image that can be used as input into steps that work on objects.

**Parameters**

- **CleanupIterations**: The number of erode/dilate iterations that are performed to remove small noise from the label image.
- **NumberOfClusters**: The number of clusters that should be generated.
- **Attempts**: The number of attempts at generating optimal clusters.
- **ClusterCenter**: The method used to select the cluster centers.

### 6.2.21 Limit 3

#### Purpose

Make sure that the image is limited to 3 channels. If it has more than 3 channels the other channels are removed.

#### Parameters

None.

### 6.2.22 Max value

#### Purpose

Return the maximum value of the two input images.

#### Parameters

None.

### 6.2.23 Max value (object)

#### Purpose

Return the maximum value in each object.

#### Parameters

None.

### 6.2.24 Mean

#### Purpose

Return the mean value in each object.

#### Parameters

None.

### 6.2.25 Merge

#### Purpose

Merge all the channels of the input images into one image. It is possible to provide between 2 and 6 input images to this step. For example if the first and second input image have 3 channels and the third, fourth, fifth and sixth image have 1 channel, a 10 channel image will be the output.

#### Parameters

None.

### 6.2.26 Min value

#### Purpose

Return the minimum value of the two input images.

#### Parameters

None.

### 6.2.27 Min value (object)

**Purpose**

Return the minimum value in each object.

**Parameters**

None.

### 6.2.28 Mlp

**Purpose**

This step uses the Artificial Neural Networks - Multi-Layer Perceptrons (MLP) machine learning algorithm to classify certain class in an image. The step can either be trained using sample images or using raster input data. After training the algorithm in the texture filter editor, an entire image can be classified using the DetectFeatures step.

- If all three input connectors are connected the SVM step is trained using raster data. The first connector is the input data to be used for the classification, the second connector is the label data to be used for training and the third connector is the raster class data to be used for training.
- If not all three input connectors are connected the SVM step is trained using the sample points as provided in the texture filter editor. The sample points are divided into two classes for the classification: positive (green circles) and negative (red circles). For optimal results make sure that you have enough sample points in the training data.

Depending on the size of the images used, this step might take longer to process. Especially when the Mlp algorithm has to be trained using multiple input images. So in the texture filter editor this means the output might not be shown instantly. While the step is processing the waiting cursor will show.

**Parameters**

- **AnnealCoolingRatio**: ANNEAL: cooling ratio. It must be >0 and less than 1. Default value is 0.95.
- **AnnealFinalT**: ANNEAL: final temperature. It must be >=0 and less than initialT. Default value is 0.1.
- **AnnealInitialT**: ANNEAL: initial temperature. It must be >=0. Default value is 10.
- **AnnealItePerStep**: ANNEAL: iteration per step. It must be >0. Default value is 10.
- **BackpropMomentumScale**: BPROP: Strength of the momentum term (the difference between weights on the 2 previous iterations). This parameter provides some inertia to smooth the random fluctuations of the weights. It can vary from 0 (the feature is disabled) to 1 and beyond. The value 0.1 or so is good enough. Default value is 0.1.
- **BackpropWeightScale**: BPROP: Strength of the weight gradient term. The recommended value is about 0.1. Default value is 0.1.
- **Epsilon**: The tolerance used for the termination criteria of the iterative MLP training procedure which solves a partial case of constrained quadratic optimization problem.
- **HiddenLayers**: The number of neutrons in each of the hidden layers. The input and output layer are added automatically by scenProc.
- **MaxIterations**: The maximum number of iterations used for the termination criteria of the iterative MLP training procedure which solves a partial case of constrained quadratic optimization problem.

- **RpropDW0**: RPROP: Initial value delta-zero of update-values delta-ij. Default value is 0.1.
- **RpropDWMax**: RPROP: Update-values upper limit delta-max. It must be >1. Default value is 50.
- **RpropDWMin**: RPROP: Update-values lower limit delta-min. It must be positive. Default value is FLT_EPSILON.
- **RpropDWMinus**: RPROP: Decrease factor eta. It must be <1. Default value is 0.5.
- **RpropDWPlus**: RPROP: Increase factor eta. It must be >1. Default value is 1.2.
- **TrainMethod**: The MLP train method. Possible values are:
  - Backprop: Back-propagation algorithm.
  - Rprop: RPROP algorithm.
  - Anneal: Simulated annealing algorithm.

### 6.2.29 Multiply

**Purpose**

Multiply the two input images by each other. Since multiplying two byte images will give large values a scaling is applied to ensure that the output is visible as a byte image again.

$$output = \frac{input1 * input2}{scale}$$

**Parameters**

- **Scale**: The scaling to apply after multiplication.

### 6.2.30 Multi Res Segment

**Purpose**

The Multi Res Segment step performs a multi resolution segmentation algorithm to divide the input image into different segments. The output of this step is a label image that can be used as input into steps that work on objects.

The segmentation algorithm will check each segments with its neighbor segments. It will then merge the two segments for which the merge cost is lowest, as long as it is below a configurable maximum merge cost. This merge cost is determined based on two factors:

1. A color factor based on the standard deviation of the colors of the merged segment.
2. A shape factor based on the shape of the merged segment. This shape factor itself is composed or a compactness and smoothness component.

With the different weight parameters of the algorithm you can determine which of the factors mentioned above matters more. The merge cost is determined by combining the color factor and the shape factor, the sum is weighted based on the color weight that determines which factor is more important.

$$cost = W_{color} * F_{color} + (1 - W_{color}) * F_{shape}$$

The color factor is determined by summing the factor per band of the image multiplied by the weight of the image. The factor per band is determined using the standard deviation.

$$F_{color} = \sum W_{band} * F_{band}$$

The shape factor is determined by combining the compactness and smoothness factor, the sum is weighted based on the compactness weight that determines which factor is more important.

$$F_{shape} = W_{compact} * F_{compact} + (1 - W_{compact} * F_{smooth}$$

Depending on the size of the images used, this step might take longer to process. So in the texture filter editor this means the output might not be shown instantly. While the step is processing the waiting cursor will show.

**Parameters**

- **ColorBandWeights**: The weight that is assigned to each of the 4 bands of the input image when calculating the color factor in the segmentation algorithm. The range is between 0.0 and 1.0. This is $W_{band}$ in the equations above.
- **ColorWeight**: The weight that determines the balance between the color factor and the shape factor in the segmentation algorithm. The range is between 0.0 and 1.0. This is $W_{color}$ in the equations above.
- **CompactnessWeight**: The weight that determines the balance between the compactness factor and the smoothness factor when calculating the shape factor. The range is between 0.0 and 1.0. This is $W_{compact}$ in the equations above.
- **MaximumTileSize**: The maximum number of pixels per tile. If the image is wider or taller than this number of pixels it will be processed in different tiles in parallel to improve the performance.
- **MutuallyBestMerge**: Only merge two segments when the merge is the best option for both segments. This improves the quality, but at the cost of extra calculation time.
- **NumberOfIterations**: The number of iterations that are performed to cluster segments. The algorithm can also terminate earlier when no more segments can be merged.
- **NumberOfSegments**: The number of segments that have been created by the algorithm. This is not an input parameter, but an output parameter to give an idea of how the segmentation worked.
- **PerformedIterations**: The number of iterations that have been performed by the algorithm. This is not an input parameter, but an output parameter to give an idea of how the segmentation worked.
- **RandomSeed**: The seed used for the random order in which the segments are considered for merging.
- **Scale**: The scale determines how big the segments will be, only segments for which the calculated factor is less than the scale are merged.

### 6.2.31 NDVI (Vegetation)

**Purpose**

NDVI stands for Normalized Difference Vegetation Index and it is a common measure for the amount of vegetation. It is calculated using near infrared and red color data, so you need to make sure your input image contains the near infrared channel as well. The NDVI value is between -1.0 and 1.0 normally, but for the byte output image it is given the range of 0 to 255. This means that 128 corresponds to a NDVI of 0.0.

$$NDVI = \frac{(NIR - RED)}{(NIR + RED)}$$

**Parameters**

- **NIR Channel**: The channel of the image that contains the near infrared data.
- **Red Channel**: The channel of the image that contains the red data.

### 6.2.32 NDWI (Water)

**Purpose**

NDWI stands for Normalized Difference Water Index and it is a common measure for the amount of water. It is calculated using near infrared and green color data, so you need to make sure your input image contains the near infrared channel as well. The NDWI value is between -1.0 and 1.0 normally, but for the byte output image it is given the range of 0 to 255. This means that 128 corresponds to a NDWI of 0.0.

$$NDWI = \frac{(GREEN - NIR)}{(GREEN + NIR)}$$

**Parameters**

- **Green Channel**: The channel of the image that contains the green data.
- **NIR Channel**: The channel of the image that contains the near infrared data.

### 6.2.33 Noise

**Purpose**

Add noise to the image.

**Parameters**

- **Mean**: The mean value of the noise to add.
- **Standard deviation**: The standard deviation of the noise to add.

### 6.2.34 Output Blendmask

**Purpose**

This step defines the output image that should be used for the blendmask in a photoreal scenery.

**Parameters**

None.

### 6.2.35 Output Night

**Purpose**

This step defines the output image that should be used for the night variant in a photoreal scenery.

**Parameters**

None.

### 6.2.36 Output Season

**Purpose**

This step defines the output image that should be used for the seasonal variants in a photoreal scenery.

**Parameters**

None.

### 6.2.37 Output Watermask

**Purpose**

This step defines the output image that should be used for the watermask in a photoreal scenery.

**Parameters**

None.

### 6.2.38 Resize

**Purpose**

Resize the image to a different size.

**Parameters**

- **Factor**: The scale factor to apply to the width and height of the image.
- **Interpolation**: The interpolation method to use while resizing.

### 6.2.39 Scale

**Purpose**

Scale the image pixel values and optionally apply an offset to it as well.

$$output = input * scale + offset$$

**Parameters**

- **Scale**: The scale value to use.
- **Offset**: The offset value to use.

### 6.2.40 Sharpen

**Purpose**

Apply a sharpen filter to the image.

**Parameters**

None.

### 6.2.41 Split 3

**Purpose**

Split the 3 channels of the image into separate images that you can use in the texture filter.

#### Parameters

None.

### 6.2.42 Split 4

#### Purpose

Split the 4 channels of the image into separate images that you can use in the texture filter.

#### Parameters

None.

### 6.2.43 Std Deviation

#### Purpose

Calculate the standard deviation of the area around the pixel. This is a measure of how much variation there is in the values of a channel and can be useful to differentiate between smooth and rough features (e.g. grass or trees).

#### Parameters

- **Sample size**: The size of the area that should be included in determining the standard deviation of the value of each pixel.
- **Scale**: The scaling to apply to the output image, but using a value higher than 1.0 you can make the standard deviation more clear.

### 6.2.44 Std Deviation (object)

#### Purpose

Return the standard deviation within each object.

#### Parameters

- **Scale**: The scaling to apply to the output image, but using a value higher than 1.0 you can make the standard deviation more clear.

### 6.2.45 Subtract

#### Purpose

Subtract the two input images from each other.

$$output = input1 - input2$$

#### Parameters

None.

### 6.2.46 Svm

#### Purpose

This step uses the Support Vector Machine machine learning algorithm to classify certain class in an image. The step can either be trained using sample images or using raster input data. After training the algorithm in the texture filter editor, an entire image can be classified using the DetectFeatures step.

- If all three input connectors are connected the SVM step is trained using raster data. The first connector is the input data to be used for the classification, the second connector is the label data to be used for training and the third connector is the raster class data to be used for training.
- If not all three input connectors are connected the SVM step is trained using the sample points as provided in the texture filter editor. The sample points are divided into two classes for the classification: positive (green circles) and negative (red circles). For optimal results make sure that you have enough sample points in the training data.

The Support Vector Machine algorithm tries to define the separation planes between the different classes in the training data. Figure 6.13 gives a graphical representation of this for data with two dimensions (X and Y) and two classes (red and blue). The data used when classifying images will have more than two dimensions in general, but then it is less easy to visualize what the algorithm does.

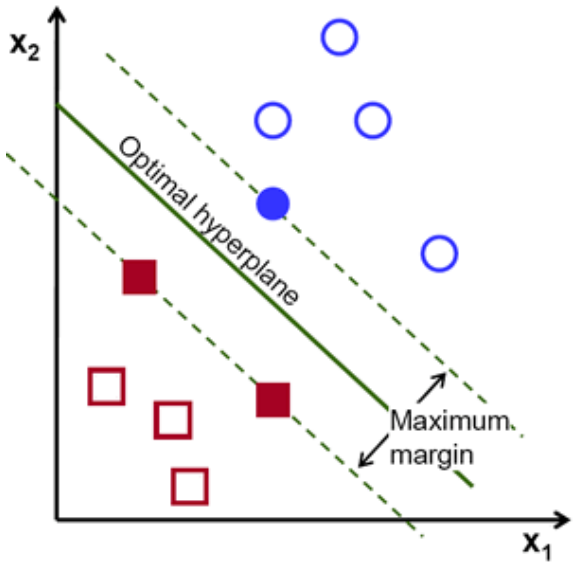



Figure 6.13: Illustration how SVM tries to define a separation plane between the data (source OpenCV documentation)

Depending on the size of the images used, this step might take longer to process. Especially when the Svm algorithm has to be trained using multiple input images. So in the texture filter editor this means the output might not be shown instantly. While the step is processing the waiting cursor will show.

**Parameters**

- **C**: Parameter C of a SVM optimization problem. This is used when the SVM type is CSvc, EpsSvr or NuSvr.
- **Coef0**: Parameter coef0 of a kernel function. This parameter is used when the KernelType is Poly or Sigmoid.
- **Degree**: Parameter degree of a kernel function. This parameter is used when the KernelType is Poly.
- **Epsilon**: The tolerance used for the termination criteria of the iterative SVM training procedure which solves a partial case of constrained quadratic optimization problem.
- **Gamma**: Parameter $\gamma$ of a kernel function. This parameter is used for KernelType Poly, RBF, Sigmoid and Chi2.

- **KernelType**: The kernel type of the SVM model. See Figure 6.14 for a comparison of the results of different kernel types on some sample training data. Possible values are:
  - Chi2: Exponential Chi2 kernel, similar to the RBF kernel.
  - Custom: Custom kernel, this type is not supported.
  - Inter: Histogram intersection kernel. A fast kernel.
  - Linear: Linear kernel. No mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option.
  - Poly: Polynomial kernel.
  - Rbf: Radial basis function (RBF), a good choice in most cases.
  - Sigmoid: Sigmoid kernel.

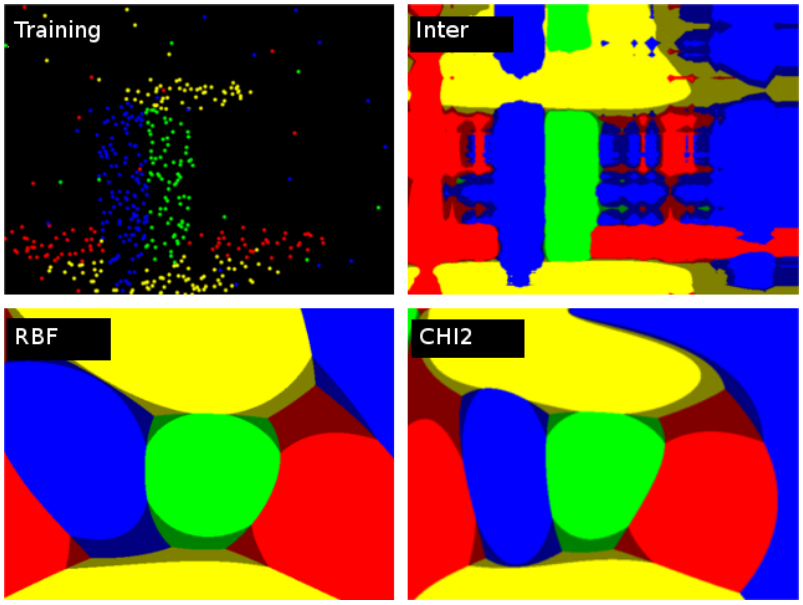



Figure 6.14: Comparison of different SVM kernels (source OpenCV documentation)

- **MaxIterations**: The maximum number of iterations used for the termination criteria of the iterative SVM training procedure which solves a partial case of constrained quadratic optimization problem.
- **Nu**: Parameter $\nu$ of a SVM optimization problem. This parameter is used when the SvmType is NuSvc, NuSvr or OneClass.
- **SvmType**: The SVM type to use. Possible values are:
  - CSvc: C-Support Vector Classification. n-class classification, allows imperfect separation of classes with penalty multiplier C for outliers.
  - EpsSvr: $\epsilon$-Support Vector Regression. The distance between feature vectors from the training set and the fitting hyper-plane must be less than p. For outliers the penalty multiplier C is used.
  - NuSvc: $\nu$-Support Vector Classification. n-class classification with possible imperfect separation. Parameter $\nu$ (in the range 0 to 1, the larger the value, the smoother the decision boundary) is used instead of C.
  - NuSvr: $\nu$-Support Vector Regression. $\nu$ is used instead of p.

  - OneClass: Distribution Estimation (One-class SVM). All the training data are from the same class, SVM builds a boundary that separates the class from the rest of the feature space.

- **P**: Parameter p of a SVM optimization problem. This parameter is used when the SvmType is EpsSvr.

### 6.2.47 TGDI (Tree/Grass)

**Purpose**

This step implements the Tree Grass Differentiation Index (TGDI) as discussed in the article A new index to differentiate tree and grass based on high resolution image and object-based methods by Qian. The two inputs of the steps are the detected edges in the image and the brightness. The index is calculated with the equation shown below.

$$TGDI = Factor * \log_{10}(Edges) * Brightness$$

**Parameters**

- **Factor**: Additional factor applied to scale the TGDI when converting it back to a gray scale image.

### 6.2.48 Threshold Binary

**Purpose**

Return maximum value when the value in the input image is bigger than the threshold, else return zero.

**Parameters**

- **Threshold**: The threshold value to use.
- **Maximum Value**: The maximum value to use.

### 6.2.49 Threshold Binary Inverse

**Purpose**

Return maximum value when the value in the input image is smaller than the threshold, else return zero.

**Parameters**

- **Threshold**: The threshold value to use.
- **Maximum Value**: The maximum value to use.

### 6.2.50 USI (Shadow)

**Purpose**

USI stands for Urban Shadow Index and it is a measure for the amount of shadow in the image. This index has been taken from the article Two-Step Urban Water Index (TSUWI): A New Technique for High-Resolution Mapping of Urban Surface Water by Wu. The index uses the RGB bands of the image and the NIR band, so you need to make sure that you use this step only when you have a 4 band image with NIR data included.

**Parameters**

- **Blue Channel**: The channel of the image that contains the blue data.
- **Green Channel**: The channel of the image that contains the green data.
- **NIR Channel**: The channel of the image that contains the near infrared data.
- **Red Channel**: The channel of the image that contains the red data.

### 6.2.51 UWI (Water)

**Purpose**

UWI stands for Urban Water Index and it is a measure for the amount of water in the image. This index has been taken from the article Two-Step Urban Water Index (TSUWI): A New Technique for High-Resolution Mapping of Urban Surface Water by Wu. The index uses the red and green color bands of the image and the NIR band, so you need to make sure that you use this step only when you have a 4 band image with NIR data included.

**Parameters**

- **Green Channel**: The channel of the image that contains the green data.
- **NIR Channel**: The channel of the image that contains the near infrared data.
- **Red Channel**: The channel of the image that contains the red data.

## 6.3 Example texture filters

This section contains a number of example texture filters to detect different feature types or to use to make photoreal scenery. Use these scripts as inspiration, you will probably need to tune attributes, like threshold values, to match the raster data you are using to get the best results.

### 6.3.1 Vegetation feature detection using back projection

This section discusses an example texture filter that can be used to detect vegetation using the back project step. This step uses sample images and then determines how similar the input image is to these samples. This filter contains the following steps:

1. The input image to use, see Figure 6.15. Note that in the panel on the right the sample images of the vegetation that have been selected by dragging on the input image are shown.
2. Determine how similar the pixel is to the sample images using the back project step. A bin size of 24 is used and 3 bands are used for the detection. See Figure 6.16.
3. Binary threshold with a value of 16 to select only the areas which are likely to be vegetation, see Figure 6.17.
4. Split the input image into 3 channels.
5. Calculate the standard deviation of the blue channel, a sample size of 5 and a scale of 10 is used. See Figure 6.18.
6. Binary threshold with a value of 15 on the standard deviation to exclude the areas with have a low standard deviation (and thus are monotonous), see Figure 6.19.
7. Minimum value of the vegetation detection and standard deviation threshold values. This means that areas with a low standard deviation are excluded from the vegetation detection. This is done to detect less grass areas. See Figure 6.20.
8. Erode step with 1 iteration to get rid of small noise, see Figure 6.21.

9. Dilate step with 3 iterations to grow the detected vegetation back to the normal size, see Figure 6.22.
10. Erode step with 4 iterations to get rid of more small detected areas, see Figure 6.23.
11. Dilate step with 4 iterations to grow the detected vegetation back (again) to the normal size, see Figure 6.24.
12. The output image, shown merged in Figure 6.25 with the input image, to see how the detection matches the input image.

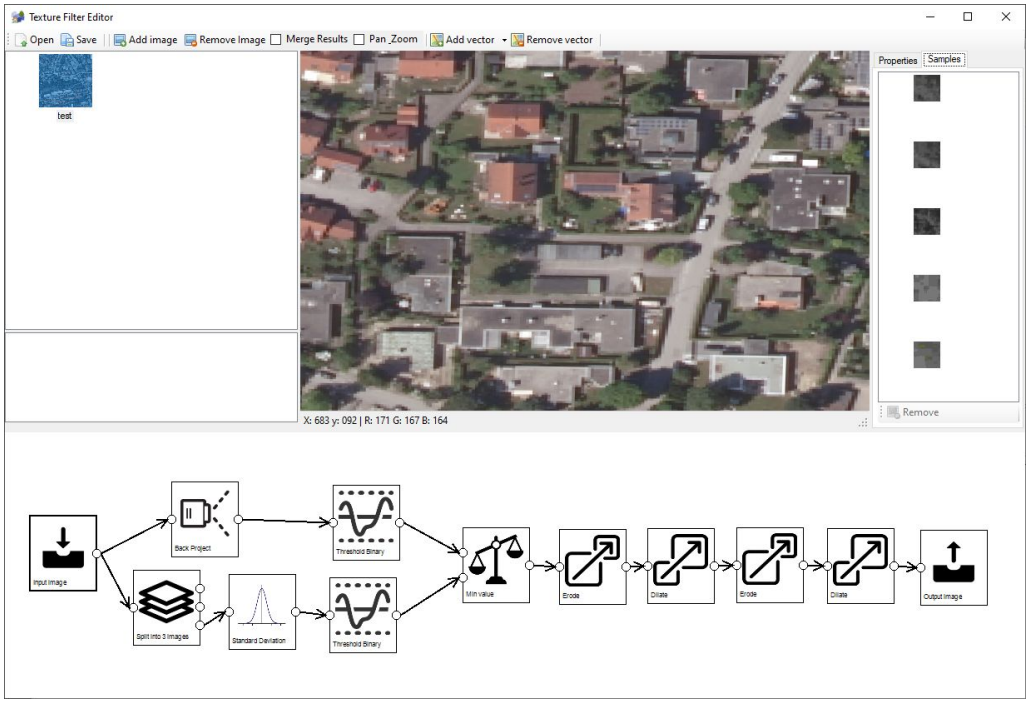



Figure 6.15: The input image for the vegetation detection

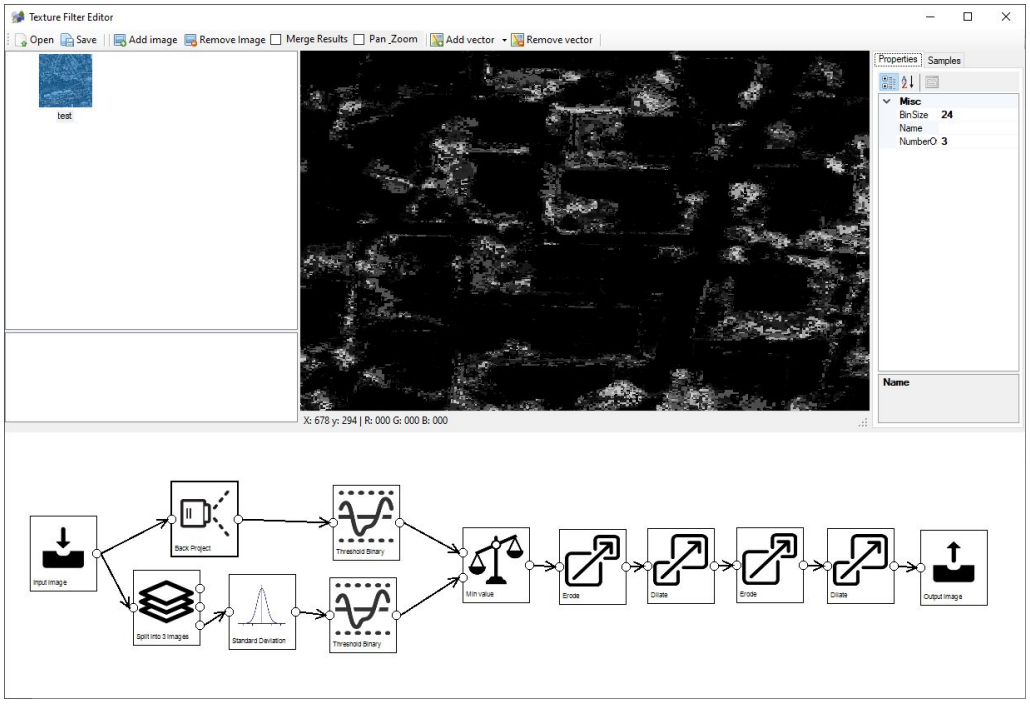



Figure 6.16: Determining which parts are similar to the sample images using the back project step

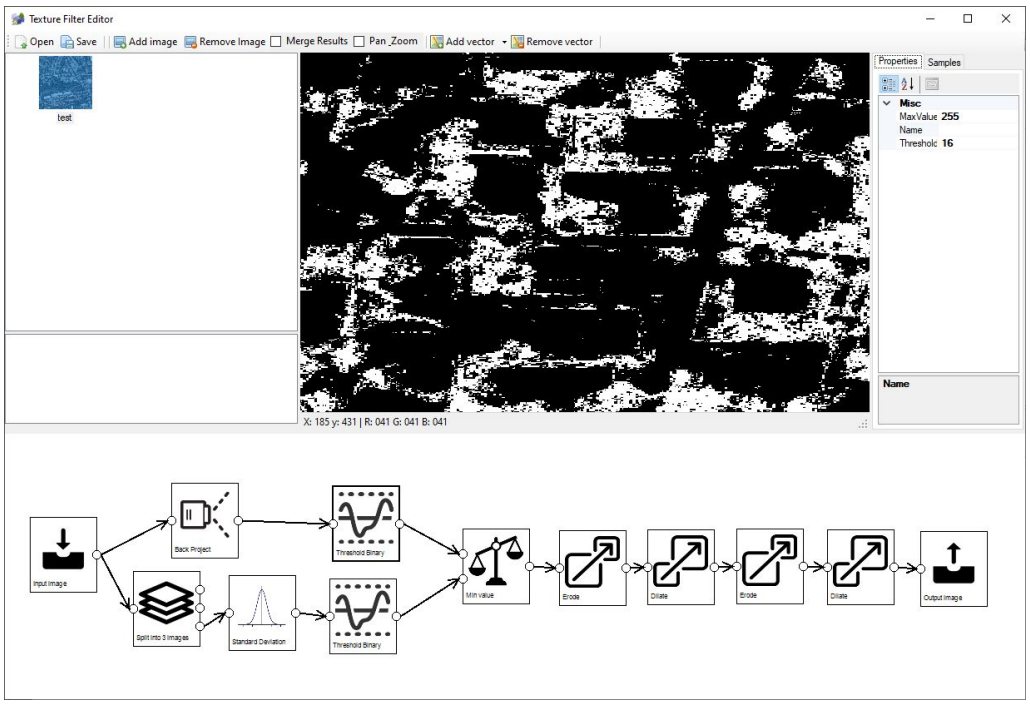



Figure 6.17: Applying a threshold to the detected vegetation

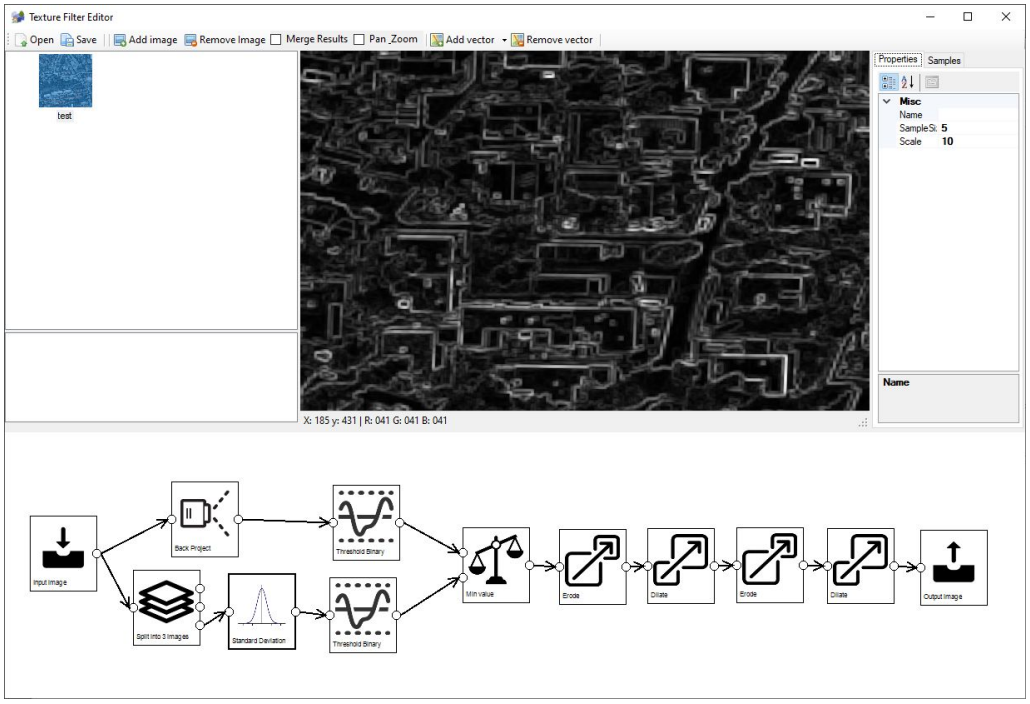



Figure 6.18: Calculating the standard deviation of blue channel of the input image

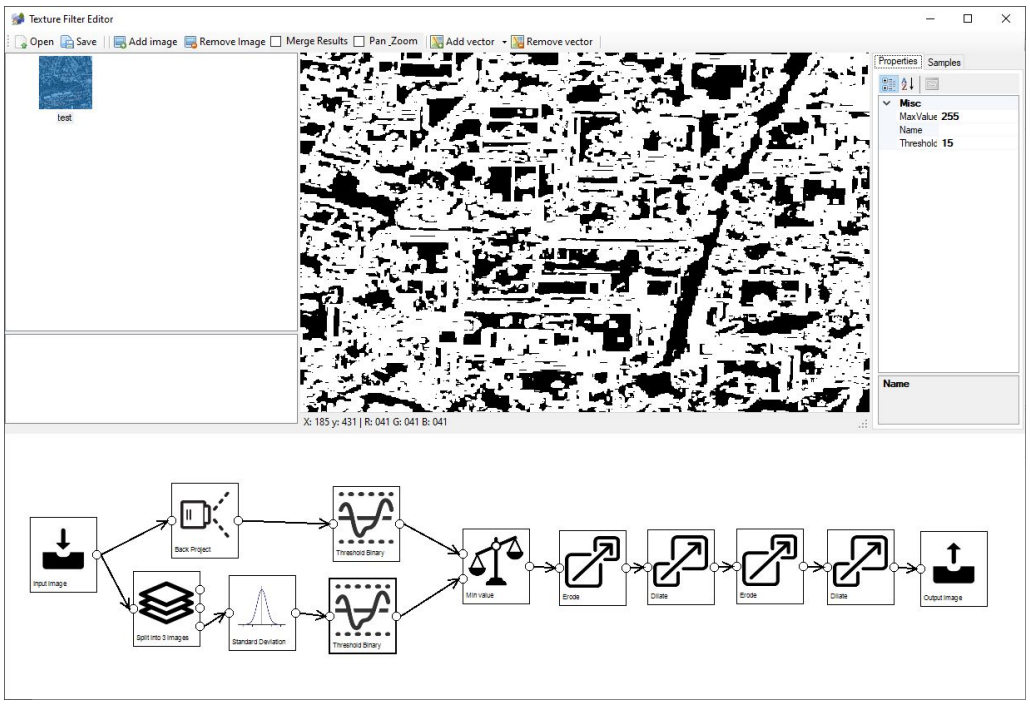



Figure 6.19: Applying a threshold to the standard deviation

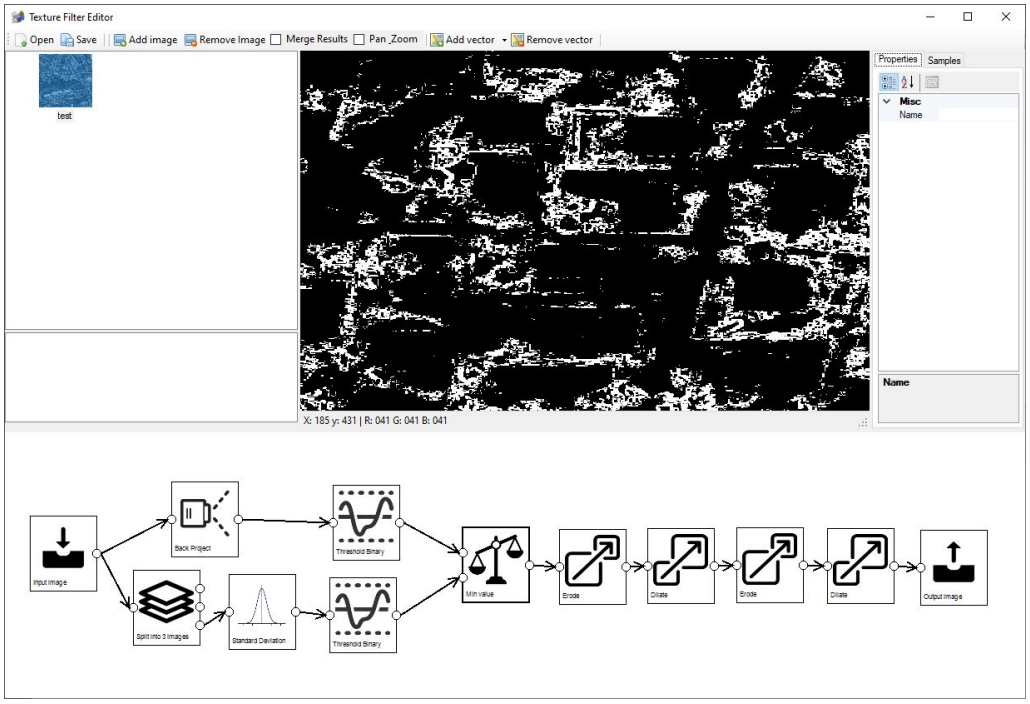



Figure 6.20: Minimum value of the standard deviation and detected vegetation

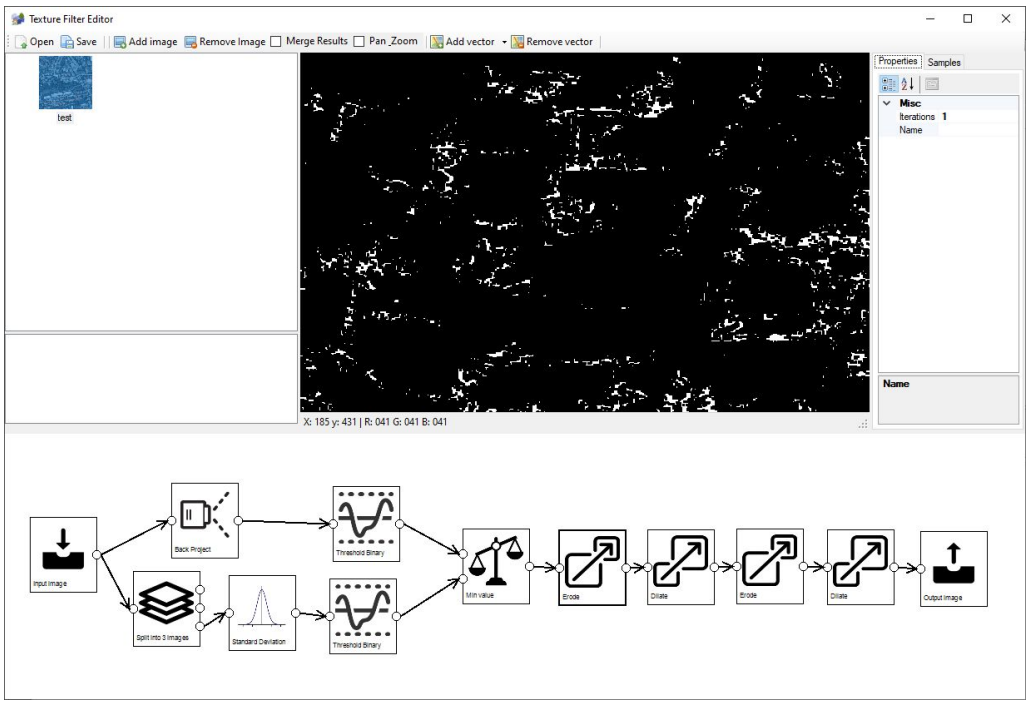



Figure 6.21: Eroding to remove noise

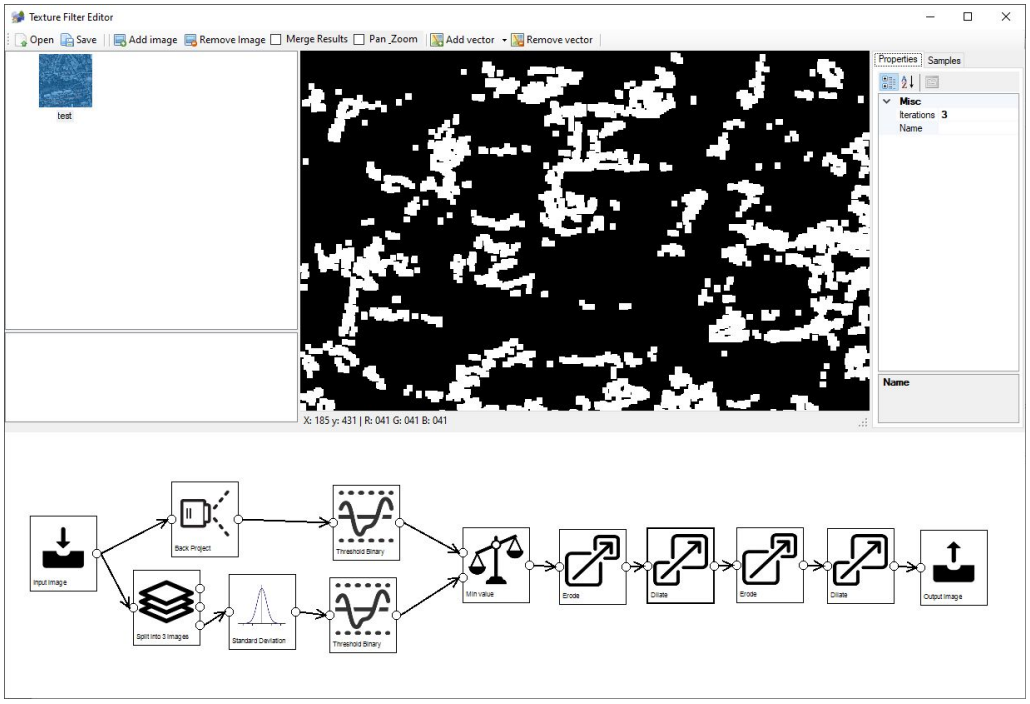



Figure 6.22: Dilating to grow the detected areas

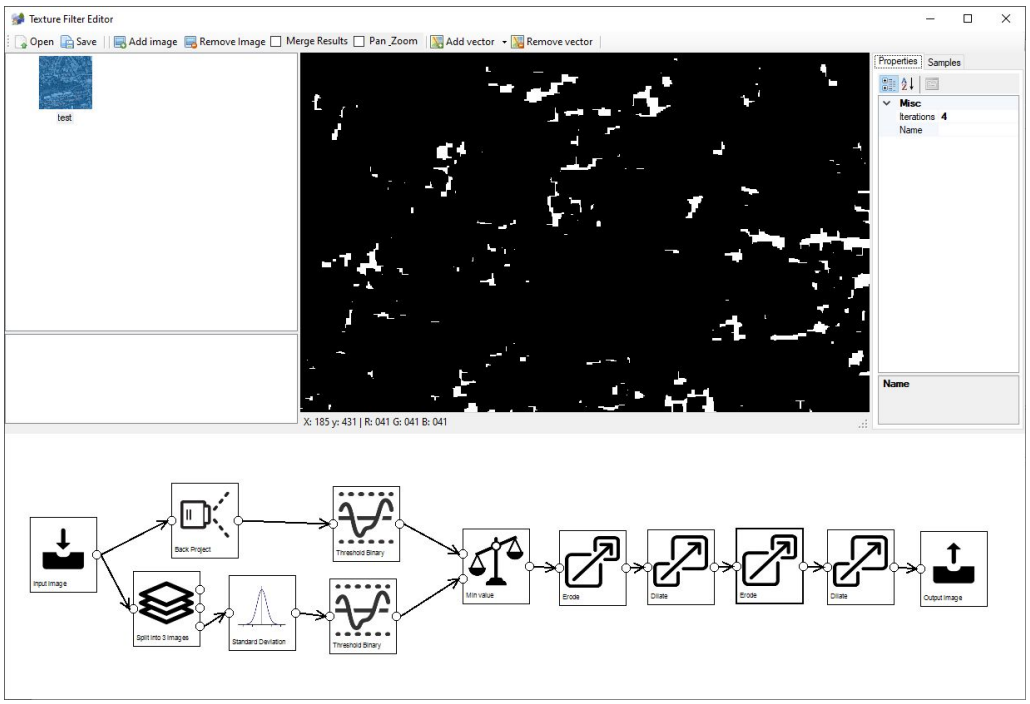



Figure 6.23: Eroding again to remove more noise

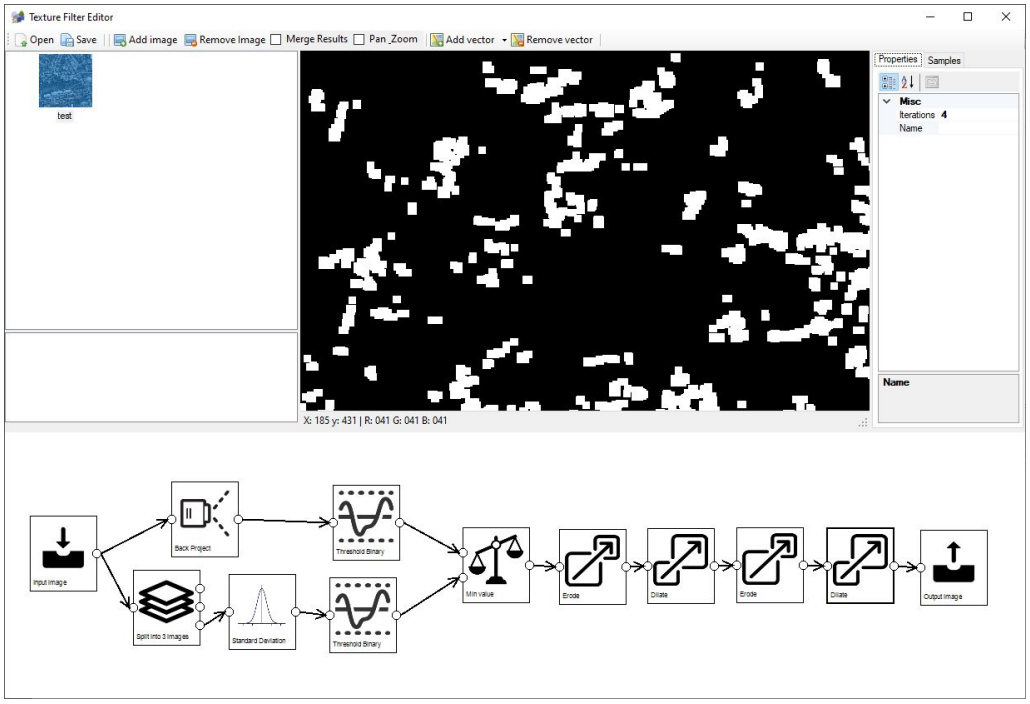



Figure 6.24: Dilating again to grow the detected areas

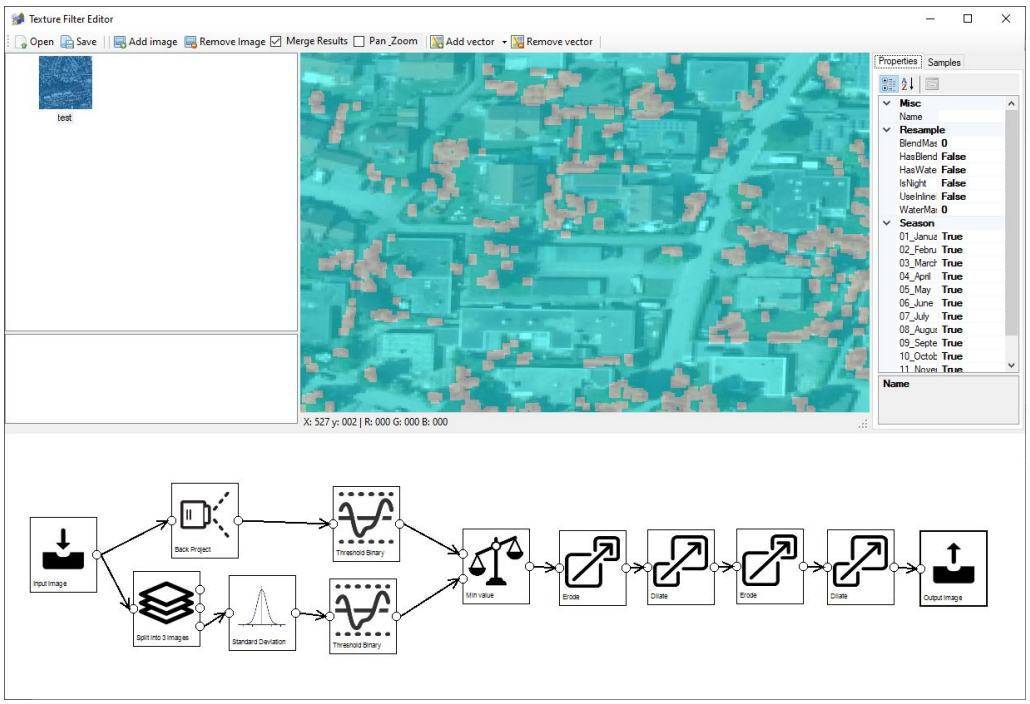



Figure 6.25: The output image with the detected vegetation merged

### 6.3.2 Vegetation feature detection using NDVI

This section discusses an example texture filter that can be used to detect vegetation based on the NDVI value. This filter contains the following steps:

1. The input image to use, see Figure 6.26.
2. Calculate the NDVI value, see Figure 6.27.
3. Binary threshold with a value of 148 to select only the areas with a lot of vegetation (remember a value of 128 corresponds to a NDVI of 0.0), see Figure 6.28.
4. Erode step with 1 iteration to get rid of small noise, see Figure 6.29.
5. Dilate step with 2 iterations to grow the detected vegetation back to the normal size, see Figure 6.30.
6. The output image, shown merged in Figure 6.31 with the input image, to see that the vegetation matches the vegetation in the image well.

See section 5.14 for an example configuration that can be used to run this texture filter.

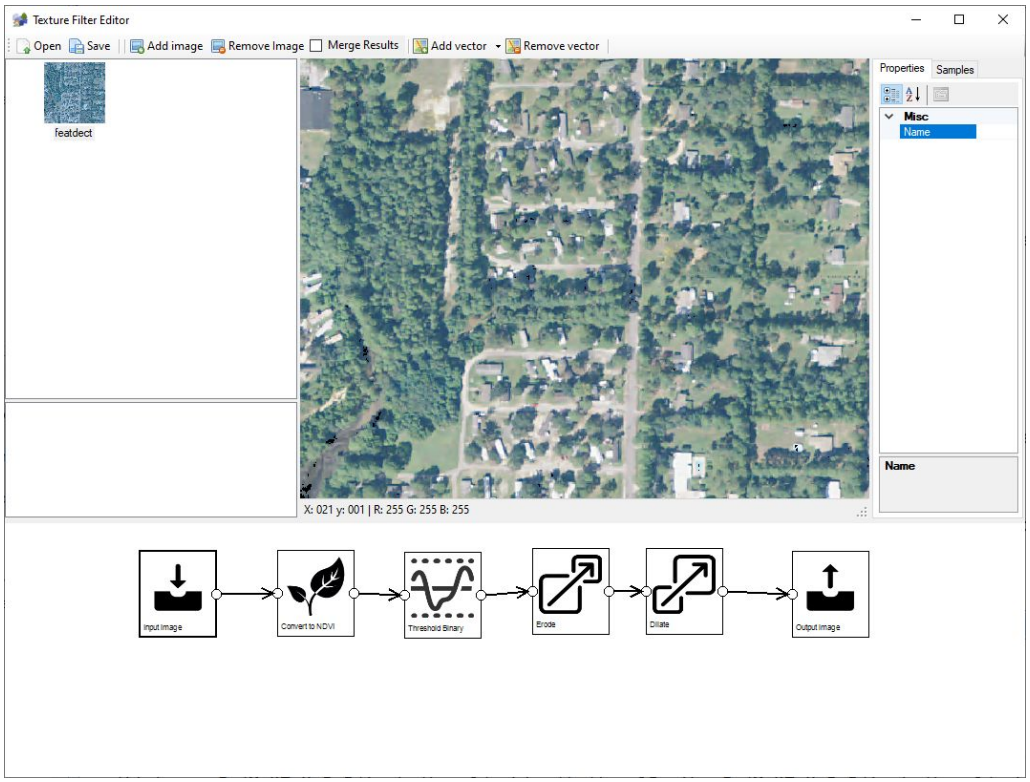



Figure 6.26: The input image for the vegetation detection

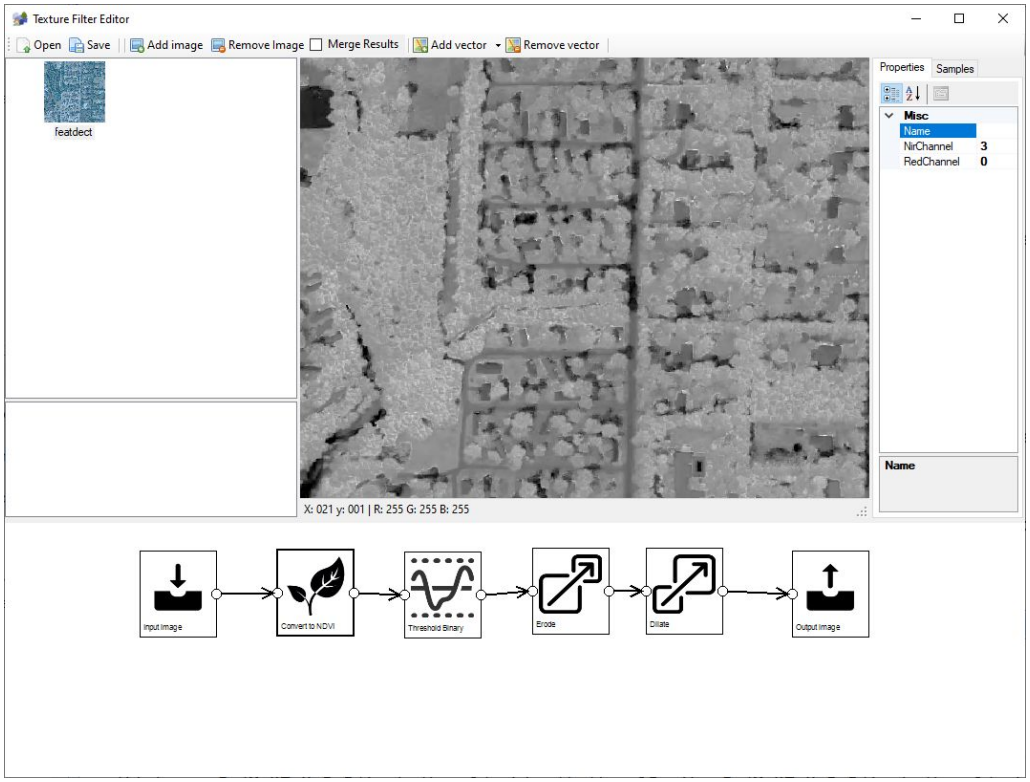



Figure 6.27: The NDVI value calculated from the input image




Figure 6.28: Applying a threshold to the NDVI value to select areas with vegetation




Figure 6.29: Eroding to remove noise from the detection




Figure 6.30: Dilating to restore the vegetation size




Figure 6.31: The final detection result merged with the input image

### 6.3.3 Water feature detection using NDWI

This section discusses an example texture filter that can be used to detect water based on the NDWI value. This filter contains the following steps:

1. The input image to use, see Figure 6.32.
2. Calculate the NDWI value, see Figure 6.33.
3. Binary threshold with a value of 216 to select only the areas with a lot of water, see Figure 6.34.
4. Erode step with 1 iteration to get rid of small noise, see Figure 6.35.
5. Dilate step with 2 iterations to grow the detected water back and to make sure that tiny holes in the water are filled, see Figure 6.36.
6. Erode step with 1 iteration to shrink the water back to the normal size, see Figure 6.37.
7. The output image, shown merged in Figure 6.38 with the input image, to see that the water matches the vegetation in the image well.

See section 5.14 for an example configuration that can be used to run this texture filter.

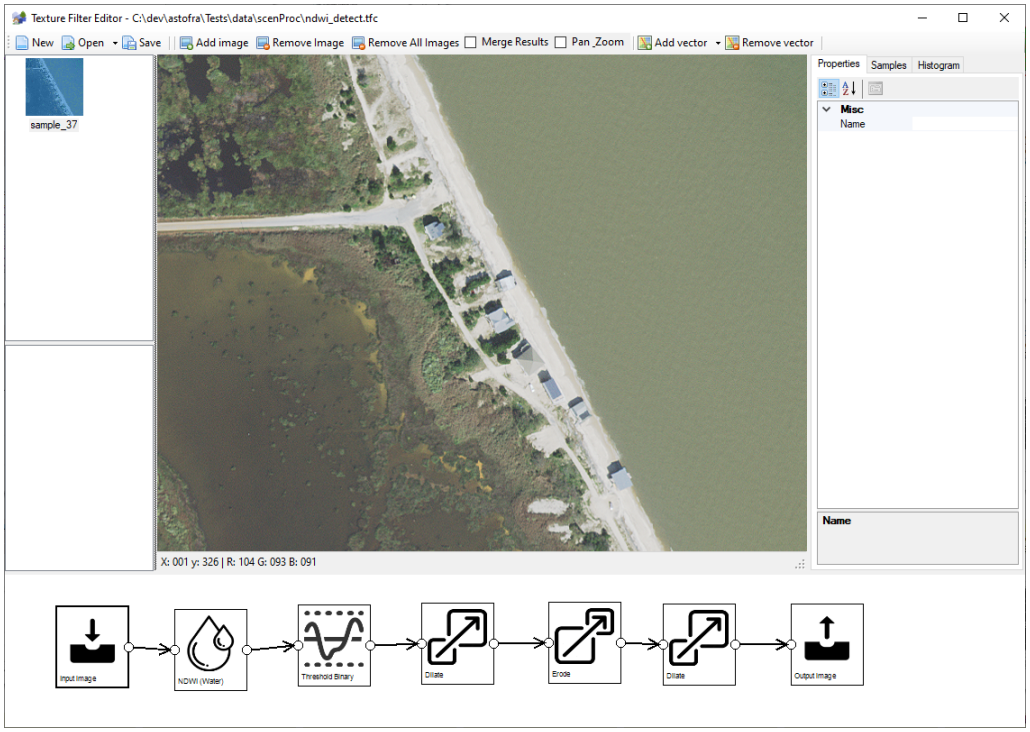



Figure 6.32: The input image for the water detection

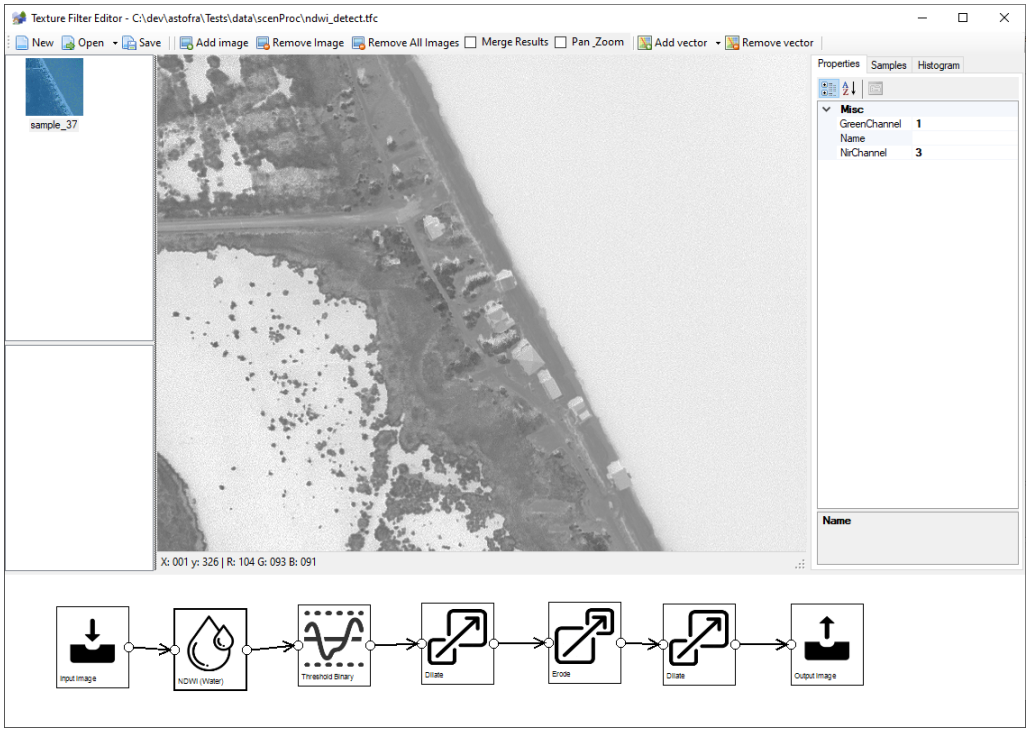



Figure 6.33: The NDWI value calculated from the input image

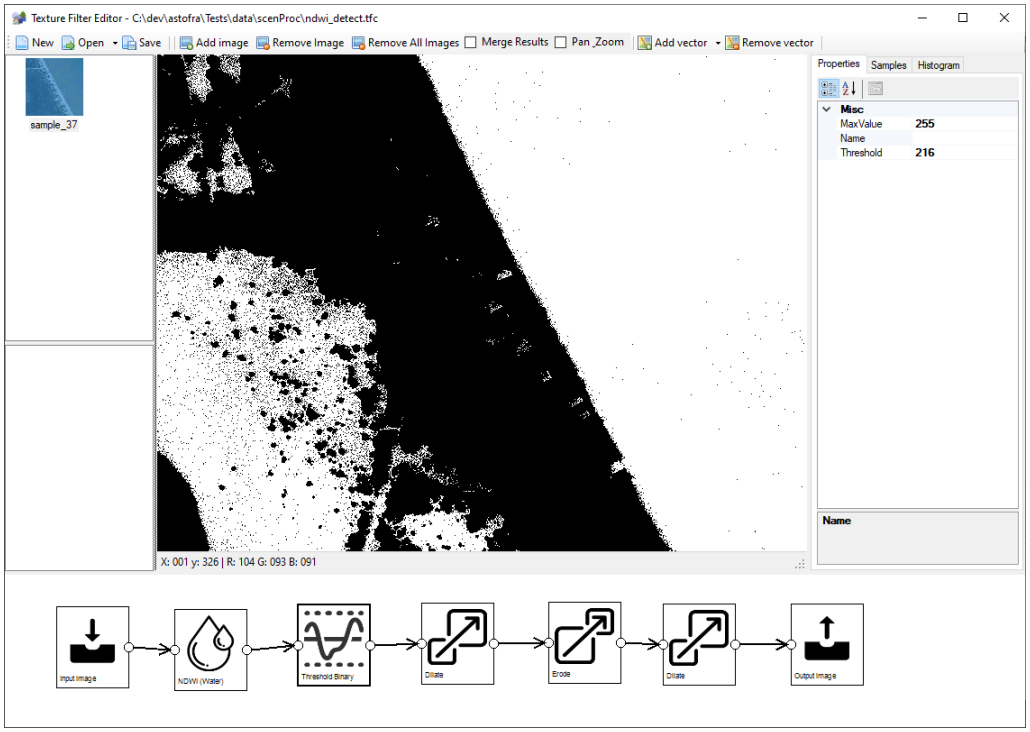



Figure 6.34: Applying a threshold to the NDWI value to select areas with water

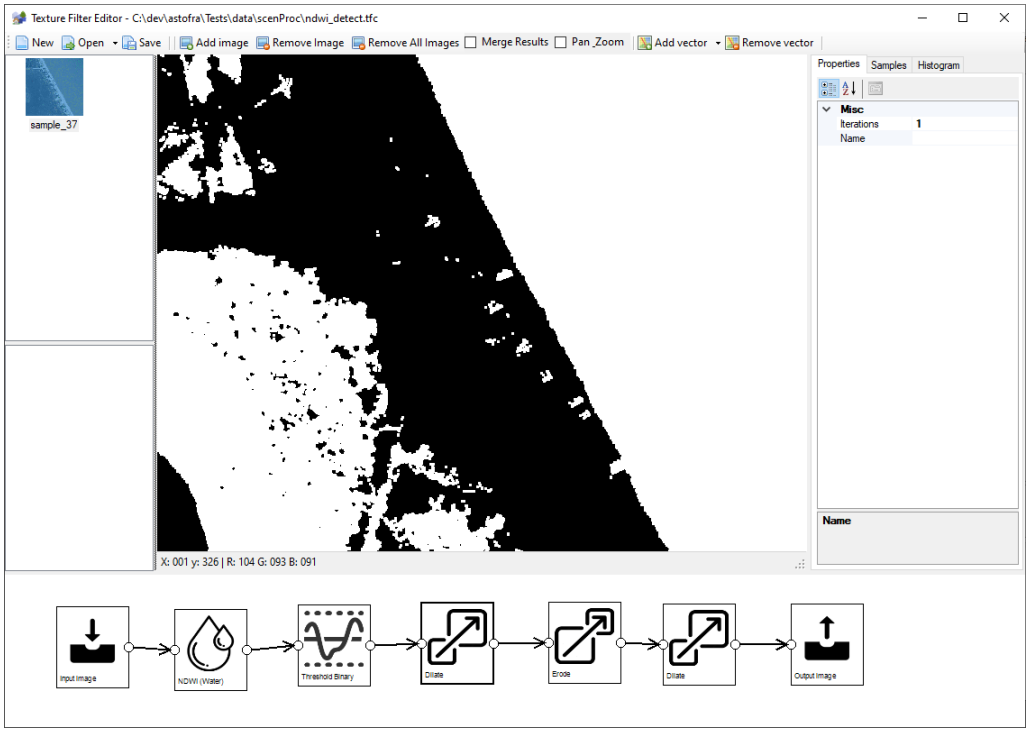



Figure 6.35: Eroding to remove noise from the detection

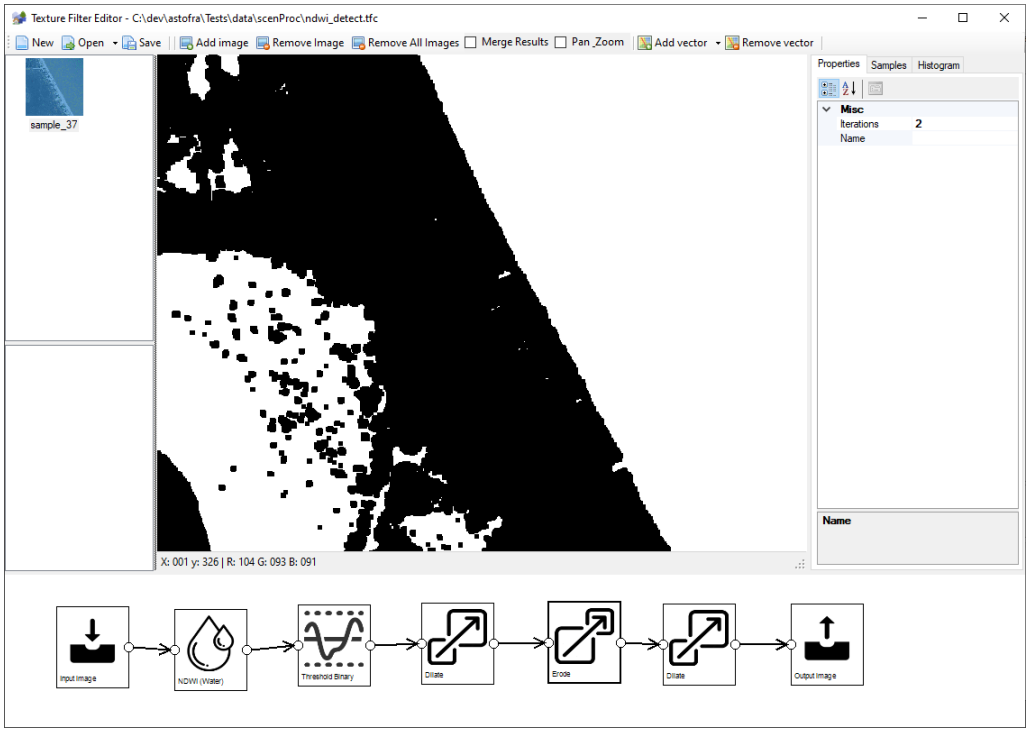



Figure 6.36: Dilating to remove small hoels from the water

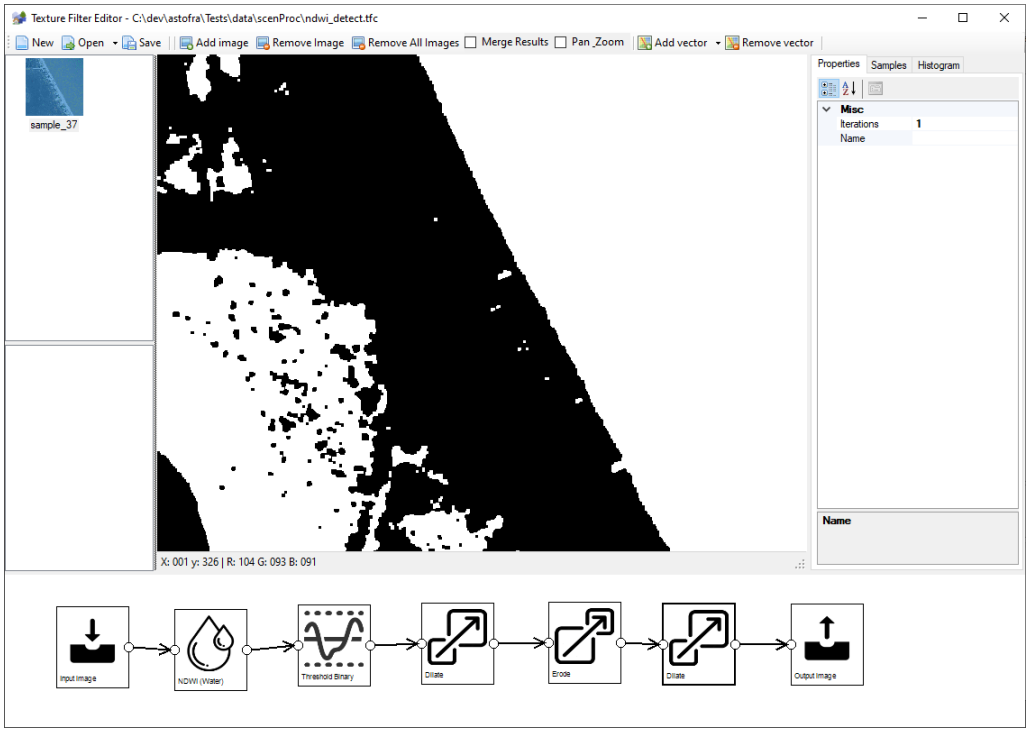



Figure 6.37: Eroding to restore correct water size

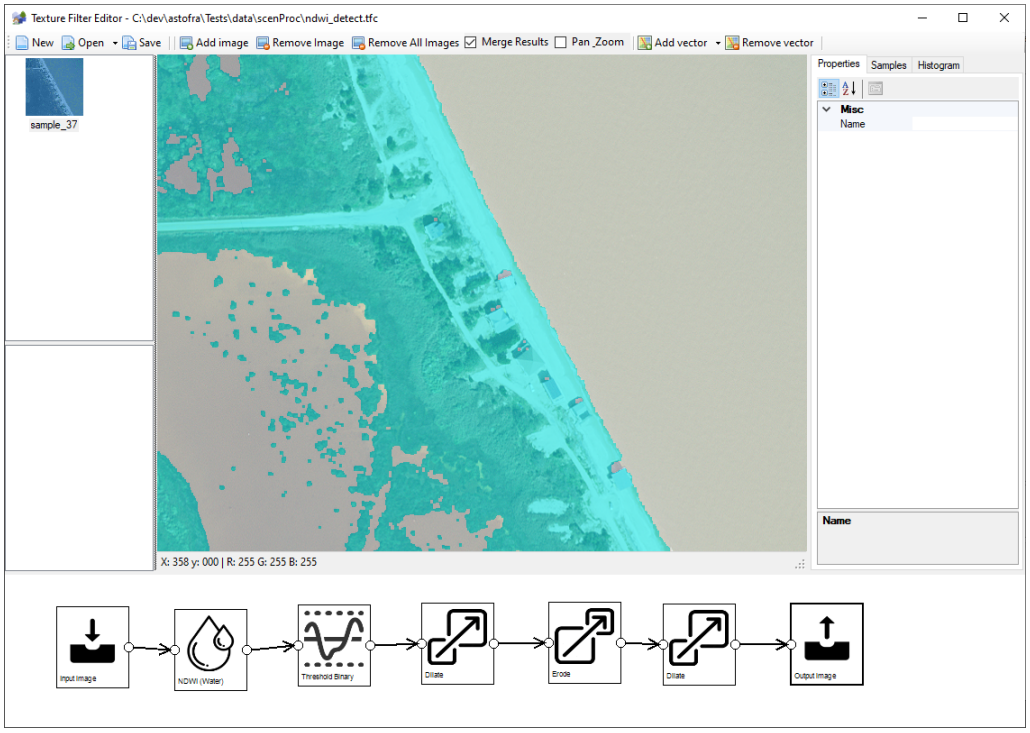



Figure 6.38: The final detection result merged with the input image

### 6.3.4 Water feature detection using TSUWI

This section discusses an example texture filter that can be used to detect water based on the Two Step Urban Water Index (TSUWI). This approach has been taken from the article Two-Step Urban Water Index (TSUWI): A New Technique for High-Resolution Mapping of Urban Surface Water by Wu. This filter contains the following steps:

1. The input image to use, see Figure 6.39.
2. Calculate the UWI value, see Figure 6.40.
3. Binary threshold with a value of 80 to select only the areas with a lot of water, see Figure 6.41.
4. Calculate the USI value, see Figure 6.42.
5. Binary threshold with a value of 170 to select only the areas with a less shadow, see Figure 6.43.
6. Minimum value of the UWI and USI values, this only selected the areas where there is water and no shadows, see Figure 6.44.
7. Erode step with 1 iteration to get rid of small noise, see Figure 6.45.
8. Dilate step with 4 iterations to grow the detected water back and to make sure that tiny holes in the water are filled, see Figure 6.46.
9. Erode step with 3 iteration to shrink the water back to the normal size, see Figure 6.47.
10. The output image, shown merged in Figure 6.48 with the input image, to see that the water matches the vegetation in the image well.

See section 5.14 for an example configuration that can be used to run this texture filter.

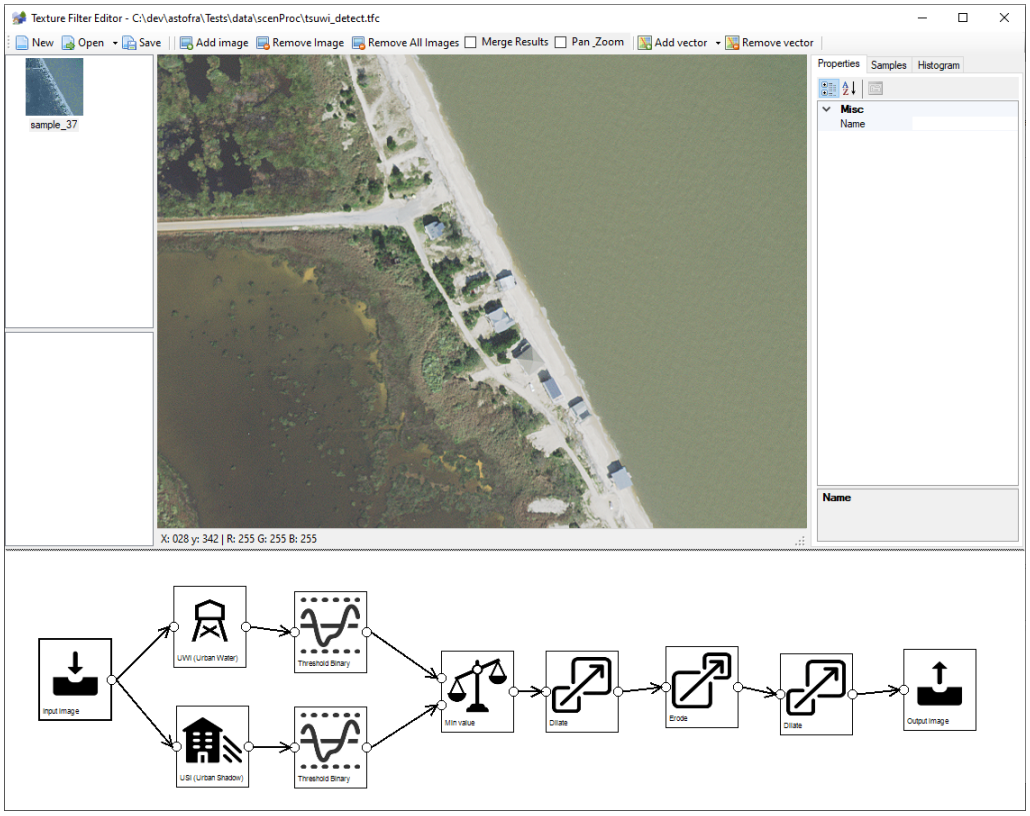



Figure 6.39: The input image for the water detection

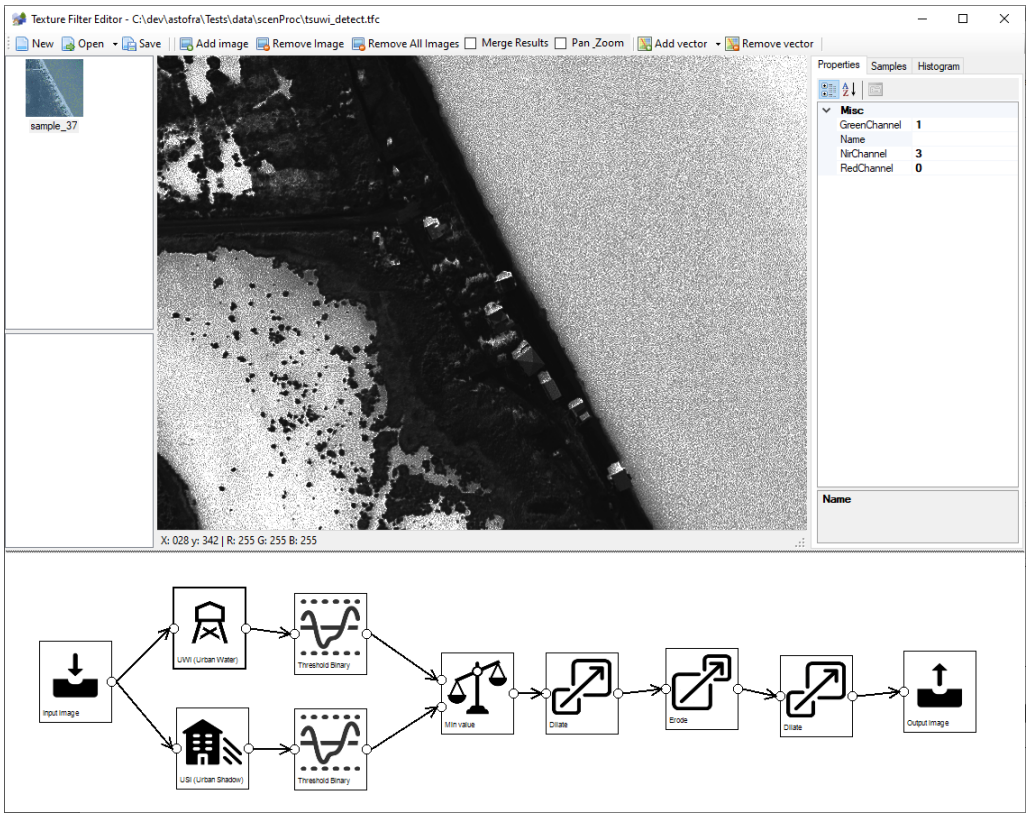



Figure 6.40: The UWI value calculated from the input image

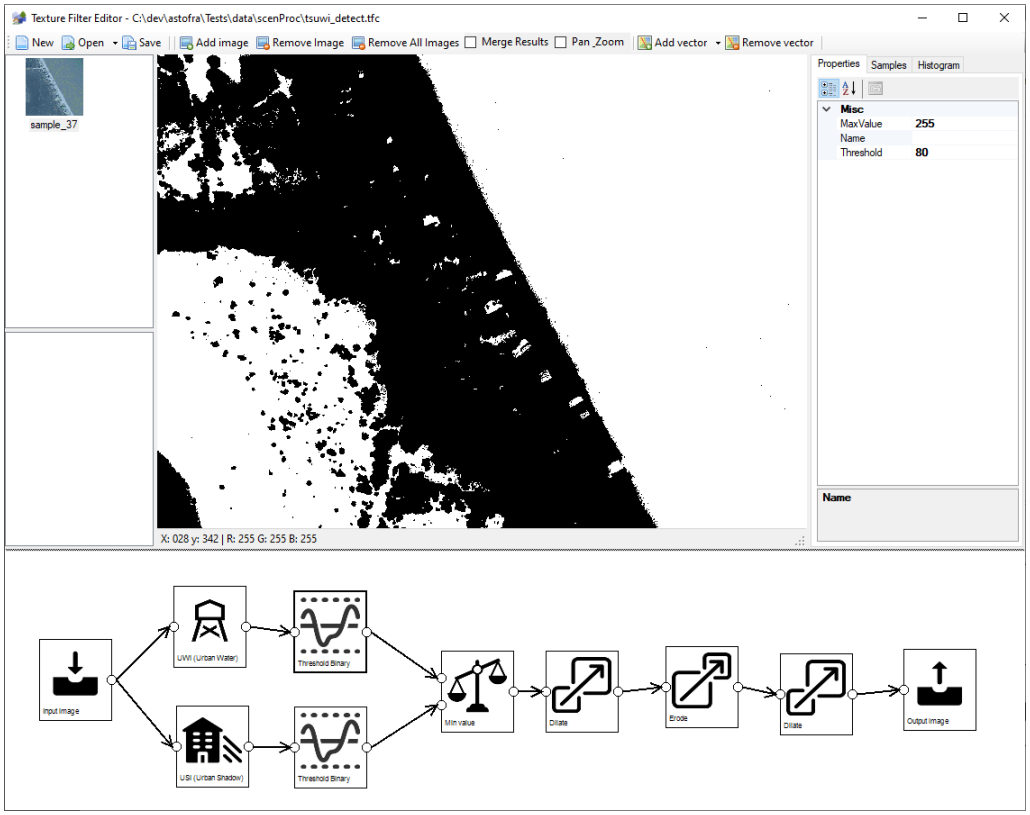



Figure 6.41: Applying a threshold to the UWI value to select areas with water

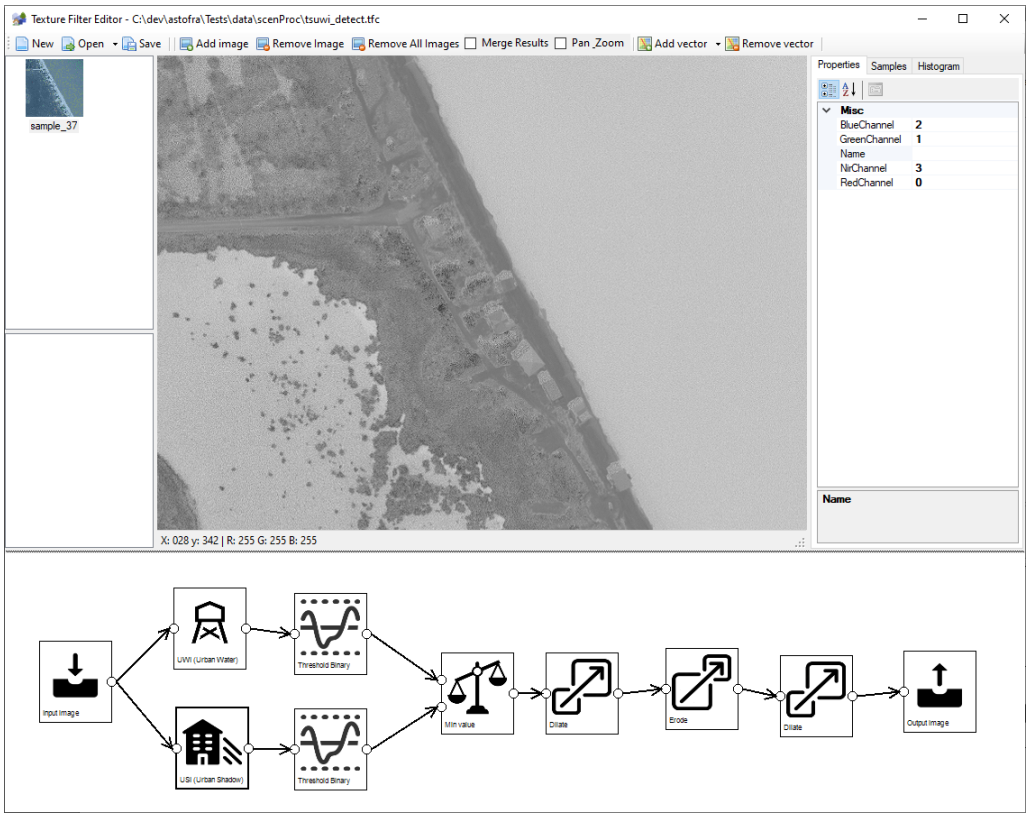



Figure 6.42: The USI value calculated from the input image

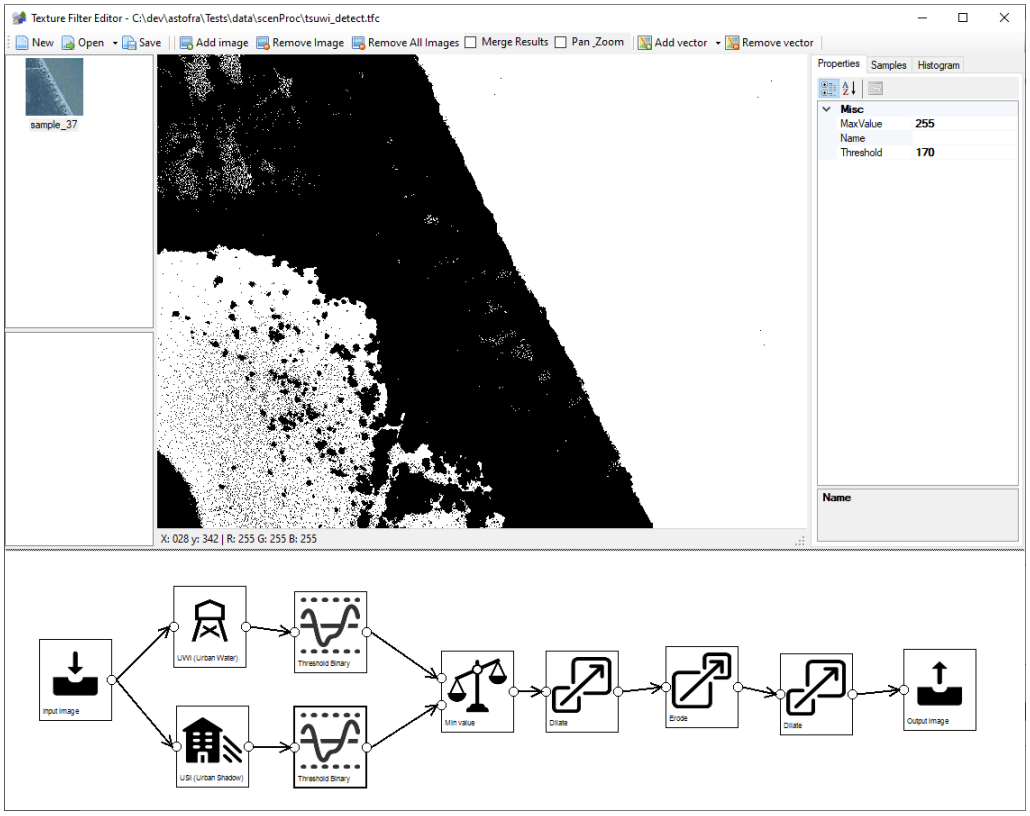



Figure 6.43: Applying a threshold to the USI value to select areas without shadow

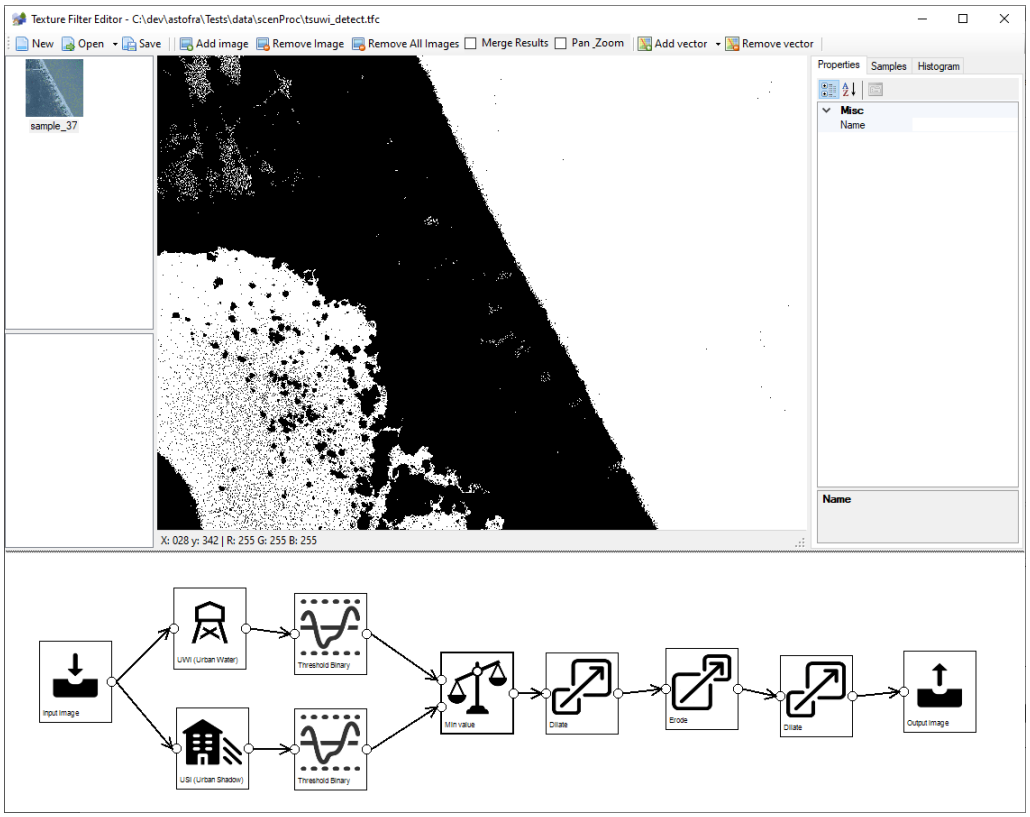



Figure 6.44: Applying minimum value to select areas with water and no shadows

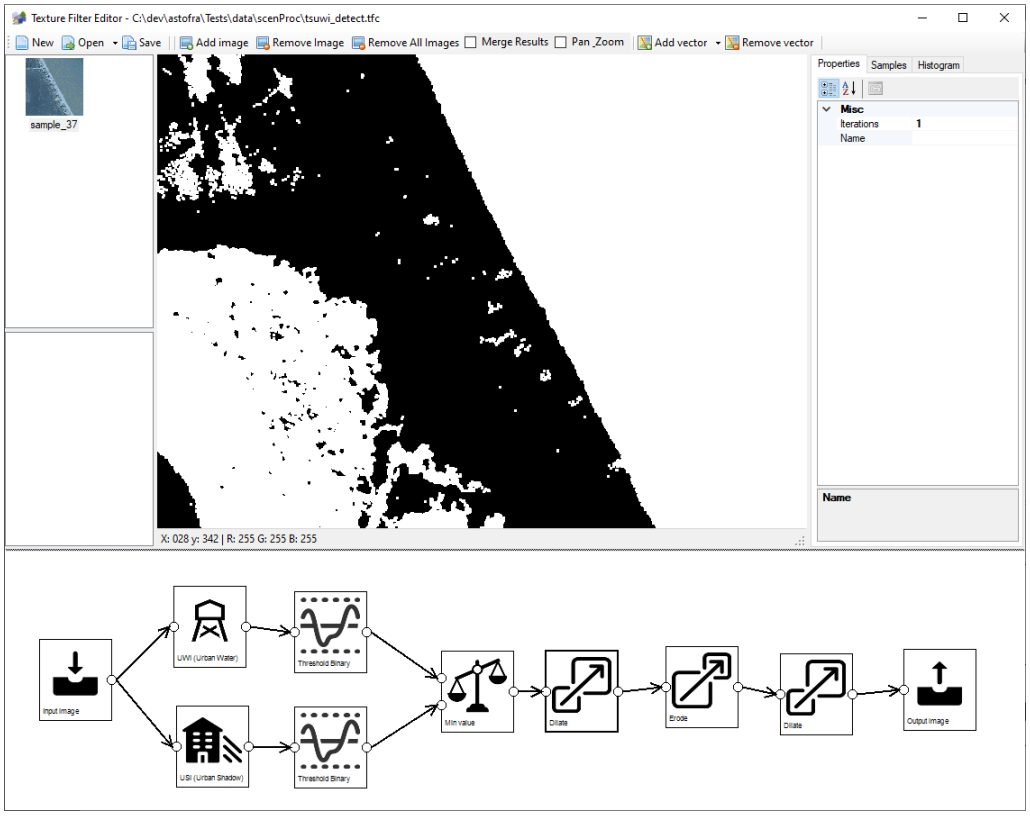



Figure 6.45: Eroding the result to get rid of noise

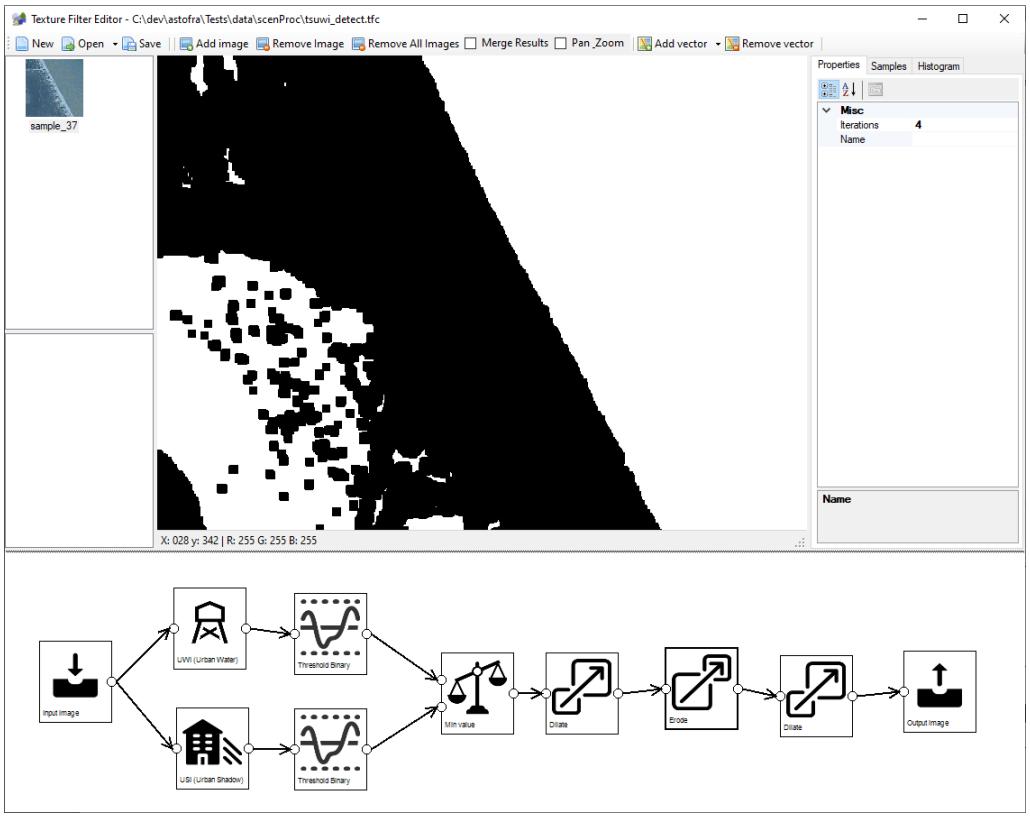



Figure 6.46: Dilating the result to fill holes in the water

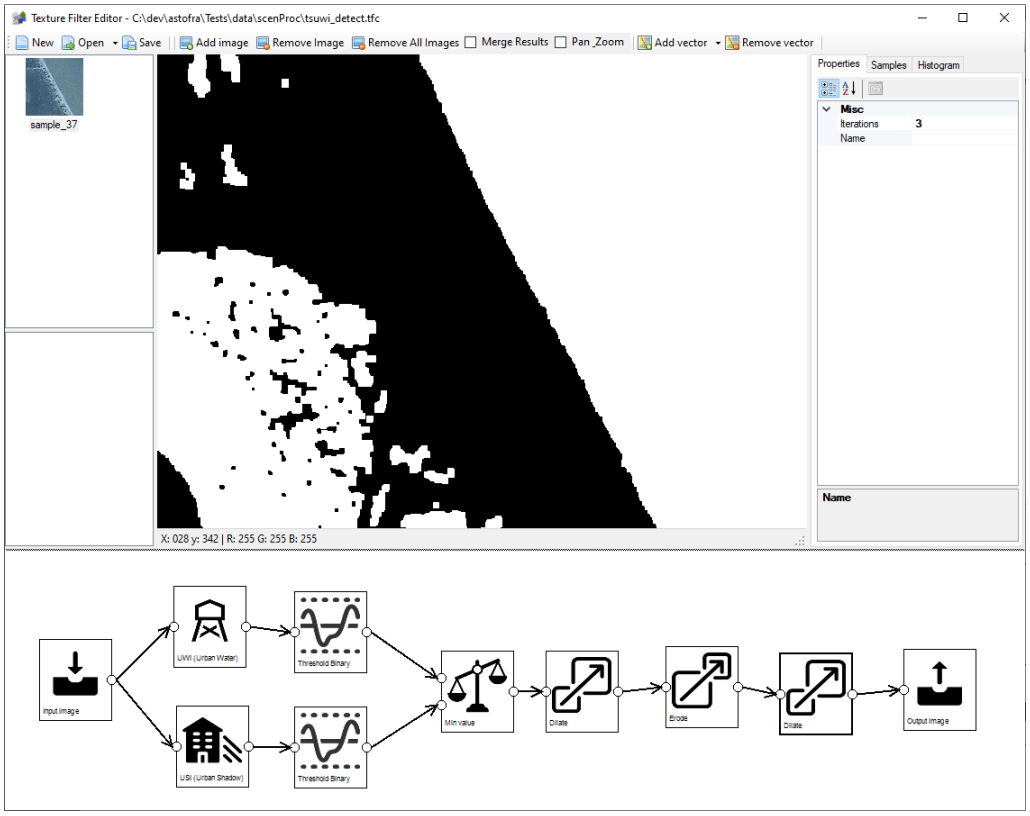



Figure 6.47: Eroding to restore the correct water size

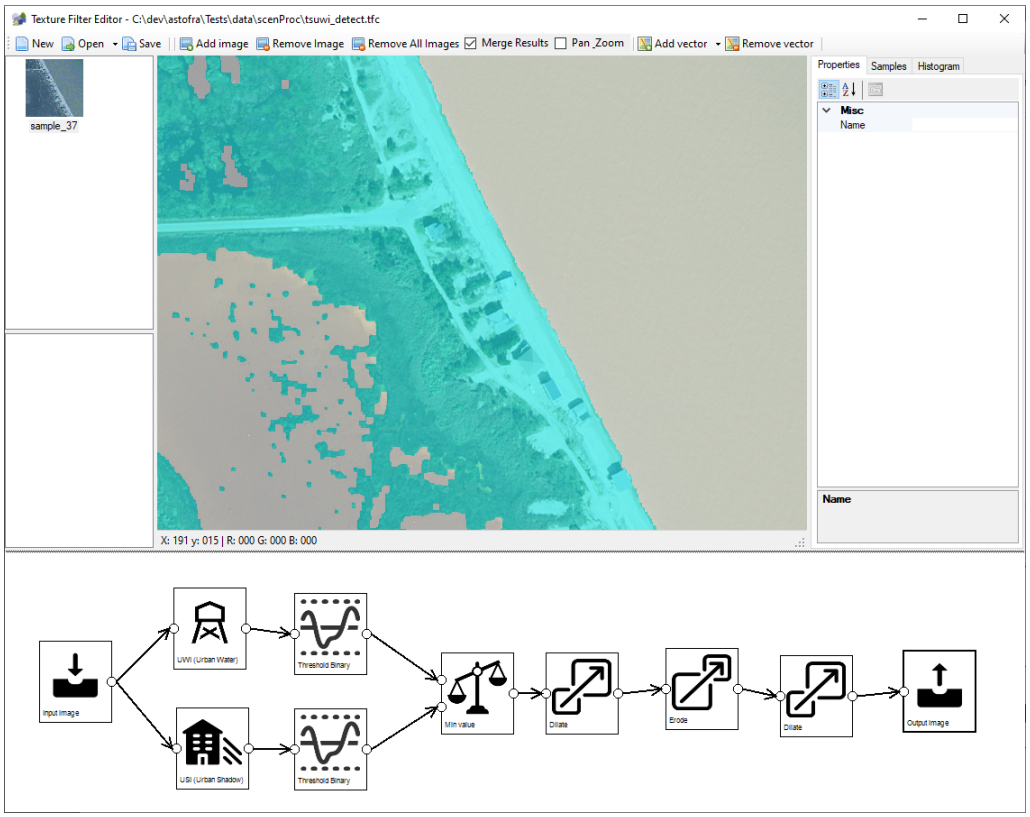



Figure 6.48: The final detection result merged with the input image

### 6.3.5 Vegetation detection using SVM machine learning

This section provides a sample script that uses the SVM machine learning algorithm to detect vegetation in an image. The advantage of using machine learning is that you don't have to tune the exact criteria to select parts of the image as vegetation yourself. You only have to define which sample points are used as training data and the machine learning will determine which criteria belong to it.

Another difference between this sample texture filter and the previous ones is that the classification is done on objects (segments) and not at pixel level. In general this gives better results, as the noise of a single pixel with an abnormal value will have less influence on the classification results. The multi resolution segmentation step in the texture filter is used to identify the segments in the image.

This filter contains the following steps:

1. The input image is a 4-band image, containing R, G, B and NIR bands, see Figure 6.49. The sample points that have been defined for the image are also shown. A number of positive sample points have been made in the vegetation we want to detect and a number of negative sample points has been made in features we don't want to detect (grass, buildings, roads).

   Defining these sample points is an iterative process. You will add more sample points to improve the classification, for example by clicking on vegetation areas that have not yet been detected or by adding a negative sample point for areas that should not be detected but currently are.

   In this example only one sample image is used for which sample points are defined. But in a normal project you would have multiple sample images, with representative samples of the area you want to classify. And you would then add sample points to each of these sample images. To see how well the classification works you would also add a few sample images without any sample points, as that is a good indication how accurate the classification is.

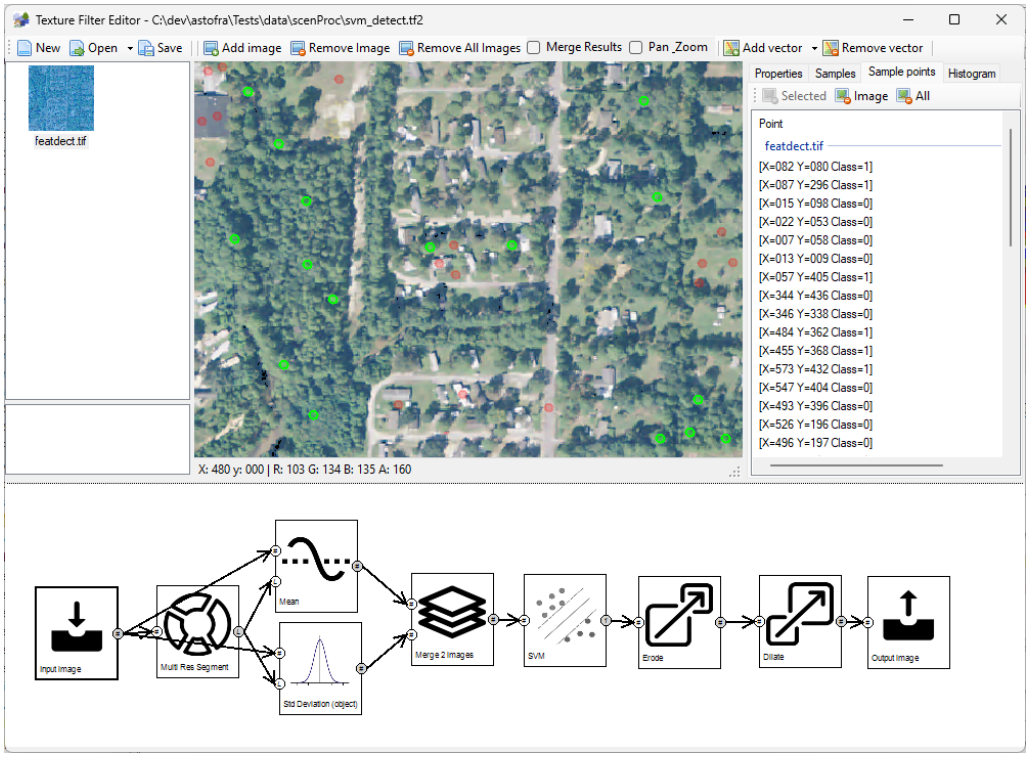



Figure 6.49: The input images showing the sample points used for the machine learning algorithm training data.

2. The segmentation of the image into different objects, see Figure 6.50. The main parameters you want to vary are the Scale parameter and the weights. The aim is to get not too many objects, so you want the Scale not to be not too small. But if the objects get too large they are less homogeneous and that affects the classification quality. So by trial and error you should find a good balance. Looking at the mean output (next step) is a good way to see if the objects are homogeneous or not.

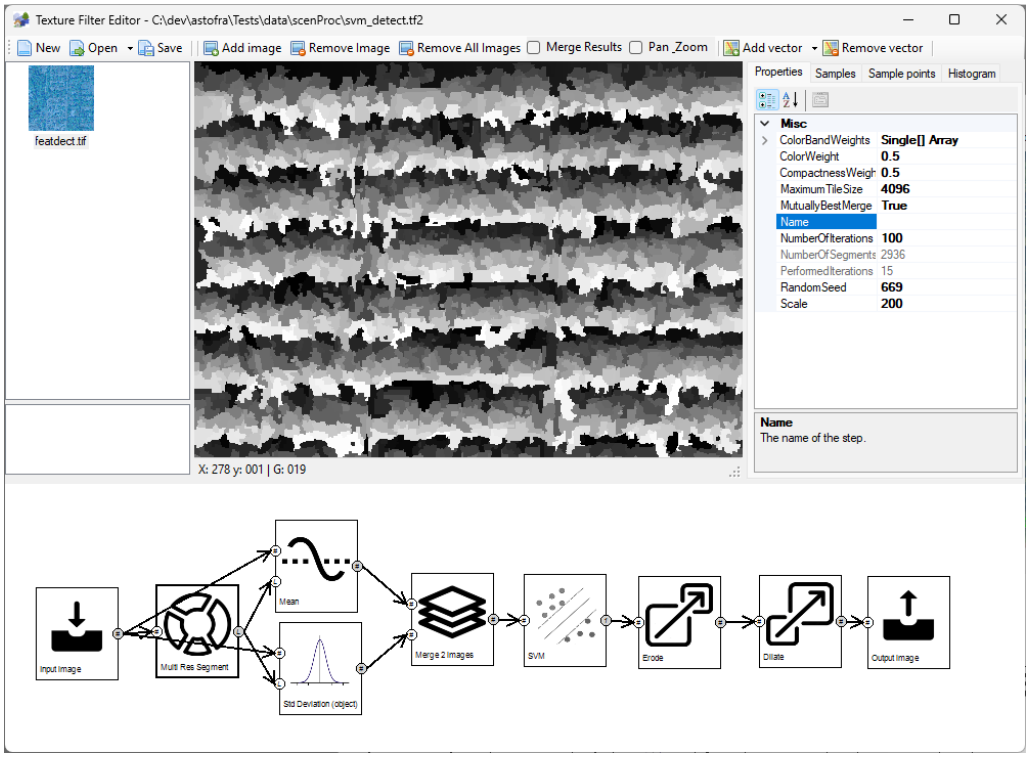



Figure 6.50: The multi resolution segmentation step that divides the images into segments

3. The mean color and NIR values for each of the objects in the image is calculated, see Figure 6.51.

4. The standard deviation of the color and NIR values for each of the objects in the image is calculated, see Figure 6.52.

5. The mean and standard deviation images are merged into one image, see Figure 6.53. Each of the input images has 4 bands (R, G, B and NIR), that means that the merged image has 8 bands in total.

6. The SVM step performs the classification using a machine learning algorithm, see Figure 6.54. There are many parameters that you can tune for this step, see section 6.2.46 for all the details. The selection of the best kernel type is probably the most important parameter that you need to decide. It takes some trial and error to see which one works best, although Rbf is probably a good one to start with.

   The machine learning algorithms will use each of the bands of the input image as a feature to consider in the classification. So it is the combination of these 8 values (Mean R, G, B, NIR and Std R, G, B, NIR) that are used to determine if an object is vegetation or not.

   When you have changed parameters of the SVM step or if you have changed the connectors of this step, the machine learning algorithm needs to be trained again. Especially when the training data is based on different sample images this might take some time. The waiting cursor is shown in the texture filter editor when the calculations are still busy. Once the algorithm has been trained generating the classification results is quicker. If you save the

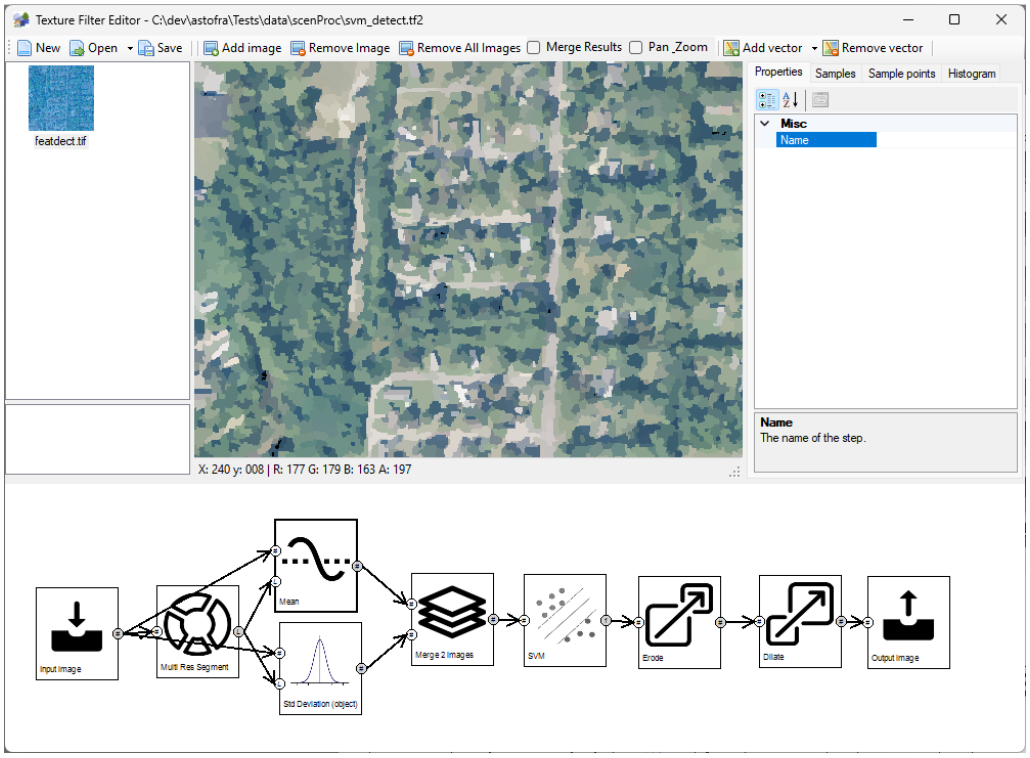



Figure 6.51: The mean color of each segment

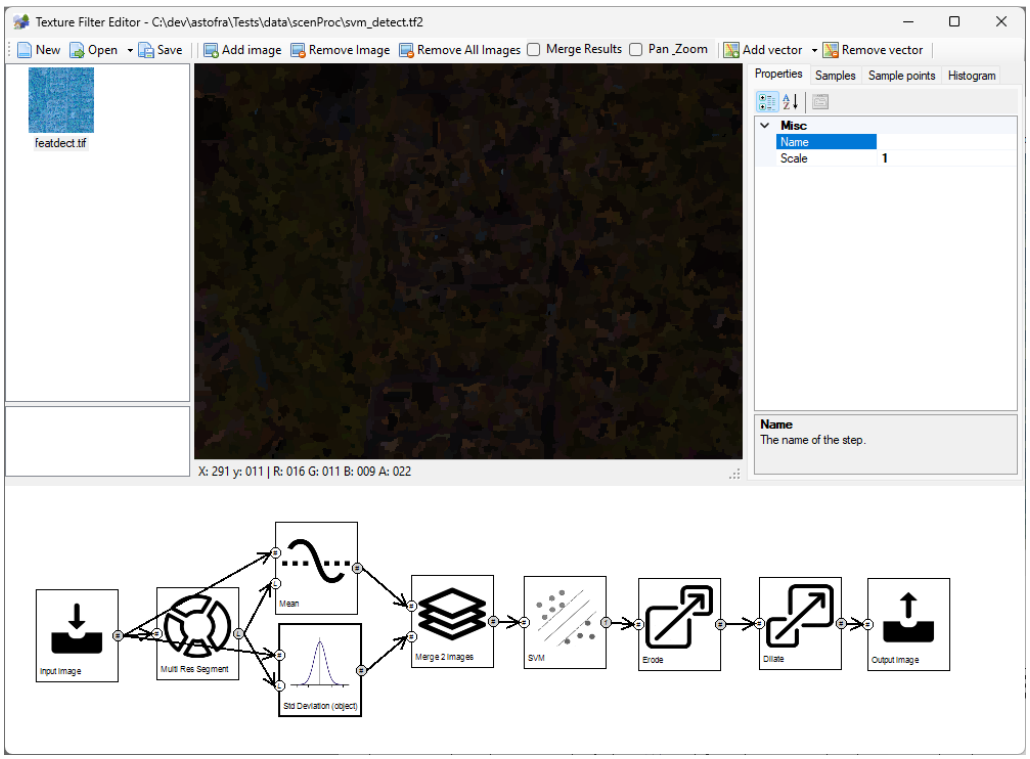



Figure 6.52: The standard deviation of each segment

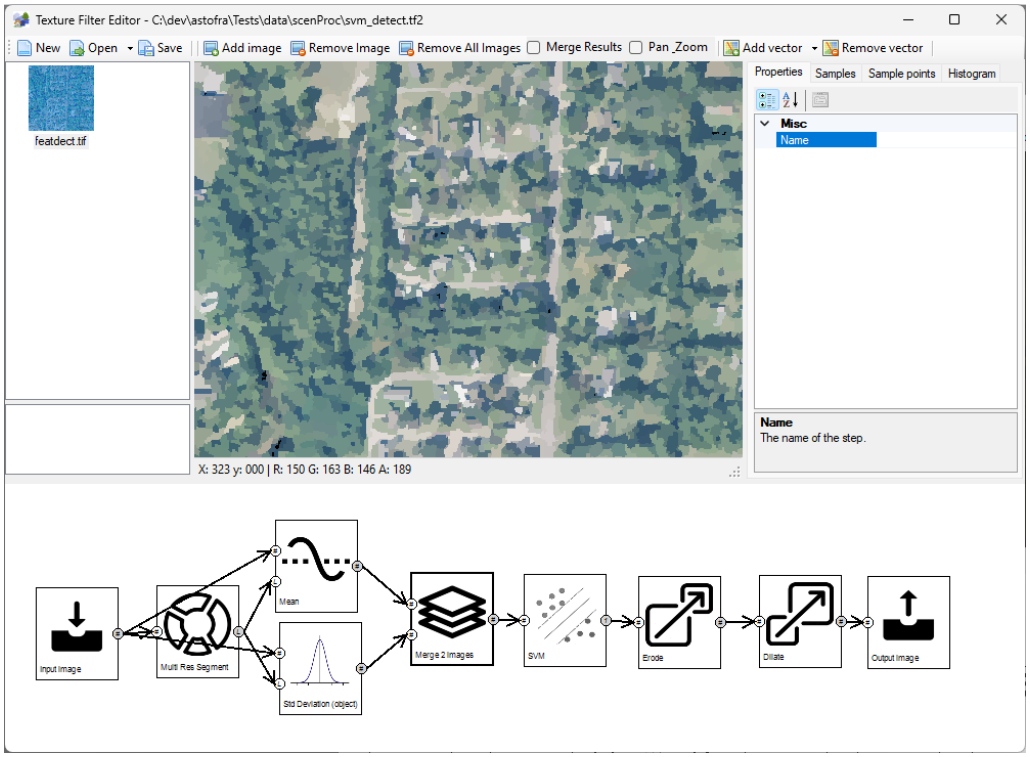



Figure 6.53: Merging the mean and standard deviation images into one image

texture filter configuration file, the training data is saved as well. So if you run the texture filter from your script the training does not have to be performed again.

7. A 2 iteration erode step is performed on the SVM output to get rid of small vegetation areas, see Figure 6.55.
8. A 4 iteration dilate step is performed to grow the detected vegetation back to normal size, see Figure 6.56.
9. The output, the detected vegetation, is shown here merged with the input image, see Figure 6.57.

See section 5.14 for an example configuration that can be used to run this texture filter.

This example only uses the color and standard deviation as features for the machine learning classification. Based on the literature, for example the article Comparing Machine Learning Classifiers for Object-Based Land Cover Classification Using Very High Resolution Imagery by Qian, suggests that better results can be achieved by using the color, standard deviation, NDVI, NDWI, brightness and maximum intensity difference as features for the classification. But as that would make this example overly complex a simplified approach has been shown here. It is easy to add additional classification features to this texture filter, by merging more images into the image that is provided to the SVM step.

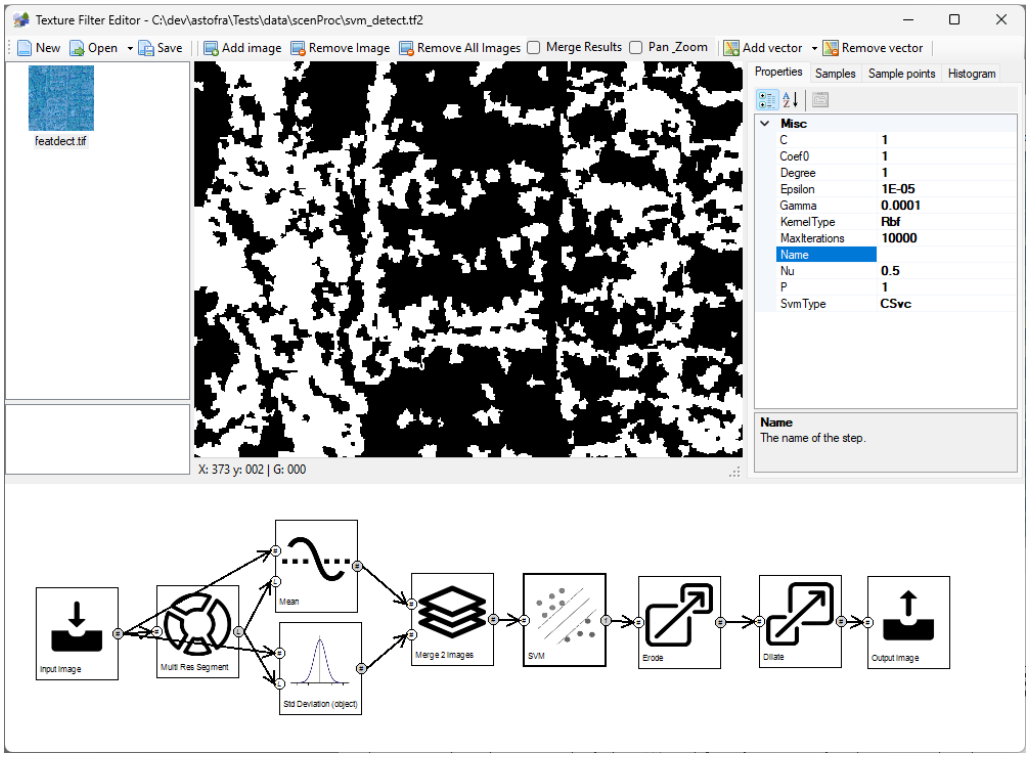



Figure 6.54: The SVM machine learning step that detects the vegetation

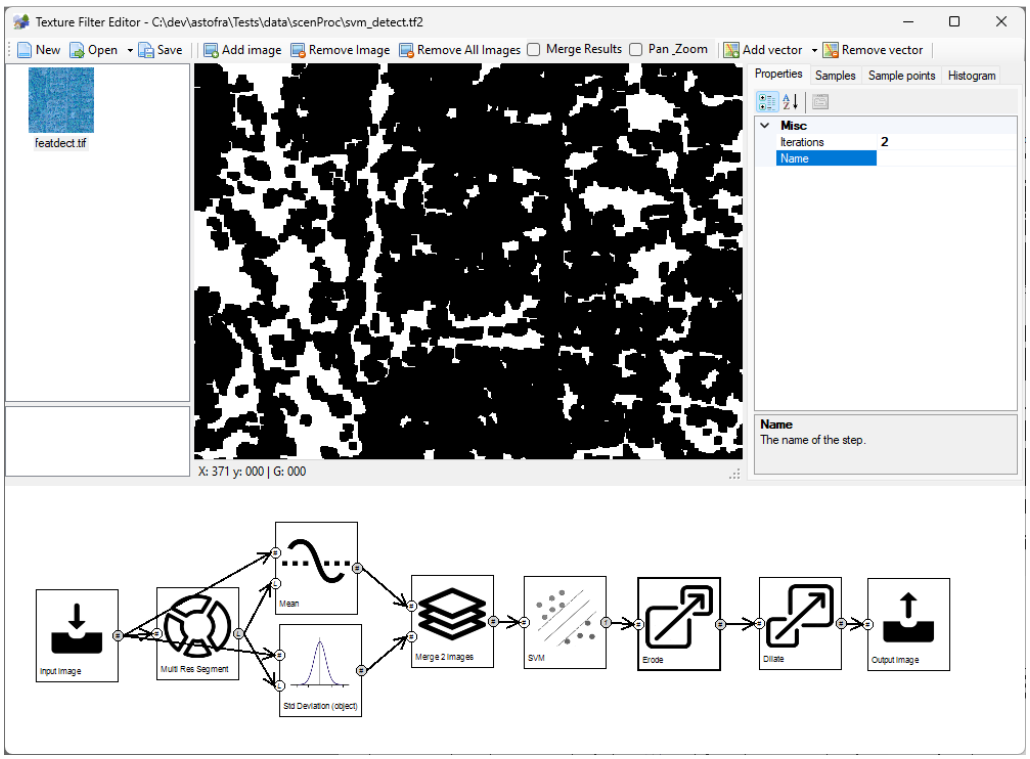



Figure 6.55: Eroding the result to filter out small vegetation

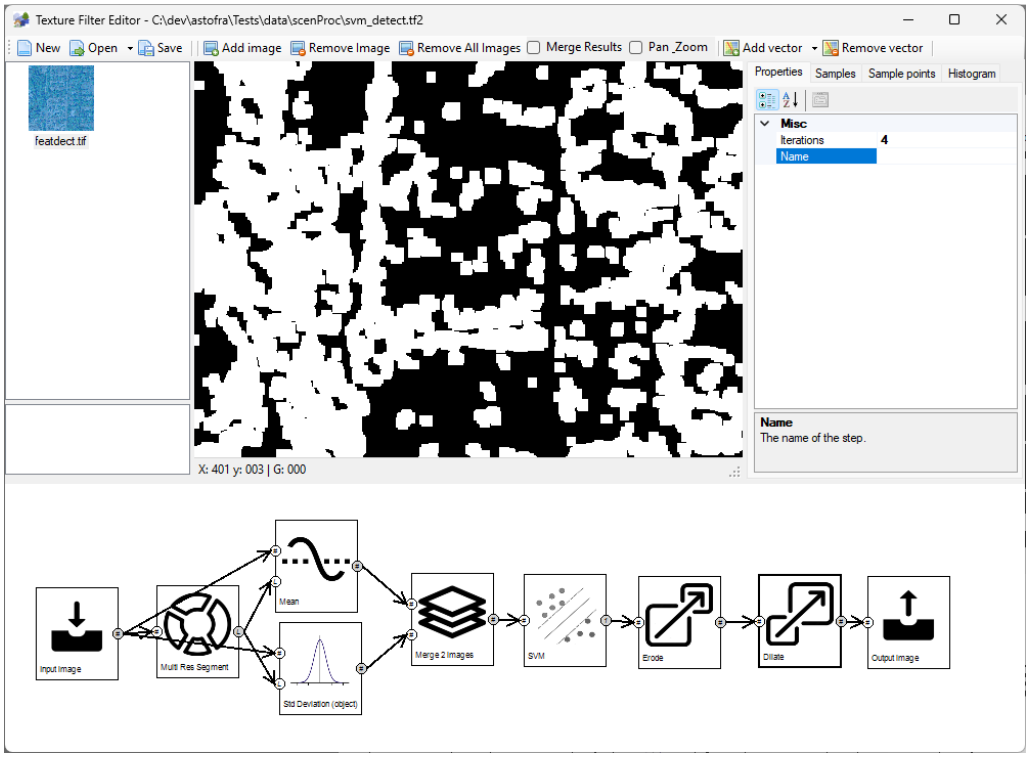



Figure 6.56: Dilating the result to grow the detection back to normal size

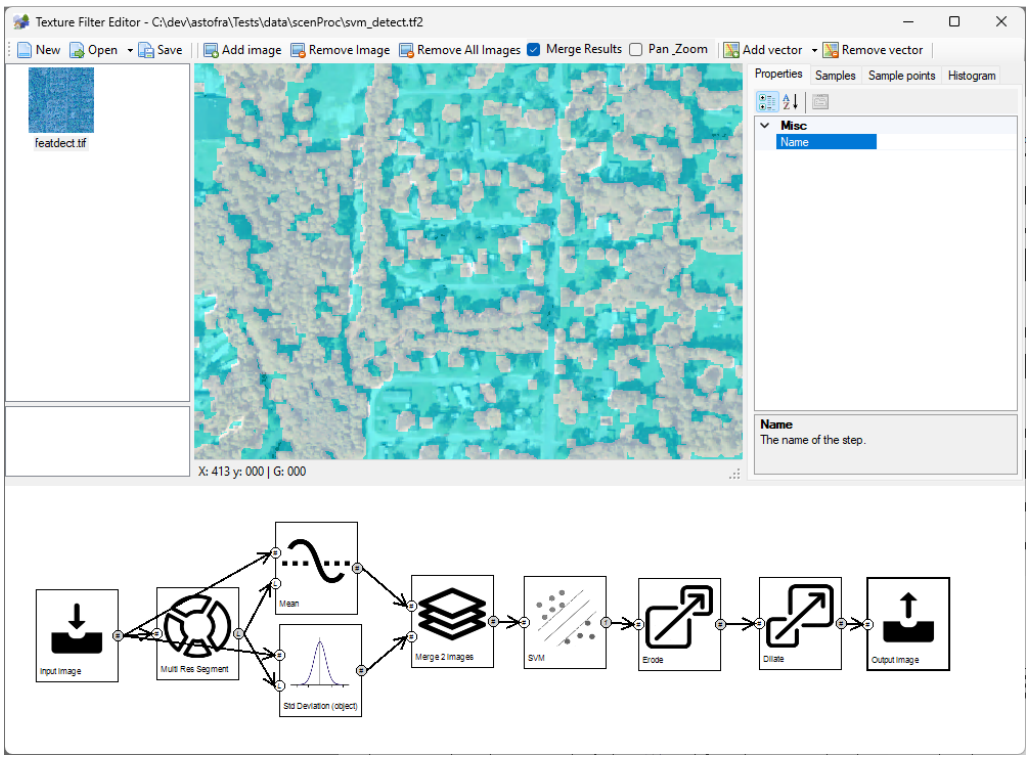



Figure 6.57: The final detection result merged with the input image

### 6.3.6 Photoreal texture filter

This section discusses an example texture filter that can be used to create photoreal scenery with a watermask. This filter contains the following steps:

1. The input image to use, see Figure 6.58.
2. A color correction to make sure that the colors of the image also look good in FS, see Figure 6.59. This image is connected to the Output image step, so that resample will use it for the imagery in the scenery.
3. An empty raster with the same size as the input image, this is used to create the watermask, see Figure 6.60. We start with a black image, since then we can burn polygons in it. If we would start with a white image it's not possible to add feature (all pixels already have the maximum value).
4. Burn the water polygons into the empty raster, see Figure 6.61. We use some blur at the edges to make the transition of the water more smooth. The water polygons used in this sample are not realistic, so they don't match the image well, it's just to illustrate the process.
5. Inverting the watermask, see Figure 6.62. This is done since resample wants the land to be white in the watermask. This image is connected to the Output watermask step, so that resample will use it for the watermask in the scenery.

See section 5.11 for an example configuration that can be used to run this texture filter.

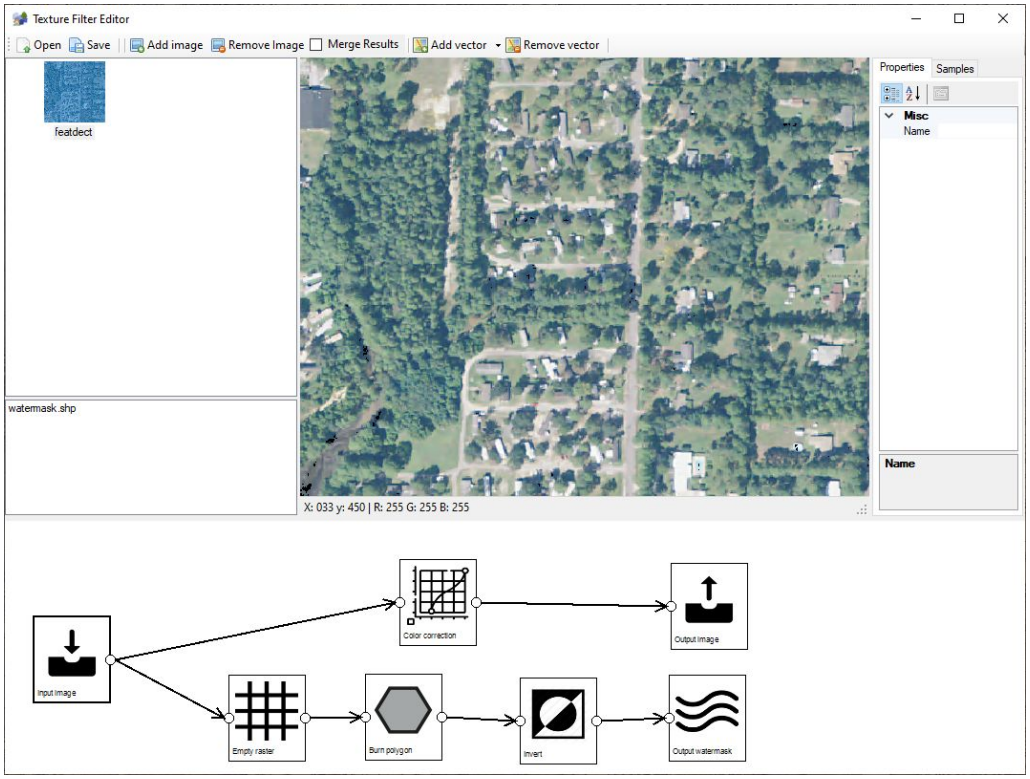



Figure 6.58: The input image for the photoreal

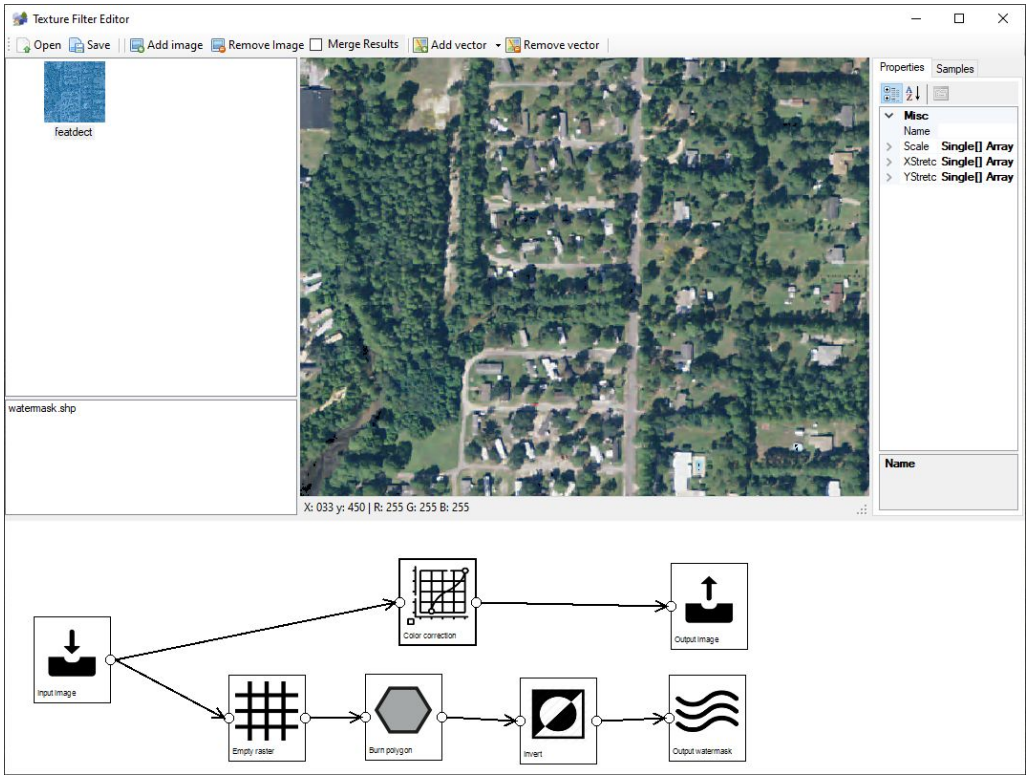



Figure 6.59: Applying color correction to the image

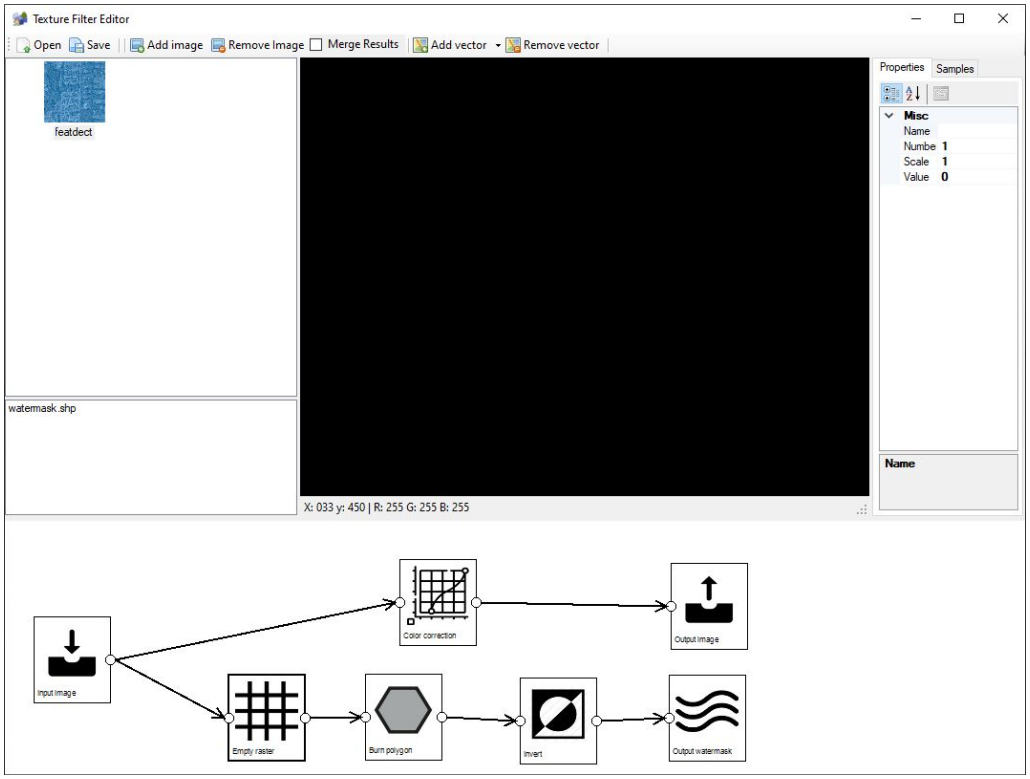



Figure 6.60: An empty raster for the watermask

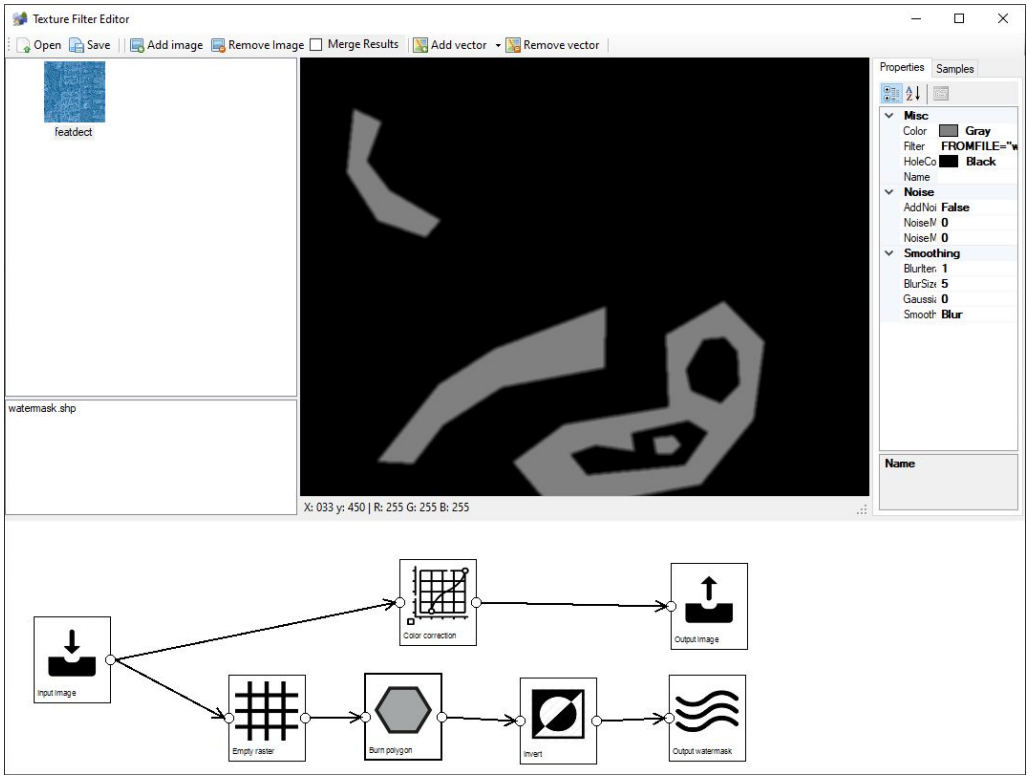



Figure 6.61: Burning the water polygons into the raster

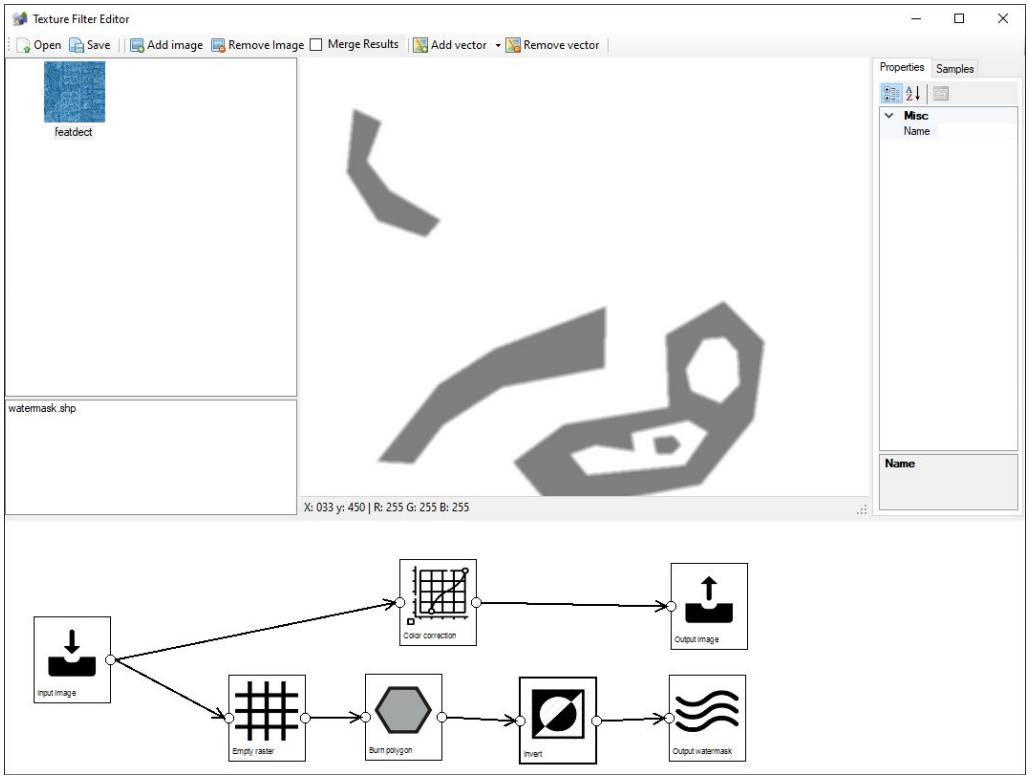



Figure 6.62: Inverting the image to get a correct watermask for resample

# Chapter 7

# 3D Buildings

Autogen buildings are a great way to populate your scenery with buildings, but they also have some drawbacks. The most important one is that autogen buildings can only be rectangular in shape. That means that if your building footprint has a different shape you either have to simplify it to a rectangle or represent it with a few rectangles. For this kind of more complex shapes scenProc also provides another option and that is the feature to create 3D buildings as MDL files. This chapter explains what kind of buildings can be made. The Create3DBuilding step that is used to make the buildings is explained in section 10.4.1.

The basic idea is that scenProc takes a footprint polygon and extrudes it to a specific height to create the walls of your building. Next a roof is added, and as discussed in the section 7.1 different roof shapes are supported. You can also add gables, see section 7.2 and additional features, see section 7.3, to your building. Finally a texture is mapped on the walls and roof of the building to give it the right looks, that is explained in section 7.4.

## 7.1 Roofs

The building generation algorithm does support three basic types of roofs, Figure 7.1 gives a graphical representation of these roofs.

**Flat roofs** are the most simple roof type and they can be applied to any building footprint shape.

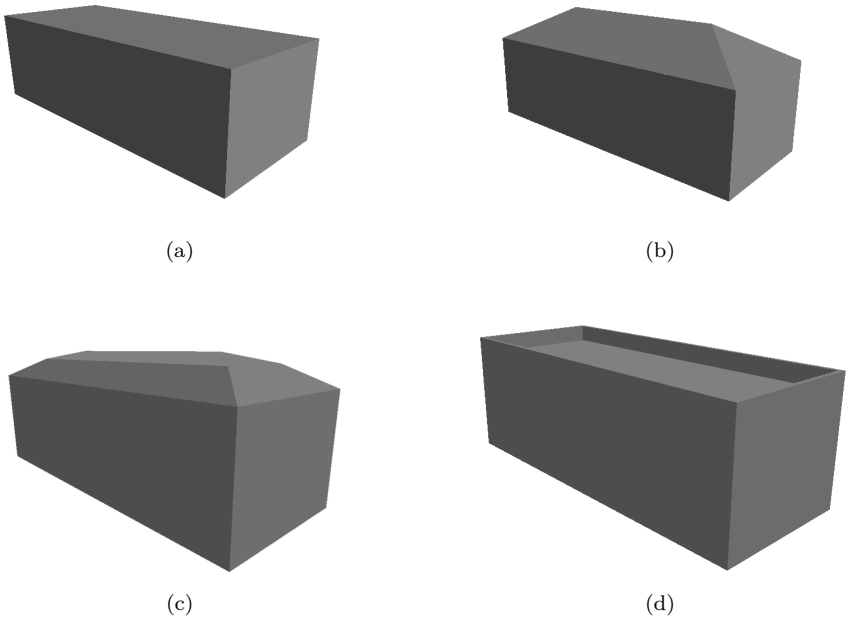
**Hipped roofs** are created by continuously shrinking the building footprint so that the skeleton of the roof is defined. This is called a straight skeleton and the edges of the skeleton are then given a slope to define the shape of the hipped roof. The definition of the slope angle for hipped roofs is given in Figure 7.2.

In general hipped roofs can be applied to any footprint shape. But if the building is very wide, it is best to keep the slope angle of the roof low, else you will get very tall roofs that look unrealistic.

**Mansard roofs** are similar to hipped roofs, but instead of using a constant slope for the edges of the skeleton, for a mansard roof two slope angles are used. Once the distance along the edge is over a certain threshold value a secondary slope angle is used. This gives roofs with two different slopes. The definition of the slope angles is given in Figure 7.2.

In general mansard roofs can be applied to any footprint shape. But if the building is very wide, it is best to keep the slope of the roof low, else you will get very tall roofs that look unrealistic.

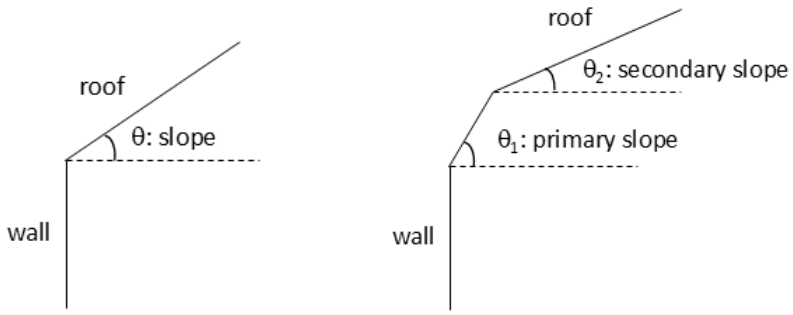
**Parapets roofs** are roofs that consist of a wall elements and a sunken roof. The definition of the parameters that are used to define a parapets roof are given in Figure 7.3.



(a) (b) (c) (d)

Figure 7.1: The basic roof types: (a) flat, (b) hipped, (c) mansard and (d) parapets

Be aware that parapets roofs use more triangles than the other roof types, so when applying this type of roof to many buildings it might influence the performance of the scenery. As a comparison for a simple rectangular building a flat roof requires 2 triangles, a hipped roof 6 triangles, a mansard roof 14 triangles and a parapets roof 26 triangles.




(a) (b)

Figure 7.2: Definition of the slope angle for (a) hipped and (b) mansard roofs

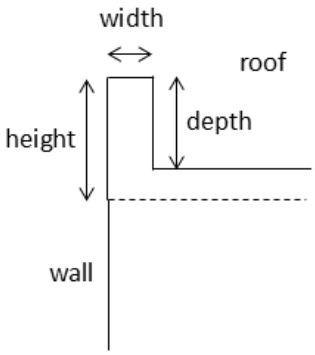
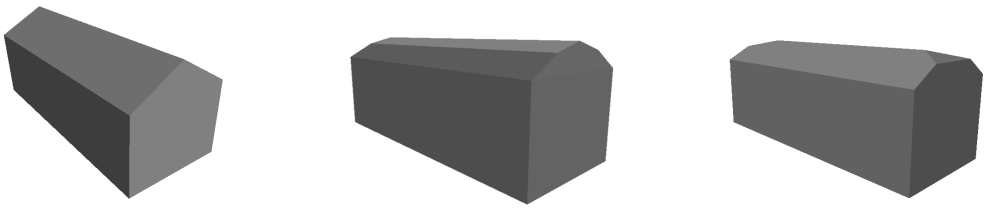



Figure 7.3: Definition of the parameters for parapets roofs

## 7.2 Gables

More various of the three basic roof types, as described in section 7.1, can be added by using gabled roofs. Gabled roof various from the previous roofs in the sense that certain edges of the skeleton will be turned into vertical walls, instead of slope roof segments. Figure 7.4 gives a graphical representation of different gabled roofs. The semi-gabled mansard roof is created by creating a wall from the segment of the roof that has the primary slope, while keeping the sloped roof for the segment with the secondary slope angle.



(a) (b) (c)

Figure 7.4: Gabled roofs: (a) gabled hipped, (b) gabled mansard (c) semi-gabled mansard

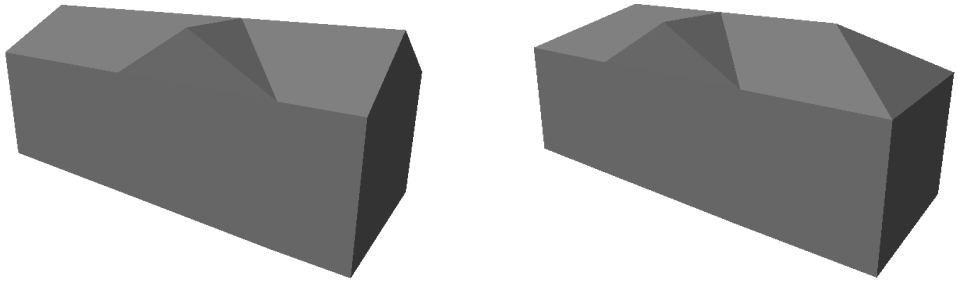
The building algorithm can automatically assign which edges should become gabled. This is done by evaluating the angles between the edges in the footprint and will assign gables to the shorter sides of a rectangle. Since this algorithm works best on footprints that are not too complex, it is advised to use gables only on footprints with not too many vertices and that contain only perpendicular angles.

When using the interactive mode to define a building footprint it is also possible to manually assign which edges should become gabled.

Another way to add gables to buildings is by applying a footprint modification. This will add an additional gable on top of the one that is assigned automatically for rectangular and L-shaped footprints.

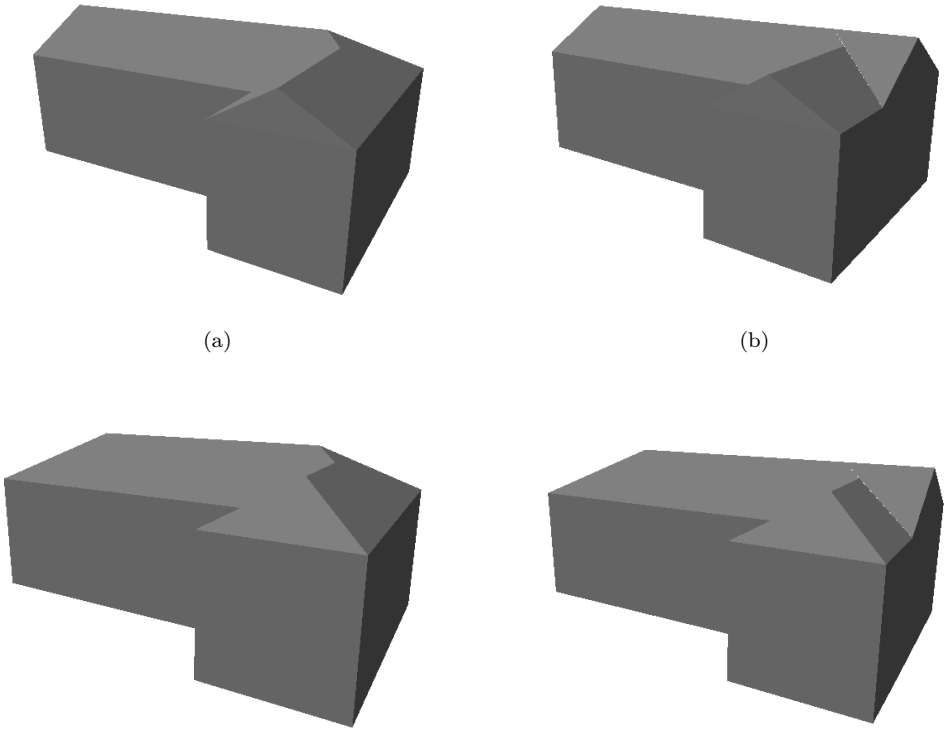
The footprint modification for rectangles does modify one of the longest edges of the footprint so that an additional gable is added. Figure 7.5 shows how this looks for a gabled and a hipped roof. You need to make sure that the polygon you run the modification on is rectangular, which means it has 4 vertices and 4 perpendicular angles.

The footprint modification for L-shaped does modify one of the edges of the footprint so that an additional gable is added. This additional edge is placed opposing to one of the legs of the L-shape. Figure 7.6 shows how this looks for a gabled and a hipped roof. You need to make sure that the polygon you run the modification on is L-shaped, which means it has 6 vertices and 6 perpendicular angles.



(a) (b)

Figure 7.5: Rectangular buildings with an extra gable (a) gabled roof (b) hipped roof



(a) (b)

(c) (d)

Figure 7.6: (a) hipped L-shaped building (b) hipped L-schaped building with extra gable (c) gabled L-shaped building (d) gabled L-shaped building with extra gable

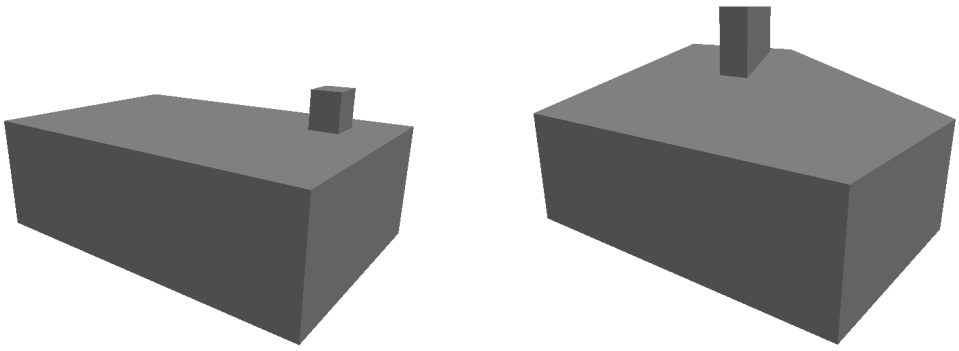
## 7.3 Features

The building algorithm can also generate additional features for the building. It is possible to add chmineys and dormers. These are explained in the sections below.

For each feature you can set a position, a direction and a size. In the interactive editing mode you can set these yourself, if you use the automatic placement logic for the features they are calculated based on the shape of the footprint.

### Chimney

A chimney can be added on any roof type, see Figure 7.7 for a graphical representation. If the roof texture configuration file that you use contains a group called ___CHIMNEY this texture is automatically selected for the chimney. It is adviced to have a relatively dark roof texure in that group, as that will look best on a chimney.



(a) (b)

Figure 7.7: Buildings with a chimney (a) flat roof (b) hipped roof

### Dormer

A dormer can only be added to a building that does not have a flat roof. Dormers can have different roof types, see Figure 7.8 for a graphical representation. The possible roof types are:

- Flat roof
- Sloped roof
- Hipped roof
- Gabled hipped roof

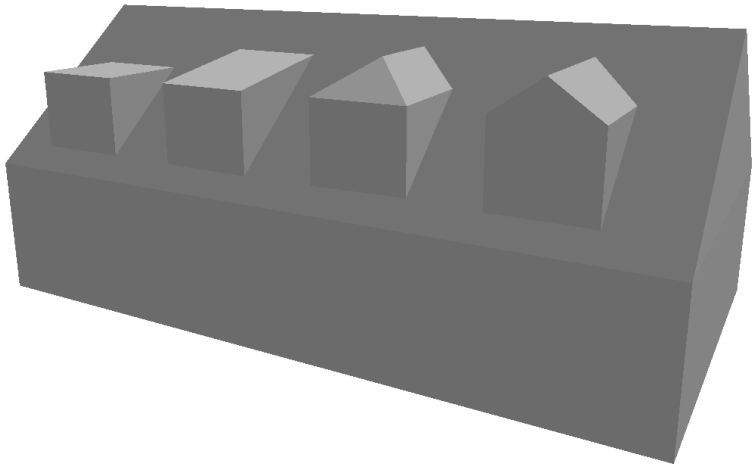


Figure 7.8: Building with dormers. From left to right (1) flat roof, (2) sloped roof, (3) hipped roof and (4) gabled hipped roof

## 7.4 Building Texture Configuration Editor

Until now only the geometry of the building has been discussed, but for the final appearance of the building in your scenery also the texture that is applied to the building is very important, see Figure 7.9 for an example of a textured building made by the building algorithm.

For autogen buildings Flight Simulator has a fixed layout of the texture that is used and therefore any custom textures you make need to follow that exact layout. The building generation algorithm does not work like that, you can use any texture layout that you want. But to allow the algorithm to map the texture realistically on the building you have to provide additional information to be able to define the optimal mapping. For example which part of the texture contains the walls and the roofs. And where the windows and doors are located, so that the algorithm can ensure that they are not split over different wall segments. That is defined in the Building Texture Configuration file. When creating a building you can specify a configuration file for the walls and another one for the roofs.

When the algorithm makes a gabled roof it tries to prevent that windows are clipped by the slope of the gable as well. Two approaches can be used for this:

1. Try to find a window in the wall texture segment that has enough empty space around it to fit on the gable wall. See Figure 7.10 for an illustration of this strategy. The yellow rectangle should contain a window, while the other parts of the gable triangle should not contain a window.
2. Try to find a area in the wall texture segment without any windows and use that on the gable wall.

In the Building Texture Configuration file you can specify what is the preferred approach of these two is. If the preferred approach fails, the algorithm will automatically try the other as fall-back.

When you are placing multiple buildings in your scenery, you also don't want that they all look the same. Therefore you can specify multiple wall and roof textures in your Building Texture Configuration file and the algorithm can randomly pick one for each building that is created. You can also create different groups within the building texture configuration file and then you can instruct the algorithm to pick a texture for a specific group. You can for example create groups for different colors of roofs (dark grey, light gray, red, etc).



Figure 7.9: Example of a building with a texture applied



Figure 7.10: Mapping strategy for gable with window

All of the topics mentioned above are configured in the Building Texture Configuration file and you can use the Building Texture Configuration Editor, see Figure 7.11, to make and edit this configuration file. The texture shown in Figure 7.12 is used as an example in this section to explain how that is done. You can see that this building texture configuration file contains 3 different wall segments.

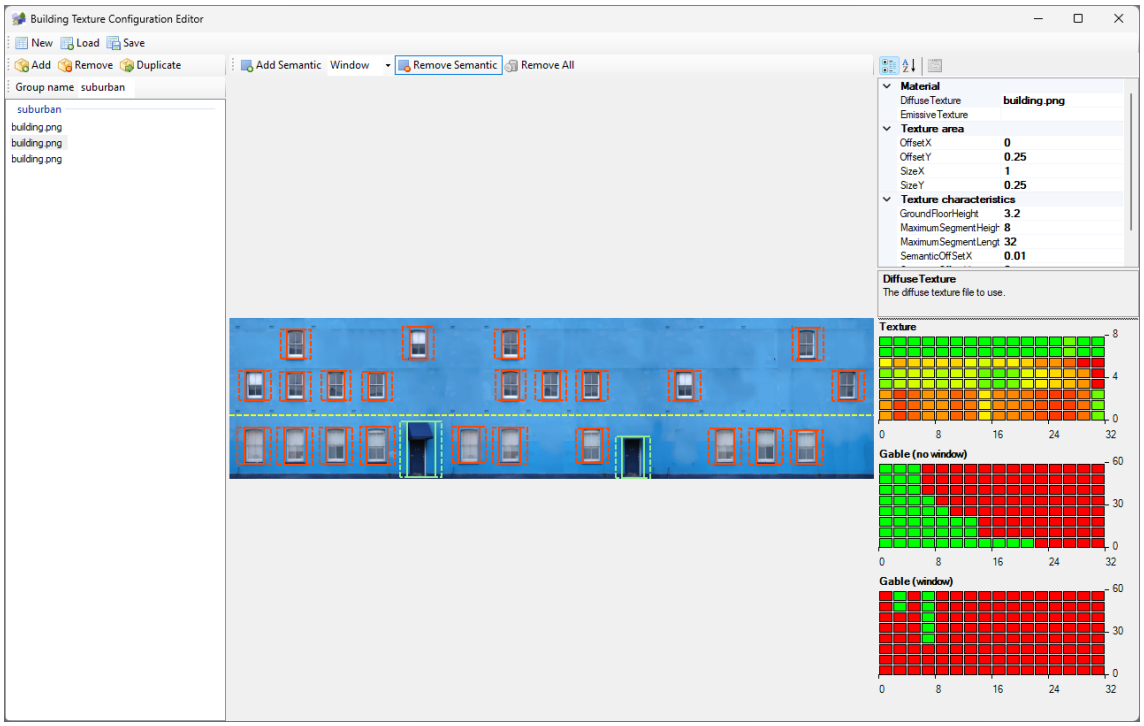



Figure 7.11: Building Texture Configuration Editor

In the Building Texture Configuration Editor you see a list of texture segments on the left side. This list contains either wall or roof segments, depending on whether this configuration file is used for walls or roofs. On the right side you see the properties of the selected texture segment and graphical quality indicators of your texture. The part in the middle is used to show the texture segment you are configuring. Each of these areas of the editor will be described below.

### Main toolbar

The toolbar at the top of the Building Texture Configuration Editor gives you access to the following functions:

- **New** creates a new empty configuration file.
- **Load** reads a saved BT2 configuration file from disk.
- **Save** writes the configuration file to disk as a BT2 file.

### Texture segment list

The list of texture segments, see Figure 7.13, lists all the texture segments of this building texture configuration file. If you click on a texture segment from the list, this segment will be shown in the central picture and the properties are shown in the property dialog. The texture list has the following buttons in the toolbar:

- **Add** will add a new texture segment to the list. After clicking Add you will get a file selection dialog to select which texture file should be used for this segment.



Figure 7.12: Example building texture

- **Remove** will remove the currently selected texture segment from the Building Texture Configuration file.

- **Duplicate** will make a copy of the currently selected texture segment and add it to the list. This can be useful if you want to add another segment that only differs slightly from an already existing one.

- **Group name** specifies the group that the currently selected roof texture segment belongs to. By changing this name you can assign it to a different group. When making the building you can specify that the algorithm should only select wall or roof texture segments from a specific group.

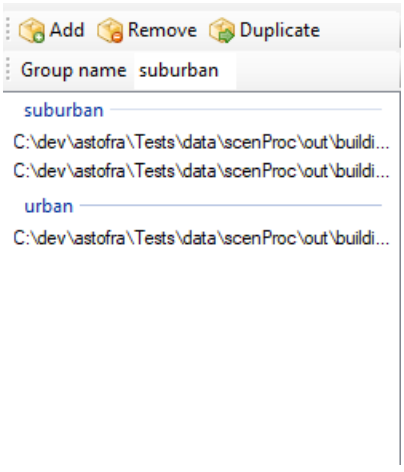



Figure 7.13: Texture segment list

### Property list

The property list shows all the properties of the selected texture segment. The following properties are available:

- **Allow horizontal flip** specifies if the texture can be flipped horizontally when applying it to the building. Allowing horizontal flipping gives more variation in the final look of the buildings, for example a door is on the left of a window on one segment and on the right in another due to the flipping. But for textures with text or signs on them the flipping can produce undesired results, so in that case this attribute should be set to false.
- **Diffuse texture** specifies the diffuse texture that is used for this texture segment (When using a PBR material this texture is used as albedo texture).
- **Emissive texture** specifies the emissive texture that is used for this texture segment. When no name is specified the building will not have an emissive texture.
- **Ground floor height** specifies the height of the ground floor in meters for wall texture segments. When the wall texture segment has to be repeated vertically the ground floor is excluded, to prevent that you get doors floating in the air, see Figure 7.14. In the texture preview the ground floor height is shown as a dashed yellow line. When the entire texture can be repeated vertically you can enter a ground floor height of zero.



Figure 7.14: Example of building where the ground floor is not repeated vertically

- **Maximum segment length** specifies what the length of the texture segment is in meters. This information is used to scale the texture correctly on the building. When the building is longer than this length the polygons are split into multiple polygons to make sure that the texture can be applied correctly.
- **Metallic** specifies the metalic level of the material. This is only used for PBR materials.
- **MetallicHasOcclusion** specifies if the metcallic map has ambient occlusion in it's green channel. This is only used for PBR materials.
- **MetallicHasReflectance** specifies if the metallic map has specular reflectance in it's blue channel. This is only used for PBR materials.
- **Metal/smooth/AO texture** specifies the metallic/smoothness/AO texture that is used for this texture segment. When no name is specified the building will not have a metallic/smoothness/AO texture. This texture is only used for PBR materials.
- **Normal texture** specifies the normal texture that is used for this texture segment. When no name is specified the building will not have an emissive texture. This texture is only used for PBR materials.

- **Offset X** and **Offset Y** specify the offset of this texture segment in the texture sheet as a percentage of the size of the texture sheet. This way you can use different segments from the same texture sheet on your building. Table 7.1 shows the offset values for the sample texture sheet shown in Figure 7.12.
- **Semantic offset X** and **Semantic offset Y** specify if an additional offset should be added in the X and Y direction around semantics (windows, doors). Such an offset can be used to prevent that a door or window is mapped directly at the corner of two walls. The offset is specified relative to the size of the texture sheet. For example a value of 0.01 means that the offset is 1% of the size of the texture. In the texture preview the window offset is shown as a red dashed rectangle.
- **Size X** and **Size Y** specify the size of this texture segment as a percentage of the size of the texture sheet. This way you can use different segments from the same texture sheet on your building. Table 7.1 shows the size values for the sample texture sheet shown in Figure 7.12.

| Texture segment | Offset X | Offset Y | Size X | Size Y |
|---|---|---|---|---|
| wall 1 | 0.00 | 0.00 | 1.00 | 0.25 |
| wall 2 | 0.00 | 0.25 | 1.00 | 0.25 |
| wall 3 | 0.00 | 0.50 | 1.00 | 0.25 |
| roof 1 | 0.00 | 0.75 | 0.50 | 0.25 |
| roof 2 | 0.50 | 0.75 | 0.50 | 0.25 |

Table 7.1: Offset and size values for the sample texture

- **Smoothness** specifies the smoothness of the material. This is only used for PBR materials.
- **Use PBR material** specifies if the material is a PBR material or a standard FSX material.
- **Window gable percentage** specifies which percentage of the gables should get a window. A value of 1.0 means that the algorithm tries to place a window on all gables, while a value of 0.0 means that none of the gables should get a window.

#### Texture preview & Semantic editing

The texture preview shows you the currently selected texture segment, see Figure 7.15. This way you can easily verify that you have entered the correct offset and size values for your segment. Besides that you can also edit the semantic definitions in the preview. These semantics define where windows, door and other elements are that the texture mapping should take into account, as you don't want such semantic features to be split on your building. To edit the semantics you use the buttons shown in the toolbar:

- **Add semantic** enables the mode where you can add new semantics. By clicking and dragging with the mouse you can draw new semantics on top of the texture. The semantic you are currently adding is shown as a green rectangle. Clicking again on the button will disable the add mode again. In the drop down list next to the button you can select the type of semantic you want to add, the following types are available:
  - Window, shown in red on the texture segment.
  - Door, shown in light green on the texture segment.
  - Other, shown in dark pink on the texture segment. Other is an element on the texture that is not a door or a window, but still should not be split when mapping the texture. For example a drainage or a mailbox attached on the wall.
- **Remove semantic** enables the mode where you can remove semantics. When this mode is enable a click inside a semantic will remove that semantic from the Building Texture Configuration. Clicking again on the button will disable to remove mode.
- **Remove all** removes all semantics from the Building Texture Configuration so that you can start with a clean sheet.




Figure 7.15: Semantic editing

### Texture quality indicators

Depending on the layout of your texture segment it might be easier or harder for the algorithm to map it on a building. For example if you have a lot of windows in the segment, it might be hard to map it on a wall without putting any window on the edge. How hard it is to map the texture also depends on the length of the wall segment that is being mapped of course. The texture quality indicators, see Figure 7.16 are a visual indication of how suitable your texture segment is to map on walls or different lengths. This should help you in designing a texture segment that is well balances, so that it can be applied on buildings with different shapes and sizes. Three different indicators are available:

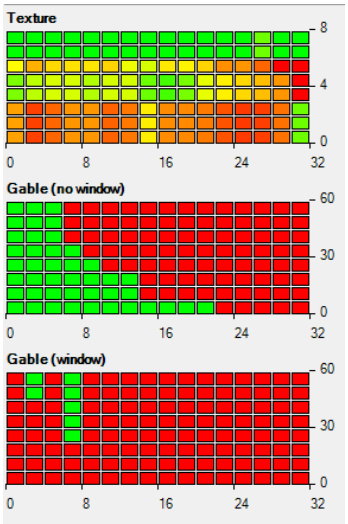



Figure 7.16: Texture quality indicators

- The **Texture** indicator shows how easy it is to map the texture on a wall or roof of a given length. The horizontal axis indicates the length of the wall, between 0 and the maximum segment length. The vertical axis indicates the vertical areas in your segment. The quality indicator shown in Figure 7.16 is for the texture segment shown in 7.15. As the ground floor of that texture segment has more windows, you can see that vertically the bottom of the texture segment is harder to map (more orange or red squares), while the top part is easier to map. Horizontally you can see that a wall of 4 meters in length is harder to map than a

wall of 16 meters.

- The **Gable (no window)** indicator shows you how suitable the texture segment is to map on a gable without any windows. The horizontal axis once again shows the length of the gable, between 0 and the maximum segment length. The vertical axis shows the slope of the gable in degrees. The quality indicator shown in Figure 7.16 is for the texture segment shown in 7.15. You can see that for gables longer than 16 meter it is very hard to map the texture, this is because these require a big area of the texture sheet without windows and at the higher slope angles the gable height might be higher than the height of the wall texture segment. But for slopes in the range of 30 to 45 degrees you can see that gables up to 8 meter in length can be mapped correctly without any windows on them.
- The **Gable (window)** indicator shows you how suitable the texture segment is to map on a gable with a window. The horizontal axis once again shows the length of the gable, between 0 and the maximum segment length. The vertical axis shows the slope of the gable in degrees. The quality indicator shown in Figure 7.16 is for the texture segment shown in 7.15. You can see that there are only a few gables where a window can be mapped correctly. This is partly due to the design of the texture segment, it only has a few windows in it that have enough empty space around them to fit on a gable. For gables with a small slope angle it is also hard to map a window, since these will not be very tall.

# Chapter 8

# Merging autogen files

You will often be using scenProc to generate autogen files for large areas. This also means you will be processing a lot of data. Often so much data that it will fill all your memory. So in that case you will probably split your data in sections and process them each in turn. This however leads to the problem on how to merge all the generated autogen files back into one scenery project. This chapter will discuss some solutions and tools to help with such problems.

## 8.1 Split data along autogen tiles

One way to prevent you have to merge your autogen files afterward is to make sure your input data is already divided exactly along the borders used by autogen tiles. To be able to do this you can use two approaches:

1. Preprocess your GIS data so that it aligns with the FS terrain tiles. You can do this in a GIS tool, but in general I don't prefer this approach since it requires extra preprocessing of your data and that can cost a lot of time.
2. If you can setup a geo data server, like PostgreSQL with PostGIS or GeoServer, you can load your data in that server. scenProc can retrieve data from such servers and if you use the terrain tile bounding area when retrieving the data you will only get the data you need for that run.

This approach will not be discussed further here, as it does not require merging of autogen files.

## 8.2 Using ImportAGN step

Another approach is to load already created autogen objects into scenProc while you are processing the overlapping areas. This way they will be nicely merged. I have used this approach in one of my projects. Let me describe the workflow.

The data I was using in that project was organized in different files. Each file contained the data for a specific region, but those regions were not aligned with the autogen grid of course. Actually each file contained the data for a specific paper map generated from the data. So each file contained the data for one sheet of the atlas of the entire country.

For this project I made one scenProc configuration file that I would use to process all data files in turn. So I was using the batch mode, as explained in chapter 9. Since one AGN file could be filled with data from different files, I used the ImportAGN step to import the autogen objects that had been generated from the previous files already.

With the EXISTINGCELLSONLY option of the ImportAGN step you can tell scenProc to only load the autogen files that overlap with the data you are currently processing. So imagine you are

processing an entire country, with this option the autogen objects in the south of the country will not be loaded into memory when you are processing some data in the north.

Below you find an example configuration file that illustrates the workflow described above. This configuration file would be used in the batch mode on each file with data in turn.

```
# @0@= myfile.gml
#
ImportOGR|@0@|*|*|NOREPROJ
#
SplitGrid|AGN|*
#
ImportAGN|texture|EXISTINGCELLSONLY
#
CreateAGNPolyVeg|type="forest"|{c9dc45ae-f240-42a9-a137-b7617452a308}
#
ExportAGN|FSX|texture
```

## 8.3 Using Autogen Merge Tool

The approach described in the previous section works fine if you want all the autogen files to end up in one texture folder because they belong to one scenery. But that approach doesn't work so well if you have two separate sceneries that border with each other. In that case you can use the Autogen Merge Tool to merge the overlapping AGN files that exist in both sceneries.

The basic idea of the Autogen Merge Tool is that you select one input folder that contains the AGN files of your project. You also select one or more other autogen folders that contain the AGN files of the bordering projects. For each autogen file in your project, the folders of the bordering projects will be checked. If they also contain autogen for that specific tile, these objects will be merged into the AGN file of your project. In the end all AGN files are written to your selected output folder.

So imagine you have three countries that border with each other and you have a scenery project for each of them. With scenProc you would then generate AGN files for each country separately, for example you could store them in the following folders:

- countryA
- countryB
- countryC

Then in your texture folders of your scenery you want to have the total merged autogen files. So for the scenery of country A you would select the countryA folder as input, while selecting countryB and countryC as bordering folders. For country B you would select the countryB folder as input, while selecting countryA and countryC as bordering folders. And I asume you can guess now what you will select for country C.

The Autogen Merge Tool can be found in the **Tools** menu of scenProc. When you click open the tool from the menu a form as shown in Figure 8.1 will appear.

In the top box you select the input folder that contains the AGN files of the project. Then in the list below you can add one or multiple folders that contain autogen for bordering areas. Use the button with the plus sign to add a new folder or use the button with the minus sign to remove a folder for the list. In the bottom box you select to which folder the merged autogen files should be written. It is advised to not use the same folder as input and output.

Finally the checkbox allows you to determine if the merge folder should only contain AGN files that exist in the input folder you selected or if it should contain all AGN files found in the input and other folders. If you are merging from two countries in the border area, you probably only

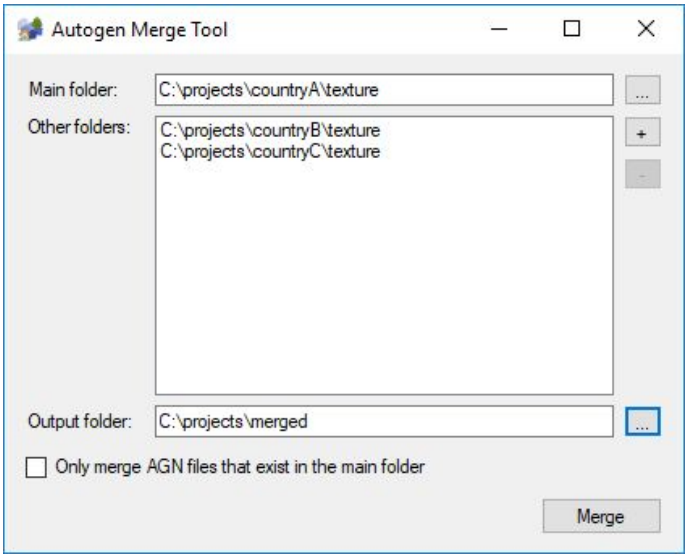



Figure 8.1: Autogen Merge Tool

want to merge the files that are within the country (checkbox checked). However if you are merging vegetation and building folders, you probably want all files (checkbox unchecked).

Then you press the Merge button and the merge operation will start. If you have a lot of autogen files it might take a while, a progress bar will show the progress of the merge operation. Once the progress bar has disappeared the merge action is finished and you will find the combined autogen files in the folder you had selected.

# Chapter 9

# Batch mode

If you want to process big amounts of data or a big area it is not always convenient to run scenProc from the graphical user interface. Therefore a batch mode is also available. To use the batch mode you need to start scenProc from the command prompt or use the scenProc Batch Runner.

## 9.1 Command line arguments

When running scenProc from the command line you can use the following command line arguments to pass additional information to scenProc:

- The name of the scenProc configuration file you want to load.
- `/run` to specify scenProc should not only load the configuration file, but also directly start processing it.
- `/log` to specify that the event log should be written to file. This argument is followed by the name of the log file to use.
- The values that should be used for batch variables.

The example below is the command to start scenProc with the myConfig.spc configuration file loaded and directly start processing the file:

```
scenProc.exe myConfig.spc /run
```

The example below does also save the event log to a file called log.txt:

```
scenProc.exe myConfig.spc /run /log log.txt
```

## 9.2 Batch variables

The way described above allows you to run configurations in batch mode, but you would still have to prepare the different configurations manually. That's not saving much time compare to running from the graphical user interface. But the real power of the batch mode is when you are also using batch variables. That way you can run the same script with slightly different content every time. For example by loading a different file with geographical data or by processing another area.

To do this you need to use batch variables in your script. When running the script they will then be filled with the values you provided. The example script below shows how you can use a batch variable for the filename that should be loaded by the ImportOGR step:

```
# @0@=area1.shp
#
```

```
ImportOGR|@0@|*|*|NOREPROJ+
```

And now you can pass the filename of the file you want to load in batch mode. So for example you can run these two commands to process two different files with the same script:

```
scenProc.exe myConfig.spc /run /log log.txt area1.shp
scenProc.exe myConfig.spc /run /log log.txt area2.shp
```

As you can see in the sample configuration code I have added a comment with `@0@=area1.shp`. When running from the graphical user interface this comment will be used to give a value to the batch variable. That way you can still run and test the script from the graphical user interface as well. And once you are happy with how it works you can batch run it on all your data or on your total area.

In this example we have substituted the filename, but you can use batch variables for any part of the configuration. It's just that at runtime the values are assigned, you could use it for a filename, coordinates or even a filter or a GUID. It's up to you how you want to use it. You can also use more than one batch variables in the same script, the only requirement is that they start numbering from 0 and that the numbers increase sequentially.

## 9.3 scenProc Batch Runner

To make it easiest to run a configuration file in batch mode, you can also use the scenProc Batch Runner tool. You can start scenProc in the Batch Runner mode by using the scenProcBatchRunner.bat file in your installation folder or starting scenProc with the following command:

```
scenProc.exe batchrunner
```

In this tool you can easily specify which script you want to run and what the values of the batch variables should be. Figure 9.1 shows the interface of the scenProc Batch Runner. Before it is explained how to use this tool, first some concepts have to be defined.

**Job** A job is a combination of a scenProc configuration file, plus the values of the batch variables that are used when executing the configuration file. A job can have the following statuses:

- Queued when it is in the queue of jobs to be processed.
- Running when it is currently being processed by a worker.
- Succeeded when it was completed without errors by a worker.
- Failed when it was completed with errors by a worker.
- Requeued when it failed before, but was automatically requeued into the list of jobs that still need to be processed.

**Worker** A worker is a process that can execute a job. A worker can have the following statuses:

- Idle when it is doing nothing and waiting for a job to process.
- Running when it is processing a job.

The scenProc Batch Runner GUI has three lists that provide you information about the status of the workers, the jobs still to be processed and the jobs that have finished processing. Each of these lists of the GUI and the corresponding buttons are discussed in the sections below.

### 9.3.1 Worker status

The worker status lists shows all the workers that are currently available and what their status is. The status information includes the status of the worker, the time it started processing the current job and which job is being processed.

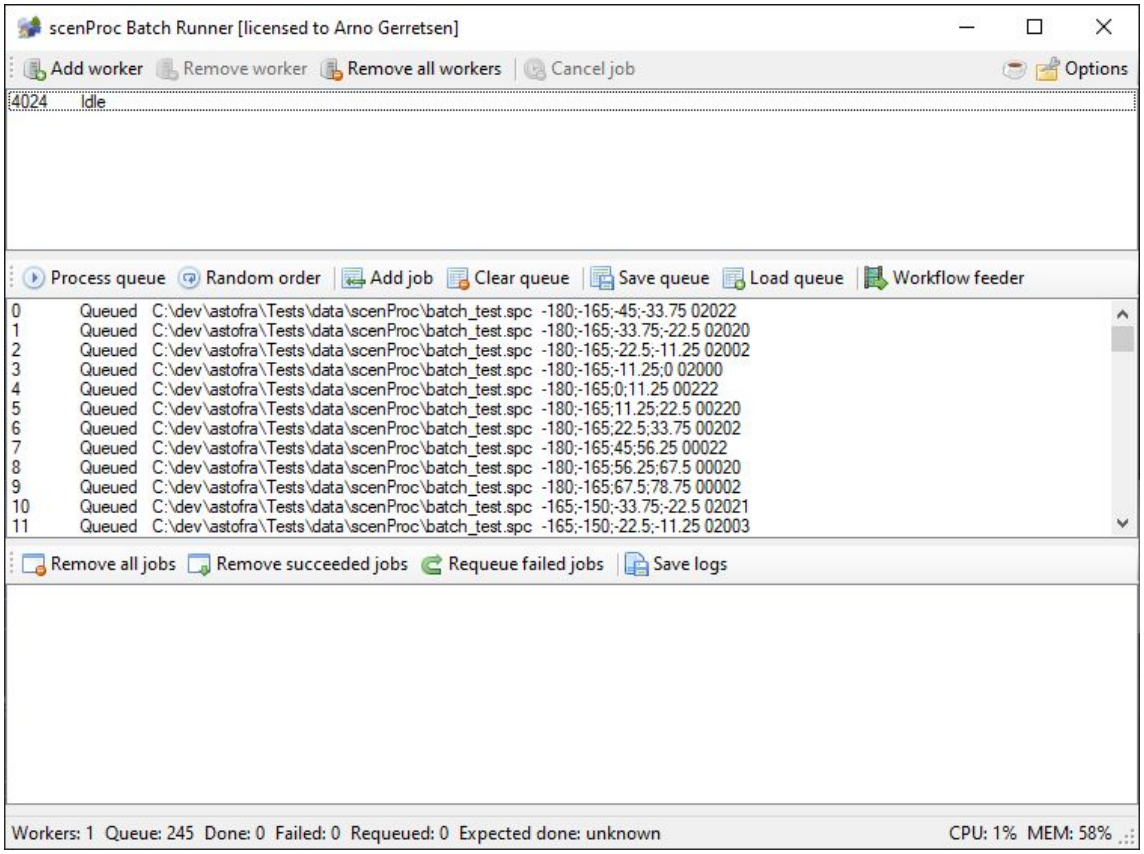



Figure 9.1: scenProc Batch Runner

The following buttons are available in the worker status list:

**Add worker** Add an additional worker to the scenProc Batch Runner. With multiple workers you can process the jobs more quickly, but more computer resources are also used.

**Remove worker** Remove the currently selected worker from the list of workers. If this worker is still running a job, that job will be canceled and get a failed status.

**Remove all workers** Remove all workers from the list of workers. If a worker is still running a job, that job will be canceled and get a failed status.

**Cancel job** Cancel the job currently running on the selected worker from the list of workers. This job will get a failed status.

### 9.3.2 Jobs to process

The jobs to process lists shows all jobs that still need to be processed and also gives you the controls to add new jobs and determine how the list is processed.

The following buttons are available in the jobs to process list:

**Process queue** Process the list of jobs, this means that when a worker is idle the next job from the list will be started.

**Random order** When this button is pressed the jobs in the list are processed in random order.

**Add job** Add a new job to the list, this is done using the form shown in Figure 9.2. For the jobs you need to select the scenProc configuration file that should be used. And you need to select one of the three options for the batch variables:

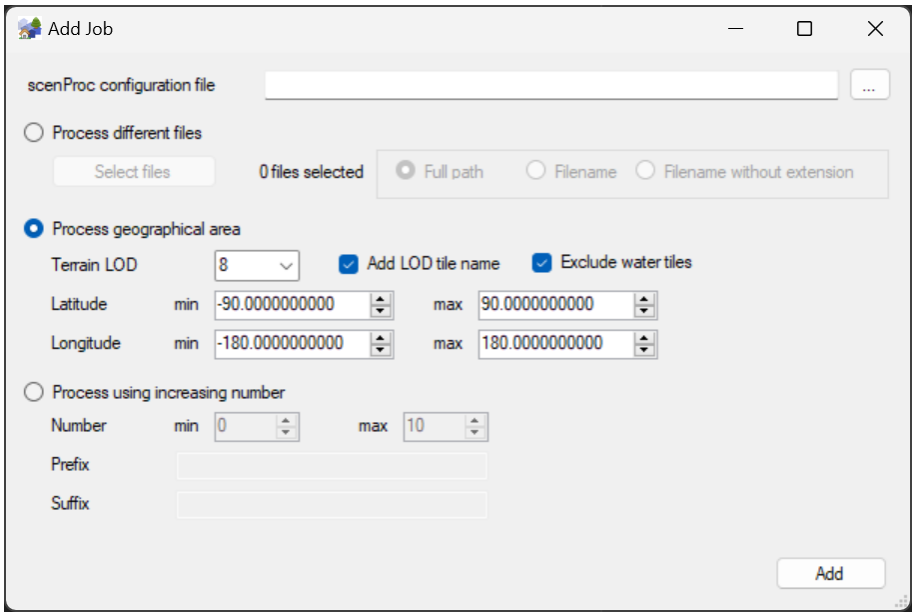



Figure 9.2: Add job form

- **Process different files** if you want to select various files that should each be passed as batch variable to the same script. This can for example be used if you have different source files with the geographical information that should all be processed by the same script. With the radio buttons you can determine if the full path of the files, only their file names or the file names without extension are added as user variable.
- **Process geographical area** if you want to run the same script, but using a different bounding box for a geographical area each time. You can specify the latitude and longitude bounds of the area you want to process and which FS tile size should be used.

The tool will then automatically create a job for each tile of the given size in the area you have specified. The **Add LOD tile name** option can be used to also add a batch variable with the name of the LOD tile according to the SDK. The **Exclude water tiles** option can be used to skip tiles that only contain water.

- **Process using increasing number** if you want to have a batch variable that increases a number each time. You can also specify a prefix and suffix to be used. This can for example be used if you would like to pass value1, value2, value3 and value4 to the script in different runs.

**Clear queue** Remove all jobs that are currently in the list.

**Save queue** Save the current list of jobs to file, so that you can reload the same list of jobs later on.

**Load queue** Load a list of jobs from a file. These jobs are added to the jobs already in the list.

**Workflow feeder** The workflow feeder is an additional tool to feed your list of jobs based on a workflow. It is described in section 9.4.

### 9.3.3 Finished jobs

The finished jobs lists shows all jobs that have finished already. It shows the status of these jobs and if you double click on a job the event log of that jobs will be shown in a seperate form.

The following buttons are available in the jobs to process list:

**Remove all jobs** Remove all jobs from the finished jobs list.

**Remove succeeded** Remove all succeeded jobs from the finished jobs list.

**Requeue failed jobs** Add all failed jobs back to the list of jobs to be processed.

**Save logs** This saves the log files of the jobs in the list to file.

### 9.3.4 Options

With the Options button at the top of the scenProc Batch Runner forum you can bring up the options screen as shown in Figure 9.3.

The following options can be set:

**Publish port** The port used by the batch runner and workers to publish messages to.

**Subscribe port** The port used by the batch runner and workers to subscribe to.

**Automatically requeue failed jobs** When checked failed jobs are automatically requeued. This is done for a maximum number of fails as specified. So when you specify a number of 5 there and the jobs fails a sixth time, it will not be requeued anymore.

**Restart worker after number of jobs** When checked the worker will restart itself after it has processed the specified number of jobs.

**Automatically save job log files to folder** When this option is enabled, the log files of the jobs are automatically saved to the specified folder.

**Sent email progress notifications** When this option is enabled the batch runner will send you email progress notification messages so that you can monitor the progress of the jobs. You can specify the interval in minutes between the notification messages. The following settings must be provided:

- The email address to send the notification to.
- The SMTP server used to send the email.
- The username of the SMTP server.

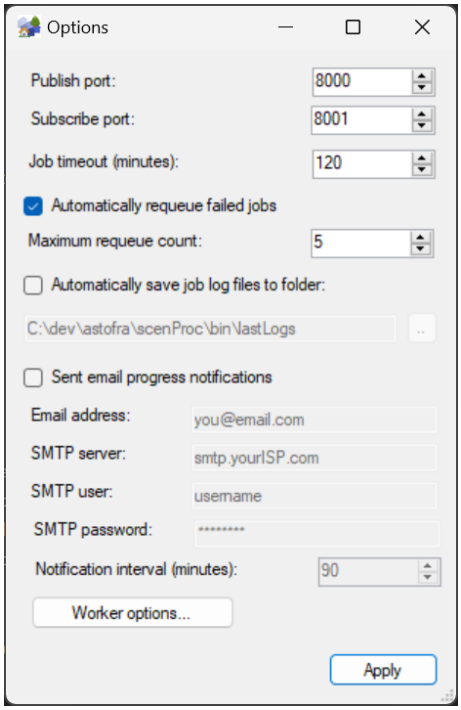



Figure 9.3: scenProc Batch Runner options form

- The password of the SMTP server.

**Worker options...** Opens the options form for the scenProc worker. Here you can specify the paths to the different compiler tools as used by the worker. See Figure 9.4 for the options form that shows. It is similar to the scenProc options form, but without the settings that are related to the scenProc graphical user interface.

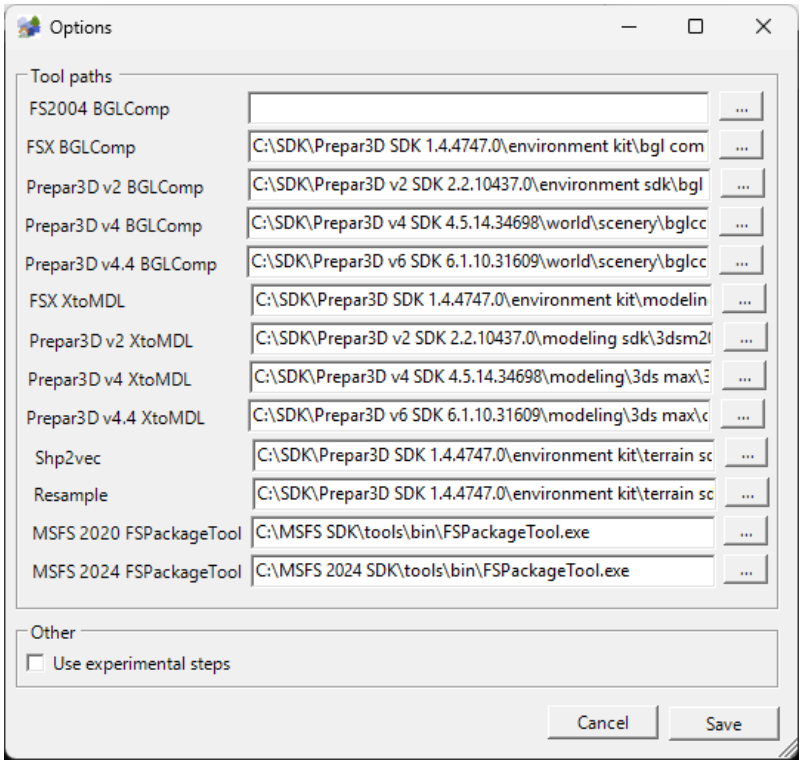



Figure 9.4: scenProc Worker options form

## 9.4 Workflow feeder

With the scenProc Batch Runner you can process many jobs in a batch run. But imagine you are working on a scenery project for a big country, where you have multiple scenProc scripts to do different jobs, like detecting vegetation, creating autogen, placing XML objects or creating photoreal scenery. Would it not be nice if you could manage the production of such a scenery more efficiently as well? This is exactly what the Workflow Feeder can do for you. You can define a workflow of different scripts and the feeder will feed them all into the scenProc Batch Runner for processing.

### 9.4.1 User interface

In Figure 9.5 you see the user interface of this tool. You can start it from the scenProc Batch Runner. This section explains what options are available and in the next section an example of using the feeder is given.

The user interface consists of four different areas:

- A toolbar at the top where you find the following buttons:

  **New workflow** This creates a new empty workflow.

  **Load workflow** This loads a saved workflow in from disk. The default extension for the workflow file is SBW (**s**cenProc **B**atch **W**orkflow).

  **Save workflow** This saves the workflow to disk. The workflow is automatically saved every minute to disk as well, so that the current processing status is not load when the program closes unexpectedly.

  **LOD size** With this combobox you can setup the LOD tile size that should be used for the workflow. Be careful if you change the LOD size the status of all existing tiles will be lost.

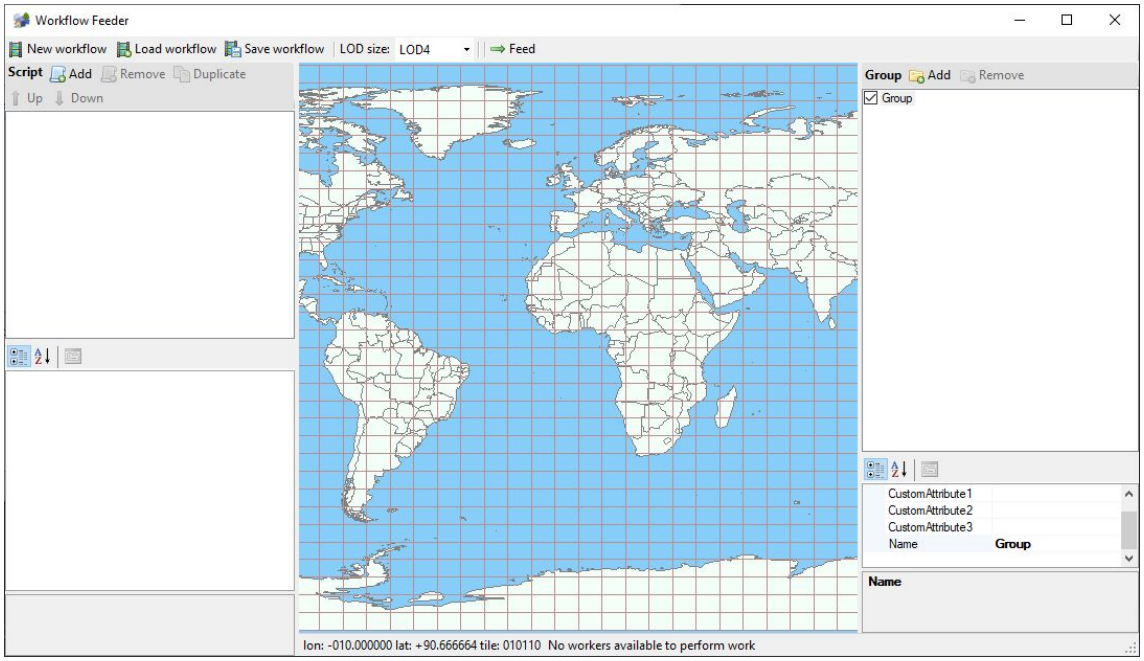



Figure 9.5: The Workflow feeder user interface

**Feed** When the feed button is pressed new jobs will be send to the scenProc Batch Runner when all batch runner workers are idle.

- On the left a list of scripts that your workflow consists of. At the left bottom the properties of the script are shown. The following buttons are available for the scripts list:

  **Add** Add a new script to the list.

  **Remove** Remove the selected script form the list.

  **Duplicate** Duplicate the selected script. This can be useful if you want to add another script that has many similar settings.

  **Up** Move the selected script up in the workflow.

  **Down** Move the selected script down in the workflow.

- In the middle a map where you can see which tiles your project consists of and what the status of running the workflow for these tiles is.

- On the right a list of the groups in your workflow. You can use the groups to run the same workflow for different projects, where each group can for example be a state or country that you want to produce. The following buttons are available for the groups list:

  **Add** Add a new group to the list.

  **Remove** Remove the selected group from the list. At least one group should always remain.

### 9.4.2 Example usage

To illustrate how the workflow feeder work, let's setup a ficticious project. In this project we will let scenProc run two different scripts. I will not discuss the content of the scripts here, this section is about the workflow we want to setup for them.

- Create autogen objects
- Create XML object

And we want to use this workflow for two different product, let's say a scenery for the Netherlands and for Belgium. So how would we configure the Workflow Feeder for this?

First we need to choose which tile size we want to use for the workflow. For each tile that the workflow feeder processes it will send all the jobs to the batch runner. So in general it would be good to have 16 to 64 jobs in each tile to make sure the processing remains balanced. Typically the script you have made will be designed to process a specific tile size in one run. Let's see that our autogen script has been designed to process a LOD11 area and the XML script was designed for LOD10. If we then select LOD8 for our workflow we will have 16 XML jobs and 64 AGN jobs in each tile. That is a good balance. So we will pick LOD8 for the workflow tile size. So let's make sure the LOD size combobox is set to that value.

#### Setting up the scripts

Now we are going to add our two scripts to the workflow. Figure 9.6 shows the interface after we have added the two scripts. For each of the scripts you can setup the following properties:

- **Job requeue count**: The maximum number of times the script is requeued when it fails.
- **Job timeout**: The timeout in minutes to use for this job. When it runs longer than this time the job will be stopped and reported as failed.
- **Log folder**: When a folder is specified here, the log file of the script are written to that folder.
- **Worker restart count**: If a value is specified here the worker is automatically restarted after processing this amount of jobs.
- **Custom attribute 1/2/3**: With these properties you can setup custom attributes that are specific to this script. You can use them later in the batch variables.
- **LOD size**: This LOD tile size at which the script should be run. Based on this size and the workflow LOD size it is determined how many jobs should be started when running the workflow.
- **Process all tiles parallel**: When this option is set to true all tiles in the workflow are send to the batch runner at once. This can be useful if there are only a few jobs to run for each tile, in that case queuing them all at once can improve the performance.
- **Script**: The name of the script that should be run.
- **Title**: Tile of the script that will be shown in the scripts list.
- **User variable 0 till 9**: With these properties you determine which values should be send as user variables to your script. You can choose from the following options:
  - LODBoundingBox: passes the bounding box for the current job.
  - CustomGroup#: passes the value of the custom group attribute with the specified number.
  - CustomScript#: passes the value of the custom script attribute with the specified number.
  - LOD#Name: passes the tile name for the center of the current job at the specified terrain tile LOD size.
  - QMID#Name: passes the tile name for the center of the current job at the specified terrain tile QMID size.

As can be seen in Figure 9.6, we have setup our autogen script to use LOD11 and we pass 3 batch variables into the script: the bounding area, the LOD11 name and the first custom group attribute. For the XML object script we use almost the same settings, except that it is setup to use LOD10.

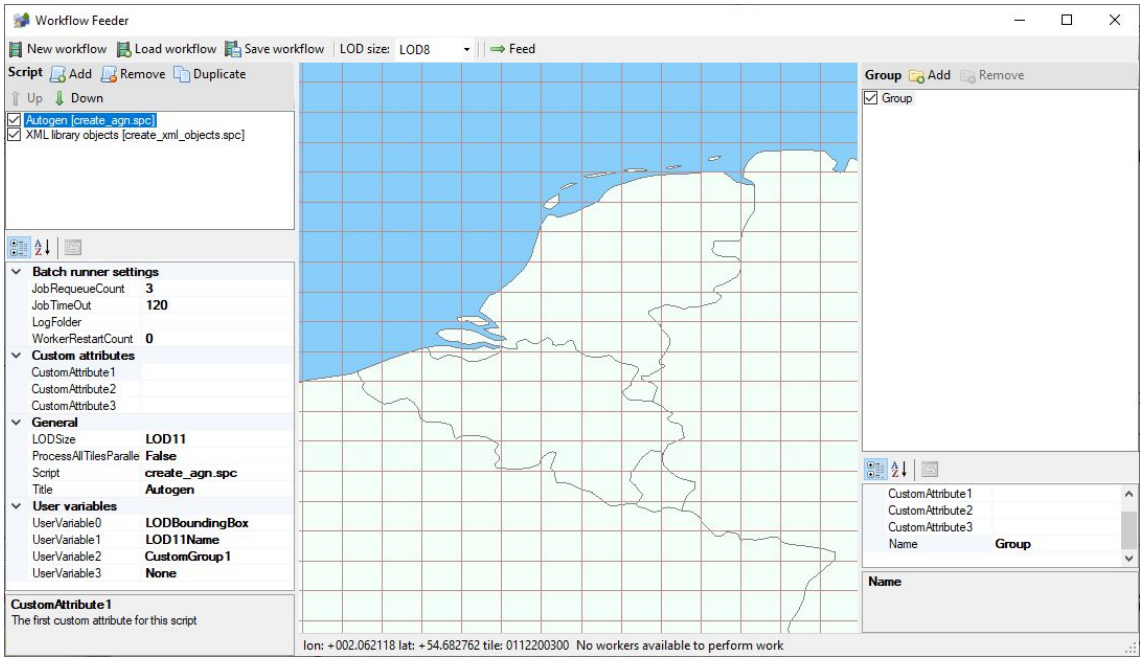



Figure 9.6: Setting up the scripts

### Setting up the groups

Now that the scripts are setup, we can define the groups and which areas should be processed. First we setup the group for the Netherlands. We change the name of the default group and specify the correct custom attribute as well. We have named it Netherlands, so that we can use that value in our script to save the files to the correct folder. Figure 9.7 shows the settings for the group.

Next we need to add the tiles that belong to this group. This is done by dragging on the map to highlight an area. When you release the mouse button you get a menu where you can specify the action to take. In this case we will select Add tile to add the selected tiles to the group. The gray tiles in Figure 9.7 are the tiles we have selected for the Netherlands group in the end. If you make a mistake when adding tiles to the group, you can drag over the added tiles to select them again and then select Remove tile from the menu, this will remove them again.

We repeat the same procedure for the Belgium group. Figure 9.8 shows the result of that.

### Running the workflow

Now we have all the scripts and groups setup it is time to start running the workflow. For this we need to make sure that in the scenProc Batch Runner worker(s) have been started that can do the work.

Next we need to decide which scripts and groups we want to run. There are checkboxes shown in the scripts and groups lists that you can use to select which scripts and groups you want to run. Sometimes you might want to run the entire workflow, but when you are updating only parts of your scenery you maybe only want to run 1 or 2 scripts.

Once all the settings are correct, you press the Feed button in the toolbar and the workflow feeder will start to feed jobs to process to the batch runner. If you hover with the mouse over the tiles in the map you can see the status of processing them. And the colors also indicate this status. Once all tiles are green they have finished successful. See Figure 9.9 for the status while the workflow is running. When a group has finished processing, the workflow feeder will proceed to the next group that is active for processing.

If tiles have failed you can click on them and select the Reset failed tiles option to set their status

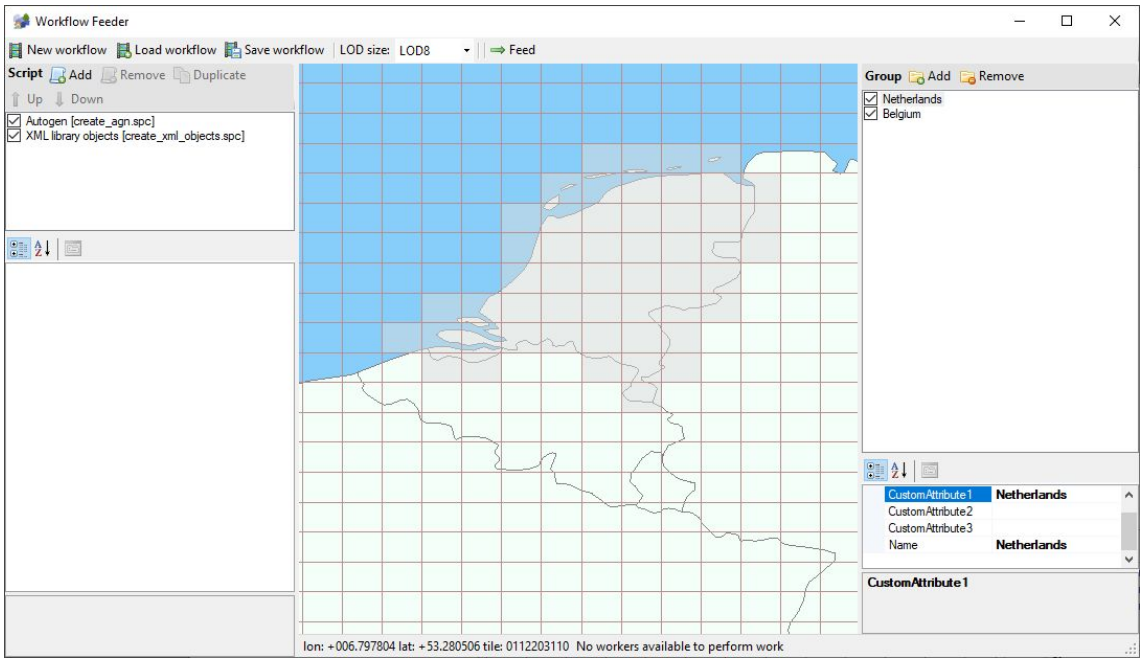



Figure 9.7: The tiles for the Netherlands group

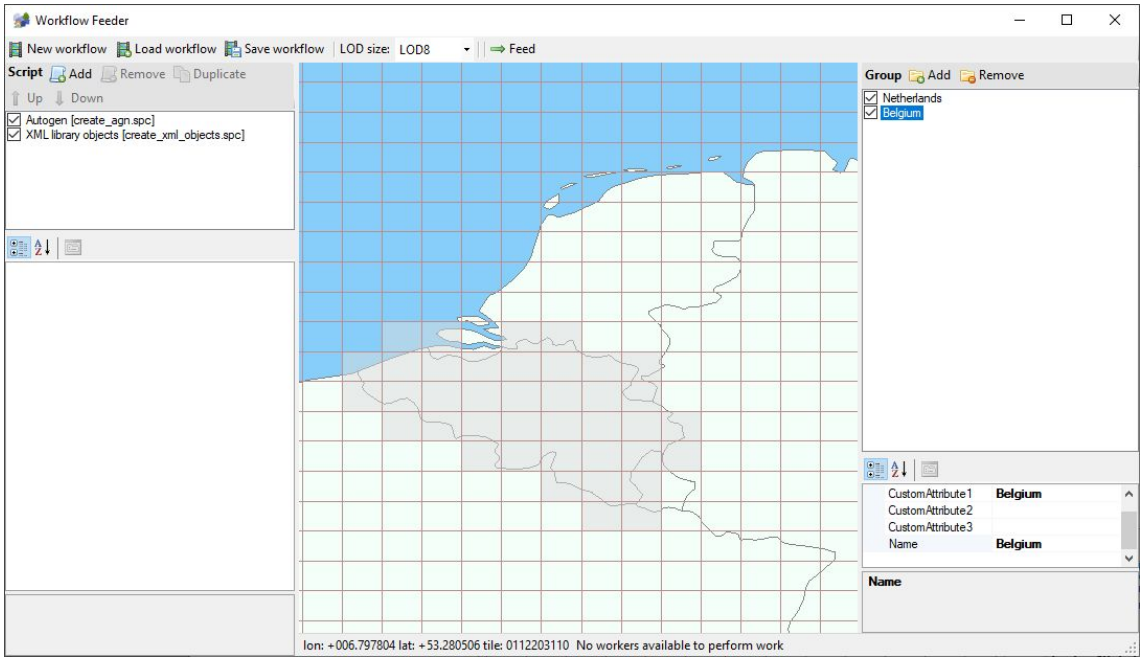



Figure 9.8: The tiles for the Belgium group

back to Not ran, so that they will be processed again.

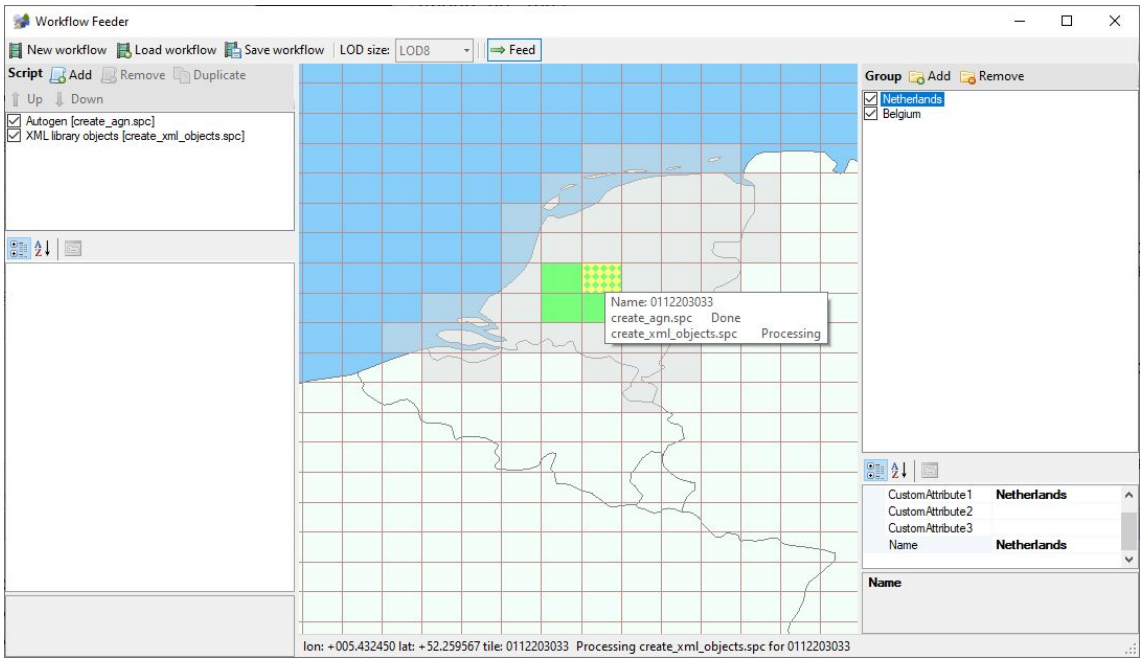



Figure 9.9: Workflow feeder status while processing

# Chapter 10

# List of available steps

In this appendix a detailed description of all available steps you can use in your scenProc configuration file is given. For each step the purpose, the arguments that you need to specify and examples are provided.

## 10.1 Importing

In this section all steps that allow you to import data into scenProc are discussed.

### 10.1.1 ImportAGN

**Purpose**

Import features from autogen files (AGN). This allows you to import existing autogen and thus add extra autogen objects to the

**Arguments**

1. The folder where the AGN files that should be read are stored
2. Optional options, the following options can be provided for this step:
    - CREATEFEATURES specifies that the autogen objects should not only be imported, but that features should also be created from them. This for example allows you to export the features to other formats like SHP. When features are not created, the autogen objects can only be exported to AGN files again.
    - EXISTINGCELLSONLY specifies that only autogen objects should be loaded from AGN files that match with autogen cells for which data is already loaded into scenProc. This prevents that autogen objects are loaded into memory that will not be modified by the current data.
    - The options below can be used to load only certain types of autogen objects. If these options are not specified all available autogen objects will be imported.
        - IMPORTGENBUILDS specifies to import generic buildings objects.
        - IMPORTLIBOBJS specifies to import library objects.
        - IMPORTRECTVEGS specifies to import rectangular vegetation objects.
        - IMPORTPOLYVEGS specifies to import polygonal vegetation objects.
        - IMPORTPOLYBUILDS specifies to import polygonal building objects.
        - IMPORTROWHOUSES specifies to import row house objects.

– NOINVERSEDISTORTION specifies that when creating features from the imported autogen object the inverse correction for the autogen distortion should not be applied. Use this option if you are creating features made from autogen files made with Annotator. If you are creating features from autogen files made with scenProc you should not use this feature.

**Example**

Import AGN files from the selected folder:

```
ImportAGN|c:\flightsim\my scenery\texture
```

Import AGN files from the selected folder that overlap with currently loaded data:

```
ImportAGN|c:\flightsim\my scenery\texture|EXISTINGCELLSONLY
```

Import only the rectangular and polygonal vegetation objects from the selected folder:

```
ImportAGN|c:\flightsim\my scenery\texture|IMPORTRECTVEGS;IMPORTPOLYVEGS
```

### 10.1.2 ImportBGL

**Purpose**

Import features from BGL files. At the moment only the following object types can be read from a BGL file:

- Library object placements, these can be imported as point feature or a polygon of the footprint of the used library object can be made.
- Car parking polygons.

The following attributes are added to these features:

| Attribute | Point | Polygon | Description |
|---|---|---|---|
| BGLOBJTYPE | x | x | The type of BGL object, either LIBRARYOBJECT or CARPARKING. |
| GUID | x | x | The GUID of the object placement. |
| HEADING | x | | The heading of the object placement. |
| SCALE | x | | The scale of the object placement. |
| ALT | x | x | The altitude of the object placement. |
| ALTISAGL | x | x | True is the altitude is AGL, False if it is MSL. |

Table 10.1: Attributes of features generated from object placements

**Arguments**

1. The filename of the BGL file to load. You can use a wildcard to select multiple files that match a pattern.
2. Bounding box of the region you want to import from the BGL file, or use * to load all data.
3. Optional options, possible values are:
   - RECURSIVE to not only search the current directory for the pattern of the filename, but also all children directories of the current directory.
   - GENERATE_FOOTPRINT to generate a footprint polygon instead of a point feature for the object placement of the object.
4. Optional, the filename of an additional library BGL file to look for objects. You can use a wildcard to select multiple files that match a pattern.

**Example**

Import all object placements from the file obplace.bgl.

```
ImportBGL|obplace.bgl|*
```

Import all object placements from the file obplace.bgl that are between N50-N51 and E005-E006.

```
ImportBGL|obplace.bgl|5;6;50;51
```

Import all object placements from files that are named ob* in the current directory.

```
ImportBGL|ob*.bgl|*
```

Import all object placements from files that are named ob* in the current directory and all children directories.

```
ImportBGL|ob*.bgl|*|RECURSIVE
```

Import all object placements from the file obplace.bgl and generate footprint polygons instead of points.

```
ImportBGL|obplace.bgl|*|GENERATE_FOOTPRINT
```

Import all object placements from the file obplace.bgl and generate footprint polygons instead of points, also look for library objects in the file lib.bgl.

```
ImportBGL|obplace.bgl|*|GENERATE_FOOTPRINT|lib.bgl
```

### 10.1.3 ImportBLN

**Purpose**

Import features from BLN files.

**Arguments**

1. Filename of the BLN file to import
2. Coordinates of the bounding rectangle that should be read. If supplied only features within this area will be imported. If you provide a * all features will be imported. The coordinates should be provided in the geodetic degrees. The coordinates are specified as:

   `minx;maxx;miny;maxy`.
3. The projection used by the vector data. This is entered as a Proj4 string. The auto completion will help you to enter these strings based on the name describing the projection. Use NOREPROJ if your data is also in geodetic coordinates and does not need to be reprojected.

**Example**

Import data from the specified file:

```
ImportBLN|c:\myfile.bln|*|NOREPROJ
```

### 10.1.4 ImportGDAL

**Purpose**

This step can import raster data from the different formats supported by the GDAL library. For example GeoTIFF and JPEG2000 files are supported.

If the raster data is not using the geodetic WGS84 projection it will be reprojected to this projection in memory. For big raster files that might give too much memory usage, so in that case you can better make sure that the raster data has the right projection before loading it.

**Arguments**

1. Filename of the file with raster data to import. It is also possible to specify the filename with a wildcard so that all files matching the pattern will be read.
2. Coordinates of the bounding rectangle that should be read. If supplied only features within this area will be imported. If you provide a * all features will be imported. The coordinates should be provided in the projection of the source data. The coordinates are specified as: `minx;maxx;miny;maxy`.
3. The projection used by the imagery data. You can specify AUTODETECT to let scenProc detect the projection data automatically. If that fails you can specify it manual using a Proj4 string. The auto completion will help you to enter these strings based on the name describing the projection. Use NOREPROJ if your data is also in geodetic coordinates and does not need to be reprojected.
4. Optional, the attribute type and category (name) of the new attribute to add to each feature read.
5. Optional, the value of the new attribute to add to each feature read.

**Example**

Import specified raster image:

```
ImportGDAL|myfile.tif|*|AUTODETECT
```

Import only raster features between N50 E5 and N51 E6:

```
ImportGDAL|myfile.tif|5;6;50;51|NOREPROJ
```

Import specified raster image and add a new attribute to the raster feature read:

```
ImportGDAL|myfile.tif|*|AUTODETECT|String;rasterType|myType
```

Import all raster images in the specified folder that match the specified search filter:

```
ImportGDAL|C:\mydata\200*.jp2|*|AUTODETECT
```

### 10.1.5 ImportINF

**Purpose**

This step can import raster data from a resample INF file. Use this approach if you don't have geo-referenced image files, but you have setup a INF file for resample with the position and resolution information.

**Arguments**

1. Filename of the INF file to import.

**Example**

Import specified INF file:

```
ImportINF|myfile.inf
```

### 10.1.6 ImportOGR

**Purpose**

This step can import vector data from many different formats using the OGR libraries. The point, lines and polygon features will be imported from the vector data, as well as all available attributes.

**Arguments**

1. Filename of the file with vector data to import. You can use a wildcard to load multiple files at once.
2. Coordinates of the bounding rectangle that should be read. If supplied only features within this area will be imported. If you provide a * all features will be imported. The coordinates should be provided in the projection of the source data. The coordinates are specified as:

   `minx;maxx;miny;maxy`.
3. A list of required attribute categories. If provided only features that have at least one of these categories will be imported. This can be useful to only import the features that you want to use later on, especially if the file you are importing contains a lot of data. If you provide a * all features will be imported. If you use FLAYER=name1;name2+ only features from layers with one of the specified names will be imported.
4. The projection used by the vector data. You can specify AUTODETECT to let scenProc detect the projection data automatically. If that fails you can specify it manual using a Proj4 string. The auto completion will help you to enter these strings based on the name describing the projection. Use NOREPROJ if your data is also in geodetic coordinates and does not need to be reprojected.
5. Optional, the attribute type and category (name) of the new attribute to add to each feature read.
6. Optional, the value of the new attribute to add to each feature read.

**Example**

Import all features from a shapefile and auto detect the projection used:

```
ImportOGR|myfile.shp|*|*|AUTODETECT
```

Import all features from a shapefile that uses UTM coordinates:

```
ImportOGR|myfile.shp|*|*|+proj=utm +zone=31 +datum=WGS84 +units=m +no_defs <>
```

Import features from the category landuse and building from an OSM file:

```
ImportOGR|myfile.osm|*|landuse;building|NOREPROJ
```

Import features from the layers water and forest from a GML file:

```
ImportOGR|myfile.gml|*|FLAYER=water;forest|NOREPROJ
```

Import only features between N50 E5 and N51 E6 from a shapefile:

```
ImportOGR|myfile.shp|5;6;50;51|*|NOREPROJ
```

Import all features from all shapefiles in the temp folder:

```
ImportOGR|c:\temp\*.shp|*|*|NOREPROJ
```

Import all features from a shapefile and add an new attribute with the name type and value of myFeat to each type:

```
ImportOGR|myfile.shp|*|*|NOREPROJ|String;type|myFeat
```

### 10.1.7 ImportSBuilder

**Purpose**

This step can import raster data from images saved from SBuilder. The geo-referencing information is read from the TXT file that comes with the image.

**Arguments**

1. Filename of the SBuilder image to import. Wildcards can be used to select multiple files.

**Example**

Import specified image file:

```
ImportSBuilder|L14X27285X27301Y16458Y16470.tif
```

Import all image files in a folder:

```
ImportSBuilder|C:\SBuilder\images\*.tif
```

## 10.2 Processing

In this section all steps to process data and derive new data are described.

### 10.2.1 AddAttribute

**Purpose**

Add a new attribute to the selected features. The attribute category and value are provided.

**Arguments**

1. The filter that selects the features to which the attribute should be added
2. The attribute type and category (name) of the new attribute
3. The value of the new attribute. There are three special values that have a different behaviour:
   - RND(x) generates a random number between 0.0 and x
   - RND(x,y) generates a random number between x and y
   - NEWGUID creates a new random GUID
   - TILELOD adds the name of the LOD of the cell the feature is in

**Example**

Add new attribute to all square polygons:

```
AddAttribute|FTYPE="POLYGON" And FNUMVERT=4 And FNUMPERPANG=4|String;BUILDINGTYPE
    ↪ |SQUARE
```

Add random heading between 0 and 360 to all point features:

```
AddAttribute|FTYPE="POINT"|Double;heading|RND(360)
```

Add random building height between 10 and 20 to all buildings:

```
AddAttribute|building=*|Double;height|RND(10,20)
```

### 10.2.2 AddAttributeIfInside

**Purpose**

Add a new attribute to the selected features when they are inside a group of selected features. The attribute category and value are provided. This can for example be used to classify building footprints based on the fact if they fall within a residential or industrial area polygon.

**Arguments**

1. The filter that selects the features to which the attribute should be added
2. The filter that selects the polygons inside which the features that get the attribute should fall
3. The attribute type and category (name) of the new attribute
4. The value of the new attribute.
5. Optional options, possible values are;
    - NOT_INSIDE indicates that attributes will be added to features that are not inside the selected polygons.
    - ADD_POLYGON indicates that the attribute should be added to the polygon feature inside which the tested feature is.
    - COPY_ATTR indicates that the attribute with the given name should be copied from the polygon to the feature that is inside it. If you want to copy multiple attributes, you can specify there names with a comma in between. See the example below for details.

**Example**

Add a new attribute to all features that fall within residential polygons:

```
AddAttributeIfInside|building="*"|landuse="residential"|String;BLDTYPE|
    ↪ RESIDENTIAL
```

Add a new attribute to all features that are not inside the forest polygon:

```
AddAttributeIfInside|type="tower"|natural="forest"|String;TWRTYPE|OUTSIDE_FOREST|
    ↪ NOT_INSIDE
```

Add a new attribute to all building polygons to indicate the building type, when a point feature indicating churches is inside the polygon:

```
AddAttributeIfInside|type="church"|type="building"|String;bldType|church|
    ↪ ADD_POLYGON
```

Copy the attribute vegType (vegetation type) from forest polygons to all point tree features that are inside the forest polygon:

```
AddAttributeIfInside|tree="*"|type="forest"|String;vegType||COPY_ATTR
```

Copy the attributes vegType (vegetation type) and vegHeight (vegetation height) from forest polygons to all point tree features that are inside the forest polygon:

```
AddAttributeIfInside|tree="*"|type="forest"|String;vegType,vegHeight||COPY_ATTR
```

### 10.2.3 AddAttributeSampleRaster

#### Purpose

Add a new attribute to the selected features where the value is determined by sampling a raster image. This can for example be used to sample the color of a building roof from imagery.

#### Arguments

1. The filter that selects the features to which the attribute should be added
2. The filter that selects the raster images that should be sampled. For each feature the raster image within which the feature is will be selected.
3. The sampling mode to use. You can use CENTER to get the raster value at the center of the feature, use AVERAGE to get the average raster value over the entire feature or use RADIUS to raster the raster in a circular areas with a given radius around the center of the feature.
4. The category (name) of the new attribute. This is the basename, the attribute name will be appended with the band index, e.g. _0 or _2. So if you specify name here, you will get attributes name_0, name_1, etc.
5. Optional offset to apply to the sampled value. This can be used when sampling elevation data to add an elevation offset.
6. Optional radius in degrees to be used in the RADIUS mode.

#### Example

Sample the average color of all buildings:

```
AddAttributeSampleRaster|building="*"|FTYPE="RASTER"|AVERAGE|color
```

Sample the elevation at the center of the building, but add a 10 meter offset:

```
AddAttributeSampleRaster|building="*"|FROMFILE="elev.tif"|CENTER|elev|10
```

### 10.2.4 AddCellAttribute

#### Purpose

Add a new attribute to the selected cells. The attribute category and value are provided.

**Arguments**

1. The filter that selects the cells to which the attribute should be added
2. The attribute type and category (name) of the new attribute
3. The value of the new attribute. There are three special values that have a different behaviour:
   - RND(x) generates a random number between 0.0 and x
   - RND(x,y) generates a random number between x and y
   - NEWGUID creates a new random GUID
   - TILELOD adds the name of the LOD of the cell
4. Optional options. Possible values are:
   - APPENDVALUE to indicate that the new attribute value should not replace but be appended to existing values.

**Example**

Add new attribute to cells based on the value of another attribute:

```
AddCellAttribute|NUMHOUSE>100|String;DENSITY|HIGH
```

### 10.2.5 AddCellAttributeFeatureCount

**Purpose**

Add a new attribute to the selected cells of which the value is the amount of features that match a specific filter. The attribute category is provided. This can for example be used to assign the best texture type based on the building year attribute of house features.

**Arguments**

1. The filter that selects the cells to which the attribute should be added
2. The filter that selects the features which should be counted
3. The category (name) of the new attribute

**Example**

Add the count of house build between 1940 and 1960 as an attribute to a cell:

```
AddCellAttributeFeatureCount|*|type="HOUSE" And BUILD>1940 And BUILD<1960|
    ↪ NUM_40_60
```

### 10.2.6 AddCellAttributeIfInside

**Purpose**

Add a new attribute to the selected cells when the center of the gridcell is inside a group of selected polygon features. The attribute category and value are provided. This can for example be used to classify gridcells based on vector data about population density and use that to set the autogen heigth.

**Arguments**

1. The filter that selects the cells to which the attribute should be added
2. The filter that selects the polygons inside which the center of the cell that gets the attribute should fall
3. The attribute type and category (name) of the new attribute
4. The value of the new attribute.

**Example**

Add new attribute to all cells that fall within residential polygons:

```
AddCellAttributeIfInside|*|landuse="residential"|String;TEXTURESTYLE|RESIDENTIAL
```

### 10.2.7 AddCellAttributeSampleRaster

**Purpose**

Add a new attribute to the selected cells where the value is determined by sampling a raster image.

**Arguments**

1. The filter that selects the cells to which the attribute should be added
2. The filter that selects the raster images that should be sampled. If the cell is covered by multiple raster images the one that contains the center of the cell will be used.
3. The sampling mode to use. You can use CENTER to get the raster value at the center of the feature or use AVERAGE to get the average raster value over the entire feature.
4. The category (name) of the new attribute. This is the basename, the attribute name will be appended with the band index, e.g. _0 or _2. So if you specify name here, you will get attributes name_0, name_1, etc.

**Example**

Sample the color oat the center of all cells:

```
AddAttributeSampleRaster|*|FTYPE="RASTER"|CENTER|color
```

### 10.2.8 BooleanFeatures

**Purpose**

This step can be used to perform boolean operations on two sets of features, for example to determine the intersection or the difference of the two sets. It can for example be used to subtract road polygons from a forest, so that the trees don't end up on the middle of the road. Or another typical use case is to subtract buildings from forest areas, as polygonal vegetation autogen will supress buildings when they overlap. Figure 10.1 shows an example where buildings have been subtracted from vegetation polygons.

**Arguments**

1. The type of boolean operation to apply, either use INTERSECTION or DIFFERENCE.
2. Filter to select the first set of features (Set A)
3. Filter to select the second set of features (Set B)

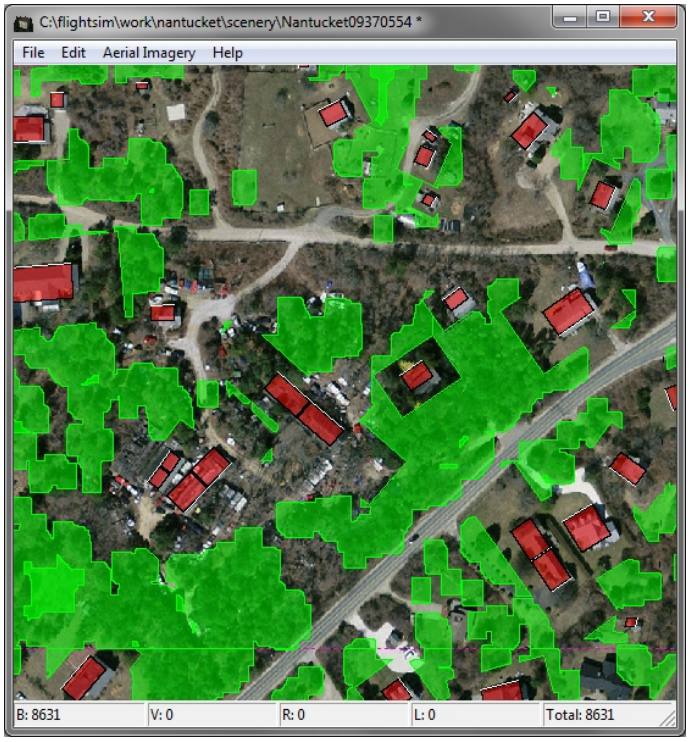



Figure 10.1: Example of features being subtracted from each other. The building polygons have been subtracted from the vegetation, where the buildings have first been scaled to give a buffer area.

4. Scale factor to apply on Set B before performing the boolean operation
5. Optional options, the following options can be provided for this step:
    - MINAREAPOLY specifies that instead of the exact polygon of the feature in Set B, the minimum area polygon (the best fitting rectangle around the feature) should be used

**Example**

Subtract all buildings from the forests:

```
BooleanFeatures|DIFFERENCE|landuse="forest"|building="*"|1
```

Subtract all buildings, using their minimum area polygon, from the forests. A scaling of 1.5 is used to get a buffer zone between the buildings and the vegetation:

```
BooleanFeatures|DIFFERENCE|landuse="forest"|building="*"|1.5|MINAREAPOLY
```

Determine the intersection of the building polygons and the border polygon of the country to only process the buildings inside that country:

```
BooleanFeatures|INTERSECTION|type="building"|border="Country"|1
```

To make it more clear what the output is with a specific first and second set, the following samples take the same data as input and show the resulting output. Figure 10.2 shows the input polygons that are used for these examples. Polygon A is shown in green and polygon B in red.

The first example uses DIFFERENCE as operation and has polygon A as the first set and polygon B as the second set. The resulting output is shown in Figure 10.3. It can be seen that the shape of polygon B has been removed from polygon A and are kept as the first set.

```
BooleanFeatures|DIFFERENCE|FROMFILE="polyA.shp"|FROMFILE="polyB.shp"|1
```

The second example also uses DIFFERENCE as operation, but reverses the polygons. So polygon B is used as first set and polygon A as second set. The resulting output is shown in Figure 10.4. It can be seen that in this case the shape of polygon A has been removed from polygon B and are kept as the first set.

```
BooleanFeatures|DIFFERENCE|FROMFILE="polyB.shp"|FROMFILE="polyA.shp"|1
```

The last example uses the INTERSECTION operator. In this case it does not matter in which order the two sets are specified, as the intersection of them is calculated. The resulting output is shown in Figure 10.5. It can be seen that in this case the overlapping parts of the two polygons are kept as the first set.

```
BooleanFeatures|INTERSECTION|FROMFILE="polyA.shp"|FROMFILE="polyB.shp"|1
```

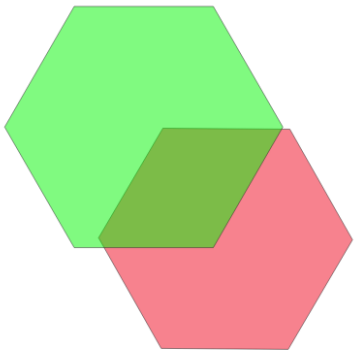


Figure 10.2: Two polygons used to explain different boolean options. Polygon A is shown in green and polygon B in red.

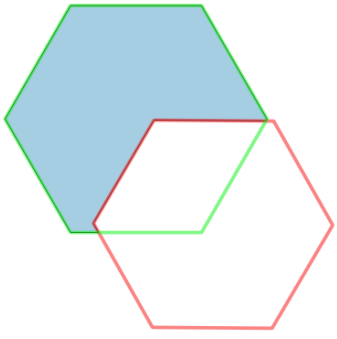


Figure 10.3: Output of boolean when the different between polygon A and polygon B is calculated.

### 10.2.9 BufferFeatures

**Purpose**

Add a buffer around the selected features. Figure 10.6 shows an example of buffered features.

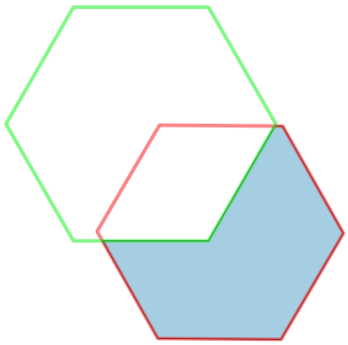


Figure 10.4: Output of boolean when the different between polygon B and polygon A is calculated.

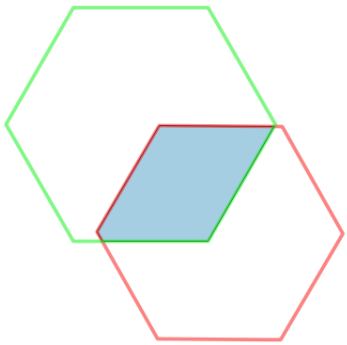


Figure 10.5: Output of boolean when the intersection between polygon A and polygon B is calculated.

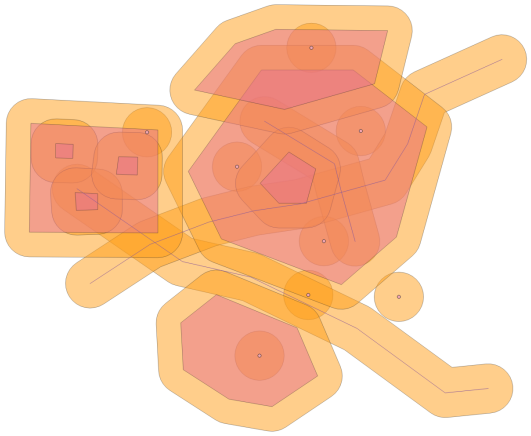


Figure 10.6: Example of buffered features.

**Arguments**

1. Filter to select the features to buffer
2. The size of the buffer in degrees
3. The attribute type and category (name) of the new attribute
4. The value of the new attribute

**Example**

Add a buffer of 0.001 degrees around all features:

```
BufferFeatures|*|0.001|String;type|buffered
```

### 10.2.10 CalibrateFeatures

**Purpose**

This step can be used to assign coordinates to data that does not have latitude and longitude coordinates assigned. For two reference points (XY) the latitude and longitude is given and the step will scale all other features correctly based on this.

**Arguments**

1. Filter to select the features that will be scaled
2. The X and Y coordinate pair of the first calibration point, the X and Y value are separated by a semicolon (;).
3. The X and Y coordinate pair of the second calibration point, the X and Y value are separated by a semicolon (;).
4. The longitude and latitude coordinate pair of the first calibration point, the longitude and latitude value are separated by a semicolon (;).
5. The longitude and latitude coordinate pair of the second calibration point, the longitude and latitude value are separated by a semicolon (;).

**Example**

Scale all features, while assigning N51.5 E6.2 to XY 0;0 and N51.6 E6.4 to XY 1000;700:

```
SCALEFEATURES|*|0;0|1000;700|6.2;51.5;6.4;51.6
```

### 10.2.11 CopyAttributeIfInside

**Purpose**

Copy attributes from on feature to another, when the second feature is inside the first one. This can for example be used to copy attributes from courser landcover data over to individual vegetation polygons.

**Arguments**

1. The filter that selects the features to which the attribute should be copied
2. The filter that selects the polygons inside which the features that get the attribute should fall, these are also the features that the attributes will be copied from
3. The category (name) that should be copied over

**Example**

Copy the tree type attribute over from landcover to vegetation data:

```
AddAttributeIfInside|vegetation="*"|landuse="forest"|TREETYPE
```

### 10.2.12 CreateRectangle

**Purpose**

Create a new rectangle polygon from a given bounding box or geographic name.

**Arguments**

1. The bounding box of the rectangle (minlon;maxlon;minlat;maxlat) or the name of the geographic region that the bounding box should be searched for online using OpenStreetMap.
2. The attribute type and category (name) of the new attribute
3. The value of the new attribute.

**Example**

Create a rectangle between N51 E5 and N52 E6 with the attribute `type` and value `area`:

```
CreateRectangle|5;6;51;52|String;type|area
```

Create a rectangle around the island of Texel in the Netherlands with the attribute `area` and value |Texel|:

```
CreateRectangle|Texel|String;area|Texel
```

### 10.2.13 DetectFeatures

**Purpose**

Detect features from raster images. This step can be used to create new vector data when no good source is available. For example forest areas can be detected from the imagery and used as input to create autogen vegetation. See chapter 6 for more information on the algorithms and approach used in the texture filters.

**Arguments**

1. The filter that selects the raster images that should be used as input
2. The filename of the texture filter configuration file that specifies how the features should be detected
3. The attribute type and category (name) of the new attribute that will be added to the detected features
4. The value of the new attribute
5. A filter to select a set of masking features. The feature detection will only be performed within these features. Specify NONE if you want to detect over the entire raster image.
6. Optional options, the following options can be provided for this step:
    - DONTPROCESSHOLES specifies that holes in detected features should not be processed, which means that only the outer rings of the polygons are processed.

**Example**

Detect vegetation polygons from raster images:

```
DetectFeatures|FTYPE="RASTER"|tree_detect.tf2|String;veg|tree|NONE
```

Detect vegetation polygons from raster images only with residential areas:

```
DetectFeatures|FTYPE="RASTER"|tree_detect.tf2|String;veg|tree|landuse="
    ↪ residential"
```

### 10.2.14 FilterDuplicateLines

**Purpose**

Remove duplicate lines from the loaded features. Duplicate lines are lines with exactly the same vertices.

**Arguments**

None.

**Example**

None.

### 10.2.15 FilterFeatures

**Purpose**

Filter out features when they are overlapping or very close to other features. This can for example be used to remove objects near roads or from the water areas.

**Arguments**

1. The filter that selects the features that you want to be filtered.
2. The filter that selects the features you want to filter (check) against.
3. The filter mode to be used, possible values are:
    - OVERLAP_BBOX: In this mode features are removed when their bounding box is overlapping with any other feature. The features are tested in the order they have been imported.
    - NOT_OVERLAP_BBOX: In this mode features are removed when their bounding box is not overlapping with any other feature.
    - DISTANCE_CENTER_SMALLER: In this mode features are removed when the distance from the center of the bounding box to the center of the bounding box of any other feature is smaller than the threshold value specified. The features are tested in the order they have been imported.
    - DISTANCE_CENTER_GREATER: In this mode features are removed when the distance from the center of the bounding box to the center of the bounding box of all other features is greater than the threshold value specified.
    - DISTANCE_FEATURE_SMALLER: In this mode features are removed when the distance from the feature to any other feature is smaller than the threshold value specified. The features are tested in the order they have been imported.
    - DISTANCE_FEATURE_GREATER: In this mode features are removed when the distance from the feature to all other features is greater than the threshold value specified.

- CENTER_INSIDE: In this mode features are removed when their center is inside of any of the features being checked against.
- CENTER_NOT_INSIDE: In this mode features are removed when their center is not inside of all the features being checked against.

4. Optional, the distance threshold used to remove a feature. This argument is used in the DISTANCE_CENTER and DISTANCE_FEATURE modes.

**Example**

Filter out all point features of trees that are within 20 meter of a road:

```
FilterFeatures|FTYPE="POINT" AND type="tree"|type="road"|DISTANCE_FEATURE_SMALLER
    ↪ |20
```

Filter out all buildings that are overlapping in bounding box with vegetation:

```
FilterFeatures|type="building"|type="vegetation"|OVERLAP_BBOX
```

Filter out all point features of lights that are within 10 meter of another light. This can be used to declutter the output of the `LineToPoint` step for example. Note that in this case the same filter is used for the features to be filtered and the features to be checked against:

```
FilterFeatures|type="light"|type="light"|DISTANCE_CENTER_SMALLER|10
```

### 10.2.16 FilterLineOnPolygon

**Purpose**

Filter out all line features that are exactly on the border of a polygon feature with the same shape.

**Arguments**

None.

**Example**

None.

### 10.2.17 HeadingFromNearestLine

**Purpose**

Adds a new attribute with a heading to the feature. The heading is based on the direction of the line that is closest to the feature. So this can for example be used to add a heading attribute to point features that don't have such an attribute and you can align them with nearby roads.

**Arguments**

1. Filter to select the features to which the heading attribute should be added
2. Filter to select the lines from which the heading should be derived
3. Name of the heading attribute that is added

**Example**

Add a heading attribute to all point features of schools based on any nearby road:

```
HeadingFromNearestLine|amenity="school"|highway="*"|HDG
```

### 10.2.18 LineToPoint

**Purpose**

Create points from a line feature. For example to place light poles along a road.

**Arguments**

1. Filter to select the lines from which the points are created.
2. The mode that should be used for creating the points, either VERTEX, SINGLE, CONTINUOUS or INTERSECTION. See the explanation below for the behavior of the different modes.
3. The distance between the points that are generated. If you enter two values the distance is randomly chosen between the entered values. This argument is not used in VERTEX or INTERSECTION mode.
4. The offset perpendicular to the line. If you enter two values the offset is randomly chosen between the entered values. This argument is not used in VERTEX mode.
5. The distance from the start of the line to the first point that is placed. If you enter two values the distance is randomly chosen between the entered value. This argument is only used in SINGLE mode.
6. The attribute type and name of the attribute that is added to the created point features.
7. The value that will be assigned to the attribute that is added to the created point features.
8. The name of the attribute that will contain the heading value of the point features.
9. Optional options, possible values are:
   - INHERITPARENTATTR when all attributes of the line should be copied over to the created point features.
   - ADDMODULUS2ATTR when a new attribute FMOD should be added that contains the value of the modulus of the number of the new point and 2. This can be used to divide the new point in two groups.
   - ADDMODULUS4ATTR when a new attribute FMOD should be added that contains the value of the modulus of the number of the new point and 4. This can be used to divide the new point in four groups.

**Example**

Create points at 100 meter intervals along a line using SINGLE mode. The first point is placed 50 meters from the start of the line.

```
LineToPoint|type="road"|SINGLE|100|0|50|String;obj|light|hdg
```

Create points at intervals of between 50 and 100 meters along a line, with an offset of 10 meter perpendicular, using SINGLE mode. The first point is placed directly at the start of the line.

```
LineToPoint|type="road"|SINGLE|50;100|10|0|String;obj|light|hdg
```

Create points at 20 meter distance using CONTINUOUS mode:

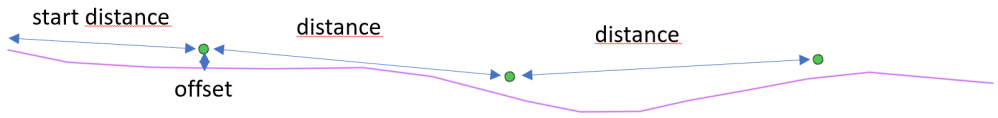



Figure 10.7: LineToPoint SINGLE mode arguments

```
LineToPoint|type="road"|CONTINUOUS|20|0|0|String;obj|median|hdg
```

Create points from the line by placing a point on each vertex of a powerline line:

```
LineToPoint|type="road"|VERTEX|0|0|0|String;obj|powertower|hdg
```

Create points at 5 meter from each intersection of roads of type primary to be used to place lights:

```
LineToPoint|highway="primary"|INTERSECTION|0|5|0|String;obj|light|hdg
```

### VERTEX mode

In VERTEX mode a point feature is created for each vertex of the line. For each point the heading will be the average of the two line segments that meet at this vertex. This can be useful if the vertices of the lines represent objects, for example the pylons of a powerline.

### SINGLE mode

The SINGLE mode is the mode that you should use if you want to place objects along the line. It will create points at a regular distance and for each point the heading will be set to the orientation at the location of the point. It is also possible to apply an offset and a start distance attribute. See Figure 10.7 for a graphical representation of the different arguments.

When using SINGLE mode the step will try to ensure that the distance between the points stay constant as well when two different lines connect.

### CONTINUOUS mode

The SINGLE mode doesn't work correctly if you want to align different objects in a continuous row, for example if you want to place a median object along a road. When using SINGLE mode the objects don't align correctly, because the heading is wrong. Therefore the CONTINUOUS mode should be used for such objects. In this mode the heading is calculated differently so that the objects will align. This however can mean that the objects don't follow the line exactly anymore (they will cut the corners a bit).

The distance and offset values should not use any randomness in this mode. So make sure the enter a constant value, else the objects will not align. The distance between the points should be equal to the size of the object you want to place in a row along the line. The start distance argument is also not used in the CONTINUOUS mode.

In this mode only a single line is processed. When the remainder of the line is too short the step will try to add a point for an object with half the distance. So let's imagine that you are placing points every 20 meters because your object is 20 meters in size. If there is only 16 meters of line left, the step will create a new point that can be used with a 10 meter long object. This process

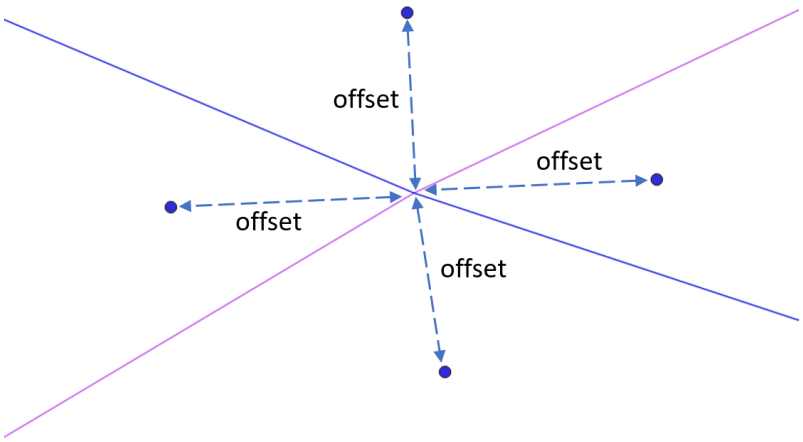



Figure 10.8: LineToPoint INTERSECTION mode

is repeated a maximum of 4 times. The attribute value will get a suffix to indicate these points are supposed to be used with a smaller object. See Table 10.2 for an overview of the suffixes used.

| **Size** | **Attribute value** | **Sample distance** |
|---|---|---|
| distance | VALUE | 20 |
| 1/2 distance | VALUE_2 | 10 |
| 1/4 distance | VALUE_4 | 5 |
| 1/8 distance | VALUE_8 | 2.5 |
| 1/16 distance | VALUE_16 | 1.25 |

Table 10.2: Attribute values for smaller sized objects

#### INTERSECTION mode

The INTERSECTION mode finds the places where the lines that meet the specified filter intersection with each other. It will then place points around this intersection according to the offset value specified. The heading of the point is such that each object will point towards the intersection. This step can for example be used to place lights around intersections only and not at other segments of a line. See Figure 10.8 for a graphical representation of this.

The algorithm places one point between each two segments that meet at the intersection. So that means at a normal 4-way crossing you will get 4 light points. A special attribute FNUMINT is added to each point that assign a random number to each point of each intersection. If you don't want all the lights, but only 1 or 2 you can use this attribute to filter out the other points.

### 10.2.19 LineToPolygon

#### Purpose

Turns line features into polygon feature. For example if you have roads or rivers as line features, but you want to process them as polygons. To be able to identify the new polygons you can add a new attribute and value to the created features.

**Arguments**

1. Filter to select the line features that will be extruded
2. The width the polygons should get in meters
3. Attribute type and name of the new attribute that will be added
4. Value of the new attribute that will be added

**Example**

Create 5 meter wide polygons for all roads of type highway and give them an attribute polygons with the value road:

```
LineToPolygon|FTYPE="LINE" And highway="secondary"|5|String;polygons|road
```

### 10.2.20 MatchLibObjectToPolygon

**Purpose**

Find a library object that fits within the given polygon. This step can be used to select from a library BGL the object that fits best in the bounding rectangle of the selected polygon, given a number of tolerances. If multiple library objects match the criteria one of them is selected randomly. The GUID, scale and heading of the library object are stored as attributes in the feature, so that in the CreateXMLLibObject step the XML to place the objects can be generated.

**Arguments**

1. Filter to select the polygons in which the library object should be fitted.
2. File names of the library BGL files that contain the library objects to use. You can use a wildcard to select multiple files.
3. The ratio error when selecting library objects with the same shape as the polygon. The ratio between the length and width of the polygon is compared with the ratio between the length and width of the library objects. If a polygon has a ratio 1.8 and the ratio error is 0.1, it means that all library objects with a ratio between 1.7 and 1.9 are considered as possible objects.
4. The scale threshold specifies that scaling that may be applied to the library object to fit in the polygon. If the scale that is needed to fit the polygon is outside this range, the library object is not considered.
5. The attribute type and category (name) of the new attribute that will be added to all features for which a library object was matched.
6. Value of the attribute to add.
7. Name of the attribute that will be added with the library object GUID.
8. Name of the attribute that will be added with the library object scale.
9. Name of the attribute that will be added with the library object heading.

**Example**

Match library objects from all BGL files in the libs folder to polygons with an area ratio of over 80%. Apply a ratio error of 0.1 and a scaling tolerance of 0.8 to 1.2. Feature that were matched get an attribute buildType=13. Attributes with the names GUID, SCALE and HDG are added to store the GUID, scale and heading of the library object.

```
MatchLibObjectToPolygon|FAREARAT>0.8|libs\*.bgl|0.1|0.8;1.2|Integer;buildType|13|
    ↪ GUID|SCALE|HDG
```

### 10.2.21 MergeGrid

#### Purpose

Merge all data from the different grid cells back into one cell. The main reason to do this, is because you want to export all data to a single file.

#### Arguments

None.

#### Example

Not applicable.

### 10.2.22 MergeRasterFeatures

#### Purpose

With this step two raster features can be merged into one new raster feature. You need to make sure that the two features you are merging have exactly the same size. This step can for example to used to create a 4 band raster feature with a NIR channel from a RGB and a CIR raster feature.

#### Arguments

1. Filter to select set A of raster features to use.
2. Filter to select set B of raster features to use.
3. Specification for 4 bands which raster band to use. Possible values are:
   - A# to select the #th band of raster A. Bands start counting from 0.
   - B# to select the #th band of raster B. Bands start counting from 0.
   - -1 to specify this band is not used.
4. The attribute type and category (name) of the new attribute
5. The value of the new attribute.

#### Example

Create a new raster with the attribute `type` and value `merged` by taking the RGB bands from raster A and the alpha band from raster B:

```
MergeRasterFeatures|FROMFILE="rasterA.tif"|FROMFILE="rasterB.tif"|A0;A1;A2;B3|
    ↪ String;type|merged
```

Create a new raster with the attribute `type` and value `merged` by taking the R and B bands from raster A and the G band from raster B. No alpha band is used:

```
MergeRasterFeatures|FROMFILE="rasterA.tif"|FROMFILE="rasterB.tif"|A0;B1;A2;-1|
    ↪ String;type|merged
```

Create a new 4 band raster feature with NIR channel with the attribute `type` and value `4band` by taking the RGB bands from raster A (RGB image) and the NIR band from raster B (CIR image):

```
MergeRasterFeatures|FROMFILE="rgb.tif"|FROMFILE="nir.tif"|A0;A1;A2;B0|String;type
    ↪ |4band
```

### 10.2.23 OffsetLine

**Purpose**

Replace a line feature by another line that has an offset perpendicular to the original line.

**Arguments**

1. Filter to select the lines that you want to offset.
2. The distance in meters of the offset.
3. Optional options, possible values are:
    - PARALLELREVERSE to create two offset lines instead of one, on both sides of the original line. The second one is reversed.

**Example**

Replace all primary road lines by a line offset 5 meter:

```
OffsetLine|highway="primary"|5
```

Replace all primary road lines by two line offset 5 meters on each side:

```
OffsetLine|highway="primary"|5|PARALLELREVERSE
```

### 10.2.24 PointToPolygon

**Purpose**

Turns point features into polygon feature. For example if you have point that represent light poles and you want to place autogen library object polygons at these points. To be able to identify the new polygons you can add a new attribute and value to the created features.

**Arguments**

1. Filter to select the line features that will be extruded.
2. The length and width the polygons should get in meters separated by a semicolon (;). It will first be checked if an attribute with this name exists, if so the value of that attribute is used, else the value entered is parsed.
3. The name of the attribute that contains the heading of the objects, you can also type in the heading value directly if all features need the same heading.
4. The heading offset to apply to each polygon on top of the heading provided by the argument above.
5. Attribute type and name of the new attribute that will be added.
6. Value of the new attribute that will be added.

**Example**

Create 8 by 8 meter polygons for point features of a light and give them an attribute polygons with the value lightpole:

```
PointToPolygon|FTYPE="POINT" And light="yes"|8;8|hdg|0|String;polygons|lightpole
```

Create 8 by 8 meter polygons for point features of a light, with a heading offset of 90 degrees and give them an attribute polygons with the value lightpole:

```
PointToPolygon|FTYPE="POINT" And light="yes"|8;8|hdg|90|String;polygons|lightpole
```

Create polygons for point features where the length and width are stored in the attribute LEN and WID and give them an attribute polygons with the value building:

```
PointToPolygon|FTYPE="POINT" And light="yes"|LEN;WID|hdg|0|String;polygons|
    ↪ building
```

### 10.2.25 PolygonToPoint

**Purpose**

Create points from polygons. For example to scatter trees inside a forest polygon or to place a point at the center of a house.

**Arguments**

1. Filter to select the polygons from which the points should be created.
2. The mode to use when placing the points, possible options are:
   - CENTER to place a point at the center of the polygon.
   - VERTEX to place a point at each vertex of the polygon.
   - INSIDE to place points inside the polygon.
3. The x and y distance between the points in degrees, separated by a semicolon (;) (only used in INSIDE mode).
4. The randomness applied to the point position in x and y direction, this is a value between 0 and 1. At 1 the randomness is maximum the distance specified. The values are separated by a semicolon (;) (only used in INSIDE mode).
5. The attribute type and name of the attribute that is added to the created point features.
6. The value that will be assigned to the attribute that is added to the created point features.
7. The attribute name of the attribute that will contain the heading of the point (not used in INISDE mode).
8. Optional options, possible values are:
   - INHERITPARENTATTR when all attributes of the polygon should be copied over to the created point feature.

**Example**

Place a point at the center of each building:

```
PolygonToPoint|type="building"|CENTER|0.0;0.0|0.0;0.0|String;type|buildcen|hdg
```

Place a point at each vertex of each lake:

```
PolygonToPoint|type="lake"|VERTEX|0.0;0.0|0.0;0.0|String;type|lakepos|hdg
```

Place tree points in all forest polygons:

```
PolygonToPoint|type="forest"|INSIDE|0.001;0.001|0.25;0.0|String;type|tree|hdg
```

### 10.2.26 ProcessElevationRaster

#### Purpose

Process elevation rasters to generate derived data from them. For example calculate the slope from an elevation raster. Figure 10.9 shows an example elevation raster that is used to illustrate the different processing that are available in the arguments section.

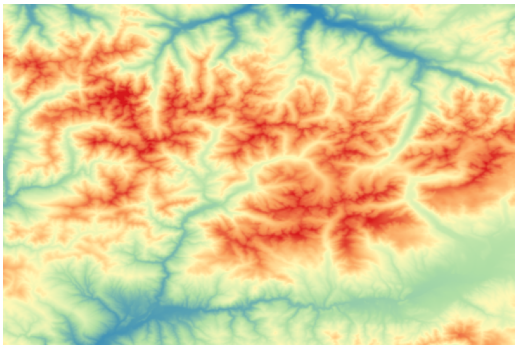
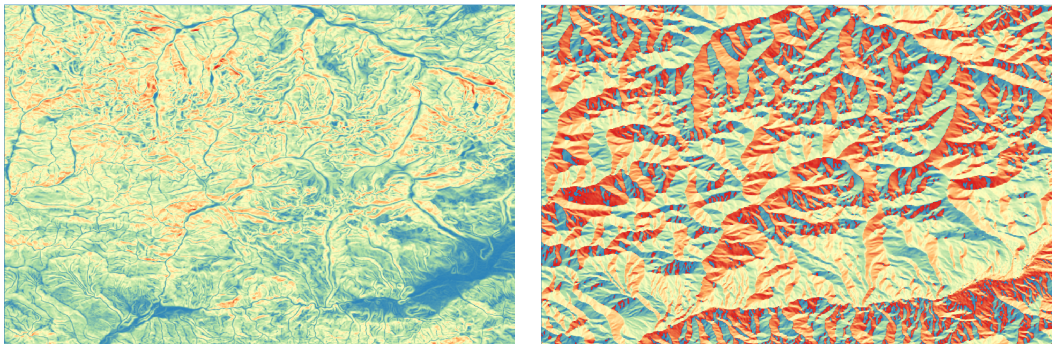


Figure 10.9: Example elevation data from the country of Andorra, used to illustrate the different processing available

#### Arguments

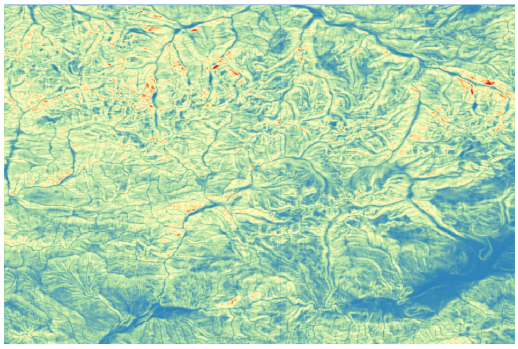
1. Filter to select the elevation raster features
2. The operation to perform on the elevation raster, possible values are:
    - SLOPE calculates the slope of the elevation raster, see Figure 10.10a for an example of the output.
    - ASPECT calculates the aspect of the elevation raster, see Figure 10.10b for an example of the output.
    - ROUGHNESS calculates the roughness of the elevation raster, see Figure 10.10c for an example of the output.
    - HILLSHADE calculates a hillshaded representation of the elevation raster, see Figure 10.10d for an example of the output.
    - TRI calculates the Terrain Roughness Index (TRI) of the elevation raster, see Figure 10.10e for an example of the output.
    - TPI calculates the Terrain Position Index (TPI) of the elevation raster, see Figure 10.10f for an example of the output.
3. The attribute type and category (name) of the attribute added to the processed elevation raster
4. The value of the attribute added to the processed elevation raster
5. Optional argument, when provided the generated raster feature uses the byte datatype. By default many of the calculated rasters use a float datatype. If you want to use the raster data in a texture filter you need to convert it to byte first. You can specify a scaling and offset for this conversion. For example the ASPECT raster map will have values between 0 and 360 degrees. By providing a scale of 0.7 the value gets in the range of 0 to 252. The TRI and TPI indices can also be negative. Let's assume they are in a range of -50 to 50. By using a scale of 2 and an offset of 128, they get in the range of 28 to 228 and therefore fit in a byte again.



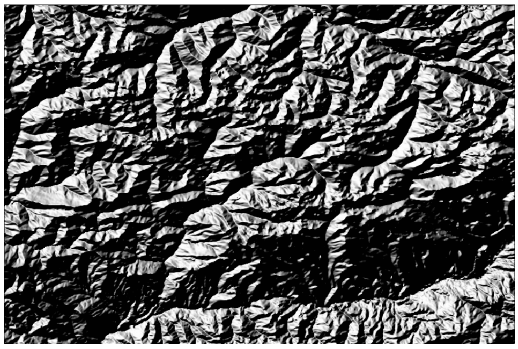
(a)



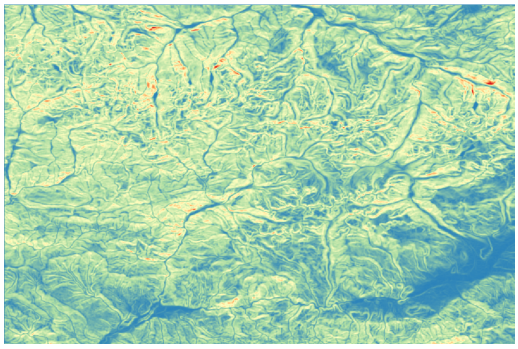
(b)



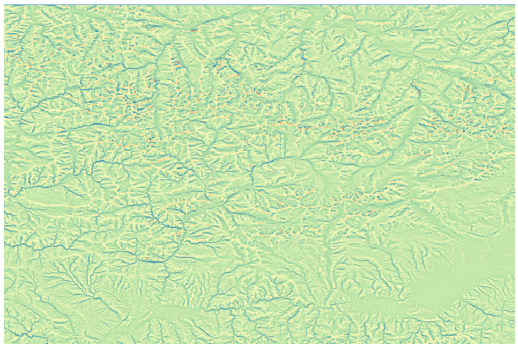
(c)



(d)



(e)



(f)

Figure 10.10: Output of the different processing options for the sample elevation raster. (a) Slope, (b) Aspect, (c) Roughness, (d) Hillshade, (e) TRI and (f) TPI.

#### Example

Calculate the slope of an elevation raster:

```
ProcessElevationRaster|FTYPE="RASTER"|SLOPE|String;type|slope
```

Calculate the TRI of an elevation raster and convert the result to a byte raster with scaling 2 and offset 128:

```
ProcessElevationRaster|FTYPE="RASTER"|TRI|String;type|tri|2;128
```

### 10.2.27 RasterToPolygon

#### Purpose

Create polygon features based on raster data. This steps create a polygon based on threshold values specified for the raster. If your raster has the data type byte you can use the DetectFeature step instead, see section 10.2.13, that step gives you much more control over which parts of the raster are turned into a polygon. Use the RasterToPolygon step for non-byte raster sets.

#### Arguments

1. Filter to select the raster features to create the polygons from.
2. The minimum and maximum threshold values of the raster used for creating the polygons.
3. Attribute type and name of the new attribute that will be added.
4. Value of the new attribute that will be added.

#### Example

Create a polygon for the area where the elevation data is between 1000 and 1500 meter, so that this polygon can be used to generate a different tree type in your autogen.

```
RasterToPolygon|FTYPE="RASTER" And FROMFILE="elev.tif"|1000;1500|String;treeType|
    ↪ pine
```

Create a polygon for the area where the population density is between 0 and 10, so that you can remove certain features in this area.

```
RasterToPolygon|FTYPE="RASTER" And FROMFILE="population_density.tif"|0;10|String;
    ↪ popdens|low
```

### 10.2.28 RemoveAttribute

#### Purpose

Remove the given attributes from features. This can be used to free memory by removing attributes from the loaded data that will not be used.

#### Arguments

- The names of the attributes that should be removed, if multiple names are provided separate them by a semicolon (;)

#### Example

Remove the attributes `date` and `creator` from all data:

```
RemoveAttribute|date;creator
```

### 10.2.29 ReplacePolygonByBuildingRectangles

#### Purpose

Autogen buildings in Flight Simulator can only be rectangular in shape. That means that more complex buildings, for example a L-shaped building, can be be represented realistically. Either part of the building would be missing or the resulting building is much too big for the original footprint. The purpose of this step is to replace such footprints by a number of rectangles. Together these rectangles will represent the shape of the footprint better.

#### Arguments

1. Filter to select the polygons that should be replaced by rectangles
2. Three parameters that determine which rectangles are acceptable. The first parameter is the minimum area ratio, the second the minimum area and the third the minimum width.
3. The weights that will be used when selecting edges to slice. Three values should be provided. These are the weights for the length, offset and parallel factor.
4. Attribute type and category (name) of the attribute that will be added to successfully processed polygons.
5. Value of the attribute that will be added to successfully processed polygons.

#### Example

The line below will select all poylgons that have a attribute BUILDCAT with a value of 3 and try to replace them by rectangles. The resulting rectangles should have a minimum area ratio of 0.8, a minimum area of 4 square meters and a minimum width of 4 meter. Successfully processed polygons will get an attribute BUILDCAT with the value of 2.

```
ReplacePolygonByBuildingRectangles|BUILDTYPE=3|0.8;4;4|0.5;2.0;1.0|Integer;
    ↪ BUILDTYPE|2
```

#### Algorithm explanation

In this section it will be described in detail how the algorithm that replaces polygons by rectangles works. Based on this information you should be able to select the right parameters for this step. The building footprint shown in Figure 10.11 will be used as example to explain the algorithm.

It is important to remember that the algorithm works best on polygons that have many parallel edges, edges at 90 degree angles and do not have very weird shapes. So try to select such polygons before running this step. Else the results might not be what you are looking for.

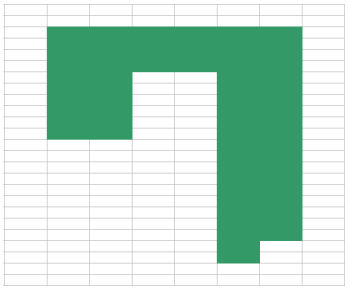


Figure 10.11: Example of complex building footprint

The basic approach of the algorithm is that it will slice the polygon along one of the edges. Each of the resulting polygons will than be checked if they are acceptable or if they need to be sliced

further. This is done by checking of the area ratio (the ratio between the area of the polygon and the area of the minimum area rectangle that fits around the polygon) of the resulting polygons is above the threshold specified in the arguments of the step. Figure 10.12 shows an edge that could be selected as the first edge to slice the example polygon around.

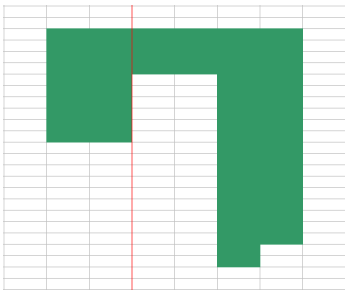


Figure 10.12: First edge to slice polygon

Figure 10.13 shows the resulting two polygons if the polygon is sliced around this edge. As you can see the left polygon is a rectangle, so the area ratio will be 1.0. This polygon is accepted as a valid rectangle. The polygon on the right is still L-shaped and has an area ratio lower than the threshold of 0.8 that is used in the example. So for this polygon the process of splitting it will continue. Figure 10.14 shows a next edge around which the feature could be split.

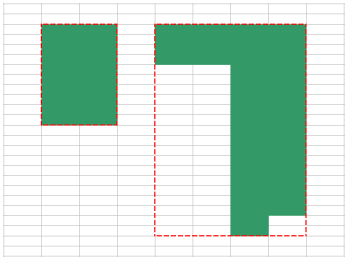


Figure 10.13: Resulting polygons

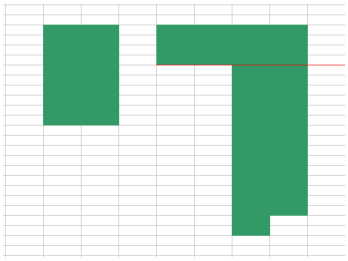


Figure 10.14: Second edge to slice polygon

Figure 10.15 shows the two new polygons that result from splitting the L-shaped polygon around the selected edge. The top polygon is a rectangle again, so that one has an area ratio above the threshold. The bottom polygon does have an area ratio lower than 1.0, but in this case it will be higher than the threshold of 0.8. So that means that this polygon is also accepted. The autogen building being generated will be slightly bigger than the polygon, due to the snap gap at the bottom, but that's acceptable. So this means we have finished splitting the complex building footprint into three rectangles and with these three we can generate our autogen buildings.

The complicate matters a bit more, some additional filtering can be done after checking the area ratio of the polygons that result from slicing. A minimum area and minimum width parameters can also be provided for the step. Any resulting polygon that has an area or width lower than these values will be dropped and not included in the resulting polygons. This is mainly useful to

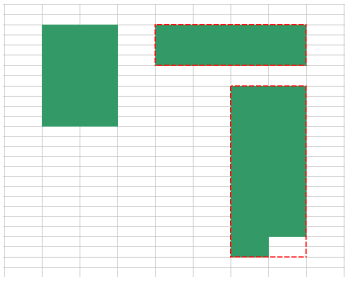


Figure 10.15: Final polygons

prevent very narrow slices or small rectangles being included. These would not result in realistic autogen objects.

In the above explanation one topic has not yet been discussed. I just drew an edge around which the polygon was split, but how is this edge selected? The algorithm will check all edges of the polygon and select the best one based on a score that is assigned to each edge. This score is based on three contributions and in the arguments you can specify a weight factor that determines how heavy each contributions adds to the final score. The following three contributions are taking into account:

1. The length of the edge. This is expressed as a value between 0.0 and 1.0. So the length of the edge is divided by the length of the polygon (in the direction of the edge). In general longer edges are a better candidate to slice around, since these are the more characteristic edges of the polygon.
2. The distance of the edge to the center of the polygon also contributes to the score. This is once again expressed as a value between 0.0 and 1.0. Here 1.0 means that the edge is at the center and 0.0 means that the edge is at the outside of the polygon.
3. The last contribution is the ratio of the distance between the closed edges that are parallel to the selected edge. This contribution is used to give an extra penalty to edge that would result in slicing the polygon in two narrow polygons of equal size. Such edges get a value of 0.0, while edges that result in polygons with different width get a higher value. These in general give more resulting polygons for autogen.

For all polygons that are successfully extracted a new attribute is added with the category and value provided. This means you can easily select them while creating the autogen buildings later on. Features that were not processed correctly remain their original attribute. A feature could for example fail to process correctly when slice around an edge doesn't return multiple polygons. To prevent an endless loop the algorithm is then stopped.

### 10.2.30 ReplacePolygonByVegetationRectangles

#### Purpose

For FS2004 autogen vegetation can only be rectangular in shape. For FSX there is a choice between polygonal and rectangular vegetation. With this step it is possible to replace a polygon by a set of rectangles that cover roughly the same area. So this allows rectangular vegetation to be created from polygons that are not rectangular. Figure 10.16 shows an example of this.

#### Arguments

1. Filter to select the polygons that should be replaced by rectangles
2. The size of the rectangle that will be created in degrees
3. The amount of overlap between rectangles, where 0.0 means no overlap and 1.0 means 100% overlap.

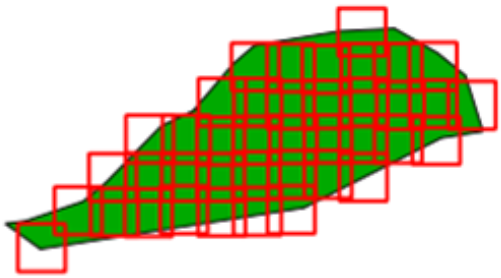


Figure 10.16: Example of polygon replaced by a number of rectangles

4. The amount of random offset added to the rectangles, 0.0 means no random offset while 1.0 means a maximum random offset equal to the size of the rectangle.

**Example**

Replace all forest polygons by rectangles 0.001 degrees in size. The rectangles will have 25% overlap and 25% random offset:

```
ReplacePolygonByVegetationRectangles|landuse="forest"|0.001|0.25|0.25
```

### 10.2.31 ScaleFeature

**Purpose**

Scale the selected polygons features, this can for example be used to reduce the size of all building footprints if the autogen buildings appear to big.

**Arguments**

1. The filter that selects the features to be scaled
2. The factor to scale the features with

**Example**

Scale all buildings with a length less than 10 meter by a factor of 0.9:

```
ScaleFeature|type="building" And FLENGTH<10|0.9
```

### 10.2.32 SimplifyFeature

**Purpose**

Simplify the selected features. By making them less complex, steps like AddAttributeIfInside will run faster on them. See 10.17 for an example how a simplified feature can look.

**Arguments**

1. The filter that selects the features to be simplified
2. The tolerance factor used for the scaling, this is a tolerance in degrees so use small values

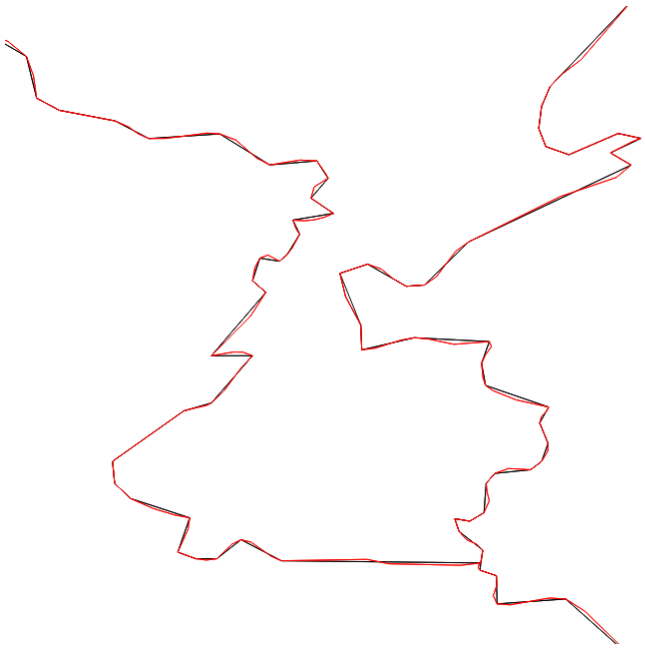


Figure 10.17: Example of simplification with tolerance 0.005 of country borders, red are the original borders, black the simplified ones

#### Example

Simplify all features with a tolerance of 0.005:

```
SimplifyFeature|*|0.001
```

### 10.2.33 SplitGrid

#### Purpose

Splits the loaded features into grid cells with the specified size. Autogen file have a fixed grid size, so therefore the data needs to be split. Another reason to split the data into multiple cells is that it allows parallel processing and therefore can improve performance.

#### Arguments

1. The grid size to be used for splitting the data, the size should be provided in degrees. If one number is specified a square cell is created. Else two numbers can be provided, which should be separated by a semicolon (;). For the grid sized used by FS autogen you can specify AGN. To get a specific FS terrain LOD grid size you can specify LOD8, LOD10, LOD13, etc. To get a specific AF2 terrain level grid size you can specify AF2LEV8, AF2LEV10, etc.

2. Coordinates of the bounding rectangle of the area that should be processed. If specified only features within this area will be split into grid cells. This can be useful if you don't want to process all loaded data, for example in batch runs. If you provide a * all features will be processed. The coordinates should be provided as latitude and longitude. The coordinates are specified as: `minlon;maxlon;minlat;maxlat`.

3. Optional filter to specify which features should not be split by the grid. These features will be put in the grid cell where the centre of the feature lies. This is for example useful for buildings, because if they are split into multiple polygons their shape can't be represented realistically anymore.

4. Optional buffer to be applied, this can be used to also include features in a gridcell that are within the buffer distance outside of it.

**Example**

Split the loaded features into grid cells with the right size for autogen:

```
SplitGrid|AGN|*
```

Split the loaded features into grid cells of 0.2 x 0.2 degrees in size:

```
SplitGrid|0.2|*
```

Split the loaded features into grid cells of 0.2 x 0.25 degrees in size:

```
SplitGrid|0.2;0.25|*
```

Split the loaded features into grid cells with the right size for autogen, but exclude building features from being split:

```
SplitGrid|AGN|*|building="*"
```

Split the loaded features that are between N51 E5 and N52 E6 into grid cells with the right size for autogen:

```
SplitGrid|AGN|5;6;51;52
```

Split the loaded features into grid cells with the size of FS LOD10:

```
SplitGrid|LOD10|*
```

Split the loaded features into grid cells with the size of AF2 level 10:

```
SplitGrid|AF2LEV10|*
```

### 10.2.34 UnloadFeatures

**Purpose**

Unload the selected features from memory. This can be used to free up memory for features that you will not use in the remainder of your configuration file.

**Arguments**

1. The filter that selects the features to be removed

**Example**

Unload all forest features:

```
UnloadFeatures|type="forest"
```

### 10.2.35 UnloadXMLObjects

**Purpose**

Unload all XML objects from memory. This can be used to free up memory for XML objects that you will not use in the remainder of your configuration file or to start creating a new BGL file with different objects afters you have exported the first one.

#### Arguments

None.

#### Example

Not applicable.

## 10.3 Create autogen

All the steps described in this section are used to create autogen, which can then be exported as a AGN file for use in Flight Simulator.

### 10.3.1 CreateAGNGenBuild

#### Purpose

Create generic autogen buildings from the selected polygonal features. Autogen generic buildings are always rectangular in shape, therefore they are created from the best fitting bounding rectangle around the selected feature.

The texture that is used for the autogen buildings can be set per autogen tile. This is done with the SetAGNBuildingTexture step. If no texture is set the default texture for the area is used by FSX.

When the autogen buidlings are viewed in the Annotator of the SDK they might appear to have the wrong heading. Depending on the latitude of your scenery, Annotator might show the wrong heading, but in Flight Simulator the buildings will appear perfectly aligned with your photo scenery.

#### Arguments

1. Filter to select the features from which the buildings should be generated
2. GUID of the roof (group) that should be used for the buildings
3. Optional options, the following options can be provided for this step:
    - MAXRATIO=## specifies that the building should be split into multiple buildings when the ratio between the length and width is bigger than the specified value. This can be useful to split long rows of houses that are represented as one polygon into multiple buildings.
4. Optional feature attribute name that contains the roof GUID. If the attribute does not existing the GUID provided in the second argument is used as fallback.

#### Example

Create autogen buildings from all building polygons and assign the roof group named "Roofs Gabled _ALL_".

```
CreateAGNGenBuild|building="*"|{5ae04eb6-934c-4f63-bb48-5e7dee601212}
```

Create autogen from all building polygons that have an area ratio of over 70%. Assign them the roof group named "Roofs Gabled _ALL_". If the ratio of the length and width is greater than 2 the building will be split into multiple buildings.

```
CreateAGNGenBuild|building="*" And FAREARAT>0.7|{5ae04eb6-934c-4f63-bb48-5
    ↪ e7dee601212}|MAXRATIO=2
```

Create autogen buildings from all building polygons and use the roof GUID from the attribute roofGuid:

```
CreateAGNGenBuild|building="*"|{5ae04eb6-934c-4f63-bb48-5e7dee601212}||roofGuid
```

### 10.3.2 CreateAGNLibObject

**Purpose**

Create autogen library objects from polygon features. For example to place light poles or churches as autogen objects. The library objects are called from the classes that are defined in the default.xml file of FS.

**Arguments**

1. Filter to select the polygon features from which the library objects will be generated
2. GUID of the object class to use. If you provide multiple GUIDs, separated by a semicolon (;), scenProc will randomly select one of the GUIDs from the list.
3. Optional feature attribute name that contains the object class GUID. If the attribute does not existing the GUID provided in the second argument is used as fallback.

**Example**

Create library objects using the wooden electricity pole object class:

```
CreateAGNLibObject|OBJECT="ELECPOLE"|{960b0c53-4824-d848-e7ea-3e9475d18c29}
```

### 10.3.3 CreateAGNPolyBuild

**Purpose**

Create autogen polygonal buildings. This type of building is rarely used, since the outline polygon should always be extruded. That makes it only suitable for buildings with a fixed width and a courtyard in the middle. This is a limitation of the FSX autogen engine.

**Arguments**

1. Filter to select the features from which the polygonal building will be generated
2. GUID of the building style. If you provide multiple GUIDs, separated by a semicolon (;), scenProc will randomly select one of the GUIDs from the list.
3. Extrusion width for the building in autogen coordinates
4. Optional: when a fourth attribute is provided it means that the building polygon is open instead of closed

**Example**

Create autogen polygonal buildings from all features marked as type polybuild with an extrusion depth of 0.01. Assign the building group named "3 Story Flat Roof Mixed".

```
CreateAGNPolyBuild|type="polybuild"|{f707c26e-32ed-4f15-85a2-81c5bc9f8b32}|0.01
```

### 10.3.4 CreateAGNPolyVeg

**Purpose**

Create autogen polygonal vegetation. This is usually used to create forest or other vegetation. The FS autogen configuration files determine which objects are shown within the polygon.

**Arguments**

1. Filter to select the features from which the polygonal vegetation will be generated
2. GUID of the vegetation group that will be used. If you provide multiple GUIDs, separated by a semicolon (;), scenProc will randomly select one of the GUIDs from the list.
3. Optional feature attribute name that contains the vegetation type GUID. If the attribute does not existing the GUID provided in the second argument is used as fallback.

**Example**

Create autogen polygonal vegetation from all forest polygons and assign the vegetation group named "Mixed Forest".

```
CreateAGNPolyVeg|landuse="forest"|{c9dc45ae-f240-42a9-a137-b7617452a308}
```

### 10.3.5 CreateAGNRectVeg

**Purpose**

Create autogen rectangular vegetation. This is usually used to create forest or other vegetation for FS2004. For FS2004 the type of vegetation is determined with the SetAGNVegetationSettings step. For FSX the type of vegetation is selected with a GUID that refers to a class in the autogen configuration files. The bounding box of the selected features is used as the rectangle of the autogen.

**Arguments**

1. Filter to select the features from which the rectangular vegetation will be generated
2. GUID of the vegetation group that will be used. If you provide multiple GUIDs, separated by a semicolon (;), scenProc will randomly select one of the GUIDs from the list.
3. Optional feature attribute name that contains the vegetation type GUID. If the attribute does not existing the GUID provided in the second argument is used as fallback.

**Example**

Create autogen rectangular vegetation from all forest polygons and assign the vegetation group named "Mixed Forest".

```
CreateAGNRectVeg|landuse="forest"|{c9dc45ae-f240-42a9-a137-b7617452a308}
```

Create autogen rectangular vegetation for FS2004 where the GUID is not required.

```
CreateAGNRectVeg|landuse="forest"
```

### 10.3.6 CreateAGNRowHouse

**Purpose**

Create autogen row houses from the selected polygonal features. Autogen row houses are always rectangular in shape, therefore they are created from the best fitting bounding rectangle around the selected feature. The height and roof type of the row houses can't be influenced.

The texture that is used for the autogen row houses can be set per autogen tile. This is done with the SetAGNRowHouseTexture step. If no texture is set the default texture for the area is used by FSX.

When the autogen row houses are viewed in the Annotator of the SDK they might appear to have the wrong heading. Depending on the latitude of your scenery, Annotator might show the wrong heading, but in Flight Simulator the row houses will appear perfectly aligned with your photo scenery.

**Arguments**

1. Filter to select the features from which the row houses should be generated

**Example**

Create autogen row houses from all building polygons.

```
CreateAGNRowHouse|building="*"
```

### 10.3.7 SetAGNBuildingHeight

**Purpose**

Set the building heights of autogen buildings for this autogen tile. Be aware that the heights can not be set per building, but only per entire autogen tile (around 1x1 kilometer in size).

For four different height categories it can be specified how likely they are to appear. These categories are:

- Buildings with 1 to 2 floors
- Buildings with 3 to 5 floors
- Buildings with 6 to 8 floors
- Buildings with 9 to 12 floors

Exactly which height of building will appear in Flight Simulator also depends on the size of your autogen buildings. For example for small buildings only buildings with 1 or 2 floors will appear, even if you set the buildings heights to only 9 to 12 floor buildings.

When you don't add this step to your configuration file only buildings with 1 or 2 floors will be generated in your autogen.

**Arguments**

1. Specify the filters for the features to test the center of the autogen tile against. Only if the center is within one of these features the specified buildings heights will be assigned. Use * to assign the building heights to all autogen tiles.
2. A semi-colon separated list of 4 values that specify how likely each height category is to appear. The values are between 0.0 and 1.0.

**Example**

Assign only 1 or 2 floor buildings to all autogen tiles:

```
SetAGNBuildingHeight|*|1.0;0.0;0.0;0.0
```

Assign a mixture of buildings heights to all autogen tiles:

```
SetAGNBuildingHeight|*|0.7;0.4;0.3;0.1
```

Assign highrise buildings to all cells marked as downtown area:

```
SetAGNBuildingHeight|type="downtown"|0.0;0.0;0.6;1.0
```

### 10.3.8 SetAGNBuildingTexture

**Purpose**

Set the custom autogen building texture that should be used for this autogen tile. Be aware that the texture can not be set per building, but only per entire autogen tile (around 1x1 kilometer in size). See the SDK for the required layout of this texture sheet.

The custom autogen texture should be placed in the main texture folder of Flight Simulator. It is also required that a night texture is present. If the day texture is called myAGNBuildings.dds, the night texture should be named myAGNBuildingsLM.dds.

**Arguments**

1. Specify the filters for the features to test the center of the autogen tile against. Only if the center is within one of these features the specified texture will be assigned. Use * to assign the texture to all autogen tiles.
2. Base name of the texture, this is the name of the day texture without extension.

**Example**

Assign a custom building texture to all autogen tiles:

```
SetAGNBuildingTexture|*|myAGNTexture
```

Assign the custom building texture only to autogen tiles that fall within a residential polygon:

```
SetAGNBuildingTexture|landuse="residential"|myResidentialTexture
```

### 10.3.9 SetAGNRowHouseTexture

**Purpose**

Set the custom row house texture that should be used for this autogen tile. Be aware that the texture can not be set per building, but only per entire autogen tile (around 1x1 kilometer in size). See the SDK for the required layout of this texture sheet.

The custom row house texture should be placed in the main texture folder of Flight Simulator. It is also required that a night texture is present. If the day texture is called myAGNRowHouse.dds, the night texture should be named myAGNRowHouseLM.dds.

**Arguments**

1. Specify the filters for the features to test the center of the autogen tile against. Only if the center is within one of these features the specified texture will be assigned. Use * to assign the texture to all autogen tiles.
2. Base name of the texture, this is the name of the day texture without extension.

**Example**

Assign a custom row house texture to all autogen tiles:

```
SetAGNBuildingTexture|*|myAGNTexture
```

Assign the custom row house texture only to autogen tiles that fall within an old town polygon:

```
SetAGNBuildingTexture|type="old_time"|myOldTownTexture
```

### 10.3.10 SetAGNVegetationSettings

**Purpose**

This step sets the FS2004 vegetation settings for an autogen tile. Be aware that these settings can only be set per autogen tile (around 1x1 kilometer in size) and not per vegetation rectangle. Also be aware that these settings only apply for FS2004. For FSX the type of vegetation is set through the vegetation GUID, whereas in FS2004 these settings are used to specify the two types of vegetation in a tile and their distribution.

**Arguments**

1. Specify the filters for the features to test the center of the autogen tile against. Only if the center is within one of these features the vegetation settings will be assigned. Use * to assign the vegetation settings to all autogen tiles.
2. The vegetation class of the first vegetation type to use. Possible values are:
   - None
   - Deciduous
   - Coniferous1
   - Coniferous2
   - Shrub wet
   - Shrub dry
   - Palm trees
3. The vegetation class of the second vegetation type to use. See above for possible values.
4. The percentage of vegetation class 1. For example 1.0 means 100% class 1 and 0% class 2, while 0.3 means 30% class 1 and 70% class 2.
5. The density of the vegetation. This is a value between 0.0 and 1.0, where 1.0 means the most dense vegetation.
6. A semi-colon separated list of the minimum height and the maximum height of vegetation class 1.
7. A semi-colon separated list of the minimum height and the maximum height of vegetation class 2.

**Example**

Assign all autogen tiles a 50-50 mix of deciduous and coniferious1 vegetation, with a 75% density and a vegetation height between 10 and 20 meters:

```
SetAGNVegetationSettings|*|1|2|0.5|0.75|10;20|10;20
```

Assign a mix of wet and dry scrubs to tiles that are within an area with scrubs:

```
SetAGNVegetationSettings|landuse="scrubs"|4|5|0.75|0.5|3;6|2;5
```

## 10.4 Create XML scenery

In this section all steps to create BGLComp XML scenery objects are described. These objects can be exported to BGL files later on.

### 10.4.1 Create3DBuilding

**Purpose**

Create a 3D building model from the footprint polygon. For more information about the 3D buildings that scenProc can create see chapter 7.

**Arguments**

1. Filter to footprint polygon features.
2. The name of the attribute that contains the height of the building. If the feature doesn't have this attribute the value of this argument is parsed as height value. If two values are provided, separated by a semicolon, a random value between those two values is used.
3. The name of the attribute that contains the altitude of the building. If the feature doesn't have this attribute the value of this argument is parsed as altitude value.
4. The roof type to use for the building. Supported values are:
   - FLAT
   - HIPPED
   - GABBLED
   - MANSARD
   - MANSARD_GABLED
   - MANSARD_HALFGABLED
   - PARAPETS
5. The name of the attribute that contains roof options 1, see 10.3 for an overview of options per roof type and 10.4 for a description of the options. If the feature doesn't have this attribute the value of this argument is parsed as slope value. If two values are provided, separated by a semicolon, a random value between those two values is used.
6. The name of the attribute that contains roof options 2, see 10.3 for an overview of options per roof type and 10.4 for a description of the options. If the feature doesn't have this attribute the value of this argument is parsed as slope value. If two values are provided, separated by a semicolon, a random value between those two values is used.
7. The name of the attribute that contains roof options 3, see 10.3 for an overview of options per roof type and 10.4 for a description of the options. If the feature doesn't have this attribute the value of this argument is parsed as slope value. If two values are provided, separated by a semicolon, a random value between those two values is used.
8. The name of the attribute that contains roof options 4, see 10.3 for an overview of options per roof type and 10.4 for a description of the options. If the feature doesn't have this attribute the value of this argument is parsed as slope value. If two values are provided, separated by a semicolon, a random value between those two values is used.

| **Roof type** | **Option 1** | **Option 2** | **Option 3** | **Option 4** |
|---|---|---|---|---|
| FLAT | - | - | - | - |
| HIPPED | slope | overhang | - | - |
| GABLED | slope | overhang | - | - |
| MANSARD | slope | overhang | slop2 | slope2 distance |
| MANSARD_GABLED | slope | overhang | slop2 | slope2 distance |
| MANSARD_HALFGABLED | slope | overhang | slop2 | slope2 distance |
| PARAPETS | height | depth | width | texture type |

Table 10.3: Roof options per roof type

| Roof Option | Description |
|---|---|
| slope | The slope angle of the roof in degrees. |
| overhang | The distance of the overhang of the roof in meters. |
| slope2 | The secondary slope angle of the roof in degrees. |
| slope2 distance | The distance from which the secondary slope is applied in meters. |
| height | The distance from the top of the building wall to the top of the parapets in meters. |
| depth | The distance from the top of the parapets to the roof in meters. |
| width | The width of the parapets in meters. |
| texture type | The texture type to apply to the parapets. Either ROOF when the roof texture of the building is applied or WALL when the wall texture of the building is applied. |

Table 10.4: Descriptions of the roof options

9. The modifications to apply, so you specify multiple values separated by a semicolon. Possible values are:
   - NONE
   - EXTRA_GABLE_RECTANGLE, with this modification a cross gabled roof is created within the rectangular footprint
   - EXTRA_GABLE_LSHAPE, with this modification 3 gables are created for an L-shaped footprint
   - ADD_CHIMNEY adds a chimney to the building
   - ADD_DORMER_FLAT adds a dormer with a flat roof to the building
   - ADD_DORMER_SLOPED adds a dormer with a sloped roof to the building
   - ADD_DORMER_HIPPED adds a dormer with a hipped roof to the building
   - ADD_DORMER_GABLED adds a dormer with a gabled hipped roof to the building
10. Optional, filename of the wall building texture configuration file to use. This configuration specifies how the texture mapping of the walls should be done. See section 7.4 for more details.
11. Optional, the name of the attribute that contains the wall group to use. The building texture configuration can contain multiple groups. If you specify a * the wall texture will be selected from any of the groups.
12. Optional, filename of the roof building texture configuration file to use. This configuration specifies how the texture mapping of the roofs should be done. See section 7.4 for more details.
13. Optional, the name of the attribute that contains the roof group to use. The building texture configuration can contain multiple groups, for example for differently colored roofs. If you specify a * the wall texture will be selected from any of the groups.
14. Optional options, the following options can be provided for this step:
    - NOAUTOGENSUPPRESSION to disable suppression of autogen by the object
    - NOCRASH to disable crash detection by the object
    - NOSHADOW to disable the display of shadows by the object
    - NOFOG to disable fog affecting to the object
    - NOZWRITE to disable Z writing of the object
    - NOZTEST to disable Z testing of the object

- ALTISMSL when the altitude of the object should be considered MSL and not AGL
- SNAPTOGROUND when the object should be snapped to the ground (MSFS only)
- SNAPTONORMAL when object attitude should be snapped to the terrain normal (MSFS only)
- COMPLEXITY_SPARSE to specify that the buildings should get image complexity sparse.
- COMPLEXITY_NORMAL to specify that the buildings should get image complexity normal.
- COMPLEXITY_DENSE to specify that the buildings should get image complexity dense.
- COMPLEXITY_VERY_DENSE to specify that the buildings should get image complexity very dense.
- COMPLEXITY_EXTREMELY_DENSE to specify that the buildings should get image complexity extremely dense.
- WRITEDEBUGOUTPUT_BBOX_ERROR to specify a debug file should be written when buildings have an unexpected bounding box.
- WRITEDEBUGOUTPUT_ERROR to specify a debug file should be written when an error occurs when generating the building.
- WRITEDEBUGOUTPUT_MOD_ERROR to specify a debug file should be written when the footprint modification results in an error.
- WRITEDEBUGOUTPUT_LAST to specify a debug file should be written for each object when processed.

**Example**

Create an untextured building with a height of 6 meter with a flat roof:

```
Create3DBuilding|building="*"|6|0|FLAT|0|0|0|0|NONE
```

Create an untextured gabled building with a height between 3 and 6 meters and a slope between 30 and 40 degrees:

```
Create3DBuilding|building="*" And FNUMVERT=4|3:6|elev|GABLED|30:40|0|0|0|NONE
```

Create an untextured gabled building with a height between 3 and 6 meters. The slope is read from the slope attribute of the feature and the overhang from the overhang feature:

```
Create3DBuilding|building="*" And FNUMVERT=4|3:6|elev|GABLED|slope|overhang|0|0|
    ↪ NONE
```

Create an untextured mansard building with a height of 6 meter, a slope between 40 and 50 degrees, a secondary slope between 20 and 25 degrees and a distance of 2 meters where the secondary slope starts:

```
Create3DBuilding|building="*" And FNUMVERT=4|6|elev|MANSARD|40:50|0|20:25|2|NONE
```

Create an untextured parapets building with a height of 6 meter, a parapets height of 1.5 meter, a depth of 1 meter and a width of 0.3 meter:

```
Create3DBuilding|building="*" And FNUMVERT=4|6|elev|PARAPETS|1.5|1.0|0.3|0|NONE
```

Create an textured building with a height of 6 meter with a gabled roof with a slope between 30 and 40 degrees. Using the specified building texture configuration files for the walls and roof:

```
Create3DBuilding|building="*" And FNUMVERT=4|6|0|GABLED|30;40|0|0|0|NONE|
    ↪ wallTextureConfig.btc|*|roofTextureConfig.btc|*
```

Create an textured building with a height of 6 meter with a parapets roof with a height between 1.5 and 2 meter, a depth between 1 and 1.5 meter, a width between 0.3 and 0.5 meter and with the parapets using the roof texture. Using the specified building texture configuration files for the walls and roof:

```
Create3DBuilding|building="*" And FNUMVERT=4|6|0|PARAPETS
    ↪ |1.5;2.0|1.0;1.5|0.3;0.5|ROOF|NONE|wallTextureConfig.btc|*|
    ↪ roofTextureConfig.btc|*
```

Create an textured building with a height of between 3 and 6 meter with a gabled roof with a slope between 30 and 40 degrees. Using the specified building texture configuration files for the walls and roof. The roof texture is selected from the group dark_roofs:

```
Create3DBuilding|building="*" And FNUMVERT=4|3;6|0|GABLED|30;40|0|0|0|NONE|
    ↪ wallTextureConfig.btc|*|roofTextureConfig.btc|dark_roofs
```

Create a building with gables and the modification for an additional cross-gable:

```
Create3DBuilding|building="*" And FNUMVERT=4|height|elev|GABLED|30|0|0|0|
    ↪ EXTRA_GABLE_RECTANGLE|wallTextureConfig.btc|*|roofTextureConfig.btc|*
```

Create a building with a chimney and a dormer with a flat roof:

```
Create3DBuilding|building="*" And FNUMVERT=4|height|elev|HIPPED|40|0|0|0|
    ↪ ADD_CHIMNEY;ADD_DORMER_FLAT|wallTextureConfig.btc|*|roofTextureConfig.btc
    ↪ |*
```

### 10.4.2 CreateXMLBridge

#### Purpose

Create XML extrusion bridges from line features.

#### Arguments

1. Filter to select the line features from which the bridges will be generated
2. Filter to select the line features that are used for the ramp of the bridge
3. The width of the bridge in meters
4. The name of the attribute that contains the height of the bridge
5. The additional scaling to apply to the height value
6. The slope of the ramp of the bridge as percentage, a value of 1.0 means 45 degrees ramp
7. The GUID of the extrusion profile to use for the bridge
8. The GUID of the material set to use for the bridge
9. The GUID of the pilar objects of the bridge
10. Optional options, possible values are:
    - NOPLATFORM to prevent the bridge from getting platforms for traffic

**Example**

Create a bridge for all motorway features, using motorway features for the ramp as well:

```
CreateXMLBridge|highway="motorway" AND bridge="*"|highway="motorway"|12|layer
    ↪ |6|0.1|8aaabdf2-4496-483a-8df1-86ea13aa950d|7ed5603a-d17a-4246-bc3d-
    ↪ baadea6909fa|bef5ea7f-4a47-4e35-8a6e-c7eb78e6ff0e
```

### 10.4.3 CreateXMLCarParking

**Purpose**

Create XML car parking objects from polygon features (for MSFS only).

**Arguments**

1. Filter to select the polygon features from which the car parkings will be generated
2. Optional options, the following options can be provided for this step:
    - NONE when you want to use no options, but still use the optional attribute for the GUID
    - NOAUTOGENSUPPRESSION to disable suppression of autogen by the object
    - NOCRASH to disable crash detection by the object
    - NOSHADOW to disable the display of shadows by the object
    - NOFOG to disable fog affecting to the object
    - NOZWRITE to disable Z writing of the object
    - NOZTEST to disable Z testing of the object
    - ALTISMSL when the altitude of the object should be considered MSL and not AGL
    - SNAPTOGROUND when the object should be snapped to the ground (MSFS only)
    - SNAPTONORMAL when object attitude should be snapped to the terrain normal (MSFS only)
    - COMPLEXITY_SPARSE to specify that the buildings should get image complexity sparse.
    - COMPLEXITY_NORMAL to specify that the buildings should get image complexity normal.
    - COMPLEXITY_DENSE to specify that the buildings should get image complexity dense.
    - COMPLEXITY_VERY_DENSE to specify that the buildings should get image complexity very dense.
    - COMPLEXITY_EXTREMELY_DENSE to specify that the buildings should get image complexity extremely dense.

**Example**

Create car parkings based on OSM parking data:

```
CreateXMLCarParking|amenity="parking"
```

### 10.4.4 CreateXMLEffect

**Purpose**

Create XML effects from point features. For example to place light poles or churches as XML objects. The library objects are called from library BGL files using the GUID of the objects.

**Arguments**

1. Filter to select the point features from which the effects will be generated
2. The name of the effect file to use
3. The name of the attribute that contains the heading of the objects, you can also type in the heading value directly if all features need the same heading
4. An additional heading offset that should be applied to all features
5. The altitude at which the library object should be placed
6. Optional effect parameters, as described in the SDK
7. Optional options, the following options can be provided for this step:
   - NOAUTOGENSUPPRESSION to disable suppression of autogen by the object
   - NOCRASH to disable crash detection by the object
   - NOSHADOW to disable the display of shadows by the object
   - NOFOG to disable fog affecting to the object
   - NOZWRITE to disable Z writing of the object
   - NOZTEST to disable Z testing of the object
   - ALTISMSL when the altitude of the object should be considered MSL and not AGL
   - SNAPTOGROUND when the object should be snapped to the ground (MSFS only)
   - SNAPTONORMAL when object attitude should be snapped to the terrain normal (MSFS only)
   - COMPLEXITY_SPARSE to specify that the buildings should get image complexity sparse.
   - COMPLEXITY_NORMAL to specify that the buildings should get image complexity normal.
   - COMPLEXITY_DENSE to specify that the buildings should get image complexity dense.
   - COMPLEXITY_VERY_DENSE to specify that the buildings should get image complexity very dense.
   - COMPLEXITY_EXTREMELY_DENSE to specify that the buildings should get image complexity extremely dense.

**Example**

Create firework effects. The heading of the features is stored in the HDG attribute and it is placed at the ground:

```
CreateXMLEffect|type="firework"|FW_show1.fx|HDG|0|0
```

Create firework effect that only show on new year:

```
CreateXMLEffect|type="firework"|FW_show2.fx|HDG|0|0|DOY=1
```

### 10.4.5 CreateXMLExcludeRectangle

**Purpose**

Create XML exclusion rectangles from the selected features. The bounding rectangle of the feature is used for the exclusion area.

**Arguments**

1. Filter to select the features from which the exclude rectangle will be created
2. The type of exclusion to create, possible values are:
   (a) EXCLUDE_ALL
   (b) EXCLUDE_BEACON
   (c) EXCLUDE_CAR_PARKING
   (d) EXCLUDE_EFFECTS
   (e) EXCLUDE_EXTRUSION_BRIDGES
   (f) EXCLUDE_GENERIC_BUILDINGS
   (g) EXCLUDE_LIBRARY_OBJECTS
   (h) EXCLUDE_SIM_OBJECTS
   (i) EXCLUDE_TAXIWAY_SIGNS
   (j) EXCLUDE_TRIGGER_OBJECTS
   (k) EXCLUDE_WINDSOCK_OBJECTS

**Example**

Create exclusion rectangles around all airport polygons:

```
CreateXMLExcludeRectangle|aeroway="aerodrome"|EXCLUDE_ALL
```

### 10.4.6 CreateXMLLibObj

**Purpose**

Create XML library objects from features, the center of the feature is used as location. For example to place light poles or churches as XML objects. The library objects are called from library BGL files using the GUID of the objects.

**Arguments**

1. Filter to select the point features from which the library objects will be generated
2. GUID of the library objects to use. If you provide multiple GUIDs, separated by a semicolon (;), scenProc will randomly select one of the GUIDs from the list. You can provide the GUID in either the FS2004 or the FSX GUID format.
3. This argument specifies the orientation of the library object. You can specify this in two different ways:
   (a) If you only want to set a heading you can provide the name of the attribute that contains the heading of the objects. If the entered name does not exist as attribute in the feature, it will be parsed as the heading value instead.
   (b) If you want to specify a heading, pitch and roll you can provide three attributes names or values separated by a semicolon.

4. An additional heading offset that should be applied to all features
5. Name of the attribute that contains the altitude value. If the feature has no attribute with this name, the name will be parsed and used as altitude value.
6. The scale of the library object. This can be specified in two different ways:
   (a) If the string enters contains a multiply ( * ) or divide ( / ) symbol it is assumed to be a calculation based on a feature attribute. The value before the multiply or divide symbol should be the attribute name to use and the value behind the symbol the factor that the value of the attribute should be multiplied or divided with. So if you enter FLENGTH/20, the scale will be calculated by dividing the length of the feature by 20. This would thus assume that the library object has a size of 20 meters at scale 1.
   (b) Name of the attribute that contains the scale value. If the feature has no attribute with this name, the name will be parsed and used as scale value.
7. Optional options, the following options can be provided for this step:
   - NONE when you want to use no options, but still use the optional attribute for the GUID
   - NOAUTOGENSUPPRESSION to disable suppression of autogen by the object
   - NOCRASH to disable crash detection by the object
   - NOSHADOW to disable the display of shadows by the object
   - NOFOG to disable fog affecting to the object
   - NOZWRITE to disable Z writing of the object
   - NOZTEST to disable Z testing of the object
   - ALTISMSL when the altitude of the object should be considered MSL and not AGL
   - SNAPTOGROUND when the object should be snapped to the ground (MSFS only)
   - SNAPTONORMAL when object attitude should be snapped to the terrain normal (MSFS only)
   - COMPLEXITY_SPARSE to specify that the buildings should get image complexity sparse.
   - COMPLEXITY_NORMAL to specify that the buildings should get image complexity normal.
   - COMPLEXITY_DENSE to specify that the buildings should get image complexity dense.
   - COMPLEXITY_VERY_DENSE to specify that the buildings should get image complexity very dense.
   - COMPLEXITY_EXTREMELY_DENSE to specify that the buildings should get image complexity extremely dense.
8. Optional, name of the attribute that contains the GUID value. If the feature has no attribute with this name, the GUID specified in the step is used.

**Example**

Create library objects using a default lighthouse object. The heading of the features is stored in the HDG attribute and it is placed at the ground with a scale of 1:

```
CreateXMLLibObj|OBJECT="LIGHTHOUSE"|{be7fa036-6133-48cd-b3a3-bdc339e6ab35}|HDG
    ↪ |0|0|1
```

Same example as above, but this time the GUID is provided in the FS2004 style as used by FS2004 library BGL files:

```
CreateXMLLibObj|OBJECT="LIGHTHOUSE"|be7fa03648cd6133c3bda3b335abe639|HDG|0|0|1
```

Same examples as above, but this time the heading is set at a fixed value of 73 degrees:

```
CreateXMLLibObj|OBJECT="LIGHTHOUSE"|{be7fa036-6133-48cd-b3a3-bdc339e6ab35
    ↪ }|73|0|0|1
```

Same examples as above, but this time the heading is set at a fixed value of 73 degrees, the pitch at 12 degrees and the roll at 30 degrees:

```
CreateXMLLibObj|OBJECT="LIGHTHOUSE"|{be7fa036-6133-48cd-b3a3-bdc339e6ab35
    ↪ }|73;12;30|0|0|1
```

Same examples as above, but this time the heading, pitch and roll values are read from the attributes HDG, PIT and ROLL:

```
CreateXMLLibObj|OBJECT="LIGHTHOUSE"|{be7fa036-6133-48cd-b3a3-bdc339e6ab35}|HDG;
    ↪ PIT;ROLL|0|0|1
```

Create library objects using a default firetower object. The objects don't suppress autogen and have a 90 degree offset compared to the heading attribute and a scale value of 1.5:

```
CreateXMLLibObj|OBJ="FTOW"|{ace3f42f-4df3-4756-9c60-6f7032f8d271}|HDG|90|0|1.5|
    ↪ NOAUTOGENSUPPRESSION
```

Create library objects where the GUID is stored in an attribute named OBJGUID, the scale in an attribute SCALE and the heading in an attribute HDG:

```
CreateXMLLibObj|OBJ="HOUSE"|{00000000-0000-0000-0000-000000000000}|HDG|0|0|SCALE|
    ↪ NONE|OBJGUID
```

Create library objects where the GUID is selected randomly from the GUIDs that are provided:

```
CreateXMLLibObj|OBJ="HOUSE
    ↪ "|{11111111-1111-1111-1111-111111111111};{22222222-2222-2222-2222-222222222222}|
    ↪ HDG|0|0|1
```

Create library objects where the scale is calculated by dividing the length of the feature by 25:

```
CreateXMLLibObj|OBJ="STORAGE_TANK"|{ace3f42f-4df3-4756-9c60-6f7032f8d271}|0|0|0|
    ↪ FLENGTH/25
```

### 10.4.7 CreateXMLRectangle

#### Purpose

Create XML rectangles from the selected features. The minimum area polygon of hte feature is used for the rectangle. The XML rectangle is only supported when exporting to MSFS.

**Arguments**

1. Filter to select the features from which the rectangle will be created.
2. The GUID of the surface to apply to the rectangle.
3. The attribute that contains the altitude of the rectangle. When the attribute does not exist the value is parsed as altitude.
4. The attribute that contains the falloff of the rectangle. When the attribute does not exist the value is parsed as falloff.
5. The priority of the rectangle.

**Example**

Create rectangles around all helipad polygons:

```
CreateXMLRectangle|aeroway="helipad"|{6C0C6528-5CF1-483A-A586-2C905CF2757E
    ↪ }|0|500|0
```

### 10.4.8 CreateXMLSimObject

**Purpose**

Create XML sim objects from features, the center of the feature is used as location.

**Arguments**

1. Filter to select the point features from which the library objects will be generated
2. The title of the SimObject container that should be placed. If you provide multiple GUIDs, separated by a semicolon (;), scenProc will randomly select one of the titles from the list.
3. This argument specifies the orientation of the library object. You can specify this in two different ways:
   (a) If you only want to set a heading you can provide the name of the attribute that contains the heading of the objects. If the entered name does not exist as attribute in the feature, it will be parsed as the heading value instead.
   (b) If you want to specify a heading, pitch and roll you can provide three attributes names or values separated by a semicolon.
4. An additional heading offset that should be applied to all features
5. Name of the attribute that contains the altitude value. If the feature has no attribute with this name, the name will be parsed and used as altitude value.
6. The scale of the library object. This can be specified in two different ways:
   (a) If the string enters contains a multiply ( * ) or divide ( / ) symbol it is assumed to be a calculation based on a feature attribute. The value before the multiply or divide symbol should be the attribute name to use and the value behind the symbol the factor that the value of the attribute should be multiplied or divided with. So if you enter FLENGTH/20, the scale will be calculated by dividing the length of the feature by 20. This would thus assume that the library object has a size of 20 meters at scale 1.
   (b) Name of the attribute that contains the scale value. If the feature has no attribute with this name, the name will be parsed and used as scale value.
7. Optional options, the following options can be provided for this step:
   - NONE when you want to use no options, but still use the optional attribute for the GUID

- NOAUTOGENSUPPRESSION to disable suppression of autogen by the object
- NOCRASH to disable crash detection by the object
- NOSHADOW to disable the display of shadows by the object
- NOFOG to disable fog affecting to the object
- NOZWRITE to disable Z writing of the object
- NOZTEST to disable Z testing of the object
- ALTISMSL when the altitude of the object should be considered MSL and not AGL
- SNAPTOGROUND when the object should be snapped to the ground (MSFS only)
- SNAPTONORMAL when object attitude should be snapped to the terrain normal (MSFS only)
- COMPLEXITY_SPARSE to specify that the buildings should get image complexity sparse.
- COMPLEXITY_NORMAL to specify that the buildings should get image complexity normal.
- COMPLEXITY_DENSE to specify that the buildings should get image complexity dense.
- COMPLEXITY_VERY_DENSE to specify that the buildings should get image complexity very dense.
- COMPLEXITY_EXTREMELY_DENSE to specify that the buildings should get image complexity extremely dense.

**Example**

Create sim objects of a airplane. The heading of the features is stored in the HDG attribute and it is placed at the ground with a scale of 1:

```
CreateXMLSimObject|type="airplane"|My Plane|HDG|0|0|1
```

Create sim objects of a airplane, where the container title is selected randomly from the provided titles:

```
CreateXMLSimObject|type="airplane"|My Plane1;My Plane2;My Plane 3|HDG|0|0|1
```

### 10.4.9 MergeScene

**Purpose**

Merge all created XML library objects and 3D building models into one big model. This requires that they all have an altitude MSL defined, else they will not follow the terrain elevation.

**Arguments**

1. Optional, the filenames of additional BGL libraries that should be used to resolve the GUID of create XML library objects.
2. Optional options, the following options can be provided for this step:
   - NOAUTOGENSUPPRESSION to disable suppression of autogen by the object
   - NOCRASH to disable crash detection by the object
   - NOSHADOW to disable the display of shadows by the object

- NOFOG to disable fog affecting to the object
- NOZWRITE to disable Z writing of the object
- NOZTEST to disable Z testing of the object

**Example**

Merge all placed XML objects in one scene:

```
MergeScene
```

Merge all placed XML objects in one scene, using lib.bgl to find additional objects and set no autogen suppression on the created merged object:

```
MergeScene|lib.bgl|NOAUTOGENSUPPRESSION
```

## 10.5 Create terrain vector scenery

In this section all steps to create terrain vector scenery objects are described. These objects can be exported to BGL files later on.

### 10.5.1 CreateMSFSExclude

**Purpose**

Create MSFS exclude polygons for roads, streetlights and buildings from polygons.

**Arguments**

1. The filter to select the polygon features from which the excludes are made.
2. The type of exclude to create, possible values are:
   - EXCLUDE_ROAD
   - EXCLUDE_STREET_LIGHT
   - EXCLUDE_POWER_LINE
   - EXCLUDE_FEATURE_POINT
   - EXCLUDE_DETECTED_BUILDING
   - EXCLUDE_MS_BUILDING
   - EXCLUDE_OSM_BUILDING
   - EXCLUDE_TIN

**Example**

Create road exclude:

```
CreateMSFSExclude|type="exclude"|EXCLUDE_ROAD
```

Create exclude for all building types:

```
CreateMSFSExclude|type="building_exclude"|EXCLUDE_DETECTED_BUILDING;
    ↪ EXCLUDE_MS_BUILDING;EXCLUDE_OSM_BUILDING
```

### 10.5.2 CreateTVecAirportBound

#### Purpose

Create airport bound terrain vectors, these are typically used to create grass background and flattens for airports.

#### Arguments

1. The filter to select the polygon features that should be used to create the airport bounds from.
2. The GUID of the airport bound type to use.
3. The name of the attribute that contains the elevation value of the polygon feature. If an attribute with that name does not exists, the provided value is parsed as elevation.

#### Example

Create airport bounds from all airport features and use the attribute elev for the elevation:

```
CreateTVecAirportBound|type="airport"|{46bfb3bd-ce68-418e-8112-feba17428ace}|elev
```

### 10.5.3 CreateTVecExclusion

#### Purpose

Create exclusion terrain vectors.

#### Arguments

1. The filter to select the polygon features that should be used to create the exclusions from.
2. The GUID of the terrain vector type to exclude.

#### Example

Create exclusions for all roads based on a polygon of the country:

```
CreateTVecExclusion|type="country"|{325dd470-b342-4d15-ac54-f67ed9f5914f}
```

### 10.5.4 CreateTVecFreewayTraffic

#### Purpose

Create freeway traffic vectors to create moving traffic on roads.

#### Arguments

1. The filter to select the lines features that should be used to create the freeway traffic from.
2. The number of lanes of the traffic.
3. The direction for the traffic, either F when the traffic should move from the first point of the line or T when it should move towards the first point of the line.
4. Optional, name of the attribute that contains the number of lanes. If this attribute does not exact the number of lanes as provided in the second attribute is used as default value.

**Example**

Create freeway traffic for all highways with 2 lanes:

```
CreateTVecFreewayTraffic|highway="motorway"|2|F
```

Create freeway traffic for all highways with the number of lanes from the lanes attribute and using 2 lanes as default:

```
CreateTVecFreewayTraffic|highway="motorway"|2|F|lanes
```

### 10.5.5 CreateTVecPark

**Purpose**

Create park terrain vectors, these are typically used for parks, golf courses and any other textured polygon.

**Arguments**

1. The filter to select the polygon features that should be used to create the parks from.
2. The GUID of the park type to use.

**Example**

Create parks for all forests:

```
CreateTVecPark|landuse="forest"|{6cea593e-64bc-4eb8-a6ca-806a03901115}
```

### 10.5.6 CreateTVecRailroad

**Purpose**

Create railroad terrain vectors.

**Arguments**

1. The filter to select the lines features that should be used to create the railroads from.
2. The GUID of the railroad type to use.

**Example**

Create railroads from all rail features:

```
CreateTVecRailroad|railway="rail"|{dde116bf-0e97-4eb6-a9ec-7ef93d9f2d0e}
```

### 10.5.7 CreateTVecRoad

**Purpose**

Create road terrain vectors.

**Arguments**

1. The filter to select the lines features that should be used to create the roads from.
2. The GUID of the road type to use.

**Example**

Create roads from all primary road features:

```
CreateTVecRailroad|highway="primary"|{325dd470-b342-4d15-ac54-f67ed9f5914f}
```

### 10.5.8 CreateTVecShoreline

**Purpose**

Create shoreline terrain vectors.

**Arguments**

1. The filter to select the line features that should be used to create the shorelines from.
2. The GUID of the shoreline type to use.

**Example**

Create airport bounds from all airport features and use the attribute elev for the elevation:

```
CreateTVecShoreline|type="shore"|{46bfb3bd-ce68-418e-8112-feba17428ace}
```

### 10.5.9 CreateTVecStream

**Purpose**

Create stream terrain vectors.

**Arguments**

1. The filter to select the line features that should be used to create the streams from.
2. The GUID of the stream type to use.

**Example**

Create streams from all river features:

```
CreateTVecStream|waterway="river"|{2d3fc985-a72b-473d-b23b-d78e72e63b53}
```

### 10.5.10 CreateTVecUtility

**Purpose**

Create utility terrain vectors, these are typically used to place power pylons and such.

**Arguments**

1. The filter to select the line features that should be used to create the utilities from.
2. The GUID of the utility type to use.

**Example**

Create utilities from all power line features:

```
CreateTVecUtility|power="line"|{c942aadb-c351-4088-925e-a732c2093b63}
```

### 10.5.11 CreateTVecWaterPoly

#### Purpose

Create water terrain vectors. The vertices of the water polygons need to have a valid elevation set.

#### Arguments

1. The filter to select the polygon features that should be used to create the water polygons from.
2. The GUID of the water polygon type to use.
3. The elevation sample mode to use. Possible values are:
   - ATTR_VALUE to get the elevation from the provided attribute or parse the value as elevation when no such attribute exists.
   - RASTER_CENTER to sample the elevation at the center of the feature from the given raster files and use it for any vertex.
   - RASTER_VERTEX to sample the elevation of each vertex from the given raster files.
   - RASTER_LINE to sample the elevation along a line that matches the polygon and define each vertex elevation based on that line, when no line matches the RASTER_CENTER mode is used instead for the feature.
4. The name of the attribute that contains the elevation value of the polygon feature. If an attribute with that name does not exists, the provided value is parsed as elevation. This value is only used in ATTR_VALUE mode
5. The filter to select the elevation raster files to use in RASTER_CENTER, RASTER_VERTEX and RASTER_LINE mode. Use NONE when in ATTR_VALUE mode.
6. The filter to select the line features to be used to define the river elevation and slope in RASTER_LINE sample mode.

#### Example

Create water polygons from all water features with a fixed elevation of 123.45 meter:

```
CreateTVecWaterPoly|natural="water"|{bcd5c182-9c8b-4c57-97bd-272cf492cbff}|
    ↪ ATTR_VALUE|123.45|NONE|NONE
```

Create water polygons from all water features with an elevation set in the elev attribute:

```
CreateTVecWaterPoly|natural="water"|{bcd5c182-9c8b-4c57-97bd-272cf492cbff}|
    ↪ ATTR_VALUE|elev|NONE|NONE
```

Create water polygons from all water features with an elevation samples from for each vertex from the raster data:

```
CreateTVecWaterPoly|natural="water"|{bcd5c182-9c8b-4c57-97bd-272cf492cbff}|
    ↪ RASTER_VERTEX|0|*|NONE
```

Create water polygons from all water features with an elevation sampled along the river line:

```
CreateTVecWaterPoly|natural="water"|{bcd5c182-9c8b-4c57-97bd-272cf492cbff}|
    ↪ RASTER_LINE|0|*|waterway="*"
```

### 10.5.12 CreateTVecWaterPolyGPS

#### Purpose

Create water polygon terrain vectors for the GPS. These are used to render water on the GPS display in the aircraft.

#### Arguments

1. The filter to select the polygon features that should be used to create the GPS water polygons from.

#### Example

Create GPS water polygons from all water features:

```
CreateTVecWaterPolyGPS|natural="water"
```

## 10.6 Create AF2 scenery

### 10.6.1 CreateAF2Building

#### Purpose

With this step you can create AF2 cultivation buildings from the selected polygon features. These can later be written to a TOC file for AF2.

#### Arguments

1. Filter to select the features from which the buildings should be generated.
2. The name of the attribute that contains the number of floors of the building. Or NONE if no attribute is to be used for the amount of floors.
3. The name of the attribute that contains the height of the building. Or NONE if no attribute is to be used for the height. This attribute is only used when the number of floors attribute does not exist in the feature.
4. A scale factor to apply to the height value from the height attribute to convert it to a number of floors. Use a value of 1 when no scaling should be applied.
5. When the number of floors and height attributes do not exist, these fallback values for the number of floors are used. A random value between the minimum and maximum value supplied is selected.
6. The roof type of the building, the possible values are defined in the AF2_config.ini file.
7. The usage type of the building, the possible values are defined in the AF2_config.ini file.
8. The name of the attribute that contains the altitude of the building. If the feature has no such attribute, the name will be parsed as altitude value.
9. Optional options, the following options can be provided for this step:
    - MAXRATIO=## specifies that the building should be split into multiple buildings when the ratio between the length and width is bigger than the specified value. This can be useful to split long rows of houses that are represented as one polygon into multiple buildings.

#### Example

Create AF2 cultivation building from all buildings with an area ratio over more than 70%. The buildings have 2 floors and a gabled roof.

```
CreateAF2Building|building="*" And FAREARAT>0.7|NONE|NONE|1|2;2|gable|residential
    ↪ |0
```

Create AF2 cultivation building from all buildings with an area ratio over more than 70%. The buildings have between 1 and 4 floors and a gabled roof. The altitude is 0.

```
CreateAF2Building|building="*" And FAREARAT>0.7|NONE|NONE|1|1;4|gable|residential
    ↪ |0
```

Create AF2 cultivation building from all buildings with an area ratio over more than 70%. The number of floors is stored in the attribute `numFloors` and a flat roof is used. When the attribute does not exist 2 floors are used. The altitude is taken from an attribute named alt.

```
CreateAF2Building|building="*" And FAREARAT>0.7|numFloor|NONE|1|2;2|flat|
    ↪ industrial|alt
```

Create AF2 cultivation building from all buildings with an area ratio over more than 70%. The height is stored in the attribute `height` and scale with a factor of 0.3 to get the number of floors. When the attribute does not exist 2 floors are used. A flat roof is used and the altitude is taken from an attribute named alt.

```
CreateAF2Building|building="*" And FAREARAT>0.7|NONE|height|0.3|2;2|flat|
    ↪ industrial|alt
```

### 10.6.2 CreateAF2Light

#### Purpose

With this step you can create AF2 cultivation lights from the selected point features. These can later be written to a TOC file for AF2.

#### Arguments

1. Filter to select the features from which the lights should be generated.
2. The color of the light, 3 values have to be given for the R, G and B components of the color. The range is between 0.0 and 1.0.
3. The intensity of the light.
4. The flashing attributes of the light, 3 values have to be given. For the period, alternate flash and duration.
5. The altitude above the ground of the light.

#### Example

Create AF2 cultivation lights from all light point features. The lights have an almost white color and are placed 10 meter above the ground.

```
CreateAF2Light|point="light"|0.8;0.8;0.8|3|0;0;1|10
```

### 10.6.3 CreateAF2Object

**Purpose**

With this step you can create AF2 objects from the selected point features. These can later be written to a TSC file for AF2. You have to make sure yourself that the object files are present in the scenery folder as well.

**Arguments**

1. Filter to select the features from which the objects should be generated.
2. The name of the object that should be placed.
3. The name of the attribute that contains the heading. If this the feature does not have an attribute with this name, the value is parsed as heading instead.

**Example**

Create AF2 objects from all tower features and place the object named generic_tower.

```
CreateAF2Object|man_made="tower"|tower|hdg
```

### 10.6.4 CreateAF2Plant

**Purpose**

With this step you can create AF2 cultivation plants from the selected point features. These can later be written to a TOC file for AF2.

**Arguments**

1. Filter to select the features from which the plants should be generated.
2. The plant height, 2 values have to be given. A minimum and a maximum height.
3. The attribute that contains the altitude of the plant. If this attribute does not exist, the value is parsed as altitude.
4. The plant group, the possible values are defined in the AF2_config.ini file.
5. Optional the plant species, the possible values are defined in the AF2_config.ini file.

**Example**

Create AF2 cultivation plants from all forest point features. The plants will be broadleaf between 10 and 20 meter in height. The altitude of the plant is 0.

```
CreateAF2Plant|landuse="forest"|10;20|0|broadleaf
```

Create AF2 cultivation plants from all forest point features where the random attribute has a value bigger than 0.5 (so that is about half of the trees). The plants will be broadleaf between 12 and 18 meter in height and use the species T08. The altitude of the plant is 0.

```
CreateAF2Plant|landuse="forest" And FRAND >= 0.5|12;18|0|broadleaf|T08
```

### 10.6.5 CreateAF2XRef

**Purpose**

With this step you can create AF2 XRef objects from the selected point features. These can later be written to a TOC file for AF2.

**Arguments**

1. Filter to select the features from which the XRef objects should be generated.
2. The name of the XRef object that should be placed. The possible values for the auto-completion are defined in the AF2_config.ini file.
3. The name of the attribute that contains the heading. If this the feature does not have an attribute with this name, the value is parsed as heading instead.

**Example**

Create AF2 XRef objects from all tower features and place the object named tower00_medium_blue_ds_05_08_08.

```
CreateAF2XRef|man_made="tower"|tower00_medium_blue_ds_05_08_08|hdg
```

## 10.7 Exporting

In this section all steps for exporting your scenery or data from scenProc are described.

### 10.7.1 ExportAGN

**Purpose**

With this step created autogen objects are exported to AGN files. These AGN files should be put in the texture folder of your scenery project, so that Flight Simulator will load and show them.

**Arguments**

1. The FS version to export to. Possible values are FS2004, FSX or P3D v2. The created autogen files for FSX and P3D v2 are the same, so they work in both versions. FS2004 uses a different AGN file format.
2. The folder to export the AGN files to. This is usually the texture folder of your scenery project.

**Example**

Export FSX autogen:

```
ExportAGN|FSX|c:\flightsim\myproject\texture
```

### 10.7.2 ExportBGL

**Purpose**

With this step created XML object are exported to BGL files. These BGL files should be put in the scenery folder of your project, so that Flight Simulator will load and show them.

**Arguments**

1. The FS version to export to. Possible values are FS2004, FSX, P3D v2, P3D v4 or P3D v4.4. This determines which BGLComp compiler is used.
2. The basename of the BGL files to be created. To this basename a number is added for each gridcell in your data.
3. The folder to export the BGL files to. This is usually the scenery folder of your scenery project.

4. Optional options, the following options can be provided for this step:
    - KEEPXML specifies that the intermediate XML files used to make the BGL file should be kept after compiling.
    - KEEPX specifies that the intermediate X files used to make MDL files should be kept after compiling.
    - KEEPMDL specifies that the intermediate MDL files of the models should be kept after compiling.
    - NODRAWCALLBATCHING specifies that models should be exported without drawcall batching.
    - NOCRASH specifies that models should be exported without crash boxes.

**Example**

Export XML objects for FSX:

```
ExportBGL|FSX|objects|c:\flightsim\myproject\scenery
```

Export XML objects for FS2004 and keep the XML files:

```
ExportBGL|FS2004|objects|c:\flightsim\myproject\scenery|KEEPXML
```

### 10.7.3 ExportBLN

**Purpose**

Export features to the BLN format.

**Arguments**

1. Filter to select the features that should be exported.
2. The basename of the file to create.

**Example**

Export all features to a BLN file with the basename myfile:

```
ExportBLN|*|myfile
```

### 10.7.4 ExportGDAL

**Purpose**

With this step raster features can be saved to disk using the GDAL library.

**Arguments**

1. Filter to select the features that should be exported
2. The image format to use.
3. The basename of the images to be created
4. The folder where the images will be created
5. A filter to select an additional set of features. Only images in cells will be saved if there are features from this second filter present. Specify NONE if you want to export all features.

6. Optional, filename of the texture filter configuration file that should be applied to the image before export.

**Example**

Export all images to GeoTIFF:

```
ExportGDAL|FTYPE="RASTER"|GTiff|image|c:\mydata\images|NONE
```

Export all images in cells that also contain forest polygons to PNG:

```
ExportGDAL|FTYPE="RASTER"|PNG|image|c:\mydata\images|type=forest
```

Export all images, applying a color correction texture filter before export:

```
ExportGDAL|FTYPE="RASTER"|GTiff|image|c:\mydata\images|NONE|color_correction.tf2
```

### 10.7.5 ExportGDALCombined

**Purpose**

With this step a raster feature that is made by combining two different rasters can be saved to disk.

This step is to be deprecated, please use the `MergeRasterFeatures` and `ExportGDAL` steps instead.

**Arguments**

1. Filter to select set A of raster features to use.
2. Filter to select set B of raster features to use.
3. Specification for 4 bands which raster band to use. Possible values are:
   - A# to select the #th band of raster A.
   - B# to select the #th band of raster B.
   - -1 to specify this band is not used.
4. The image format to use when saving.
5. The basename of the images to be created
6. The folder where the images will be created
7. A filter to select an additional set of features. Only images in cells will be saved if there are features from this second filter present. Specify NONE if you want to export all features.

**Example**

Create a new raster by taking the RGB bands from raster A and the alpha band from raster B:

```
ExportGDAL|FROMFILE="rasterA.tif"|FROMFILE="rasterB.tif"|A0;A1;A2;B3|GTiff|image|
    ↪ c:\mydata\images|NONE
```

### 10.7.6 ExportMSFS

**Purpose**

With this step created scenery objects are exported to XML and glTF files so that they can be put in a package with the MSFS packager tool.

**Arguments**

1. The MSFS version to export to, possible values are: MSFS2020 and MSFS2024.
2. The basename of the XML and glTF files to be created. To this basename a number is added for each gridcell in your data.
3. The project name of the package that will be created.
4. The folder to export the XML and glTF files to. Instead this folder a sub-folder `modellib` will be created for all glTF files and a sub-folder `scene` for the XML placement files.
5. Optional options, possible values are:
    - COMPILE_PACKAGE specifies that the package is compiled with the fspackagetool after saving it.

**Example**

Export files to MSFS:

```
ExportMSFS|MSFS2020|objects|project|c:\flightsim\myproject\PackageSources
```

### 10.7.7 ExportOGR

**Purpose**

With this step features can be exported to the different vector file formats supported by the OGR library. This way the data as processed by scenProc can be viewed in GIS programs or other tools that support these formats. The supported formats include Shapefile, GML, KML (for viewing in Google Earth) and many more.

For all details on the different formats that OGR supports and the limitations of the drivers for each of these formats, please check the OGR formats page on the GDAL website.

The features are exported in different layers, for each gridcell and feature type a layer is created. So there are maximum 4 layers per gridcell. The layer name is appended by the gridcell number and a suffix for the feature type. It depends on the format driver that is used if these layers are written as separate files or all in one big file.

**Arguments**

1. Filter to select the features that should be exported.
2. The OGR format that should be exported to, see the auto completion help for a list of all supported formats.
3. The filename of the OGR datasource that will be created.
4. The basename used for the layers that are created.
5. Optional, the output projection used for the file that is written. When no specified or NOREPROJ the data will be written as WGS84 geodetic coordinates.
6. Optional options, the following options can be provided for this step:
    - APPENDFILES specifies that the features should be appended to an existing file when it exists. The default behavior is to overwrite a file when it already exists.

- ADDSPECIALATTR specifies that the special attributes that are calculated by scenProc should also be included when exporting. This means that attributes like FWIDTH, FLENGTH or FAREARAT are add as attributes to the file.
- EXPORTCELLS specifies that the scenProc gridcells and their attributes should also be exported. They are placed in a separate layer with a polygon in it for each gridcell.

**Example**

Export all polygon features to a Shapefile:

```
ExportOGR|FTYPE="POLYGON"|ESRI Shapefile|c:\myProject\myfile.shp|layerName
```

Export all polygon features to a folder containing Shapefiles:

```
ExportOGR|FTYPE="POLYGON"|ESRI Shapefile|c:\myProject\shp|layerName
```

Export all point features to a KML file, including all special attributes:

```
ExportOGR|FTYPE="POINT"|KML|C:\myProject\myfile.kml|layerName|ADDSPECIALATTR
```

Export all features to a GML file, including the gridcells:

```
ExportOGR|*|GML|c:\myProject\myfile.gml|layerName|EXPORTCELLS
```

### 10.7.8 ExportResampleElevationBGL

**Purpose**

Create an terrain elevation BGL file using the resample tool from the selected raster elevation data.

**Arguments**

1. The filter to select the raster features with the elevation data.
2. The filter to select the masking polygon features. Or use NONE if you don't want to use a mask.
3. The folder where the BGL file should be saved.
4. The basename of the BGL file to create.

**Example**

Create an elevation BGL file from the raster file elev.tif:

```
ExportResampleElevationBGL|FROMFILE="elev.tif"|NONE|c:\myscenery|elev
```

### 10.7.9 ExportResamplePhotorealBGL

**Purpose**

Create an terrain photoreal BGL file using the resample tool from the selected raster elevation data. A texture filter is used to define colour corrections, watermask, blendmask and seasonal variants. See chapter 6 for more information on the texture filter configuration.

**Arguments**

1. Filename of the texture filter configuration to use.
2. Filter to select the raster features with the image data. Or if you use EMPTY## where ## is the number of a terrain LOD an empty raster file with the resolution of the specified LOD tile is used as input.
3. The filter to select the masking polygon features. Or use NONE if you don't want to use a mask.
4. The compression value to use for the output BGL (a value between 0 and 100).
5. The no data values to use, you need to enter R;G;B values. Use a value of -1 if you don't want to set a no data value for that band.
6. The minimum and maximum LOD value that resample should generate. Use -1 to let resample detect this automatically.
7. The channel numbers of the blend mask and water mask when these are stored internally in the image file. Or use -1 when they are not stored internally.
8. Indication whether the variation of the output image is for day (DAY) or night (NIGHT).
9. The season definition for each Output Image and Output Season step in the texture filter, the definition for each output is separated by a semicolon (;). The season definition can be:
   - * to indicate that this image is for all seasons of the year
   - A comma separated list of months that apply, e.g. 12,1 indicates the image is used in December and January.
   - A dash separated range of months, e.g. 2-5 indicates the image is used from February until May.
10. The folder where the BGL file should be saved.
11. The basename of the BGL file to create.
12. Optional options, possible values are:
    - INVERSEMASK in case the mask polygons should be inversed.
    - KEEPSRC to keep the source files that resample uses (INF file and images).
    - USERVAR=var to pass a user variable value into the texture filter configuration.
    - BUFFER=val to set the buffer size in degrees that should be applied to empty rasters.
    - DISPOSEIMAGES will force a dispose of the raster data after using it, only use this option if the raster feature is not used again by another step.

**Example**

Create a photoreal BGL file from the raster file image.tif. The file resample.tf2 contains the texture filter configuration to use:

```
ExportResamplePhotorealBGL|resample.tf2|FROMFILE="image.tif"|NONE
    ↪ |100|-1;-1;-1|-1;-1|-1;-1|DAY|*|c:\myscenery|photoreal
```

Create a photoreal BGL file from the raster file image.tif for LOD 9 to 14, with white as no data and with 85% compression. The file resample.tf2 contains the texture filter configuration to use:

```
ExportResamplePhotorealBGL|resample.tf2|FROMFILE="image.tif"|NONE
    ↪ |85|255;255;255|9;14|-1;-1|DAY|*|c:\myscenery|photoreal
```

Create a photoreal BGL file from the raster file image.tif. The file resample.tf2 contains the texture filter configuration to use. The first output image is shown from April until September and the second the rest of the year.

```
ExportResamplePhotorealBGL|resample.tf2|FROMFILE="image.tif"|NONE
    ↪ |100|-1;-1;-1|-1;-1|-1;-1|DAY|4-9;10-3|c:\myscenery|photoreal
```

Create a photoreal BGL file from the raster file image.tif. The file resample.tf2 contains the texture filter configuration to use. The texture filter configuration stores the blend mask in channel 4 and the water mask in channel 3 of the generated output image:

```
ExportResamplePhotorealBGL|resample.tf2|FROMFILE="image.tif"|NONE
    ↪ |100|-1;-1;-1|-1;-1|4;3|DAY|*|c:\myscenery|photoreal
```

### 10.7.10 ExportResampleSeasonBGL

#### Purpose

Create an terrain season BGL file using the resample tool from the selected vector data.

#### Arguments

1. Filter to select the vector polygons that define the area for which the season should be defined.
2. The season definition for spring or NONE when the spring season is not used. The season definition can be:
   - * to indicate that this image is for all seasons of the year
   - A comma separated list of months that apply, e.g. 12,1 indicates the image is used in December and January.
   - A dash separated range of months, e.g. 2-5 indicates the image is used from February until May.
3. The season definition for summer or NONE when the summer season is not used.
4. The season definition for fall or NONE when the fall season is not used.
5. The season definition for winter or NONE when the winter season is not used.
6. The season definition for hard winter or NONE when the hard winter season is not used.
7. Filter to select the areas for the hard winter season. When NONE is used for this filter the entire areas as selected by the first filter is used for hard winter.
8. The folder where the BGL file should be saved.
9. The basename of the BGL file to create.
10. Optional options, possible values are:
    - KEEPSRC to keep the source files that resample uses (INF file and images).

#### Example

Create a season BGL file from the border polygons:

```
ExportResampleSeasonBGL|type="border"|4-6|7-9|10-12|1,3|2|NONE|c:\myscenery|
    ↪ season
```

### 10.7.11 ExportTSC

**Purpose**

With this step created AF2 objects and cultivation elements are exported to TSC files. TOC cultivation file are also exported as part of this step. These TSC files should be loaded into the AF2 simulator.

**Arguments**

1. The folder where the TSC and TOC files will be written.
2. The basename used for the files. If there are multiple tiles in the loaded data, the geographical location of lower left and upper right corner of the tile will be appended to the basename.

**Example**

Export AF2 TSC file with the name myproject to the folder out.

```
ExportTSC|out|myproject
```

### 10.7.12 ExportTVecBGL

**Purpose**

Export created terrain vector objects to a BGL file using the shp2vec tool.

**Arguments**

1. Folder to save the BGL file to.
2. Basename of the BGL file to save.
3. Optional options, possible values are:
   - REPLACECELLS specifies that the terrain vectors should replace existing scenery, instead of append to them.
   - QMID4FOLDER specifies that the output BGL file should be organised in folders with QMID4 names, like the default FS terrain vector scenery is.
   - QMID7NAME specifies that the output BGL files should be named according to QMID7 tiles. You need to make sure you data uses this size in the SplitGrid step as well then.
4. Optional prefix to use for the filename of the BGL file that is created.

**Example**

Export the terrain vectors to a file with the basename tvec:

```
ExportTVecBGL|c:\myscenery|tvec
```

Export the terrain vectors to a file with the basename tvec and organise by QMID4 folder and QMID7 name:

```
ExportTVecBGL|c:\myscenery|tvec|QMID4FOLDER;QMID7NAME
```

## 10.8 File management

### 10.8.1 CopyAGN

#### Purpose

Copy AGN files from one folder to another folder. For this step to work you need to make sure that you use AGN sized tiles in your script.

#### Arguments

1. Filter to select which grid cells should be processed.
2. From folder.
3. To folder.

#### Example

Copy all AGN files from folderA to folderB:

```
CopyAGN|*|c:\scenery\folderA|c:\scenery\folderB
```

### 10.8.2 CreateDirectory

#### Purpose

Create the specified directory if it does not yet exist.

#### Arguments

1. The directory to create.

#### Example

Make sure the texture folder of your scenery exists:

```
CreateDirectory|c:\myscenery\texture
```

### 10.8.3 PackageDirectory

#### Purpose

Create a ZIP file package of the specified directory.

#### Arguments

1. The directory to create the package from.
2. The directory where the ZIP file should be create.
3. The basebane of the ZIP file to create.
4. Folders to exclude from the ZIP file, separated by a semicolon (;).
5. Files to exlcude from the ZIP file, separated by a semicolon (;).

**Example**

Create a package scenery.zip from the scenery folder.

```
PackageDirectory|c:\FS\myscenery\|c:\output|scenery||
```

Create a package scenery.zip from the scenery folder, excluding the folders temp and WIP and excluding all files that contain ABC or DEF in the name:

```
PackageDirectory|c:\FS\myscenery\|c:\output|scenery|temp;WIP|ABC;DEF
```

## 10.9 Debugging

### 10.9.1 CreateTestGrid

**Purpose**

Create a regular test grid of point features with a 1 x 1 degree area.

**Arguments**

1. Number of points in each row and column

**Example**

Create a test grid of 10 x 10 points:

```
CreateTestGrid|10
```

### 10.9.2 CreateTestPoint

**Purpose**

Create a test point at the given location

**Arguments**

1. The position of the point (lon;lat)

**Example**

Create a test grid at N12.34 E23.45

```
CreateTestPoint|23.45;12.34
```

### 10.9.3 PrintCellAttributes

**Purpose**

Print cell attributes to the event log. This can be useful for debugging to see which attributes are assigned.

**Arguments**

1. Filter to select the cells for which the attributes are printed.
2. Optional options, possible values are:
    - ADDSPECIALATTR to specify the special attributes should also be printed.

**Example**

Print the attributes of all cells:

```
PrintCellAttributes|*
```

Print the attributes of all cells that are urban and include the special attributes:

```
PrintCellAttributes|type="urban"|ADDSPECIALATTR
```

### 10.9.4 PrintFeatureAttributes

**Purpose**

Print feature attributes to the event log. This can be useful for debugging to see which attributes are assigned.

**Arguments**

1. Filter to select the features for which the attributes are printed.
2. Optional options, possible values are:
    - ADDSPECIALATTR to specify the special attributes should also be printed.

**Example**

Print the attributes of all features:

```
PrintFeatureAttributes|*
```

Print the attributes of all building features and include the special attributes:

```
PrintFeatureAttributes|building="*"|ADDSPECIALATTR
```

# Chapter 11

# Support & Consultation

## 11.1 Support forum

If you have any problems while using scenProc or if you have suggestions or other feedback to improve the tool, please let me know. You can either contact me directly or visit the scenProc subforum at FSDeveloper.com.

## 11.2 Reporting crashes

In case a crash happens during running scenProc you will get a dialog to report the crash as shown in Figure 11.1. Please send in error reports in this way, as that will ensure that I get all the details of where the crash happened. That's much more efficient than telling me in the forum that something crashed.

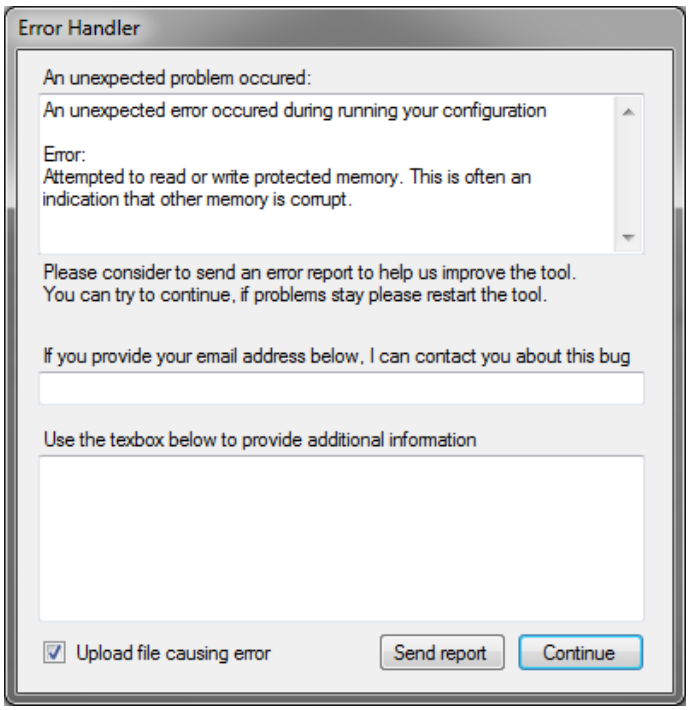



Figure 11.1: Error reporting dialog

## 11.3 Consultation

scenProc is a powerful but complex tool. This manual provides a lot of information and samples on how to use the tool. But if you need more support in getting ready to use scenProc for your project feel free to contact me directly to discuss the possibilities for consultation for your project. Especially if your project goals are in line with new features that I have planned for scenProc I might be willing to cooperate on the project. But also in other cases I can help you get started specifically for your project.

# Chapter 12

# User license

(c) 2011-2026 SceneryDesign.org / Arno Gerretsen

This software program is distributed without charge to other addon developers. Redistribution of the original ZIP file is allowed. You are NOT allowed to sell this software program or ask money for its distribution. Scenery and autogen files created with this software program can be used in commercial scenery projects if you wish.

The copyright and any intellectual property relating to this program remain the property of the author.

The software distributed in this way may represent work in progress, and bears no warranty, either expressed or implied.

## 12.1 Third party libraries

scenProc makes use of a number of third party libraries for specific functions. For example for the configuration script editor and for reading and writing of specific GIS formats. Some of these libraries have special license conditions, these are listed below.

### 12.1.1 GDAL

Copyright (c) 2000, Frank Warmerdam

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

### 12.1.2 ECW JPEG 2000

ECW JPEG 2000 SDK FREE USE LICENSE AGREEMENT IMPORTANT-READ CAREFULLY: This Earth Resource Mapping Limited ("ERM") End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and ERM for the ECW JPEG 2000 SDK software product under this Free Use License Agreement , which includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). The SOFTWARE PRODUCT also includes any updates and supplements to the original SOFTWARE PRODUCT provided to you by ERM. Any software provided along with the SOFTWARE PRODUCT that is associated with a separate end-user license agreement is licensed to you under the terms of that license agreement. By installing, copying, downloading, accessing or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT. The intent of this license is to allow unlimited decompression and limited compression (500MB per image) of ECW JPEG 2000 images within free or commercial applications. The Software Product is protected by patents, copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The Software Product is licensed, not sold. Rights to use patents, including ERM's large DWT and streaming imagery patents, are given only for use with the ECW JPEG 2000 SDK and not for other uses.

Granted Rights 1) GRANT OF LICENSE. This EULA grants you the following limited, non-exclusive rights: a) Software Product. You may install and use the enclosed ECW JPEG 2000 SDK with Unlimited Decompression and Limited Compression (Less than 500MB) SOFTWARE PRODUCT to design, develop, and test software application products for use with the Enhanced Compression Wavelet image technology. Software application products that can be build using this Software Product come under one or more of the following three (3) types: (1) "Server Software" that provides services or functionality on a computer acting as a server (and, the computer running the Server Software shall be referred to as the "Server"); (2) "Client Software" that allows a computer, workstation, terminal, handheld PC, pager, telephone, "smart phone," or other electronic device (each of the foregoing a "Device") to access or utilize the services or functionality provided by Server Software or provide other functionality and; (3) "Software Development Kits" (SDK's) which are software products similar in intent to this software product. b) You may install and use an unlimited number of copies of the ECW JPEG 2000 Runtime, which consists of files contained in the "runtime" directory for use in "Client Software". You may reproduce and distribute an unlimited number of copies of the ECW JPEG 2000 Runtime for use in "Client Software"; provided that each copy shall be a true and complete copy, including all copyright and trademark notices, and that you comply with the Distribution Requirements described below. You may not use Software Product for development or distribution of "Server Software". c) Sample Code. You may modify the sample source code located in the SOFTWARE PRODUCT's "examples" directory ("Sample Code") to design, develop, and test your Application. You may also reproduce and distribute the Sample Code in object code form along with any modifications you make to the Sample Code, provided that you comply with the Distribution Requirements described below For purposes of this section, "modifications" shall mean enhancements to the functionality of the Sample Code. You are not permitted to change the ECW JPEG 2000 file format. 2) Distribution Requirements. You may copy and redistribute an unlimited number of copies of the ECW JPEG 2000 Runtime, and/or Sample Code in object code form (collectively "REDISTRIBUTABLE COMPONENTS") as described above, provided that (i) you distribute the REDISTRIBUTABLE COMPONENTS only in conjunction with, and as a part of, your Application; (ii) your Application adds significant and primary functionality to the REDISTRIBUTABLE COMPONENTS; (iii) any distribution of the ECW JPEG 2000 Runtime includes each and every runtime file distributed as a single set; (iv) you do not permit further redistribution of the REDISTRIBUTABLE COMPONENTS by your end-user customers (v) you do not use ERM's name, logo, or trademarks to market your Application without prior approval by ERM in writing; (vi) you include a valid copyright notice on your Application; and (vii) you indemnify, hold harmless, and defend ERM from and against any claims or lawsuits, including attorneys' fees, that arise or result from the use or distribution of your Application. 3) COPYRIGHT. All title and intellectual property rights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video,

audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by ERM or its suppliers. All title and intellectual property rights in and to the content which may be accessed through use of the SOFTWARE PRODUCT is the property of the respective content owner and may be protected by applicable copyright or other intellectual property laws and treaties. This EULA grants you no rights to use such content. All rights not expressly granted are reserved by ERM.

4) PRERELEASE CODE. The SOFTWARE PRODUCT may contain PRERELEASE CODE that is not at the level of performance and compatibility of the final, generally available, product offering. These portions of the SOFTWARE PRODUCT may not operate correctly and may be substantially modified prior to first commercial shipment. ERM is not obligated to make this or any later version of the SOFTWARE PRODUCT commercially available. 5) SUPPORT SERVICES Support Services as described here are optional. Licensee may join the ERM Developer Network for a fee. Other than this service, ERM provides no technical support as part of this license.

6) U.S. GOVERNMENT RESTRICTED RIGHTS. The SOFTWARE PRODUCT and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the US Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Earth Resource Mapping Limited.

7) DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS. a) Rental. You may not rent, lease or lend the SOFTWARE PRODUCT. b) Software Transfer. You may permanently transfer all of your rights under this EULA, provided you retain no copies, you transfer all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity), and the recipient agrees to the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade, any transfer must include all prior versions of the SOFTWARE PRODUCT. c) Termination. Without prejudice to any other rights, ERM may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts. d) No Warranties. ERM EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE PRODUCT. THE SOFTWARE PRODUCT AND ANY RELATED DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU. e) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ERM OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ERM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ERM'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR US$5.00; PROVIDED HOWEVER, IF YOU HAVE ENTERED INTO A ERM SUPPORT SERVICES AGREEMENT, ERM'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU. f) This Agreement may only be modified in writing signed by authorized representatives of ERM. All terms of any purchase order or other ordering document shall be superseded by this Agreement. If any provision of the Agreement is found void or unenforceable, the remainder will remain valid and enforceable according to its terms. If any remedy provided

is determined to have failed for its essential purpose, all limitations of liability and exclusions of damages set forth in this Agreement shall remain in effect. g) This Agreement shall be construed, interpreted and governed by the laws of Western Australia. h) ERM reserves all rights not specifically granted in this Agreement.

### 12.1.3 MrSID

1008 Western Avenue, Suite 200 Seattle, WA 98104 Phone: 1-866-725-5211 or 206-652-5211 Fax: 206-652-0880

LizardTech Computer Software License Agreement for MrSID Decode SDKs revised 8 February 2010

THIS COMPUTER SOFTWARE LICENSE AGREEMENT ("Agreement") is entered into by and between CELARTEM, INC., an Oregon corporation doing business as LIZARDTECH, with a principal business address at 1800 SW First Ave, Suite 500, Portland, OR 97201 ("LIZARDTECH"), and a corporation with a principal business address at , Tel. No: , Fax No: ("LICENSEE"). The Agreement is effective ("Effective Date") as of the date executed by the later party to execute. RECITALS LIZARDTECH is party to a Technology Licensing Agreement with THE REGENTS OF THE UNIVERSITY OF CALIFORNIA ("UNIVERSITY") through which it has an exclusive right to commercialize technology for storage and retrieval of large digital images, originally developed at the Los Alamos National Laboratory ("LANL"), including all patent rights arising under U.S. Patent No. 5,710,835 ("Storage and Retrieval of Large Digital Images") and certain foreign patents pending ("TECHNOLOGY"). The U.S. Government and UNIVERSITY have certain reserved rights in the TECHNOLOGY as set forth in Appendix A of this Agreement. LIZARDTECH is engaged in the business of designing, developing, and marketing MrSID Decode SDKs (including the GeoExpress Decode SDK and the LiDAR Compressor SDK), computer software and related products arising from or developed based on the TECHNOLOGY ("DSDK"). DSDK consists of libraries that allow licensees to write software, or engineer a process, that enables end-users to view MrSID-formatted files and other supported file types. LICENSEE desires to design, develop or market software products that use DSDK to permit (among other functionality developed by LICENSEE) an end-user to view *. sid-formatted files and other supported formatted files ("Licensed Products"). In consideration of the premises and mutual covenants of this Agreement, LIZARDTECH agrees to license DSDK to LICENSEE for LICENSEE's use in the development of Licensed Products, and LICENSEE's subsequent sublicensing of DSDK with the Licensed Products pursuant to the terms and conditions which follow. 1. LICENSE GRANT 1.1 DSDK. In addition to the patent rights described above, DSDK and any and all associated media, printed materials, and "online" or electronic documentation provided with DSDK are protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties, and is licensed, not sold. LICENSEE shall not modify, reverse engineer, disassemble or decompile or otherwise seek to discover the source code or trade secrets of DSDK. LICENSEE shall not use DSDK to develop products or allow others to develop products that encode files into the MrSID format. 1.2 Development License Grant. LIZARDTECH hereby grants to LICENSEE a nonexclusive license to install DSDK on a reasonable number of CPUs in LICENSEE's own facilities only for its own internal use and development of Licensed Products that use DSDK to view *.sid files. 1.3 Distribution License Grant. Subject to the terms and conditions of this Agreement, LIZARDTECH hereby grants to LICENSEE a nonexclusive, worldwide, nontransferable right to distribute DSDK in object code format with the Licensed Products. For avoidance of ambiguity, LICENSEE is not permitted to distribute DSDK source code libraries. LICENSEE may distribute DSDK with the Licensed Products to sublicensees (and such sublicensees may further sublicense DSDK with the Licensed Products to other sublicensees) provided: (a) a notice regarding LIZARDTECH or its licensors' ownership rights shall be provided with the Licensed Products as set forth in Subsection 2.2 below; (b) DSDK shall only be sublicensed under license terms as set forth in Subsection 1.3(d) below and any and all distribution of DSDK with the Licensed Product does not cause, or could be interpreted or asserted to cause, DSDK to become subject to the terms of any Open Source license, including but not limited to the GNU Public License; (c) any and all distribution of DSDK shall not (i) create, or purport to create, any obligations for LIZARDTECH or its li-

censors with respect to DSDK; or (ii) grant, or purport to grant, to any third party any rights to or immunities under LIZARDTECH or its licensors' intellectual property or proprietary rights in DSDK; (d) DSDK shall be sublicensed to any and all sublicensees subject to a license agreement that provides LIZARDTECH and its licensors with the same protections and requirements as set forth in Sections 1.1, 1.3, 1.4, 2, 3, 4, 6.1 and Appendix A. For the avoidance of ambiguity, LICENSEE and any and all sublicensees may not sublicense DSDK separately from the Licensed Products. 1.4 Rights in DSDK. LIZARDTECH and its licensors retain all right, title to, and ownership of all applicable intellectual property rights such as patent, copyrights and trade secrets in DSDK (including Updates as defined in Section 2.3) and any associated documentation. Without limiting its rights in any way, LIZARDTECH hereby specifically reserves the worldwide nonexclusive right to develop, use, reproduce and distribute DSDK directly to other integrators, distributors and/or end-users. DSDK, including features and related information, are unpublished software, trade secret, confidential or proprietary information of LIZARDTECH or its licensors. DSDK is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), any use, modification, reproduction, release, performance, display, or disclosure of DSDK by the U. S. government shall be solely in accordance with the terms of this Agreement. Except as expressly permitted pursuant to Section 1.3, LICENSEE shall not rent, sell, lease, disclose or otherwise provide DSDK, associated documentation or any related information to any third party and will use such efforts, and in no event less than a commercially reasonable effort, to protect DSDK, associated documentation or related information as LICENSEE uses to protect its own trade secret, confidential or proprietary information. 2. OBLIGATIONS OF LICENSEE 2.1 Trademark Usage. LIZARDTECH grants to LICENSEE a nonexclusive, nontransferable limited license to use and display LIZARDTECH's trademarks, logos or other elements of its branding (collectively "Trademarks") in connection with LICENSEE'S license rights granted hereunder with respect to DSDK, provided, however, such use shall be subject to the terms in the Trademarks: Limited License provision located in the Terms of Use at www.lizardtech.com. LICENSEE agrees to abide by such terms and LICENSEE further agrees to monitor the above referenced Trademarks: Limited License provision for any updates or amendments to such terms. 2.2 Proprietary Rights Notice. LICENSEE must include in all Licensed Products all ownership/copyright, Trademark, trade secret and other intellectual or proprietary rights notices accompanying the Licensed Products in the About Box for all Licensed Products, or any other similar location where LICENSEE places proprietary rights notices pertaining to third-party software incorporated in the Licensed Products. At a minimum, LICENSEE shall include the following notice in all Licensed Products: Portions of this computer program are copyright Â© 1995-2010 Celartem, Inc., doing business as LizardTech. All rights reserved. MrSID is protected by U.S. Patent No. 5,710,835. Foreign Patents Pending. 2.3 Licensed Product Support and Updates. LICENSEE is responsible for the support of all Licensed Products. LICENSEE agrees to use commercially reasonable efforts to implement Updates provided by LIZARDTECH into the next scheduled release of the Licensed Products. LIZARDTECH may, but is not obligated to, provide such Updates to DSDK. Updates shall mean maintenance revisions that correct identified errors in, or provide bug fixes for DSDK and may also include support for additional file formats. 3. DISCLAIMER OF WARRANTIES. LIZARDTECH AND ITS LICENSORS PROVIDE DSDK "AS IS" AND WITH ALL FAULTS. LIZARDTECH, ON BEHALF OF ITSELF AND ITS LICENSORS, DISCLAIMS ALL WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO ANY MATTER WHATSOEVER RELATING TO THE DSDK, INCLUDING BUT NOT LIMITED TO ANY (IF ANY) IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF REASONABLE CARE OR WORKMANLIKE EFFORT, OF RESULTS, OF LACK OF NEGLIGENCE, OR OF A LACK OF VIRUSES, ALL WITH REGARD TO DSDK. THERE IS NO WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION, AUTHORITY, OR NONINFRINGEMENT WITH RESPECT TO DSDK. 4. LIMITATION OF LIABILITY. IN NO EVENT SHALL LIZARDTECH OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES FROM ANY CAUSE WHATSOEVER, WHETHER RESULTING FROM LOST PROFITS, DATA, USE OR REVENUE, OR FOR ANY INCIDENTAL, DIRECT, INDIRECT, SPECIAL, CONSEQUEN-

TIAL OR PUNITIVE DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. This limitation of liability shall apply regardless of the form of action whether in contract or in tort, including by negligence or any other basis. 5. TERM AND TERMINATION 5.1 Term and Termination. The initial term of this Agreement will commence as of the Effective Date of this Agreement. Either party may terminate this Agreement upon thirty (30) days prior written notice to the other party if the other party is in material breach of any provision of this Agreement and fails to cure such breach within the thirty (30) day period. Either party may terminate this Agreement without cause upon the delivery of thirty (30) days prior written notice of termination to the other party. 5.2 Obligations Upon Termination. Upon any termination of this Agreement, all licenses granted to LICENSEE shall terminate and LICENSEE must return to LIZARDTECH or destroy all copies of DSDK in its possession, custody, or control, whether modified or not. In the event this Agreement is terminated without cause, LICENSEE may (a) retain a reasonable number of copies of Licensed Products, not to exceed five (5) copies, for purposes of its internal use solely to facilitate customer support for existing customers only; and (b) exhaust its current stock of Licensed Products over a period of no more than thirty (30) days after the date of termination. Any remaining inventory at the end of such period must be destroyed. 5.3 Survival Provisions. Except as otherwise provided herein, the provisions of Sections 3, 4, 5.2, 5.3, 6.2, 6.3 and 6.6 of this Agreement survive any termination or expiration of this Agreement. 6. MISCELLANEOUS 6.1 Export Controls. LICENSEE shall not export or reexport DSDK or any direct product thereof without the appropriate United States or foreign government export licenses, notifications or approvals. 6.2 Governing Law, Jurisdiction and Dispute Resolution. This Agreement shall be governed by and construed under the laws of the State of Oregon, USA without regard to conflict of laws provisions. Any disputes under this Agreement shall be resolved either in the federal or state courts located in Multnomah County, Oregon, or under the Commercial Arbitration Rules of the American Arbitration Association in an arbitration proceeding to be held in Portland, Oregon. The prevailing party in any dispute under this Agreement will be entitled to its attorney fees. 6.3 Entire Agreement. This Agreement together with the attached Appendices sets forth the entire agreement and understanding of the parties relating to the subject matter herein and merges all prior discussion(s) between them. No modification of or amendment to this Agreement will be effective unless set forth in writing signed by officers of both parties hereto. 6.4 Notices. Any notice required or permitted by this Agreement shall be in writing and either delivered by hand or sent by prepaid, registered or certified mail, return receipt requested, or by nationally recognized overnight courier service, addressed to the other party at the address shown at the beginning of this Agreement or at such other address for which such party gives notice hereunder. Such notice will be deemed to have been given when delivered or, if delivery is not accomplished due to action or inaction of the addressee, when tendered. 6.5 Assignment and Binding Effect. LICENSEE may not transfer or assign its rights or obligations under this Agreement without the prior written consent of LIZARDTECH, except to a successor in interest or purchaser of all or substantially all of LICENSEE's assets which specifically assumes the obligations of this Agreement. LICENSEE will notify LIZARDTECH within ten (10) days of such event. Subject to the foregoing sentence, this Agreement will be binding upon and inure to the benefit of the parties hereto, their successors and assigns. 6.6 Partial Invalidity and No Waiver. If any provision of this Agreement is held to be invalid by a court of competent jurisdiction, then the remaining provisions will nevertheless remain in full force and effect. The parties agree to renegotiate in good faith any term held invalid and to be bound by the mutually agreed substitute provision. No waiver of any term or condition of this Agreement will be valid or binding on either party unless the same will have been mutually assented to in writing by an officer of both parties. The failure of either party at any time to enforce any of the provisions of the Agreement, or the failure to require at any time performance by the other party of any of the provisions of this Agreement, will in no way be construed to be a present or future waiver of such provisions, nor in any way affect the validity of an effort by either party to enforce each and every such provision thereafter. LIZARDTECH LICENSEE By: By: Name: Name: Title: Title: Date: Date:

APPENDIX A GOVERNMENT RESERVED RIGHTS 1. Los Alamos National Laboratory. Some of the TECHNOLOGY incorporated in the Software was developed in part through a project at the Los Alamos National Laboratory (LANL) funded by the U.S. Government, managed under con-

tract by the UNIVERSITY. The MrSID TECHNOLOGY, subject of U.S. Patent No. 5,710,835, is under exclusive commercial license to LIZARDTECH. The U.S. Government and the UNIVERSITY have certain reserved rights in the TECHNOLOGY as set forth in this Agreement. (a) The U.S. Government has a nonexclusive, nontransferable, irrevocable, paid-up license to practice or have practiced throughout the world, for or on behalf of the United States, inventions covered by the UNIVERSITY's Patent Rights, and has other rights under 35 U.S.C. Â§ 200-212 and applicable implementing regulations and under the U.S. Department of Energy (DOE) Assignment and Confirmatory License through which the DOE's rights in the TECHNOLOGY were assigned to the UNIVERSITY. (b) Under 35 U.S.C. Â§ 203, the DOE has the right to require LIZARDTECH to grant a non-exclusive, partially exclusive or exclusive license under the Patent Rights in any field of use to a responsible applicant(s) upon terms reasonable under the circumstances, if LIZARDTECH does not adequately attempt to commercialize the MrSID Technology. See, 37 CFR 401.6. (c) LIZARDTECH maintains a discount program for sales of Software to the U.S. Government or any agency thereof or any U.S. Government contractor who certifies that its purchase of the Software is for or on behalf of the U.S. Government. (d) The UNIVERSITY may assign its rights in its License with LIZARDTECH. (e) The UNIVERSITY makes no warranty or representation as to the validity or scope of its Patent Rights, nor that the Software will not infringe any patent or other proprietary right and has no obligation to bring or prosecute any actions for patent infringement to protect LICENSEE's use of the Software. The UNIVERSITY has no obligation to furnish any know-how, technical assistance, or technical data to LICENSEE. 2. Termination of LIZARDTECH Rights in TECHNOLOGY. Should LIZARDTECH's rights in the TECHNOLOGY under its license with the UNIVERSITY for any reason terminate during the term of this Agreement, such event will automatically operate as an assignment by LIZARDTECH to the UNIVERSITY of all LIZARDTECH's rights, title, and interest in the license. In such case, if LICENSEE is not in default of the terms and conditions herein, it may elect to continue this Agreement as an Agreement directly with the UNIVERSITY on the same terms.

### 12.1.4 OpenCV

Copyright (C) 2000-2019, Intel Corporation, all rights reserved.

Copyright (C) 2009-2011, Willow Garage Inc., all rights reserved.

Copyright (C) 2009-2016, NVIDIA Corporation, all rights reserved.

Copyright (C) 2010-2013, Advanced Micro Devices, Inc., all rights reserved.

Copyright (C) 2015-2016, OpenCV Foundation, all rights reserved.

Copyright (C) 2015-2016, Itseez Inc., all rights reserved.

Third party copyrights are property of their respective owners.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall copyright holders or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or

business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

### 12.1.5 NetTopologySuite

**Preamble**

NTS is derivative work of JTS Topology Suite.

**Project license**

Copyright (C) 2005-2018 NetTopologySuite team

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**JTS Topology Suite licensing**

As of version 1.15 JTS Topology Suite is released using the following licenses:

The Eclipse Foundation makes available all content in this project ("Content"). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of either the Eclipse Public License 1.0 ("EPL") or the Eclipse Distribution License 1.0 (a BSD Style License). For purposes of the EPL, "Program" will mean the Content.

If you did not receive this Content directly from the Eclipse Foundation, the Content is being redistributed by another party ("Redistributor") and different terms and conditions may apply to your use of any object code in the Content. Check the Redistributor's license that was provided with the Content. If no such license exists, contact the Redistributor. Unless otherwise indicated below, the terms and conditions of the EPL still apply to any source code in the Content and such source code may be obtained at http://www.eclipse.org.