

# Master Thesis

**Design and implementation of an automated workflow to  
provide a zoomable web mapping application using artistic  
styles**

in fulfilment of the requirements for the degree of

**Master of Science**

*Submitted by:* Maximilian Konrad Hartl

*Date of birth:* 30.12.1987

*Matriculation number:* 3516979

*Date of submission:* 25.08.2015

*Supervisor:* Prof. Dr.-Ing. habil. Dirk Burghardt





## **Task of Master Thesis**

**Course of Studies:** Cartography

**Name of the Graduand:** Maximilian Hartl

**Topic:** Design and implementation of an automated workflow to provide a zoomable web mapping application using artistic styles

### **Goals of this Study:**

Currently web maps are often used with standardised styles developed for orientation and navigation purposes. The aim of the work is to develop an automated workflow for the derivation of web maps with an individual, theme related look and feel. A literature review should include current approaches of non-photorealistic image rendering in computer graphics. Based on that, concepts for customised styles with either artistic or application related styling should be developed. An analysis of vector and raster based approaches should be carried out in regard to the derivation of custom styled maps depending on influencing factors such as geo data and image processing, geometry type, etc.

Aim of the practical work is the further development towards automation of existing interactive workflows for the derivation of maps with artistic styling. This requires the identification of currently necessary interactive processing steps during data preparation, preprocessing and image processing. In addition, initial investigations should be carried out to develop a processing chain for application related labelling. Furthermore a workflow for scale dependent rendering and map presentations should be designed. Final aim is the creation of an interactive web mapping application using standardised client-server architecture and a Web Map Tile Service.

There should be submitted two printed versions together with the digital version on CD. The digital version should include the text description and all required data and software to run the prototype. It is encouraged to publish the thesis on the publication server Qucosa of SLUB. The major findings will also be presented in the form of an A2 colour poster.

**Supervisors:** Prof. Dr.-Ing. habil Dirk Burghardt (TU Dresden)

**Beginning of thesis:** 27.1.2015

**Date of submission:** 30.6.2015

Prof. Dr-Ing. habil Dirk Burghardt



# Abstract

Although proprietary and free web map applications have become an important part of the daily life, individual map styling has been neglected for a fairly long time. With the latest possibilities of custom adjustment provided by many services and some interesting artistic experiments, this is about to change. In the context of artistic cartography and custom map styling, this work explores the possibilities of employing an automated process for the generation of WMTS compatible map tiles with an artistic styling. Web mapping standards and techniques of non-photorealistic rendering (NPR) are considered as well as traditional cartographic representations. Furthermore, existing vector- and raster-based processes are analyzed including an interactive workflow with the open-source image editing software GIMP, which is examined with respect to its drawing capabilities. Based on that, a concept for an automated rendering process is developed and influencing factors along with input parameters are discussed. An experimental automated processing is implemented using GIMP and its Python scripting interface to create single maps and seamless map tiles for the use in a WMTS application. Different drawing techniques of GIMP like brushes, dynamics and masks are applied during the rendering process. Geodata is taken from the freely available OpenStreetMap project and it is stored in a geodatabase. Furthermore, the GIS capabilities of the database are used to implement custom query procedures for the creation of seamless tiles, feature simplification and generalization that makes a preprocessing of the data unnecessary. Additionally randomization methods for the estrangement and abstraction of its geometry to emulate a hand-drawn appearance are created and tested based on SVG vector graphics and non-photorealistic rendering techniques. The rendering and abstraction results are evaluated and discussed regarding their contribution to an artistic appearance. The generated tiles are generated in a WMTS compatible way and presented in a web mapping application.



# Declaration of authorship

I hereby declare that the thesis I am submitting on the subject

*Design and implementation of an automated workflow to provide a zoomable web mapping application using artistic styles*

is entirely my own unaided work. All direct or indirect sources used are acknowledged as references.

Dresden, August 25th 2015

---

Maximilian Hartl





# Acknowledgements

I would like to express my deepest gratitude to Professor Dirk Burghardt for his guidance as well as critique and feedback. He was always an inspiration in the various discussions we had about my thesis.

Furthermore I would like to thank my friend Mariko for her support with the English language, my brother Ferdinand for his advice with many technical questions and Latex, and Sebastian for various discussions and tips. Additionally, I would like to express my appreciation to all the people who offered me a place to stay during the times I came to visit Dresden for my thesis. Thank you Aaron, Daniel and Rico for your hospitality and flexibility. And finally, I would like to thank all the people who showed their manifold support during my studies.

And last but not least, thanks to all the OpenStreetmap contributors and open-source developers. Without the data and many versatile tools, this work would have been much harder.



# Contents

<b>Task Description</b>	<b>III</b>
<b>Abstract</b>	<b>V</b>
<b>Declaration of authorship</b>	<b>VII</b>
<b>Acknowledgements</b>	<b>IX</b>
<b>Table of Contents</b>	<b>XIII</b>
<b>List of Figures</b>	<b>XVI</b>
<b>List of Tables</b>	<b>XVIII</b>
<b>Glossary</b>	<b>XX</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Objectives . . . . .	2
1.3 Structure of the work . . . . .	2
<b>2 Cartographic context and NPR</b>	<b>5</b>
2.1 Definitions . . . . .	5
2.2 Non-photorealistic rendering . . . . .	7
2.2.1 NPR in computer graphics . . . . .	7
2.2.2 Hand-drawn rendering . . . . .	11
2.3 Cartographic communication and visualization . . . . .	12
2.4 Cartography and art . . . . .	15
2.5 Web Mapping . . . . .	17

<b>3</b>	<b>Image editing and web map rendering</b>	<b>19</b>
3.1	Image editing software . . . . .	19
3.2	Raster image editing with GIMP . . . . .	20
3.2.1	Interactive editing . . . . .	20
3.2.2	Plug-ins and scripting . . . . .	28
3.3	Map rendering . . . . .	32
3.3.1	Web map rendering techniques . . . . .	32
3.3.2	Comparison of vector and raster based approaches . . . . .	35
3.3.3	An interactive workflow with GIMP . . . . .	37
<b>4</b>	<b>A concept for automated processing</b>	<b>41</b>
4.1	Comparison of influencing factors of a vector and raster based approach . .	41
4.2	Requirements and goals of automation . . . . .	43
4.3	Definition of input parameters . . . . .	45
4.4	Concept for an automated process with GIMP . . . . .	45
4.5	Technical components . . . . .	47
4.5.1	JSON . . . . .	47
4.5.2	OpenStreetMap . . . . .	48
4.5.3	Database environment . . . . .	49
4.5.4	SVG as exchange format . . . . .	50
4.5.5	Web map tile service . . . . .	51
<b>5</b>	<b>Implementation of an automated process with GIMP</b>	<b>55</b>
5.1	Geodata setup . . . . .	55
5.1.1	Data acquisition . . . . .	55
5.1.2	Data import and update . . . . .	56
5.2	Sketch rendering . . . . .	59
5.2.1	Displacement and randomization . . . . .	59
5.2.2	Hand-drawn emulation . . . . .	63
5.2.3	Hatching . . . . .	65
5.3	Automated processing . . . . .	68
5.3.1	Configuration files . . . . .	69
5.3.2	Script structure . . . . .	72
5.3.3	Feature Processing . . . . .	77

5.3.4	Image rendering . . . . .	82
5.3.5	Output and WMTS . . . . .	84
<b>6</b>	<b>Processing results and evaluation</b>	<b>87</b>
6.1	Results . . . . .	87
6.2	Limitations and potential . . . . .	93
<b>7</b>	<b>Conclusion</b>	<b>99</b>
7.1	Summary . . . . .	99
7.2	Outlook . . . . .	100
	<b>Appendix A Images and Tables</b>	<b>103</b>
	<b>Appendix B Compact disc content</b>	<b>111</b>
	<b>Bibliography</b>	<b>113</b>



# List of Figures

2.1	Tonal rendering examples . . . . .	8
2.2	Pen and ink rendering . . . . .	11
2.3	Sketchy style examples by Wood et al (2012) . . . . .	12
2.4	Artistic map abstraction and stylization by Isenberg (2013) . . . . .	17
3.1	GIMP Paint tool options . . . . .	22
3.2	GIMP brush types examples . . . . .	23
3.3	GIMP dynamics options . . . . .	24
3.4	GIMP dynamics example stroke . . . . .	25
3.5	GIMP path drawing . . . . .	26
3.6	GIMP mask example . . . . .	26
3.7	GIMP filter examples . . . . .	27
3.8	GIMP text drawing . . . . .	28
3.9	Schema seamless image creation with GIMP . . . . .	28
3.10	Mapbox map style examples . . . . .	34
3.11	Mapzen map style examples . . . . .	35
3.12	Stamen map style examples . . . . .	36
3.13	Embroid style by Dong (2015) . . . . .	38
4.1	Workflow of the concept . . . . .	46
4.2	WMTS Tile indexing grid . . . . .	52
5.1	Schema random curve controlpoints . . . . .	62
5.2	Schema line handy . . . . .	65
5.3	Polygon disjoint cut points . . . . .	66
5.4	Schema hatching . . . . .	67

5.5	Sketchy polygon with hatching . . . . .	68
5.6	Project files structure . . . . .	69
5.7	Workflow script structure . . . . .	74
5.8	Simplified UML diagram of the core rendering classes . . . . .	76
5.9	Workflow detail tiles . . . . .	78
5.10	Workflow detail map . . . . .	79
5.11	Tile bounding box buffering . . . . .	80
5.12	Polygon generalization schema . . . . .	80
5.13	Polygon generalization example . . . . .	81
5.14	Schema database functions . . . . .	81
5.15	Schema text rendering . . . . .	84
5.16	Schema tile naming . . . . .	85
6.1	Embroid tiles . . . . .	88
6.2	Chalk tiles . . . . .	89
6.3	Embroid map . . . . .	90
6.4	Sketchy map . . . . .	90
6.5	Text label rendering methods . . . . .	91
6.6	Text label rendering example . . . . .	92
6.7	Brush comparison . . . . .	94
6.8	Polygon union problem . . . . .	95
6.9	Polygon union problem example . . . . .	96
A.1	Full UML diagram of the rendering classes . . . . .	106
A.2	WMTS application OpenLayers . . . . .	108
A.3	WMTS application QGIS . . . . .	109



# List of Tables

2.1	Categorization of NPR techniques . . . . .	9
3.1	GIMP brush types . . . . .	23
4.1	<i>PostGIS</i> to SVG conversion . . . . .	51
4.2	Zoom level and tile coverage dependency . . . . .	53
5.1	osm2pgsql command arguments . . . . .	57
5.2	osm2pgsql database schema . . . . .	57
5.3	Methods for random points on line . . . . .	61
5.4	Random curve controlpoints . . . . .	63
5.5	Line jittering . . . . .	64
5.6	Lines handy . . . . .	65
5.7	Polygon jittering examples . . . . .	66
5.8	JSON configuration file top-level properties . . . . .	70
5.9	JSON configuration file <b>map</b> properties . . . . .	70
5.10	JSON configuration file <b>style</b> properties . . . . .	71
5.11	JSON style file top-level properties . . . . .	71
5.12	JSON style file common feature properties . . . . .	72
5.13	JSON style file <b>lines</b> feature properties . . . . .	72
5.14	JSON style file <b>polygons</b> feature properties . . . . .	73
5.15	JSON style file <b>stroke_line</b> properties . . . . .	73
5.16	JSON style file <b>stroke_hachure</b> properties . . . . .	73
5.17	JSON style file <b>polygons</b> text properties . . . . .	73
5.18	JSON style file <b>effect</b> text properties . . . . .	74
A.1	Random distribution examples . . . . .	104

A.2	Random point displacement . . . . .	105
A.3	Selected OSM line features . . . . .	107
A.4	Selected OSM polygon features . . . . .	108

# Acronyms

**API** Application Programming Interface.

**DPI** Dots per Inch.

**DTM** Desktop Mapping.

**GIS** Geographic Information System.

**GPS** Global Positioning System.

**GUI** Graphical User Interface.

**KVP** Key-Value Pair.

**LOD** Level of Detail.

**NPR** Non-photorealistic Rendering.

**OGC** Open Geospatial Consortium.

**OSM** OpenStreetMap.

**PHP** PHP Hypertext Preprocessor.

**PNG** Portable Network Graphics.

**SQL** Structured Query Language.

**SVG** Scalable Vector Graphics.

**UGC** User Generated Content.

**URL** Uniform Resource Locator.

**VGI** Volunteered Geographic Information.

**WMS** Web Map Service.

**WMTS** Web Map Tile Service.

**XML** Extensible Markup Language.

# 1 Introduction

## 1.1 Motivation

Digital maps are ubiquitous today. Newspaper websites and social media platforms feature them as well as mobile applications or navigation systems. But despite their various applications, possibilities for custom styling which is required by many users in various contexts have not kept up with this development. Not all applications benefit from customized styles of course, but some recent developments have shown the potential of such individual styling. The creation of individual styles is driven by many different forces and not only by people with an cartographic background. This demonstrates the relevance of map design for an actually designed product with an highly individual or artistic look-and-feel. These maps are not intended to communicate spatial information in a precise way, but to convey a different meaning that is created by the applied styling. Their main purpose is an appealing design that looks not computer generated but hand-made. Such maps can be used in a specific context that is related to a theme or topic. For pictures and photographs, image editing software provides many tools to modify existing images or create new ones with a specific style that can be created using different painting tools. These images can be crafted to be very close to hand-made created products. How these capabilities from image editing can be used and incorporated into a cartographic process of creating maps is therefore a relevant research question that should be examined in this work.

## 1.2 Objectives

The aim of this work is to implement an automated rendering workflow for the creation of artistic styles that can be displayed on map tiles in a zoomable web mapping application. Based on an existing interactive workflow and considering techniques from non-photorealistic rendering, a concept for an automated process should be proposed and implemented. As the existing interactive workflow is based on GIMP, the scripting interface of this software should be examined to create an experimental scripted processing. The processing should be scalable to any size of input data with as little as possible or no processing. This allows the continuous updating of the data for a long-term use. Important input parameters have to be identified and categorized in order to define them properly for the rendering process. The results have to be analyzed regarding their use in a tile based, seamless application. Furthermore, a solution to define the application of different stylings should be developed. In addition, the text rendering capabilities of GIMP should be examined regarding their styling options and potential for map tile creation. All of this should demonstrate that it is possible to render custom styled maps with an open-source image editing software.

## 1.3 Structure of the work

As an introduction in the topic, this work provides at first an overview of the field of non-photorealistic rendering in computer graphics and relates it to the classical cartographic context of communication and visualization. Furthermore, the connection of cartography and art is taken into account and various opinions on the topic are discussed. In chapter 3, the focus is set on the field of image editing and different image editing softwares are introduced. From this selection, the open-source software GIMP is picked and examined in detail regarding its interactive drawing techniques like brushes, dynamics or image masks as well as its scripting capabilities. In particular, the Python programming interface is taken into further examination. Subsequently, techniques of web map rendering, especially for custom styling are introduced with examples and different technologies are illustrated. These approaches are then compared taking their nature as being either raster- or vector based into account. Finally, an existing interactive workflow is analyzed and important

steps are extracted as a base for an automated workflow.

A concept for such an automated rendering process is then developed in the following chapter. First, influencing factors are identified with respect to a vector- or raster based approach. Secondly the Requirements of such a workflow are defined based on the general rules for web maps. Thirdly, a schematic concept is proposed and comprising technologies are introduced, such as JSON, OSM, a PostGIS geodatabase environment, SVG and the Web Map Tile Service specification. The following chapter 5 is then dealing with the actual implementation of the automated workflow. At first, the data environment setup is explained, including the acquisition of data and its import into the database storage as well as its updating. That followed, various techniques for the estrangement of vector geometry are introduced. Important functions for the displacement and randomization are explained as they provide the base for the emulation of hand-drawn geometries as well as hatching techniques. Eventually the automation process and its components are illustrated. Important configuration parameters are introduced and the structure of the Python script is clarified. Feature processing components which include the data queries and generalization steps are accounted for in detail before the actual rendering techniques are explained. To conclude, the structure of the output results and how tiles can be stored in a WMTS compatible way is contained in the last section of this chapter. The results are evaluated in the next chapter 6 which includes some examples, before limitations of the process and involved components are discussed. Based on that, a conclusion is given that sums up the results and gives an outlook on further research questions.





## 2 Cartographic context and NPR

### 2.1 Definitions

#### Web Mapping

With the establishment of the World Wide Web as a new communication medium, the field of Internet Cartography evolved from the classical cartographic disciplines including geographic information science. Client-server interactions have started to play an important role in modern cartography even for GIS applications. In addition, commercial web mapping applications like Google Maps have become an important part of a digital life. However, web mapping is strongly influenced by open standards and technology (Peterson, 2008). Today the development of web mapping is significantly driven by a large variety of mapping services that provide an Application Programming Interface (API) which are provided on a commercial and open-source base. Furthermore, the concept of Web 2.0, which is still in a state of continuous evolution, plays an important role today. With user generated content and other additional values provided, new sources for geodata as well as social information emerges (Gartner, 2009).

#### Web Mapping Services

It is a common way to store geographic information in a GIS databases. These infrastructures became increasingly popular since the 1980s. With their increasing use, a standardized technology was required facilitate the distribution and serve this data to end users displayed on a web map. For this reason, the Open Geospatial Consortium (OGC) defined a set of standards for the distribution of data in 1999. Standards for services were defined to supply geodata via the Web and extract information from the data stored on a server (Peterson, 2012). All mapping services include a standard set of functions to obtain

metadata and the actual data that is provided and served upon a client's request. As an OGC standard they are featured in most GIS applications and web mapping APIs.

### **Computer Graphics**

The field of computer graphics can be described as “any use of computers to create and manipulate images” (Ashikhmin et al., 2009). In general, the aspects of computer graphics can be categorized in three areas: (1) The modeling that deals with the mathematical description of geometric shapes in way that allows the storage in a digital format. (2) The rendering which handles the creation of images from the model and (3) animation which is a technique to create the illusion of motion using sequences of images. Furthermore, fields like user interaction, virtual reality, visualization, image processing 3D scanning and computational photography are related fields. A majority of applications employing computer graphics can be found in the fields of video games, visual effects, animated movies or information visualization (Ashikhmin et al., 2009). For the creation of digital maps, the capabilities of computer graphics technology have always played an important role. However, the focus in computer graphics is more on automated processes whereas cartographic visualization cannot always be fully automated (Jones, 2013).

### **Non-photorealistic Rendering**

Non-photorealistic computer graphics describes an area of scientific and technological interest that deals with the generation of images which appear to be handmade. These images are characterized by using randomness, ambiguity or arbitrariness rather than completeness and coherence. Even though it covers all aspects of computer graphics, it mostly involves the rendering process which is referred to as Non-photorealistic Rendering (NPR) (Strothotte and Schlechtweg, 2002). With such individual and abstract rendering, NPR has applications in many different fields like technical manuals or medical illustrations where non-photorealistic depictions offer a level of clarity or vitality that is difficult to cover with photorealism (Winkenbach and Salesin, 1994). Therefore NPR can either serve as an artistic abstraction or an efficient and concise simplification.

## 2.2 Non-photorealistic rendering

### 2.2.1 NPR in computer graphics

Photorealism has been the driving force behind computer graphics for a very long time. With images being rendered from three-dimensional scenes with a simulation of physics and light, they are assessed regarding how close to a photograph they are. Achieving acceptable results can be considered as the main aim of photorealistic rendering in computer graphics. Non-photorealistic rendering (NPR) however is following a different approach. Instead of depicting the information about reality as it is, the focus is on communicating the information contained in an image scene by creating an illusion of reality. Stimulating the human perception by stylization and communication is in the center of this rendering technique. This is achieved by adapting techniques that have been used by artists to the field of computer graphics to emphasize or expose specific features and leave out dispensable information. Whereas photorealistic images can be classified by a Level of Detail (LOD) non-photorealistic ones are classified by level of abstraction. The level of abstraction is aimed at the intended purpose which can either be an artistic appearance or a simplification to convey relevant information. This could for example be the NPR version of a medical X-ray or MRI image that should illustrate an injury for a layman without a background in medicine (B. Gooch and A. Gooch, 2001). Aside from the medical example, different purposes can be identified that constitute the need of a non-photorealistic rendered image (Strothotte and Schlechtweg, 2002). The first one is the simulation of human intelligence or the emulation of human facilities to create hand-drawn graphics. An example is the application of different drawing techniques like lines and hatching with one tool, e.g. a pencil to achieve a distinguishable effect. How to make the computer decide which technique to use goes into the direction of artificial intelligence (AI) research. Aside from this scientific challenge, the conveying of meaning through communication of information is one of the most important qualities of NPR. The superiority of such images has been proven by many studies but it has to be assessed whether viewers are able to perceive the meaning that was intended to be transferred by the image. Additionally, the relationship between language and pictures can be clarified. The usage of images instead of words rather than using images as supplementary to words is a subject that needs to be evaluated. Finally, NPR offers a range of opportunities for

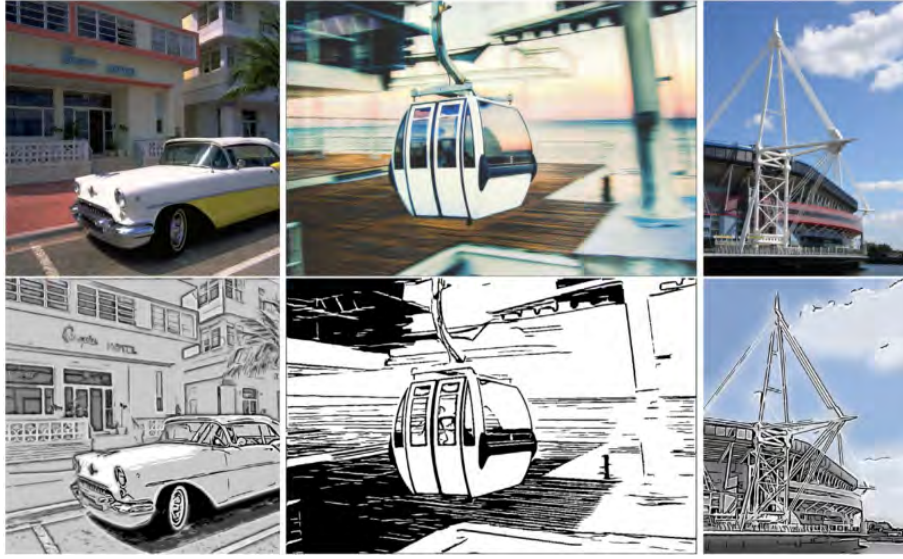


Figure 2.1: Tonal rendering examples by Rosin and Lai (2013)

new products and services for example for the emerging market of e-books. Many existing books feature hand-drawn graphics and scanning them for digital copies is not always a satisfying solution as they would lack possibilities for further editing or interaction.

According to Gooche and Gooche (2001) the research in NPR can be split into three main fields. The first is artistic media simulation which focuses on modeling physical properties of artistic mediums. This includes the medium itself, which could be described as the technique an artist employs to paint an image like oil, a pencil or watercolor. Additionally, an applicator is defined which resembles the tool that the artist uses to apply the medium on a substrate which could be a canvas for example. The second main field of research NPR is user-assisted image creation which deals with enabling a software user to interactively draw artistic images. No artistic skills should be required as the actual rendering techniques are controlled by a software. Artistic tools are emulated by a computer and natural components are added by random parameters for example. Thirdly automatic image creation deals with the automated creation of images that have a predefined communication goal. Example applications for a tonal artistic minimal rendering using lines and blocks are explained by Rosin and Lai (2013) some examples are shown in figure 2.1.

Tateosian and Healey (2004) describe NPR as a two-staged process. The general workflow is that an input imagery is chosen first which is then processed and rendered in some artistic style. The variety of NPR is based on the decisions that are made in these stages. The processing is not only depending on the style of art that is simulated but on the

Stylized lightning	Silhouettes and Edges	Pen-and-ink, Hatching and Engravings	Volume Illustration
Tone Shading, Cartoon Rendering	Edge Classification, Object Space Methods, Image Space Methods, Hybrid Approaches	Textures, Direction Fields	Traditional Paradigms, Alternative Approaches

Table 2.1: Categorization of NPR techniques by Sayeed and Howard (2006)

question if simulating the act of creating artwork, the physical behavior of the media or the visual characteristics of an artistic style in in the focus. Drawing with different brush-strokes can be considered belonging to the first category, whereas the second category covers the modeling of physical properties of the drawing tool. The third category is exemplified in the field of pen-and-ink drawing in NPR.

The techniques of NPR can be further distinguished into pixel or raster-based approaches and geometry based approaches. Whereas raster based approaches are applied in two-dimensional space, geometry based approaches are relevant for three-dimensional non-photorealistic rendering. However, many two-dimensional techniques are actually based on renderings from three-dimensional geometric data. Many applications of NPR are abstractions of existing photography or rendered three-dimensional scenes in computer graphics. An approach of an categorization has been made by Sayeed and Howard (2006) is shown in the following table 2.1.

Stylized lightning methods can be used for shading of technical or volume rendered illustrations or highlighting of colored surfaces. Silhouette and edge techniques are constructed from line strokes and can express information in a concise manner. These are used to influence perception, the aesthetic value or an imitation of an artistic human drawing style. Whereas edge and image space approaches are raster-based techniques using image-processing methods, object space methods are directly operating on the underlying 3D geometry. Hybrid approaches are combining both techniques. Pen-and-ink is the traditional human illustration method using only pen strokes and has been mentioned before.

For filling different applications of hatchings can be used which can be applied as stroke textures or direction fields to indicate the orientation of an object. For volumetric datasets, two approaches can be identified which are the feature enhancement through traditional volume rendering and alternative reconstruction techniques.

Semmo et al. (2015) analyze the aspects of NPR in three-dimensional cartographic spaces and visualization techniques for basic feature types are identified and categorized. These include buildings, water surfaces, green spaces, road networks and digital terrain models. Furthermore, a visualization pipeline is described that involves generalization of three-dimensional geodata. Different illustration techniques are discussed regarding their use for cartographic design and parameters and algorithms that can contribute to a hand-drawn cartographic rendering in 3D are presented. This demonstrates the potential that NPR has in cartography and its relevance to the traditional visualization techniques.

Based on these findings, scientific theories of art using NPR tools can be defined. The human vision is able to understand and interpret new styles of representation without effort. New stroking techniques can be constructed by skilled artists in a way that they are clear and unambiguous. Making use of that, messages can be transferred without information about the encoding (Hertzmann, 2010). However, it has been shown that users can very well distinguish between real hand-drawn and computer generated imagery. Therefore it is important that the creators of these images consider some rules in order to avoid ambiguity. They should know what they are aiming for with their work and be aware of the audience. Furthermore it is crucial to seek high quality results, to focus on NPR techniques that represent material properties, to know and work with the object models, to avoid obvious patterns and regularities and to pay attention to line and dot placement as well as the used tools and their parameter definitions (Isenberg et al., 2006). With the focus of NPR being on emphasis of certain features and their abstraction at variable ranges, Cartography in general can be considered as a NPR technique for the depiction of spatial data as well. Instead of displaying all available information, maps are usually a simplified selection of features that were selected from an actual image of an area or recorded using technologies like GPS. Communicating data is one of the main aims of maps, however the artistic stylization is a field that has not found too much respect in research yet.

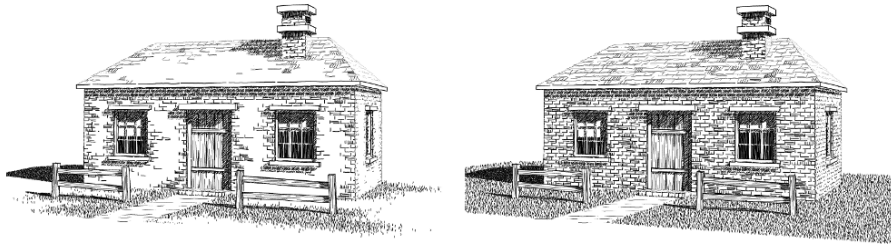


Figure 2.2: Pen and ink rendering by Winkenbach and Salesin (1994)

### 2.2.2 Hand-drawn rendering

The most famous hand-drawn rendering technique in NPR is probably pen-and-ink drawing. This approach has first been introduced by Winkenbach and Salesin (1994), defining algorithms and techniques to achieve a certain artistic styling. “Stroke textures” for the creation of tone and texture are used as well as “controlled-density-hatching” for free-form-surfaces. These renderings can be considered as simplification of the original image content and are often preferred in technical drawings. “Indication” can be used, a technique that focuses on rendering of details along existing features to make it more interesting (Tateosian and Healey, 2004). Figure 2.2 shows two renderings of a polygonal model with indication and without.

The generation of hand-drawn styles is not restricted to brush and stroke renderings though. Another approach is the use of textures in a 3D object space. It is based on texture generation and a pigment based lightning mode which has been proven as a suitable technique to imitate watercolor images using Perlin-Noise images even though it does not mimic the actual painting process very well (Lum and Ma, 2001). Artistic rendering effects can be applied to given images and achieved using sophisticated image processing algorithms for simplification. The result are abstract images that contain sufficient information from the original images. These images have been proven to be beneficial for the task of memorizing faces for example. However, a unique answer for artistic rendering cannot be given and the algorithm is influenced by various parameters. Aside from the tonal balance, the total number of tone and the retainment of general lines play an important role (Rosin and Lai, 2013).

A framework for a vector-based hand drawn rendering technique was developed and evaluated by Wood et al (2012). For the visualization of information, a sketchy rendering style

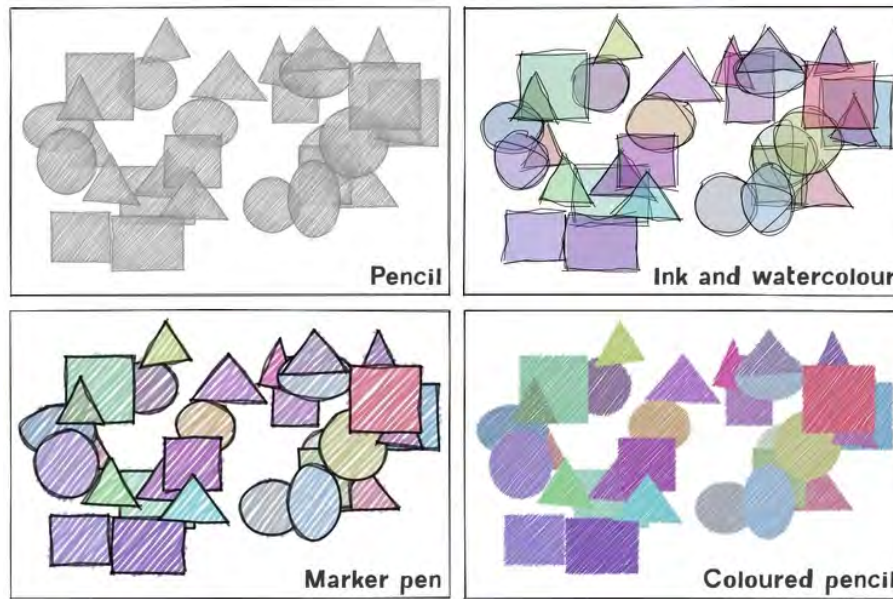


Figure 2.3: Sketchy style examples by Wood et al (2012)

was implemented as a library for the visual arts programming language Processing<sup>1</sup>. This style named Handy is applicable to drawing primitives like lines, polygons and ellipses. It is intended to enhance the idea of imprecision, the aesthetics and the narrative qualities of a visualization. This is of special interest for rapid digital prototyping of interfaces for example, where it is crucial to suggest that the design process is not yet finished. Furthermore, the “sketchyness” is considered as a visual variable that can be altered and carry information. Eventually, the a hand-drawn rendering can be a tool to set this style apart of ordinary computer generated designs and its visual appeal is used in various applications as it reinforces the perception of simplicity. An example of the Handy rendering technique is shown in figure 2.3.

## 2.3 Cartographic communication and visualization

Cartographic communication has always played an important role in cartography and modern trends have made the communication with maps more feasible. Nevertheless, a basic understanding of the processes and methods of cartographic model generation and the efficient and accurate communication of spatial data is required. This enforces the fulfillment of obligations for both, the creation of cartographic presentation forms and

<sup>1</sup><https://processing.org/>



the cartographic communication process. The map is the central information carrying element in the cartographic communication schema which describes the information flow going from the map maker via the map to the map user who is receiving a mental image of reality that influences his behaviour (Lechthaler, 2010).

With the automation in modern Cartography, new purposes of communication and visualization of spatial relationships have emerged. Computer graphics have played an important role in this process by providing opportunities for the interaction with graphic displays. This enables the user to define parameters for a visualization but requires awareness for cartographic design (Jones, 2013). However, the role of the designer is defined by abilities an automated process can not completely emulate. This includes the definition and structuring of a problem in a specific context as well as the evaluation and the stage of decision-making (C. H. Wood and Keller, 1996). Hence, no or only little guidance can be provided by the computer regarding the map design but the power of cartographic communication strongly depends on choices being made in terms of selecting parameters for symbolization, colors and many other aspects. An important role in map design play cartographic symbols and their effects on maps achieved by their manipulation (Jones, 2013).

In graphic symbology, symbols are classified according to the type of two-dimensional spatial object that is represented. This can be any of the graphic primitive types which are point, line and area. The representation of an object often depends on the level of generalization and can be modified to communicate different types of information (Jones, 2013). These graphic variables have been pointed out by Bertin (1983) who has established the cartosemiotic foundations that are still of major concern for modern Cartography. They include hue, lightness or value, size, shape, texture orientation and location in the context of point, line or area symbols. In the following paragraph, the variables with relevance to the non-photorealistic rendering of geodata are introduced and discussed.

Hue means the use of different colors as they are defined by name. It is the major variation of color, mostly used for the representation of qualitative information. A large number of different colors however may imply the risk of confusion for the map reader. Minor variations of color are understood to relate to similar categories. Another aspect of color is the lightness or value as it may be referred to. It describes a value that can be continuously varied from light to dark and is most suitable for the representation of ordinal and

numerical data. Too many different levels should be avoided as it is hard for the user to memorize them. They should be naturally allocated for interpretation if possible. Similar to the lightness, the saturation defines the actual value of color that is used or the color intensity. Size is predominantly used for the variation of point and sometimes also for line objects. It can vary between as a visualization of ordinal, numerical or nominal data, e.g. by relating the size of a symbol to an actual value that is represented. Applying variations in size for areas is not easy as it results in location distortion. Like the size, the shape is also mostly referring to point objects. Variations in shape are usually communicating information about the quality of an object at a location in the map. Textures or patterns on the other hand are of most use for area objects, though sometimes used for line symbols as well. It describes the internal graphical structure of an object which can be composed of a distribution of other graphic primitives. Examples would be hatching with lines as a area fill or a sequence of dashes for lines. The graphic variable of orientation can either be used to represent a qualitative attribute or a spatial property like direction. It is also mostly applied to point objects as the orientation of lines and areas is usually enforced by their location. This is the last of the graphic variables and can be considered as the one with the least amount of cartographic freedom. Variations are usually the result of map generalization or another projection that is being used (Jones, 2013).

With respect to the technique of NPR which was introduced in chapter 2.2, the graphical variables have to be evaluated from another perspective. Maps can of course already be considered as non-photorealistic depictions but the focus that NPR has on abstraction is not taken into account. The artistic abstraction or simplification of image data has a significant impact on the map rendering and the cartographic design. It is a striking characteristic of NPR that the graphic variables are mixed and strongly influenced by randomization. Most of the variables are therefore uncoupled from their traditional effects, meanings or initial use. The use of color hue is a method to create more abstract appearances of an actual map image. Variations in saturation and lightness can contribute to a rendering that is far from the natural properties. This could for example provide the styling for a drawn or comic like rendering. Lightness of a brush or area can simulate brush behavior that is depending on the applied pressure while drawing for example. The size is no longer used as a quantitative factor but as a random component that can suggest different drawing speeds or brush pressures as well. This relates more to small variations, strong variations may still be perceived as a qualitative attribute. Shapes of objects are

subject to randomization which can dramatically change the outline of areas for example. Variations in shape are no longer obligatory related to an objects attributes but may be intended to simulate a hand-drawn nature. Textures or patterns can be mixed or randomized for artistic or hand-drawn effects, for example hachure lines. The orientation is also another factor that contributes to a hand-drawn appearance by randomization and can strengthen the effects of a suggested artistic drawing method. Finally the location of an objects becomes less important as it can be randomly displaced. The amount of randomization or displacement is always supposed to be related to the degree of abstraction.

## 2.4 Cartography and art

There is a consent in Cartography that aside from technological and scientific approaches, the artistic approach has always played an important role as well (Cartwright, 2009; MacEachran, 1995). However, with the emerging of digital technologies and a resulting shift into the direction of scientific visualization, the aspect of art has been neglected in favor of technology (Cartwright et al., 2009). According to Field (2009), “a good map is a product of design and a pleasing map takes on an aesthetic appearance”. He argues that this originates from the experience people have with historic maps which were actually crafted by hand.

Today, the use of maps has become ubiquitous and with it map art as a part of graphic media. As maps are a prominent part of the daily life, they are for example extensively featured in newspapers as a medium for information. The flood of maps has introduced many aspects that are not originally connected with the making of maps and many map artists have been influenced by different kinds of geographic data visualizations. As a result, various artists have engaged the map as an expressive medium. Thus map art was established that has itself detached from an university-educated elite of cartographers (D. Wood, 2006). Furthermore, maps have made their way into contemporary art as an evidence for investigations that are communicated from the artist to the user. This is influenced by a crossover in mapping techniques that is greater today than it was ever before (Watson, 2009). Therefore Cartwright (2009) addresses that Cartography is different from other scientific disciplines as it can create products with “an art a technology or a

science 'flavor'". How to make art as relevant as the other fields is considered important to be brought up in the discussion about the direction of Cartography. The dualism of scientific and artistic aspects of Cartography has been discussed and summarized by Krygier (2000). He demands a critical view on the automation of map design as well as on the subjective view on Cartography as a craft which neglects the importance of methodologies that allows to create accurate and objective maps. This requires a reconsideration that is demanded to be accounted for by cartographers today.

The process of map design needs to be reconsidered in terms of aesthetics and visual expression which is strongly interlinked with artistic aspects. Aesthetics have been ignored due to their minor role in the process of cartographic design and its existence independent from geographic information. Decreasing cartographic standards as a result of the reduction to the output of a Geographic Information System (GIS) have been subject to many complaints (Kent, 2005). Nevertheless, "the art of Cartography is in the doing" (Field and Demaj, 2012), in the process of creating a map for someone. Therefore, what is really missing is the awareness for the human activity of design that is involved in map making. This is something that technology is not able to solve in the same way as a human being. Unveiling the purpose of a map product is communicated though the art of applying all elements of map design in a meaningful way. This can be summarized as map design being therefore the central aspect of cartography, comprising all scientific, technological and artistic aspects. Finally, this is embraced by the definition the International Cartographic Association has devised for Cartography. It is specified as "the discipline dealing with the art, science and technology of making and using maps"<sup>1</sup>.

An example of a map design process that focuses on map abstraction with applied aesthetics is given by Isenberg (2013) who describes a process of artistic map alteration. It is supposed to be an experimental play involving map abstraction and no precise cartographic depiction. However, it can be seen as a symbiosis of a technical and an artistic claim. Scientific techniques of data generalization need to be combined with illustrative techniques of NPR to achieve a result that can be considered as appealing in terms of aesthetics and art. A watercolor-like styling was chosen based on a substrate-simulation technique. Figure 2.4 shows an example of geodata that was rendered using this approach.

Art in cartographic representations has always be evaluated in the context of the map.

---

<sup>1</sup><http://icaci.org/mission>

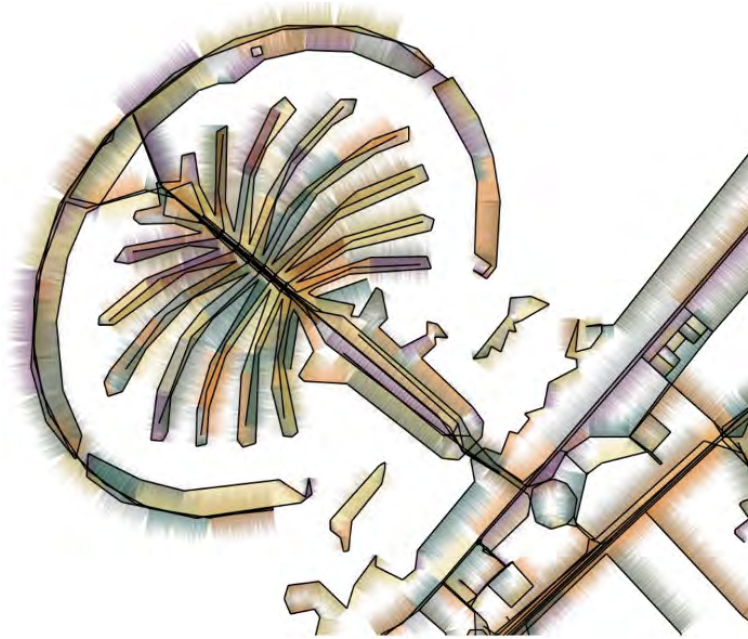


Figure 2.4: Artistic map abstraction and stylization by Isenberg (2013)

Sometimes it can support conveying a message from the map maker or map artist. However, sometimes the correct depiction of geodata is the main concern of a map and too much artistic styling can make it hard for the map user to focus on the actual content. Furthermore, abstraction always introduces an amount of uncertainty which might be desired but this effect needs to be considered seriously against the loss of information. With artistic abstraction being one of the aims of non-photorealistic rendering the application of art in Cartography can most certainly be considered as a rendering technique but Cartography by nature always includes abstraction that depicts the world as a non-photorealistic image.

## 2.5 Web Mapping

Web mapping can be defined as “the process of designing, implementing, generating and delivering maps, geospatial data and web map services on the World Wide Web” (S. Li et al., 2011). Maps and visualisations of geospatial data that are delivered via the Web require a different design and production approach compared to other digital screen maps or paper maps, the traditional cartographic medium (Cartwright, 2003). However, with the invention of the World Wide Web, the Internet, access to map data for end users has become much easier. Additionally, the term Web GIS is used for applications that

provide a set of functionality for spatial data analysis and data exploration (S. Li et al., 2011). The technology of web mapping or online mapping is strongly driven by Application Programming Interfaces (APIs) (Peterson, 2012). Many commercial map providers have developed APIs to integrate their data in a variety of web applications. When Google established their service Google Maps in 2005 and added a map-based search to their Internet search engine many other providers followed this approach.

Today a variety of map types exists from normal maps to satellite imagery and 3D-maps (Schmidt and Weiser, 2012). Many mapping APIs combine geographic data in so called mashups to create multi-scale panable maps interlinked with Web 2.0 content. Popular mapping APIs include the services from Google (Google Maps), the Bing Maps API or the OpenStreetMap (OSM) API. Furthermore, powerful JavaScript libraries exist like OpenLayers or Leaflet which provide more generic APIs to create client based web applications. These APIs have been evaluated in terms of their performance in previous work and one of the most significant differentiation criteria is most certainly the availability. Commercial providers all require payment for their services once you reach a certain amount of data traffic (Peterson, 2015). Furthermore, one is dependent on the companies infrastructures and the data may not be under your control.

Web 2.0 is another trend that is playing a role in web mapping. As Gartner (2009) defines it as Web mapping 2.0, the impact of this principle in cartography is obvious. An essential idea of the concept of Web 2.0 was to allow users more than just the retrieval of information. More characteristics are a rich user experience, user participation, dynamic content, metadata, Web standards and scalability. Furthermore, openness, freedom and collective intelligence by user participation have to be mentioned as well. This contributes to the development of the so called semantic Web with defined meta data that extends the functionality of applications on the Internet. In web mapping possible applications include spatial search engines, geotagging, or geoblogging and mashups created with different APIs (Gartner, 2009). Furthermore, OSM is a great example for spatial user-generated content (UGC) with a collaborative approach. It is set up as a free editable map of the world where users can upload collected geo data and add annotations to it (Cartwright, 2008).

## 3 Image editing and web map rendering

### 3.1 Image editing software

As image editing has many different fields of application, a huge variety of different software is available for the editing and creation of digital images. Many of these raster image based programs are used for the enhancement and digital editing of photographs and therefore feature a lot of methods to optimize photographic images. Operations that affect the color like balancing, contrast and brightness adjustment or the saturation are mainly used in editing as well as different filters based on image processing methods or artistic effects. But image editing software can also be used to create new images and the techniques for drawing with the assistance of a computer are also available in many image editing softwares. The scope of functionalities varies greatly from comprehensive and powerful applications like Photoshop<sup>1</sup> or GIMP to more rudimentary ones like Paint.NET<sup>2</sup> or Pinta<sup>3</sup>. Photoshop is probably the most famous image editing software, developed and published by Adobe and used in many commercial and academic applications. It features many editing and drawing functions and has many available plug-ins and a scripting interface. A license needs to be purchased from Adobe for the current version CC 2015 but the version CS2 which is more than ten years old is also available for free download. An open-source alternative to Photoshop that runs on all major operating system platforms is GIMP<sup>4</sup>. It is an acronym for GNU Image Manipulation Program and the software is looking back on a long history with the first version being deployed in 1995. Many devel-

---

<sup>1</sup><http://www.photoshop.com/>

<sup>2</sup><http://www.getpaint.net/index.html>

<sup>3</sup><http://pinta-project.com/>

<sup>4</sup><http://www.gimp.org/>

opers have contributed to the raster graphics editors for photo and image manipulation. Aside from GIMP, a lot of open source software is available for digital painting or drawing, e.g. Krita<sup>1</sup> or MyPaint<sup>2</sup>. These applications feature tools especially for the creation of digital images by drawing or painting on a screen canvas. The focus is mainly on artistic brushes and input drawing devices like graphic tablets are supported to create hand-drawn appearing graphics. However, these applications are optimized for on screen drawing and lack any interfaces for automation.

## 3.2 Raster image editing with GIMP

The source code for GIMP is freely available and the software is included as a standard application in many Linux distributions. It is published under the General Public License (GPL) which allows users to access and alter the source code without restrictions. GIMP provides a wide range of capabilities that include using it as a paint program for image creation as well as image manipulation tasks like photo retouching, image composition, and image conversion. A large number of drawing tools, image processing and color filters are available and the software is equipped with powerful layer ordering and masking features. Furthermore it provides functionality for batch processing and it can be expanded by a variety of plug-ins that are also freely available or custom scripts that are accessing the internal GIMP functions<sup>3</sup>. Having such an advanced scripting interface made the software the choice for the aim of this work as it is relatively simple to create an extension using the available functionalities. The working principle of the interface will be analyzed in section after the interactive drawing editing are presented.

### 3.2.1 Interactive editing

The following section focuses on the different interactive editing capabilities of GIMP. These can be performed by a user inside the graphical user interface on new or existing images. For the aim of this work, there are particularly techniques which will be examined:

---

<sup>1</sup><https://krita.org/>

<sup>2</sup><http://mypaint.intilinux.com/>

<sup>3</sup><http://docs.gimp.org/2.8/en/introduction.html>



- Drawing techniques which include the use of painting tools and brushes
- Filters and masks that affect the entire image or a selected area
- Text editing
- Creation of seamless images

### **Drawing techniques**

Painting tools are the most important tools for interactive drawing. They include different stroke painting tools like the pencil, the paint brush or the airbrush which make use of different brush types. Also tools for cloning and erasing are provided, as well as tools for lighten- or darkening, smudging or gradients. In contrast to pixel value modifications like filters that are applied to the entire image or selected areas, the painting tools enable the user to modify the color of desired pixels on the canvas. All tools can be employed using an input device like a computer mouse or a graphics tablets and applied to an image that is opened in the application's canvas for editing. The user is able to define drawing properties like the color, brush size or other variable parameters in in the tool options dialog which is shown in figure 3.1.

To draw with a painting tool, it is essential to have a brush that defines how pixels are colored along the painting trajectory. A brush is defined in a separate raster image file called pixmap and parameters like size, color, spacing and drawing behavior are set in the toolbox options of the paint tool that is using the selected brush. The depiction of the brush is painted on the canvas at the position of the drawing tool pointer each time the the input device's control button is clicked. To be painted repeatedly along a movement trajectory the control button has to be continuously held down during the movement of the stylus across the image. A set of brushes comes with the default installation of GIMP and custom brush image files are stored within the application data folder of the operating system's user directory where brushes from other sources (e.g. websites) can be stored as well. All brushes are editable in GIMP and new brushes can easily be created by drawing a greyscale image with different levels of black between zero and 255 defining the opacity of the brush and serving as a substitute to the color that is selected later in the toolbox. There are different types of brushes existing in GIMP which are different in their composition. All brush types are compared in the following table 3.1 and example

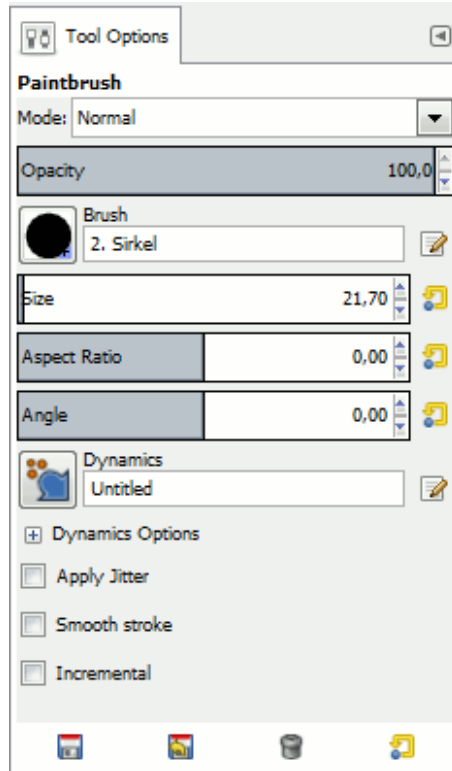


Figure 3.1: GIMP Paint tool options

strokes using the GIMP standard brushes are demonstrated in figure 3.2.

Brushes can be further parameterized by using Dynamics which define the rendering behavior of a brush when it is drawn along a trajectory. They provide the possibility to modify the brush appearance and apply a more “realistic” looking rendering to a brush stroke. It is for instance possible to let the size or opacity of the brush vary according to the pressure that is applied to a pen on a graphics tablet or to the speed of the mouse pointer. Combinations of available brush effects like size or opacity and defining parameters like pressure or velocity are displayed in a table as a mapping matrix inside the Paint Dynamics editor dialog inside GIMP. Each column in this table represents a stylus action except for the fade and random options. For each combination, a designated curve that can be modified for a custom adjustment of the rendering effect exists. These dialogs are shown in figure 3.3.

For the case that a drawing device like a graphics tablet is not available to get decent input effects, GIMP comes with the option to emulate the Dynamics when stroking along a predefined path. Emulating simulates a natural movement of the stylus with varying pressure at the beginning and the end of a stroke or different drawing speeds depending

Type	Description	Format
Ordinary brushes	Pixmap brush with greyscale values substituted by selected color	.gbr
Color brushes	Similar to ordinary brushes but works with a raster image that is drawn as it is	.gbr
Image hoses/pipes	Animated brush, can consist of multiple image layers. Parameters for the layer rendering, like random selection of single layers can be set	.gih
Parametric brushes	Created from basic geometric shapes with the GIMP brush editor, parameters like radius, hardness angle can be set	.vbr

Table 3.1: GIMP brush types



Figure 3.2: Brush type examples (ordered as in Table 3.1) using the GIMP standard brushes Oils 01, z Pepper, Chalk 03 and Star

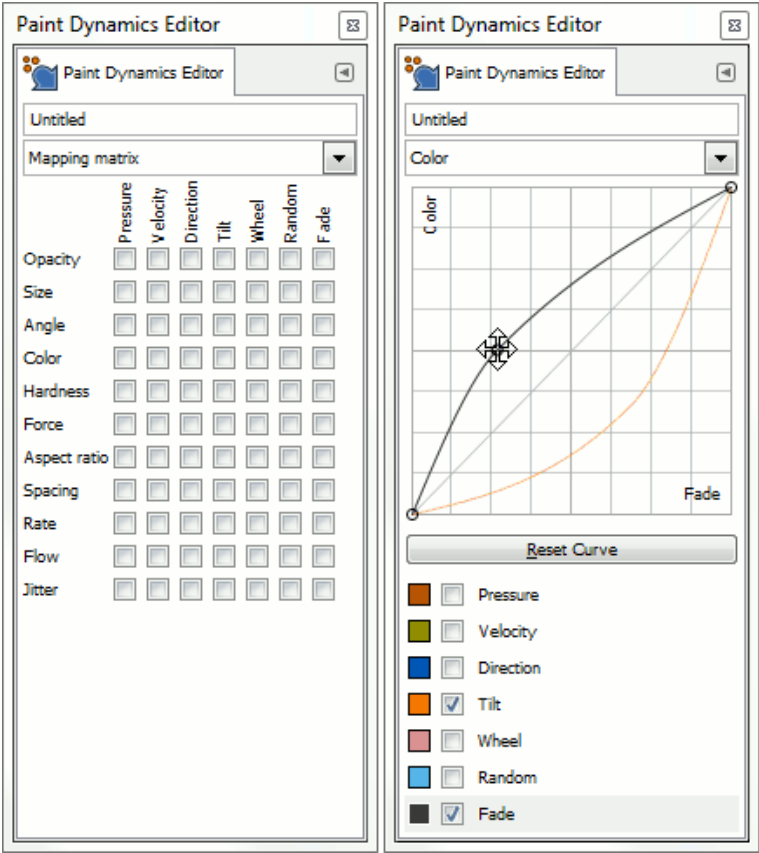


Figure 3.3: GIMP dynamics options



Figure 3.4: GIMP dynamics example stroke with brush Chalk 03

on the curvature of the stroke line. The process of creating such kinds of paths will be explained in the next paragraph. A range of dynamics are already available in a GIMP standard installation and creating Dynamics by oneself requires some practice and experience. In figure 3.4, a sample stroke is depicted using the standard Dynamics called “Basic Dynamics” which is available in a standard GIMP installation. For this illustration, the brush Chalk 03 which is also the third in figure 3.2 is used but with emulated Dynamics applied.

For the drawing of shapes, GIMP provides two options. Basic geometric shapes like rectangles or circles can be drawn using the selection tools for rectangles or ellipses. Other shapes can be drawn using the Free Select Tool which allows the user to select any area on the image. Both approaches are working on a raster base and it is possible to define a outline stroke using either a solid color or a brush as well as a fill color. To each shape that was created with a selection tool, a fill and a stroke can be applied. The fill can be selected from the fore- and background color or an available fill pattern. Stroking offers two options, either stroking as a simple line with a specified line width or stroking using a paint tool like the paint brush. The latter also provides the possibility to emulate brush dynamics.

It is also possible to draw vectors inside GIMP using the Paths Tool which also allows the creation of Bezier curves. Paths that are drawn on the canvas are stored in a layer that is separated from the image layers. It is also possible to create paths from the raster selections that have already been introduced. Furthermore options exist for exporting the created vectors into a Scalable Vector Graphics (SVG) image file and import vectors from a SVG file generated with a software that creates standard SVG graphics. The stroking options for paths are the same as for selection shapes. Figure 3.5 shows the original vector path, stroking using the line option and stroking using the paint tool option with the



Figure 3.5: GIMP path drawing: vector path (left), line stroke (center), Brush stroke with emulated dynamics (right)



Figure 3.6: GIMP mask example

Chalk 03 brush and emulated brush dynamics.

### Masks and filters

All these shapes can be drawn into different layers and a common way to define which segment of a layer should be finally rendered is using layer masks. A mask is defined by the area of a layer that has non zero raster cell values. This layer is connected to another underlying image layer and as a result only the raster cells of the underlying layer are rendered in the final image which have a corresponding value in the mask. This is a everyday tool in image editing, especially for photo editing and allows the creation of creative image compositions by using it for selective colorization for example. This use of layer mask is illustrated in figure 3.6.

Another important feature of GIMP is the application of filters on entire images or selections. Filters are raster-based operations which employ methods of image processing to achieve a specific effect. GIMP has many different filters pre-installed and more can be added through plug-ins. Most filters are actually implemented as a plug-in. Filters are available from the Filters menu inside the GIMP user interface and ordered into categories. For artistic effects, GIMP has an own category that implies filters for cartoon,

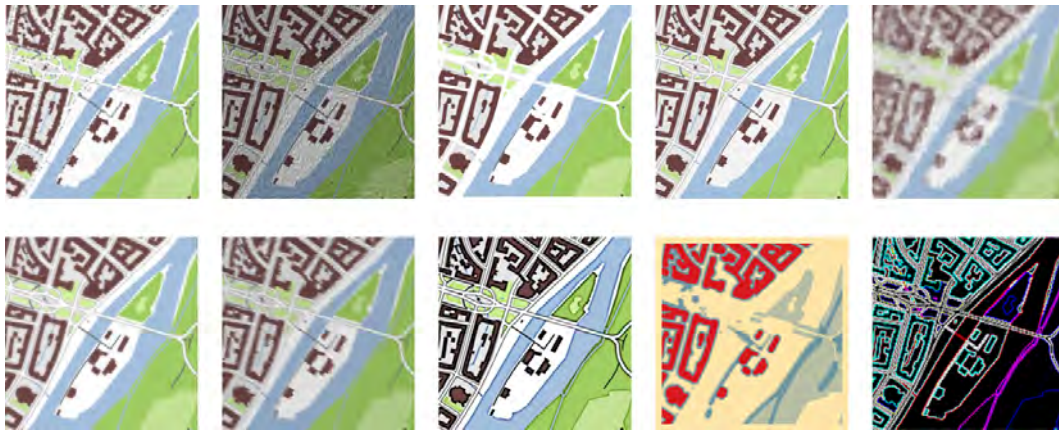


Figure 3.7: GIMP filter examples (top row, left to right): Cubism, Clothify, Oilify, Canvas, Weave (GIMPpressionist); bottom row, left to right: Van Gogh, Embroidery (GIMPpressionist), Cartoon, Obama Hope, Predator.

oil or glass effects on the one hand. Some filters on the other hand are adding a texture that suggest a classic artistic canvas like structured paper. Most filters are working with a random component that produces an irregular appearance for the effect. For this reason, however, they are not deterministic and not reproducible. Figure 3.7 shows some examples of GIMP’s artistic filters.

### Text

GIMP has a text tools that allows to place text directly on the canvas in a separate layer inside a rectangular frame. Options are available on a toolbar to change the font, font size or the alignment of the text. The typed text is automatically rendered in a pixel raster. An important feature of text layers is that their outline can be converted to a vector path. This offers the same editing possibilities as for shapes which is described in a previous paragraph. Image 3.8 shows an example text frame and the resulting outline paths.

### Seamless images

For the rendering of tiles it is utterly important to have seamless images which can be positioned repeatedly adjacent to each other. This contributes to a borderless texture that can be used over all tile images without visible margins. GIMP also provides an option to achieve that. The first step is to apply an offset to the image (grey) which is a half of its width in X- and a half of its height in Y-direction. This results in a

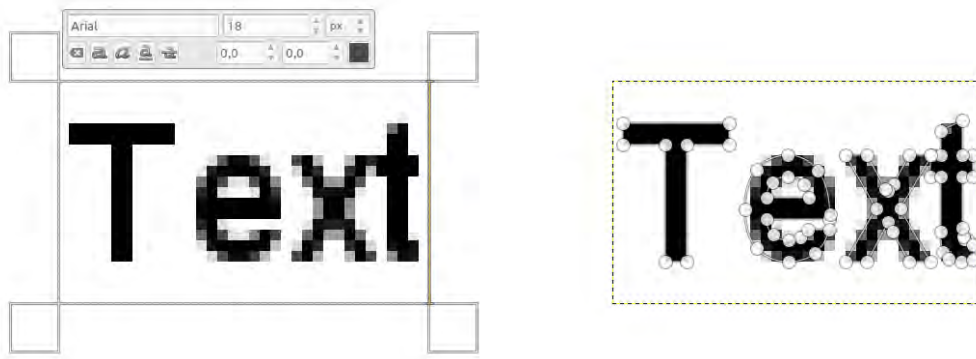


Figure 3.8: GIMP text drawing

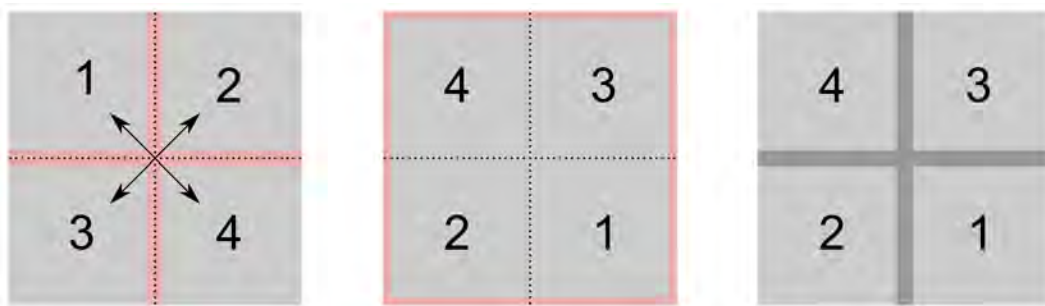


Figure 3.9: Schema seamless image creation with GIMP

rearrangement of the four image parts that are shown in figure 3.9 in their initial (left) and new position (center). These image parts are the result of splitting the original image along a horizontal and vertical centerline which is represented by the dotted line. After this operation is finished the edges that were on the inside first are now on the outside and are seamlessly fitting with other versions of the original image that have been processed the same way. However, there are new visible transitions inside the new image now along the split lines (right). This area (dark grey) has to be edited manually e.g. with the clone tool to make them less obvious or even hide them completely which does require some effort and practice.

### 3.2.2 Plug-ins and scripting

As comprehensive as the interactive possibilities of GIMP are, so is its plug-in interface. The flexible architecture of GIMP allows a simple integration of extensions and the available functions can either be automations that are triggered from the user interface and via a console or plug-ins and scripts written in a high-level programming language that



can perform complex image processing tasks. In fact, many of the functionalities that are available through the GIMP user interface are actually plug-ins. They are implemented in C as GIMP is itself or as scripts which do not require compiling. Scripts can be written in Python, Perl or Scheme, a Lisp dialect and experimental packages are also existing for Ruby and C#. Many plug-ins developed by other users can be found online at the official GIMP plug-in repository or at other websites.

For this project, the Python script interface of GIMP was chosen which is called Python-Fu. It offers almost the same scope of functionalities as the C plug-ins without the need of diving into the entire software's architecture. The Scheme interface which is called Script-Fu and the most popular way that GIMP scripts are created was not used because Python is more versatile and does also provide packages for database access and geometry processing which are used in the script. It is also much faster than Scheme. The only drawback of the GIMP Python module is its setup on a Windows operating system as it is not bundled with the standard GIMP installation package. However, as the implementation was carried out in a Linux environment, required packages are easy to install additionally (Peck, 2008).

The Python-Fu interface consists of a set of Python modules acting as a wrapper to the GIMP Python package called *libgimp* for Ubuntu Linux. It is imported via the *gimpfu* module at the beginning of a script file and enables access to the GIMP module and the procedural database registry (PDB). The GIMP module provides procedures and functions like constructors, configuration information and other operations as well as GIMP object types. These include image objects and methods which operate on so called drawables like channels and layers as well as other functions. The procedural database is a registry of GIMP and plug-in procedures that can be accessed. All procedures are listed in the Procedure Browser that can be explored from the GIMP Graphical User Interface (GUI). A Python-Fu script always has the same structure which consists of the import of the *gimp* module, the definition of the main method and the register function which connects the script function with GIMP. Along with the registration parameters which include information like the plug-in name, the author or the menu path, input parameters can be defined in this part as well. Arguments like an existing image, a specific drawable or other variables can be passed to the script for operating with it. All common variable types like Integer, String or Boolean are available aside from the GIMP objects.

The following code shows an example “Hello World” Python-Fu GIMP script which takes an input String that has “Hello World” as its default value and prints it to the console:

```
1  #!/usr/bin/env python
2  from gimpfu import *
3
4  def echo(input):
5      print "echo:", input
6
7  register(
8      "console\_echo",
9      "A console echo test",
10     "A console echo test taking an input String parameter",
11     "Test Author",
12     "Test Author",
13     "2015",
14     "<Toolbox>/Xtns/Languages/Python-Fu/Test/Console Echo",
15     "",
16     [(PF_STRING, "input", "Input String", "Hello world")],
17     [],
18     echo
19 )
20
21 main()
```

First the register function is called that connects the script with GIMP. It takes the following parameters:

- Name of the plug-in for external use (command-line)
- Short description\*
- More detailed description\*
- Author\*
- Copyright\*
- Date\*
- Menu path

- Image types\*
- Parameter list (parameters are defined by type, name, display name and default value)
- Return values
- Function to call inside the script as the main function

\* optional (can be left blank)

After that, the *main* function is called which runs the plug-in. The function defined in the register function is serving as the entry point which is the *echo* function in this case. Plug-ins and scripts have to be stored in a directory that GIMP is aware of. This can either be the default directory in the system's user directory or any other location that is known by GIMP. Additional locations can be added to GIMP in the Preferences menu and are scanned for files after a restart of the program. Scripts can then be executed either from inside the GIMP GUI or from the system console command-line interface. The example script could be started inside GIMP from the menu path that is defined in the second row of the register function. A dialog would then appear that allows the user to enter a value for the *input\_string* parameters. A command line call that would invoke the script and print the String *"This is a Python-Fu script"* on the console is for example:

```
1 python-fu-console-echo RUN-NONINTERACTIVE "This is a Python-Fu script"
```

### Comparison of functionalities

The functions and procedures that are available are very comprehensive and most of the interactive drawing techniques from chapter 2.2 have an equivalent scripting function. However, there are some limitations in scripting that need to be mentioned. As GIMP is an open-source software, it is under continuous development and not yet complete. Some of the scripting procedures are behaving unpredictable, for example the path stroking function did not work properly with the applied styling in GIMP version 2.8.12 which was the stable release at the time this project started. Therefore version 2.8.14 was used but this version still does not implement interface functions for the emulation of brush dynamics which is available from the user interface since version 2.6 and is explained

in a previous section. With this function, a more irregular and hand-drawn appearance of image elements, especially stroke outlines would be possible and an improved artistic rendering could be achieved in the future. Furthermore, the layer capabilities of the user interface are more flexible than the scripting procedures which do now allow renaming of layer groups for example. This can be useful, especially in a project like this with a highly complex layer structure.

### 3.3 Map rendering

The rendering of cartographic information is one of the most important fields of cartography. All cartographic tools used in Desktop Mapping (DTM) or in the context of GIS software provide a wide range of styling capabilities mainly based on vector data. While these tools are primarily used by users with an expert background in the geographic field, many of the techniques used for the rendering of web map applications are more diverse and end user friendly. Some of these techniques will be introduced in the following section.

#### 3.3.1 Web map rendering techniques

At the beginning of the development towards web map applications, the styling of the services was all based on pre-rendered raster tiles (Schmidt and Weiser, 2012). Today, a variety of rendering techniques exist that allow a custom styling by users or developers. Taking standards that are defined by the Open Geospatial Consortium into consideration, Styled Layer Descriptor (SLD) has to be mentioned for the styling of geo data that is provided as a Web Map Service (WMS). WMS is another OGC standard that generates a map in an image format via the Web. The markup language SLD is based on Extensible Markup Language (XML) and used to symbolize the map layers of an WMS (Cerba and Cepicky, 2012). Different styles can be defined and it is possible to enable the map user to select which style should be used in a mapping application. The style information can be stored on the server along with the geo data or sent by the client for a WMS request (Burdziej, 2011). However, the application of SLD for a WMS requires a more sophisticated geo data infrastructure running a server software like GeoServer and can be considered as software that is mainly used in a cartographic or GIS context for the visual

representation of map data.

Also providers of proprietary map data like Google offer possibilities to apply an individual styling to their maps for the end user. This allows users to modify the presentation of the standard Google Maps base map by changing the visual display. Level of variation differ from selecting various base maps to a style syntax which is available from the API. Basic visual parameters like the fill or stroke color of specified map features can be defined using this syntax<sup>1</sup>. This is possible since Google introduced the use of vector tiles in 2010 for their mobile mapping API for performance reasons and today it is used for all of their map products (Peterson, 2014). Another provider for custom maps is CartoDB which also offers different base map styles. But the actual styling possibilities are restricted to the thematic content which can be added to own projects through their web interface or by employing an API.

An approach that is based on the technology of vector tiles is implemented by Mapbox, an “open source company” that provides a mapping platform for “anyone to design and publish custom maps”<sup>2</sup>. Mapbox developed a specification for these tiles based on Google’s data exchange format Protocol Buffers<sup>3</sup>. Featuring this technology, custom map styles can be defined using CartoCSS, a map styling language also developed by Mapbox to define the visual appearance of a map in a similar way like Cascading Style Sheets (CSS) is used for web pages. With Mapbox Studio, an open-source desktop application is provided to generate vector tiles with a custom styling defined in CartoCSS. Geo data served by Mapbox which is based on OSM data can be used along with custom data from various sources. The resulting tiles are uploaded to Mapbox and presented in a web map using their Javascript API and the Mapnik renderer which is also used for the OSM project. Users are paying for the amount of styles they create and different data storage capacities but a free pricing plan with limitations is also available for test cases (Mauldin, 2015). Many map styles are available from Mapbox itself which can be used as base maps for own projects and also extended or modified. They also include some artistic styles like a comic-like or pencil drawn appearance. Some selected styles are shown in figure 3.11 which contains images of the same area of the inner city of Dresden (*Mapbox map styles* n.d.).

---

<sup>1</sup><https://developers.google.com/maps/documentation/javascript/styling>

<sup>2</sup><https://www.mapbox.com/about>

<sup>3</sup><https://www.mapbox.com/developers/vector-tiles/>



Figure 3.10: Mapbox map style examples

Another provider of custom styled web maps is Mapzen, also open-source mapping company. Their vector tile service is also based on Mapbox vector tile specification but with Tangram they are using their own map engine based on OSM data. The maps can be displayed as two- or three-dimensional and some of their demo styles also include artistic approaches. These are displayed in images shown in figure 3.11, again for the same area of Dresden (*Mapzen map styles* n.d.).

A famous example for a map with a very strong artistic effect is provided by Stamen, a design studio that is specialized on interactive design and data visualization projects, including maps<sup>1</sup>. It is the watercolor map, which features a styling that imitates a watercolor painting. The tiles for the map are created in a raster-based process that is described on their website<sup>2</sup>. The source data is rendered as raster images using Mapnik which are then processed using masks, image processing operations and different filter techniques. It is not described how the data is stored, however due to the fact that the single tiles can be obtained from an Uniform Resource Locator (URL) and the depicted geographic

<sup>1</sup><http://stamen.com/studio>

<sup>2</sup>[http://content.stamen.com/watercolor\\_process](http://content.stamen.com/watercolor_process)



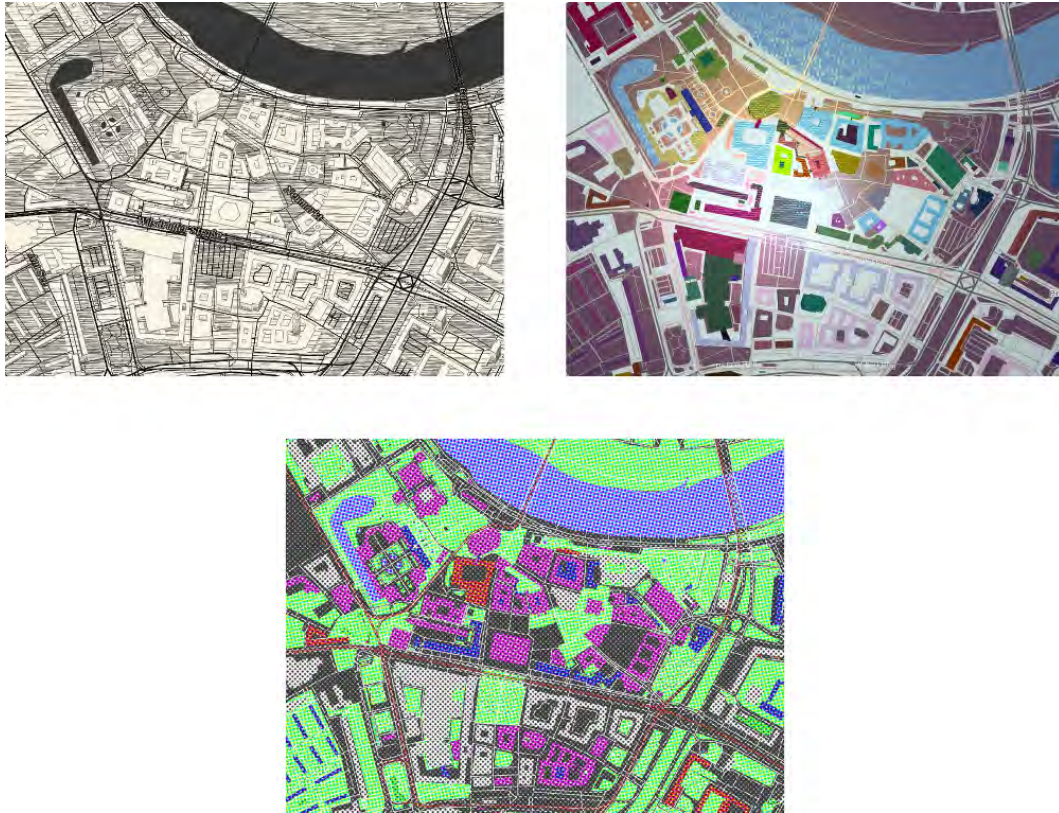


Figure 3.11: Mapzen map style examples

data is not up-to-date with the current OSM data it can be assumed that the tiles are pre-rendered and provided as a Web Map Tile Service (WMTS). Along with the watercolor style, Stamen provides another set of tiles which is called Toner. Example images of both styles can be see in figure 3.12 (*Stamen map styles* n.d.).

### 3.3.2 Comparison of vector and raster based approaches

Geographic data that is not satellite imagery is most commonly stored as vector data. So the source data of web maps can be considered to be stored in a vector format. The different methods of representing this data in a web application that were already discussed are now compared in this section.

The technique of using vector tiles for the rendering of web maps has a number of advantages compared with raster-based approaches which are indentified by Gaffuri (2012) and summarized in the following list:

- Users can retrieve thematic and semantic information for map objects



Figure 3.12: Stamen map style examples

- Geoprocessing can be performed
- Map content can be personalized
- Cartographic methods like generalization can be performed on the client side
- True integration of data from different servers is possible (no layer mash-ups)
- Improved interaction between data providers and data users
- No preprocessing/caching of raster tiles necessary, support for “live” data
- New innovative cartographic visualization techniques

Another advantage is the separation of text from the underlying layers. This allows for easy switching between labels displayed in different languages as it is implemented in Google Maps for example (Peterson, 2014). However the greatest disadvantage of vector tiles is performance. Factors such as client device memory or processing and connection capacities need to be considered. This is the main reason why pre-rendered raster tiles have been preferred for a long time. However, with improving technological capabilities and maturing digital mapping methods, vector tiles are becoming an increasingly acceptable approach. Nevertheless, existing frameworks are mainly dealing with raster data and there is a need to establish standardized and integrated approaches to support efficient vector web mapping (Gaffuri, 2012).

Pre-rendered tiles do not consume a significant amount of performance as they are generated in a separate process. But storing pre-rendered tiles has high demand regarding the capacity for server-side storage. It is almost impossible to store a full-scale map of the entire world as pre-rendered raster tiles (Peterson, 2012). Furthermore the rendering



of the tiles can be a time-consuming process that can involve complex image processing operations. Also, the rendered images are representing the data at the time the image was rendered. An update of a tile requires the entire rendering process to run again.

A hybrid approach is the rendering of raster images from vector data directly on the server once a tile is requested. This method allows to serve image tiles only upon request and at the current state of the data. For performance reasons, tiles can be cached so they are not created each time a user requests an area but are stored in a cache on the server (Sample and Ioup, 2010). This is a technique that is also implemented for the OSM map tiles in the *mod\_tile* PHP Hypertext Preprocessor (PHP) module along with the map rendering software Mapnik.

In conclusion it can be said that all three rendering methods have their use case. However, with the ongoing development towards infrastructural improvements like computation capacities and Internet bandwidth and emerging technologies that are pushed further by uprising mapping companies like Mapbox, a trend to vector tiles is obvious. Nevertheless, the most individual styling can be achieved by applying elaborate image processing operations. The creation of the watercolor map from Stamen is a striking example for this postulation.

### 3.3.3 An interactive workflow with GIMP

A concept for an interactive workflow to create custom styled maps with GIMP was developed by Dong (2015) at the Institute of Cartography at the TU Dresden. An approach is described how to create a map with a styling that imitates Chinese embroidery. This embroid styled map is derived from OSM-based Shapefiles provided by Geofabrik that were preprocessed with the proprietary GIS software ArcGIS applying selection, classification and categorization operations. Following this step, the geo data is transformed into the SVG format using the open-source software TileMill which is provided by Mapbox. A preliminary styling is also applied to the different SVG layers at this stage to get a raster image with a color fill of the polygon areas. These areas are used for the definition of image masks later. Finally, the SVG layers are imported into GIMP and rendered using the path stroking or mask functions. Before the final rendering, some preparations have to be done in GIMP. At first, custom brushes were created to imitate a needle-like stroking of lines as well as custom brush dynamics. Furthermore, images for the background and



Figure 3.13: Embroid style by Dong (2015)

polygon fill are produced beforehand making use of the custom brushes (Dong, 2015). A final version of the map is shown in figure 3.13.

The embroid styling demonstrates the feasibility of artistic map rendering with GIMP and is using the drawing techniques that were introduced in the previous section 3.2. However, as described in the previous paragraph, it requires many manual and interactive processing steps to carry out a map creation. Especially the fact that the map creation is taking place in a multi-software environment with three different software tools makes the interactive processing long and cumbersome. Storing the original geo data in a Shapefile format has several disadvantages. First, it is necessary to obtain Shapefiles from a data source, in this example OpenStreetMap. Secondly, only selected features and areas are already preprocessed and freely available on the Internet, e.g. from Geofabrik. Nevertheless, using Shapefiles as input format enables the processing to make use of other geo data sources than OSM if the data is available in this format. The preprocessing, which is carried out using ArcGIS could also be performed with an open-source tool like Quantum GIS (QGIS) that does not require to purchase a software license. The geometry of the geo data is not processed at all which would be necessary for small scale maps that require some sort of simplification to avoid cluttered outlines for example. TileMill is currently no longer in active development by Mapbox and was succeeded by their new software Mapbox Studio. This software offers the same styling possibilities as TileMill along with some new features and could be used instead. However, the intermediate step of using TileMill could be

avoided by making use of the “Path to Selection” function in GIMP and using the SVG exporting capabilities in QGIS for example. Finally, there is no text displayed on the map that is related to the content or linked to the original data. Text can only be added manually in GIMP using the text tool that renders any text the user is typing onto the canvas.



## 4 A concept for automated processing

The main goal of this work is to develop a automated process for the derivation of artistic styled maps based on principles of non-photorealistic rendering. For this purpose, a concept is developed in the following chapter which takes the aspects that have been discussed previously into account. Influencing factors of a possible vector or raster based approach are examined, requirements and goals are specified and input parameters are identified. Based on that, a schema for a concept employing the open-source software GIMP is introduced followed by an explanation of key technologies that support an implementation of the postulated concept.

### 4.1 Comparison of influencing factors of a vector and raster based approach

At first, several factors that influence a possible automated rendering process for the derivation of custom styled maps need to be considered. Based on the examination of map rendering techniques in section 3.3.1, aspects of a vector and raster based approach are taken into account and compared in the following sections.

Geographic data that is used in maps is usually stored as vectors as it is a common standard format in geo science. Geometric calculations and spatial operations can be performed well on vector data employing mathematic functions and topological relations can be defined. Furthermore, many tools exist to store and manage vector data e.g. using GIS software and a large number of sources for geographic vector data exist. Geographic raster data is either a depiction of visible or non-visible natural characteristics in satellite or aerial imagery or derived from vector data that was rendered as an image using a certain styling for the geometric primitives points, lines and polygons. Unless a processing is based on

raster data derived from natural influences, a vector to raster conversion has to take place first to generate raster data. For all data, the data quality is an important issue. This includes aspects of accuracy, completeness and level of detail.

The further processing steps are highly distinguishable as well. Vector data on the one hand allows a direct modification of the source geometry. Therefore generalization or simplification processes are based on vector operations. However, vector data can serve as a skeleton for subsequent raster based operations like stroking with defined styles or even brushes. This is often used in applications of non-photorealistic rendering as seen in section 2.2. For a raster based approach, image processing operations could be employed, which are available in many different software packages and offer a large variety of techniques as it makes up an entire field of science itself. Some examples include, filters like Gaussian-blurring, erosion and dilatation for a raster based generalization or more sophisticated techniques like the application of a Perlin-Noise image (see link to the processing of the Stamen Watercolor map, section 3.3.1). An advantage of raster based processes is that they are deterministic unless they include a random component, like some GIMP filters or NPR techniques for example. Deterministic processes are especially important for the generation of tiles and mosaicing them together. Randomization would require the option to define a seed which is possible in many programming languages but is not always implemented for operations that are available in software products.

Taking the geometry types into account, vector data allows for a more distinct styling as it is possible to define more specific rendering techniques and style options. Furthermore, the appearance of vector data is easier to change and principles from the variations of graphic variables can be employed easily. In contrast, Raster based approaches allow only pixel related operations and therefore geometry types cannot be efficiently distinguished during the rendering process. For raster images, it is a practical technique to apply pixel based masks rather than geometry type specific rendering. Masks can then serve as a selection of pixels to which a filter or simply a fill can be applied. Text rendering is not included in many applications but also an important issue for maps. A label is usually associated with a certain feature and placed on the map image according to some constraints. This is usually a vector based method where a text is placed along a path, e.g. street names.

In conclusion, vector based approaches can be considered as more specific and allow more processing options. This might however, also require more processing steps and resources

than image processing operations that can have a better computing performance. As the end product of a rendering process is always a raster image, keeping to a raster processing chain might also be reasonable. A direct vector based rendering of map tiles is possible (see Mapbox, section 3.3.1) but common standards for such an approach have still to be devised yet for the geospatial world. To achieve an artistic rendering in a process that is based on the scripting interface of GIMP, a combination of aspects from both approaches is applied. The data is stored in a vector format and converted into raster graphics using the raster based drawing techniques of GIMP. The processing is mainly operating on the vector data but raster based operations are used for post-processing e.g. for label placement.

## 4.2 Requirements and goals of automation

Based on the influencing factors of an automated processing, requirements can be defined that characterize a possible automated workflow, employing the capabilities of GIMP. These requirements are summed up in the following list and solutions are proposed in the subsequent paragraph:

- Non interactive processing
- Styling options
- Tile creation for deterministic mosaicing
- Artistic appearance
- Available geodata
- Data processing capabilities (selection, simplification, randomization)
- Platform independence (e.g. for server side implementation)

A non-interactive processing can be achieved employing the scripting interface of GIMP. Using the Python-Fu capabilities, the rendering steps of the manual processing (described in section 3.3.3) will be reproduced. Variable styles should relate to GIMP functionalities like color, stroke size and used brush. Parameters like graphic variables and different mask images should be possible to be defined by these options. Tile creation is the most crucial point in the rendering process. For a satisfying mosaicing of the generated tiles, their boundaries must not be visible in the final map image. This can be achieved by paying

attention to the following aspect: the rendering of each tile must always take adjacent tiles into account, thus the rendering of the content must be deterministic for each tile. A possible solution to this problem is the concept of metatiles which is implemented in GeoWebCache <sup>1</sup>. GeoWebCache is a tiling server that is integrated in GeoServer, an open-source server software for publishing geodata as WMS for example. This is however, mostly used for the label placement and for the rendering with GIMP, a simple buffering around the requested area of the tile is also sufficient. Adjacent geometric features are incorporated in the tile rendering process and guarantee seamless transitions between tiles. At this point it needs to be mentioned that no drawing functions of GIMP can be used that have a random component like some filters or parameters for dynamics. An artistic appearance can be achieved using GIMP as it provides a variety of drawing techniques that have an artistic application. The example of the interactive process can already be interpreted as an artistic rendering approach and using different styling parameters, a variety of artistic renderings are possible to implement. Geodata used for a rendering process should be easy to obtain, store and being used in a public application. Therefore, no legal constraints in working with the data should exist to limit the freedom of modifying and publishing the data Online. Keeping in mind the scalability of a web application, the data should also be available for the entire world with an adequate density. For the data processing, GIS capabilities can be used which allow for common cartographic operations like selection or simplification. Possible randomization techniques need to have a defining component that can make them reproducible for the tiling problem mentioned before. Finally, platform independence is guaranteed by using GIMP, a software that is available for all common platforms including its scripting capabilities.

Taking these requirements into account, the goals of an automated process can be defined and will be listed in the following. First, an implementation with GIMP should be achieved, using the scripting interface. Second, variable styling should be realized using input parameters that are defined in a human and machine readable format and relate to the GIMP functions. Third, the generation of seamless tiles should be possible that can be incorporated into a Web Map Tile Service and finally, a test of GIMP's text rendering capabilities should be included. To compare the automation with the interactive process, two modes for rendering are desired. One for web map compatible tiles and one for the rendering of a single map, as it is created in the interactive workflow.

---

<sup>1</sup><http://geowebcache.org/docs/current/concepts/metatiles.html>



### 4.3 Definition of input parameters

Based on the foregoing investigations, the input parameters necessary to control an automated processing can be defined. These input parameters have to be stored in a human and machine readable format so that it can be modified by the user and parsed by the automation script. In general there are two categories of input parameters that can be differentiated, parameters for the scripting and parameters for the styling.

Scripting parameters cover all input arguments that are required for the scripting procedure. These include specifications like the spatial extent, defined as a bounding box with upper left and lower right geographic coordinates, inside which geodata should be rendered. Additionally, the scale for single maps or the desired zoom levels for tiles are defined. Furthermore, the database connection credentials should be stored as well to avoid hard coding them inside the script. It is also important to have the selection parameters for geodata features provided for the database querying but these have to be associated with the styling parameters already. They must also include information about the zoom level at which a feature should appear in a map or not.

The styling parameters are stored along with the feature rendering information separately and define the rendering style. Styling parameters encompass all parameters that are essential for the rendering. This contains general styling specifications like the name of the style or whether the geometry should be abstracted or not. Furthermore, a background image for the map can be set. For the features, the GIMP related parameters can be defined. Each feature type class has a distinct set of parameters that can be specified. These include settings like color, brush size, brush name and other drawing options for geometric primitives. Polygons can be defined to be either rendered filled with the mask on a predefined image, with a fill color or a customized fill that can consist of hatching with lines.

### 4.4 Concept for an automated process with GIMP

Having the influencing factors, requirements, goals and input parameters defined, a concept for the realization of an automated processing featuring GIMP can be proposed based on the findings previously discussed in this chapter.

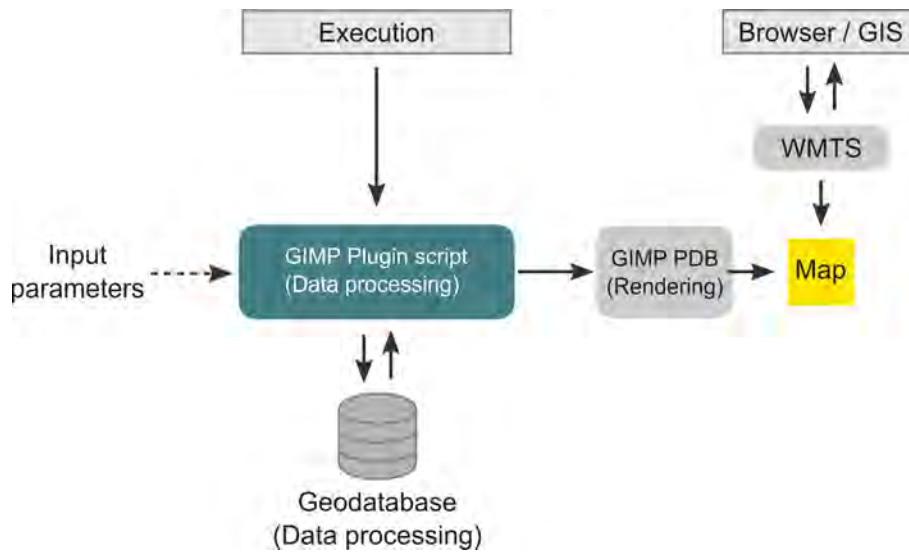


Figure 4.1: Workflow of the concept

As the main component of an non-interactive rendering process with GIMP, a Python-Fu plug-in script manages all important operations. Input parameters must be specified and available in a data format that is both, human and machine readable. Users should be enabled to set different parameters for variable styling and the processing script needs to be able to parse these. The script needs to have access to a specified data source that can provide usable geodata in an exchange format that GIMP can interact with. Based on the available geodata and the input parameters, a single map or map tiles should be rendered using the GIMP interface functionalities. At this point different techniques for the rendering of tiles or single maps must be implemented, taking into account that tile extents need to be calculated from a tile grid scheme to be georeferenced. The geodata should be obtained from a GIS enabled data source (geodatabase) and processed using cartographic (generalization) and non-photorealistic (abstraction) operations. With this setup the NPR techniques are restricted to vector based sketching and brush stroking techniques. The final map is either composed of line and polygon geometries that are rendered into a single map which is intended for using in print or just on screen, or seamless tiles that can be served on the web via a standardized service that allows the use of the data inside a web application. Figure 4.1 illustrates the structure of the concept that was described in this paragraph. For an analysis of the text rendering capabilities of GIMP, an experimental processing should be able to print feature-related texts on the map canvas.

To realize the implementation of such an automated process, the following technologies and specifications are chosen for an implementation that employs a vector based approach with the GIMP scripting interface:

- JSON as exchange format for the configuration
- OpenStreetMap (OSM) as a data source for geographic vector data
- PostgreSQL and *PostGIS* for the data storage with data processing capabilities
- SVG as interchange format for the geometry of geographic data
- The Web Map Tile Service (WMTS) standard for the serving of tiles

Nevertheless, the processing does include manual steps as well but these are reduced to a minimum. At first, custom brushes and seamless textures have to be created by hand in GIMP as described in 3.2. Second, the configuration files have to be edited manually and thirdly, the import and update of data requires interaction. These steps could be scripted, though. Finally, the results have to be transferred on a server with an WMTS installed which may not be necessary if the rendering is already performed on a server.

The components of the proposed technical environment are introduced and described in the following sections and an experimental implementation is described in the next section.

## 4.5 Technical components

In this section, practical solutions for all components that are involved in the conceptual design except GIMP are introduced and justified.

### 4.5.1 JSON

For a human and machine readable parameter configuration, the Javascript Object Notation (JSON) is a suitable data interchange format. As such it can be defined as a text format that is used to exchange data between different platforms. It is independent from any programming language and similar to XML which is also used for data exchange (Bassett, 2015). However, JSON has some significant benefits when it comes to performance and data storage. It needs less code than XML to define elements which reduces the

file size quite significantly. In addition its performance when being parsed is much faster compared to XML due to its universal data structure. It is based on literals containing a collection of name and value pairs, similar to an object, list or array in various languages<sup>1</sup>. All these reasons make JSON a suitable format to configure the input parameters of an automation script. JSON files can be edited by users to modify these parameters and they can simply be parsed as input arguments to a code file. Furthermore, it is possible to validate JSON files based on a schema to test if it has all necessary attributes.

#### 4.5.2 OpenStreetMap

As a source for geodata, OpenStreetMap (OSM) is often chosen by many academic and commercial projects. OSM is considered as the main driving-force in challenging the traditional geodata business. It was launched in 2004 and is often referred to as “the Wikipedia map of the World” (Ramm et al., 2011) as the collection of geodata is no longer limited to specialists from a field related to geosciences. Data can be collected using portable satellite navigation devices like the Global Positioning System (GPS) which is an inexpensive technology that is available to many people with the wide-spread use of smartphones today. The goal of the project is to set up a free editable map database covering the whole world and make the data available under an open content license. In fact, the Open Data Commons Open Database Library (OdbL) is used which allows to copy, distribute, transmit and adapt the data, as long as an obvious credit to OSM and its contributors is added<sup>2</sup>. For researchers, it offers an unique dataset covering the entire world (Arsanjani et al., 2015). The data that is collected is considered as Volunteered Geographic Information (VGI) which is a special form of User Generated Content (UGC). UGC is greatly driven by the advances of the Internet, social media, smart device technology like smartphones and telecommunication. It has an impact on how people are interacting today via the Internet and the most popular forms include messaging and social media content like blogs or videos (Mooney and Corcoran, 2013). However, even though the OSM data is collected in a collaborative manner there are some shortcomings with this kind of data that need to be take into account. First of all, there is no assurance of the data quality. OSM features are described by users using tags of key-value pairs and there are no strict rules how they have to be used or in which

---

<sup>1</sup><http://json.org/>

<sup>2</sup><http://www.openstreetmap.org/copyright>

combination. This is part of the collaborative principle that OSM is founded on (Ramm et al., 2011). The crowd is supposed to act as the controlling mechanism to validate the data. Furthermore, the positional accuracy and the completeness of the data can vary significantly and mechanisms for quality assurance have been a subject of research where a more detailed overview is provided (Goodchild and L. Li, 2012).

Since OSM data is used in many other projects that are related to this work and introduced in section 3.3.1, it was decided to use it for this project as well. The free and simple access to the data is considered as advantage which makes it easy to reproduce the processing that is explained in this work on other platforms. Many tools exist to work with the data and perform transformation or filtering processes which makes it a sufficient resource for such a project. Furthermore, using the data in an application that is more focused on an artistic representation rather than a precise depiction of geodata, OSM data can be considered as suitable. For the processing, only line and area features were used which provide a basic content of a map. These cover the road network, rivers and major areas of landuse, including buildings.

#### 4.5.3 Database environment

*PostGIS* is a commonly used tool and also used by the OSM project itself on the server together with the Mapnik OSM tile renderer <sup>1</sup>. It was initially developed by Refrations Research and is now maintained by the Open Source Geospatial Foundation (OSGeo). As an extension to the Structured Query Language (SQL) database language PostgreSQL, it adds common GIS functionalities to edit geometry and perform spatial operations. Furthermore spatial measuring can be carried out as well as spatial relationship analyses and all geometry data is spatially indexed. The range of functionality is completed by customized functions that can be implemented in the PL/PGSQL query language and stored in the database which are performed on the database server. Many of the functionalities are adopted from or based on other projects like the Geometry Engine Open Source (GEOS) or the Geospatial Data Abstraction Library (GDAL) (Obe and Hsu, 2015).

As *PostGIS* is supported by OSM and many applications that deal with the data, it provides a good solution to store the data and make GIS operations available. In this project, it is used for exactly these two reasons. Exchange formats to transfer obtained

---

<sup>1</sup><http://wiki.openstreetmap.org/wiki/PostGIS>

OSM data to a spatial database are existing and the capabilities of *PostGIS* meet all the requirements that were stated earlier for the data processing concept. Furthermore, it could also be used as a storage for other datasets than OSM as most common geospatial formats can be converted into a *PostGIS* database.

#### 4.5.4 SVG as exchange format

With GIMP featuring the import of vector data in the SVG format, it has to be considered how to make use of this data type efficiently. SVG is a markup format that is based on XML to describe vector graphics. It is an open-source standard of the W3C consortium and supported in all major web-browsers today. Many applications exist for the creation or editing of SVG data, like Adobe Illustrator or Inkscape. The coordinates for the image geometry are stored in a screen coordinate system which means that the origin is on the top left. This is a difference to most geographic coordinate systems but the coordinates can be easily transformed (Lienert et al., 2012).

For the realization of an automated workflow, the geodata from OSM needs to be converted from geographic coordinates to screen coordinates. Furthermore, the coordinates defining a geometry can either be stored in absolute coordinates relative to the screen coordinate system or in coordinates relative to the first point of the feature. The latter is used to reduce the actual disk size of large geometry (*SVG W3C recommendation* n.d.). A conversion can be achieved using *PostGIS*, which was introduced in the previous chapter. The SVG geometry is always returned as a path which is generally a representation of a shape. However, a path can have different defining command elements which determine its representation. SVG paths returned by *PostGIS* are either a straight line ( $L = \textit{lineto}$ ) for (multi-)lines or a closed line ( $Z = \textit{closepath}$ ) for (multi-)polygons that start at a defined position ( $M = \textit{moveto}$ )<sup>1</sup>. Table 4.1 compares the well-known-text (WKT) representation of *PostGIS* and a SVG geometries (examples from Wikipedia<sup>2</sup>).

For the geometry estrangement that was used to achieve a hand-drawn abstraction, the paths are converted into curves. These are defined by the curve command ( $C = \textit{curveto}$ ) and can also be open or closed. A cubic Bézier curve, which is composed of the line points and the bezier control points is drawn on the canvas. To achieve a non-regular outline,

<sup>1</sup><http://www.w3.org/TR/SVG/paths.html>

<sup>2</sup>[https://en.wikipedia.org/wiki/Well-known\\_text](https://en.wikipedia.org/wiki/Well-known_text)

	PostGIS (WKT)	SVG
Line	LINESTRING (30 10, 10 30, 40 40)	M 30 -10 L 10 -30 40 -40
Multi-Polygon	POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))	M 35 -10 L 45 -45 15 -40 10 -20 Z M 20 -30 L 35 -35 30 -20 Z

Table 4.1: *PostGIS* to SVG conversion

the control points can be displaced.

#### 4.5.5 Web map tile service

The Web Map Tile Service is a standard defined by the Open Geospatial Consortium (OGC). OGC Standards are intended to document in detail the technical aspects and rules for implementations that provide solutions to geospatial interoperability problems. They are defined, discussed, tested and approved by members of the OGC in a formal process. Some of these standard have also been turned into international ISO standards. The most commonly implemented OGC standards include the following:

- Web Map Service (WMS) for requesting map images from a database
- Web Feature Service (WFS) for feature access and manipulation on a database
- Web Coverage Service (WCS) for accessing geospatial images
- Web Processing Service (WPS) for geospatial processing
- Geographic Markup Language (GML), a XML-based interchange format

Typically, OGC standards are either interface (web services) or encoding (exchange formats) standards (Reed, 2011). All interface standards feature two functions:

- **GetCapabilities:** Provides capabilities of the server like file formats, map layers and display method
- **GetMap:** Handles the database request, creates a map based on the capabilities

Many services also support a set of other functions for e.g. feature information requests or a map legend (Peterson, 2012)).

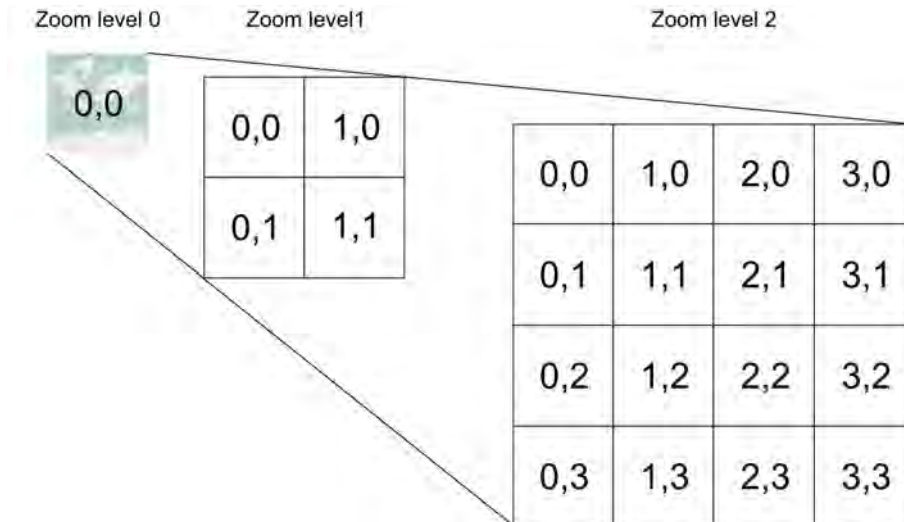


Figure 4.2: WMTS Tile indexing grid of a Mercator projection (after Clouston and Peterson, 2014)

An OGC standard for an interface that is capable of serving predefined image tiles is the Web Map Tile Service (WMTS) implementation which is a complementation to the WMS interface standard. The difference is however that the WMS standard is delivering custom maps with dynamic styling and content whereas the WMTS standard is designed for scalability and serving static data (*OpenGIS WMTS Specification* n.d.). Furthermore it is an advancement of the Tile Map Service (TMS) standard. Available tiles are advertised to a client from the service metadata and specific tiles can be requested. The client is forced to mosaic the requested tiles and clip them together to a final map image. This shifts the most processing capabilities from the server to the client, if the tiles are pre-rendered. They can also be rendered by the server on request and caching strategies can be applied to reduce network bandwidth usage (Kresse and Danko, 2012). The result can be viewed in a web mappig application as a “Slippy map” (Palazzolo and Turnball, 2012).

To fulfill the requirements of a WMTS representation it is crucial that the generated tiles have a default tile size of 256 by 256 pixel and are indexed in a structure that allows georeferencing. For this aim a tile space is defined that consists of an indexing grid that is specified for each zoom level as illustrated in figure 4.2. In the WMTS specification this is called the tile matrix.

The tile grid is always covering the entire world and equal to the tile indexing technique of Google and other major Online map providers, the origin of the tile grid is located in



Zoom level	Tile coverage	Number of tiles
0	1	1
1	2 x 2	4
2	4 x 4	16
3	$1^n \times 1^n$	$2^{2n}$

Table 4.2: Zoom level and tile coverage dependency (Source: openstreetmap.org)

the upper left corner. The tile next to the origin always has a X and Y coordinate of zero and tile indices are increasing towards the right and the bottom of the grid. This is the base for the naming convention applied to all tiles. The actual file name is the number of the tile in Y-direction or its row number in the defined grid. All tiles of one column are sitting inside this directory that is labeled by the tile number in X-direction. The column directories are then stored in a directory that is named by the zoom level (Clouston and Peterson, 2014). This results in a hierarchal image pyramid with the zoom level of zero on the top. Each zoom level has a different tile grid and therefore the tile coverage and number of tiles is significantly increasing with the zoom level. As the number of tiles in X and Y direction are doubled with progressing zoom levels, the number of tiles for each zoom level is quadrupled which is illustrated in table 4.2, adapted from the OSM website<sup>1</sup>.

---

<sup>1</sup><http://wiki.openstreetmap.org/wiki/SlippymaptilenamesZoomlevels>



## 5 Implementation of an automated process with GIMP

### 5.1 Geodata setup

OpenStreetMap (OSM) data, which was described earlier in section 4.5, was chosen as the source of geodata for the implementation of an automated map rendering process. The dataset at the current state is freely available and can be obtained from various sources. To edit, process and convert the data into GIS compatible formats like the proprietary ESRI Shapefile or an PostGIS database, several open-source tools exist and were used for the data processing in the described workflow. A *PostGIS* enabled *PostgreSQL* database is used for the storage of the OSM data as it allows for performing GIS operations on the data like basic generalization operations. It also allows to add custom functionality to the database by implementing custom database functions. How to get, import and update an OSM dataset into such a database environment is described in the following sections.

#### 5.1.1 Data acquisition

To obtain map data from the OSM database, a huge variety of download possibilities exist. The easiest way is via the web map interface where it is possible to export data by specifying a bounding box which determines the desired download area. This service however is restricted by a maximum size of an area of 0.25 square degrees<sup>1</sup> but OSM offers other alternatives for downloading the data. One of them is the download of the planet file which is a complete and regularly updated dump of the OSM database. Another way

---

<sup>1</sup><http://wiki.openstreetmap.org/wiki/Xapi>

is using the read-only OSM *Extended API (XAPI)* which is implemented in the *Overpass* API. The APIs allow for the definition of a bounding box area for download and even complex queries. Nevertheless, users are encouraged to use third party services which provide custom extracts of the data like *Geofabrik* or *Mapzen* for downloading the data to keep the load on the project servers to a minimum<sup>1</sup>). These datasets are mostly preprocessed excerpts and do not contain all of the available geometry data which is the reason the Overpass API was used for downloading the data used in this project. A sample console command using WGET to get all the data inside a bounding box covering the city of Munich would be for example:

```
1 wget -O munich.osm "http://overpass.osm.rambler.ru/cgi/xapi_meta?*["bbox
    =11.2,47.9,11.8,48.3]"
```

The `-O` parameter in the command defines the output file name which is following. As the URL, the API query is passed with the desired bounding box lower left and upper right coordinates. It is also possible to enter the URL directly in a web browser to trigger the download. Data exported from these sources are provided in a human readable OSM XML file meta format which has a `.osm` file ending. Alternatively the `.pbk` (Protocolbuffer Binary Format) is used which has some performance and storage advantages to support future extensibility and flexibility.

### 5.1.2 Data import and update

For the storage of the geodata, a *PostGIS* enabled *PostgreSQL* database is used. The environment has to be set up with a database user having read permissions. For this work, PostgreSQL 9.3.9 and version 2.1.4 of the *PostGIS* extension are used. Both are readily available as standard packages for all Linux distributions.

#### Import

For the import of a downloaded OSM dataset into a PostGIS database, the tool *osm2pgsql* was used which is also available as a package on a Ubuntu Linux system. The tool is accessed from the command line and the command arguments are explained in table 5.1.

<sup>1</sup>[http://wiki.openstreetmap.org/wiki/Downloading\\_data](http://wiki.openstreetmap.org/wiki/Downloading_data)

Argument	Effect
-s	Slim mode, required for the ability to update the dataset later
-c	Create data, would remove existing data from the database
-d	Database name, following
-U	Database user, following
-W	Forces password prompt for the database user
-H	Database host

Table 5.1: osm2pgsql command arguments

Table name	Content
planet_osm_point	Points (POIs, addresses, etc.)
planet_osm_line	Lines (roads, route relations)
planet_osm_polygon	Polygons (areas and multipolygons)
planet_osm_roads	Simplified subset of the planet_osm_line table for faster rendering

Table 5.2: osm2pgsql database schema

An import command to an empty PostGIS enabled database called *osm\_munich* would be for example:

```
1 osm2pgsql -s -c -d osm_munich -U gis -W -H localhost munich.osm
```

### The osm2pgsql database schema

The osm2pgsql tool creates four database tables which contain the geometry data as well as a number of other columns. All tables have a similar structure and provide a column *osm\_id* holding the original unique OSM id, a way column for the geometry and a *z\_order* column for the vertical ordering of the features. Additionally, a large number of columns containing the OSM tags as Key-Value Pair (KVP) are created which are populated with NULL if no tag value exists. Table 5.2 gives an overview of the created tables.

With the slim-mode enabled for import to allow a later update of the database, additional tables are created which contain OSM raw data that is not used for rendering (Ramm et al., 2011).

## Update

Data in a *PostGIS* database that was imported using the steps described before can be updated so it is not required to create a new database each time the data should be set to the current state. For an updating process it is necessary to have an OSM changeset which contains only the differences of two datasets. These files are readily available for the planet file or other subsets that are provided by third party services. For custom datasets that were downloaded from the OSM API the changeset file can be created using the tool *osmosis*. This is another command line tool available from standard Linux packages that provides OSM processing functionalities like the filtering and extraction of specific features from a dataset. It can also be used to import and update OSM data into a *PostGIS* database. For the processing of the data in this project only the function to create a changeset from two datasets was used. A sample command that creates a changeset of two different datasets would be:

```
1 osmosis --read-xml file="munich_1.osm" --read-xml file="munich_2.osm" --  
   derive-change --write-xml-change file="munich_diff.osc"
```

This changeset can afterwards be processed with *osm2pgsql* to add all changes to an existing database. A command for the execution of such a step for the changeset created in the step before and the database *osm\_munich* that has already been used in the previous examples is shown in the following line:

```
1 osm2pgsql -a -d osm_munich -U gis -W -H localhost -s munich_diff.osc
```

The only parameter that has not already been used in the import command is the *-a* which stands for the append mode that is used here instead of import.

## 5.2 Sketch rendering

A rendering to generate irregular shapes of the input geometry to achieve a more natural and hand-drawn appearance of the original OSM data is taking place before the geometry is passed to the GIMP module functionalities and works with the SVG format that has been examined in section 4.5. This chapter will now provide a detailed explanation of the methods and algorithms that deliver sketchy rendered versions of the input geometries. All numeric values and geometry dimensions are given in screen resolution pixel units. At first, different displacement methods will be presented that act on points or the nodes of geometries consisting of points. After that, the randomization of these point composed geometries, like lines and polygons will be illustrated using some examples. Lastly, the technique to create hachure lines for a computed polygon fill will be explained and illustrated. The code that will be explained in this chapter can be found in the `sketch.py` and `emphrandomize.py` Python modules in the sketching directory of the `gimprenderer` project.

### 5.2.1 Displacement and randomization

#### Random distributions

The Python programming language features a collection of different functions for the generation of pseudo-random numbers making use of various probability distributions. These randomly created numbers serve as a viable resource for the alteration of geometric coordinates in this implementation along with other parameters that can be specified<sup>1</sup>. For the aim of this work, two distribution types were identified as most suitable for the desired purpose. The first distribution used is the Uniform distribution which is also the standard method to quickly generate random numbers in Python. It provides values between zero and one with a equal probability of each output value. The second distribution used is the Beta distribution which also returns values between zero and one but with a probability pattern that can be defined using the shape parameters Alpha and Beta. If not explicitly mentioned, three is set as the standard value for Alpha and Beta in all further examples. In appendix A.1, the probability density functions of selected distributions are listed with

---

<sup>1</sup><https://docs.python.org/2/library/random.html>

their input values. Additionally an example image of applying the method to a set of 10000 points that are randomly generated and placed inside a rectangular area along a horizontal line is shown. Only the distribution along the x-axis is determined by the particular random distribution, along the y-axis there is the Uniform distribution used for all example images.

### **Displacement and randomization**

The displacement of points is a main functionality of the random sketch rendering that was implemented and used in the geometry processing of this project. Relocating points or vertices of lines and polygons randomly using a small offset accounts for a more irregular and thus a more naturally appearing outline. As a part of the sketching that makes use of randomization functions, the code of the function `emphdisplace_point()` can be found in the `randomize.py` module file. This function computes a randomly displaced new position of a given point inside a specified area using three different methods that can make use of two different random distributions each. These methods are described and analyzed in appendix A.2.

The computation time in table A.2 was determined by calculating the average computation time of 1000 runs of calculating 10000 displacements of a point in one loop. For the examples using a Beta distribution an Alpha and Beta shape parameter of the value 5 was chosen. Taking the computation time into account, using the polar coordinates equation for the calculation of the circle is the fastest approach. This is an important issue that needs to be considered as the calculation of tiles can be a time consuming process. However, using a Beta distribution instead of an Uniform distribution is preferred and selected as the method for the final rendering, mainly because it can be considered to produce the most homogeneous appearance.

### **Randomization methods**

Random functionalities have also been implemented for other purposes than calculating the displacement of a single point. In order to apply a point displacement to a line it is crucial to have sufficient line points that are available for displacement. However, this is not guaranteed as the granularity of the input OSM geodata varies on a large scale. For this reason a function was created that adds points to a simple line segment that is







Method	Distribution	Example	Computation time (in ms)
Equal	None		147.496
Uniform	Uniform		116.822
Equal uniform	Uniform		289.864
Equal beta	Beta		395.178

Table 5.3: Methods for random points on line

defined by only two points. This function, called *random\_points\_on\_line*, is also part of the *randomization.py* module and takes a line defined by two points, the number of points that should be added and a definition of the method used for the distribution of the points as input arguments. If a number is specified, the generated points are either homogeneously distributed in equal distances from each other or randomly distributed. Random scattering can either be relative to the entire line or line segments which are calculated according to the number of specified points in the result. As an additional option the distribution method can be chosen from uniform or Beta. In the final processing this function can be combined with the point displacement function to change the positions of the randomly generated points relative to the line. Example results are listed in table 5.3 with the average computation time of 1000 runs adding 5 points to a line. In these images, five points are added to an original line using different distribution methods. The “Equal” and “Uniform” methods are applied to the entire line whereas the other two methods use equally sized sub segments of the original line on which a point is randomly placed. Similar to the point displacement function, a processing using Beta distribution delivers the most homogeneous result and was chosen in the final processing chain in this project.

Another randomization method was implemented to convert a linestring into a curve with a smooth and rounder outline. This contributes to a more hand-drawn look of a line and is realized using the SVG curve commands for paths. For each point of the line string that is not the start or the end point, two random curve control points are computed and a valid SVG path with curve parameters is returned. Two methods have been tested for the computation of these control points positions which take the value *d* as an input

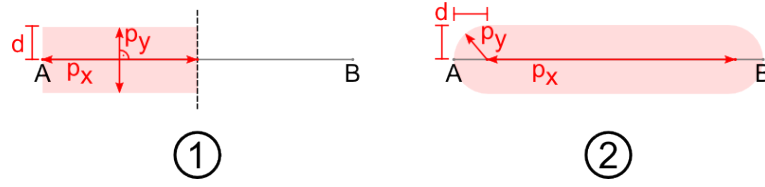


Figure 5.1: Schema random curve controlpoints

parameter that determines the maximum orthogonal distance of a control point from the line. The first approach uses orthogonal lines to determine the position of the control points. A simple line that is defined by two points is split into two equal segments at the center between the two points that define the line. The position of the random point relative to the segment is then calculated using the defined random distribution, which can be either Uniform or Beta. At this position, the final location is generated using the defined random distribution again to shift the position that was calculated in the first step orthogonally to the segment in negative or positive direction using the input parameter as the maximum distance the new point can have relative to the segment. The second approach features polar displacement and can also be split into two steps. At first, a random position is generated on a segment that is constructed from the two points which lie on the straight line between the two original points and are the given distance  $d$  away from their respective closest original point. Again either Uniform or Beta distribution can be used for the generation of the random location. In the second step, the position from the first step is displaced using the polar point displacement function explained earlier in this section with  $d$  as the displacement threshold. Nevertheless, there are two peculiarities that need to be considered. The first is that the areas of potential placement for the two generated control points are overlapping which creates peaks on the line that let the curve appear more disturbed. Secondly, if the given distance is equal or larger than the distance between the line points, a half of that distance is used for displacement instead of the input value. This was implemented to reduce the disturbance of short line string segments. Figure 5.1 explains the schema of the two methods applied on two points A and B that make up a straight line. Areas of the probability distributions  $p$  in  $x$  and  $y$  direction are colored in red and the given maximum distance threshold is labeled  $d$ . In table 5.4, the methods are illustrated with some example results and compared using the average computation time of 1000 function runs where 2000 control points are calculated for each original point.

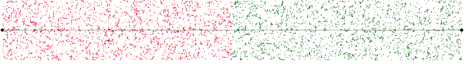



Method	Distribution	Example	Computation time (in ms)
Orthogonal	Uniform		105.166
Orthogonal	Beta (2,2)		178.452
Polar	Uniform		99.251
Polar	Beta (2,2)		156.406

Table 5.4: Random curve controlpoints

### 5.2.2 Hand-drawn emulation

To emulate a hand-drawn structure of the lines, not only points need to be randomly disturbed but also more complex geometries like lines and polygons that are created using points as their vertices. Furthermore it is desirable to transform these linestrings into curves which look more similar to a manual human drawing style. Several functions have been implemented in the sketching module and explained in the sections before which achieve this aim. They are either used in the processing as they are or in combination with other functions. The relevant functions will be described in the following section.

#### Lines

For the final processing of lines, an operation was implemented that achieves the effect of a jittered line with a hand-drawn appearance. In the code, the functions *add\_points\_to\_line()*, *displace\_point()* and *random\_controlpoints()* which have been explained in the sections before are combined in the function *jitter\_line()* which is part of the sketch.py module. It takes a value for the maximum distortion as input parameter. As the first step, random points are added to a linestring if the distance between two points is greater than the squared value of the input threshold. Subsequently, the points are being randomly displaced within a threshold distance before random curve control points are added for each point of the linestring. Table 5.5 shows some examples of Figure 5.1 which are alienations using this function with different input values on a line with a length of 300 pixels.

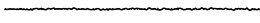

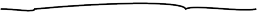
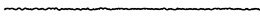

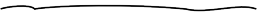
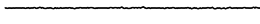


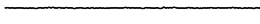


Name	d=2	d=5	d=10
Orthogonal			
Orth. Beta			
Polar			
Polar Beta			

Table 5.5: Line jittering

Another function that renders lines is the *line\_handy()* function which was inspired by the line rendering function used in the Handy library <sup>1</sup>, a hand-drawn sketchy rendering library that was introduced in section 2.2.2. The steps of this function have been implemented in Python using own methods and they are used to process hachure lines to fill polygons with. The generation of these lines will be explained later but the rendering of each single line will be described in this part as it features random alteration as well to create irregular and bowed lines. The process is dealing with simple lines only, that are constructed as a straight line connecting two defining points and takes input parameters which are factors for the maximum displacement distance, the roughness of the displacement and the bowing of the central point. Two points are then added to the line, one at the center and one at a random position between 65 and 85 percent of the line from A to B. The center point is then shifted randomly along a orthogonal vector with the length  $1/200$  in either positive or negative direction. Similarly, the other added point is randomly shifted along a orthogonal vector but here the input displacement parameter  $d$  is used to determine the length of the vector. Additionally, the start and end point are displaced within a radius of  $d$  as well. Finally the line is smoothed using a Catmull-Rom Bezier spline. Figure 5.2 illustrates the working principle with the areas of the probability distributions  $p$  of the midpoint  $m$  and the second point  $n$  on a straight line from A to B. These are located around the points A and B, along an orthogonal line at the midpoint in  $y$  and in the area of point  $n$  in  $x$  and  $y$  direction and they are indicated using a pale red color. The given maximum distance threshold is  $d$ . Some rendering examples are listed in table 5.6 on a straight line from point A to B with the varying input parameters  $d$  and  $b$  that have been used on a line of length 300 pixel.

<sup>1</sup><http://www.gicentre.net/software/#!/handy/>

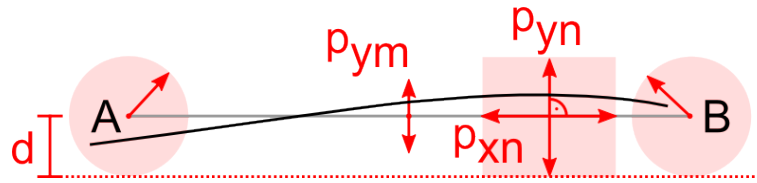


Figure 5.2: Schema line handy

d	b=1	b=2	b=5	b=10
1				
2				
5				
10				

Table 5.6: Lines handy

## Polygons

For processing polygons and multipolygons a separation of edge lines is required to use the same functions as for linestrings adequately. This is implemented in the *jitter\_polygon()* function and works in two steps. At first the outlines of the multipolygon are split into segments at vertices where the angle between the two edge lines defined by the vertex to the vertex behind and from the vertex to the vertex ahead exceeds a given threshold. The relevant vertex positions where an example polygon was split two times using a threshold angle of 90 and 120 degrees are illustrated in figure 5.3. The latter is used as the default value in the processing. Afterwards the split segments are distorted using the *jitter\_line()* method that is described before. In table 5.7 some example renderings of polygon shapes are shown including the input distortion value *d* and the applied method for line jittering. The disjoin angle was 120 degrees using the Polar Beta method on the polygon which bounding box dimensions are 500 by 400 pixels.

### 5.2.3 Hatching

So far, only line alienation has been considered. However, for the filling of polygons another technique was developed that was also inspired by the Handy rendering library.

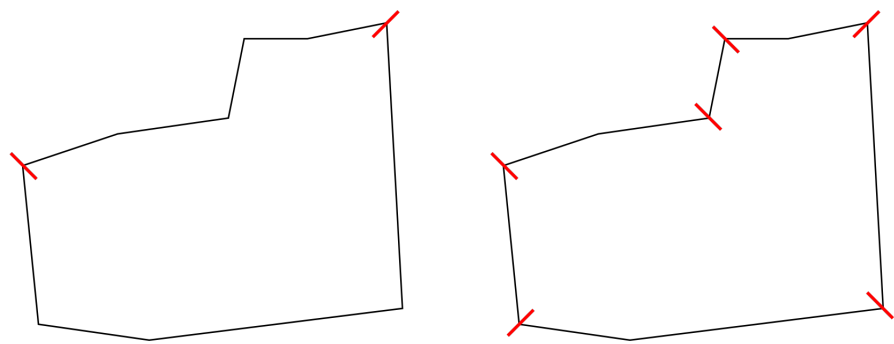


Figure 5.3: Polygon disjoin cut points with disjoin angle 90 (left) and 120 (right)

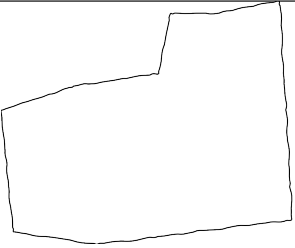
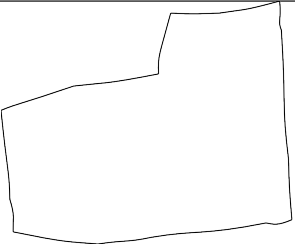
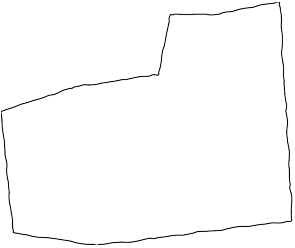
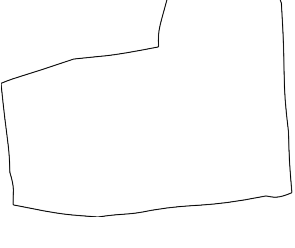
Method	d=2	d=5	d=10
Curve			
Displace curve			

Table 5.7: Polygon jittering examples

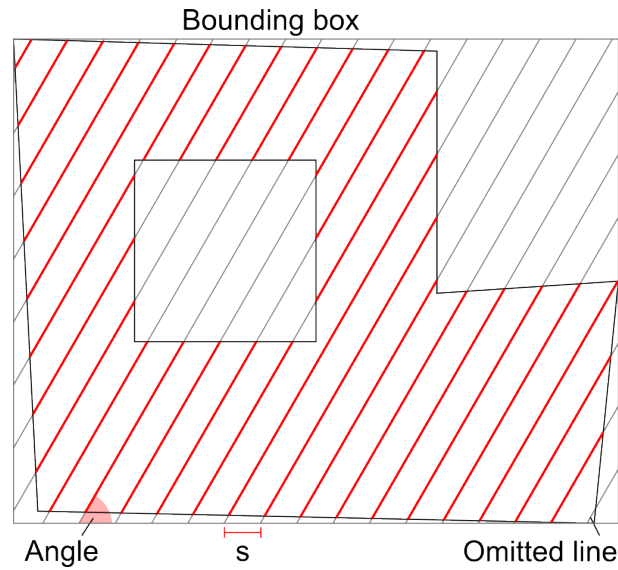


Figure 5.4: Schema hatching, angle = 60

The polygon is not filled with a specific color but a computed hatching or so called hachure lines which substitute a shape filling. These hachure lines are parallel lines which are aligned in a certain inclination and an equal distance from each other. They are computed from the bounding box of the input shape and a specified angle that can be a value between 0 and 180 and is relative to the y-axis. At first the lines are generated for the bounding box, covering the extent of the shape. After that, these lines are clipped with the shape of the polygon to get only the lines which are inside of the polygon. Lines with a line length below a threshold were omitted to avoid over-cluttering of the lines. For the geometric calculations that were required in the code to perform the clipping, the clipping function provided by the Shapely package was used which is equipped with specific functions to perform spatial operations. Figure 5.4 shows the calculation example for a multipolygon with a hole, using an angle of 60 degrees and a spacing  $s$  of 30 relative to the bounding box dimensions of 500 by 400.

### Sketching

The sketchy rendering of polygon shapes is now achieved by the combination of the two techniques that were explained earlier in this section. There are two parts that require alteration: the polygon outline and the hachure lines making up the polygon fill. An alteration of the first is produced using the polygon disjoint with subsequent jittering as shown in table 5.7. The hatching lines are alienated differently using the *line\_handy()*

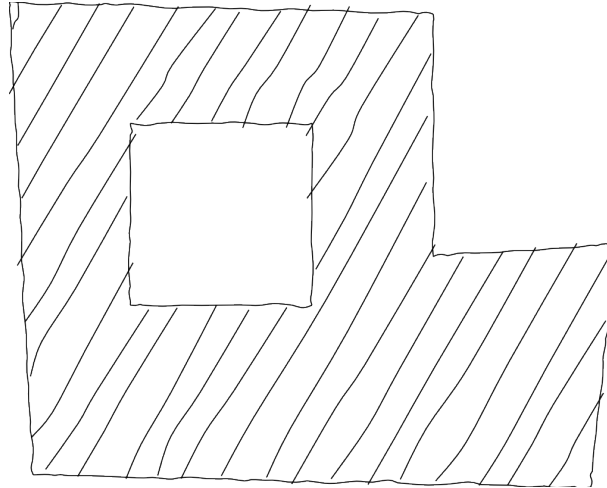


Figure 5.5: Sketchy polygon with hatching

function, discussed in the section before. Figure 5.5 gives an example of the polygon from figure 5.4 altered using the final processing.

### 5.3 Automated processing

This chapter deals with the use of the GIMP Python-Fu scripting capabilities that are implemented to realize an automated workflow for the map and tile rendering. At first, the script parameters that are defined in the configuration files and control the process are explained. After that, a general overview over the script's code structure is given including an illustration of all the implemented Python classes and modules. Interfaces used by these components and how they are connecting with the database and the configuration files are described as well. Following this, the geometry processing chain and the rendering methods are then explicated in detail before finally the format and structure of the output results are explained. The processing was conducted on a Ubuntu 14.04 LTS Linux system with GIMP version 2.8.14 and Python version 2.7.6 installed. Also Eclipse with the PyDev plugin was used as Integrated Development Environment (IDE). For the spatial functions the Shapely <sup>1</sup> and for the SVG conversions, the svgwrite <sup>2</sup> Python packages were used.

---

<sup>1</sup><http://toblerity.org/shapely>

<sup>2</sup><http://pythonhosted.org/svgwrite/>



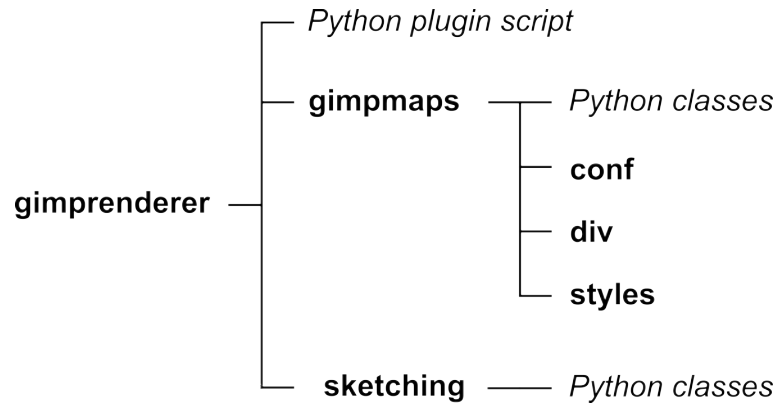


Figure 5.6: Project files structure

### Project structure

All files that are introduced in the following sections are available on the compact disc and online via Github <sup>1</sup>. Figure 5.6 illustrates the general structure of the project directory. The Relevant files are contained in a folder called `gimprenderer` which contains the GIMP Plugin python script and two subdirectories. The first is named `gimpmaps` and contains the rendering classes for tile and map creation along with additional directories. These contain the configuration files (`conf`), other files like JSON schemas, and the style files (`style`). The second folder `sketching` contains all Python classes that are used for the geometry estrangement.

#### 5.3.1 Configuration files

All parameters for the scripted rendering in the GIMP Python-Fu module are defined in JSON files which are stored in the `conf` directory within the code or at any other location that is specified when executing the script. For a map or tile rendering process, there are two required JSON files which will be introduced in the following section.

### Map configuration

The JSON file that defines the general properties and parameters that can be configured for the rendering is located in the `/conf` directory of the `gimpmaps` root directory and can have any name. The following tables 5.8 and 5.9 show the top-level and detail properties.

<sup>1</sup><https://github.com/mucximilian/gimpmaps>

Property	Description
<code>osm_db</code>	Database connection credentials
<code>map</code>	Map image(s) rendering options
<code>style</code>	Redering style name (as named in the <code>/styles</code> directory)
<code>out_dir</code>	Output directory

Table 5.8: JSON configuration file top-level properties

Property	Description
<code>bounding_box</code>	patial extent of the area to render as map or tiles. Given as a bounding box with upper left and lower right coordinates
<code>tiles</code>	Map image(s) rendering options
<code>image</code>	Style related rendering options
<code>create_xcf</code>	Defines if a GIMP <code>.xcf</code> image is created additionally to the <code>.png</code> output(s)

Table 5.9: JSON configuration file `map` properties

A full example of a configuration file for an example rendering can be found on the compact disc or Github.

Only one of the parameters `tiles` or `image` are parsed by the code for the rendering. This depends on whether map tiles or a single map should be rendered.

### Style configuration

The file `style.json` defines the rendering style of the the OSM features at specific zoom levels and a background image for the map or tile images. It can either be placed in the `/style` directory or at any other location that is defined in the configuration file. Table 5.11 and the following show the top-level and detail properties. Again, full example of a configuration file for an example rendering can be found on the compact disc or Github.

The lines and polygons properties consist of an array of feature types and the assigned

Property	Description
<code>text</code>	Defines if text is rendered
<code>sketchy</code>	Defines if sketch rendering is used
<code>style_path</code>	The path to the style definition JSON and image files (default is in the <code>/style</code> directory within the project)
<code>style_name</code>	Name of the style to use
<code>polygon_fill</code>	Defines the type (mask, hachure or fill), if the fill color and whether an outline should be drawn around polygons or not
<code>image_tile_span</code>	Images used for masking must have a size that is an integer multiple of 256 pixels (the default tile size). This is the factor that defines how many tiles one masking image covers

Table 5.10: JSON configuration file `style` properties

Property	Description
<code>name</code>	Name of the style
<code>background</code>	The image used for the map background
<code>features</code>	All OSM features that should have a visual representation (Lines, Polygons). Has a <code>lines</code> and <code>polygons</code> property.
<code>text</code>	All OSM features (Polygons) that should have a text label. Has a <code>polygons</code> property.

Table 5.11: JSON style file top-level properties

Property	Description
<code>z_order</code>	Horizontal order of overlapping objects
<code>zoom_min</code>	The minimum zoom level at which a feature is drawn
<code>zoom_max</code>	The maximum zoom level at which a feature is drawn
<code>osm_tags</code>	A list of OSM key-value pairs

Table 5.12: JSON style file common feature properties

Property	Description
<code>stroke_line</code>	The stroking options for the line

Table 5.13: JSON style file `lines` feature properties

rendering style. Tables 5.13 and 5.14 show each feature type’s property objects. The line properties of the simple line and polygons are explained in the tables 5.15 and 5.16.

Currently the text rendering is only implemented for polygon features, therefore only a polygons property is existing which consists of an array of feature types and the assigned rendering style. Table 5.17 illustrates its property objects.

### 5.3.2 Script structure

Creating a script for the automated generation of web map tiles or single map images is one of the main goals of this work. The Python-Fu scripting interface introduced in chapter X is used to achieve this and with the script and the corresponding modules that are implemented, the user is able to automatically generate tile images or a single map image from OSM geodata. The code for the GIMP plug-in is split up into different components which are illustrated in figure 5.7. It gives an overview of the code modules (dark green), the processing steps (light blue) and their interactions with other components. A detailed explanation of this diagram is provided in the following paragraphs.

As described in chapter X, a GIMP plug-in script file has to be stored either in the system user’s GIMP plug-ins directory or at any other location that GIMP is aware of. For this project, the file *create\_gimpmap.py* is the main script for the GIMP plug-in that is imple-

Property	Description
<code>stroke_line</code>	The stroking options for the outline
<code>stroke_line</code>	The stroking options for the hatching
<code>image</code>	The image that is used for masking
<code>fill_color</code>	The color with which the polygon is filled

Table 5.14: JSON style file `polygons` feature properties

Property	Description
<code>color</code>	Color for the line stroke
<code>brush</code>	Name of the GIMP brush
<code>brush_size</code>	Size of the brush in pixel
<code>dynamics</code>	GIMP dynamics applied to the brush

Table 5.15: JSON style file `stroke_line` properties

Property	Description
[same as for <code>stroke_line</code> ]	
<code>angle</code>	Angle of the hachure relative to the Y-axis
<code>spacing</code>	Distance between two hachure lines

Table 5.16: JSON style file `stroke_hachure` properties

Property	Description
<code>font_size</code>	Size of the font
<code>color</code>	Color of the font
<code>stroke_line</code>	Stroking options for the polygon outline
<code>font</code>	Name of the font
<code>effect</code>	Text rendering method

Table 5.17: JSON style file `polygons` text properties

Property	Description
<code>buffer_size</code>	Size of the buffer around the text in pixel
<code>name</code>	Name of the effect
<code>buffer_color</code>	Color of the buffer

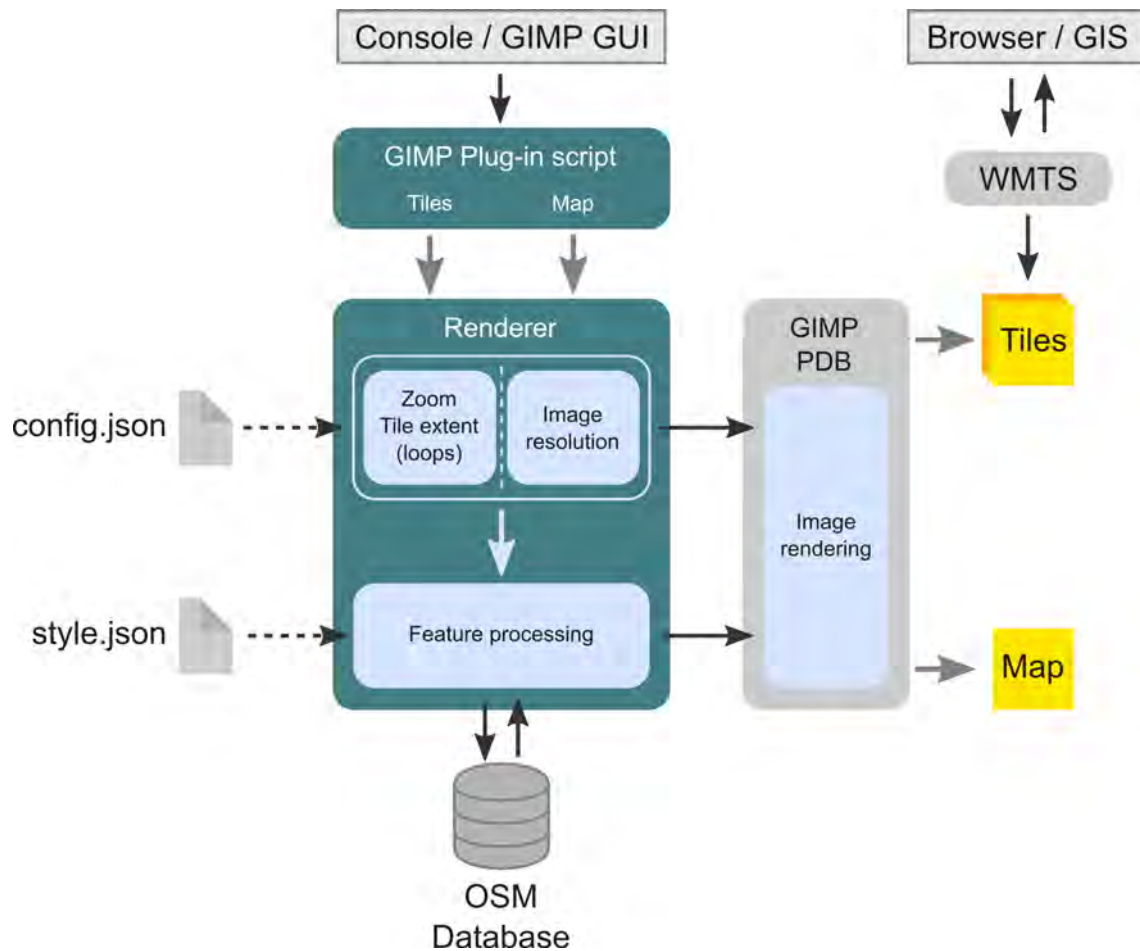
Table 5.18: JSON style file `effect` text properties

Figure 5.7: Workflow script structure

mented using the Python-Fu scripting interface to add the map creation functionalities to GIMP. It serves as the entry point of the process and can be triggered either from the system console command-line interface using the following command

```
1 gimp -i -b '(python-fu-create-gimpmap RUN-NONINTERACTIVE "config.json"
2 "tiles")' '(gimp-quit 1)'
```

or from inside the GIMP graphical user interface by navigating to *Scripts* → *CreateGIMPmaps* inside the GIMP window's menu bar. This opens a dialog which requires the same input parameters as the console command that can be defined in text fields. The execution of the plug-in script to initiate the processing requires two input parameters that need to be passed to the script:

- The location of the JSON configuration file which provides the required data for the desired rendering. This includes the spatial extent and other variables like the style definition file that should be used for the rendering style or the output directory
- The desired output format which is either a collection of tile images or a single map image

Depending on whether tiles or just a single map should be rendered, the plug-in script is instantiating the respective class that implements the rendering method which are *TileRendererGimp* for tile rendering and *MapRendererGimp* for map rendering. The classes are components of the *gimpmaps* Python module which is available in the same folder as the GIMP script. These renderer classes inherit common functionalities like the parsing of configuration files or feature querying from their abstract parent classes named *TileRenderer* and *MapRenderer* which are also depending on the *Renderer* class (*renderer.py*). This class is the base class for the entire process. As the first step, the configuration JSON file is parsed to get all necessary input parameters using the functions from the *Renderer* class. For the subsequent rendering of images, the *TileRenderer* or *MapRenderer* classes are calling their *render()* method that defines the steps for the image creation. This is the main differentiation during the rendering process for tiles or a single map and at this point, the *draw()* function is used to manage the image rendering. The map tiles are generated inside a loop for each tile extent that is calculated from on the full spatial extent defined in the configuration file and the zoom levels (see sloppy maps tiling scheme, chapter X).

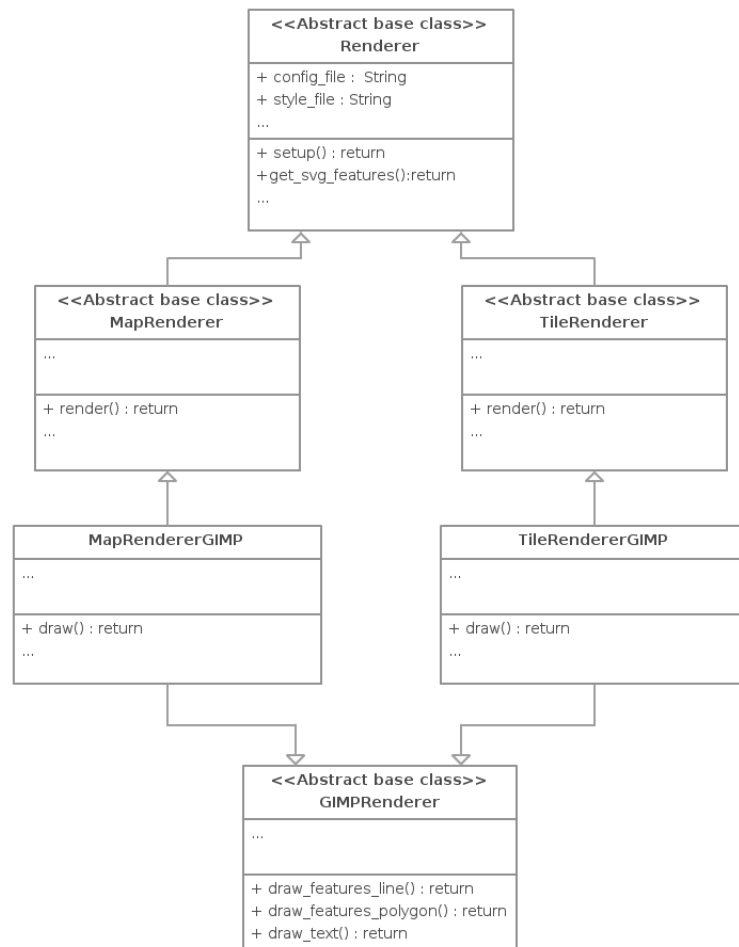


Figure 5.8: Simplified UML diagram of the core rendering classes

For a single map, the zoom level that is equivalent to the defined scale is calculated for the styling as well as the pixel dimensions of the output map image. This is calculated from the full extent in meters, the given scale and the output print quality which is set to 300 Dots per Inch (DPI) by default. At this point, the map or tile image is created in a non-interactive GIMP session employing mainly *GIMP Procedural Database* (PDB) functions which are provided by the GimpRenderer class. Tile images are uniformly created with the standard tile size of 256 pixels. The class dependencies are implemented in this way because there are tile and map rendering classes that do not use the GIMP rendering module but create SVG files. This was implemented for test reasons as the debugging of the functionalities was challenging due to the fact that it can only be started from inside GIMP and not in an IDE. Figure 5.8 gives an overview of the classes in a simplified UML class diagram. A more detailed diagram can be found in appendix A.1.



Now the map features are processed iterating the features that have an associated styling defined in the style file for the given zoom level in the order of their `z_order` value. For each line and polygon feature the geometry that is contained inside the image bounding box is queried from the geodatabase. Geometries are not split so the feature's full geometry is returned which might also be mostly outside the bounding box extent. The transformation from geographic coordinates to image coordinates is taking place inside the *PostGIS* enabled database and this data can then be modified applying estrangement methods described in the previous section 4.5. The geometry is then added to the image through PDB functions again with the drawing parameters for the styling applied. A more detailed description of the feature processing and image rendering is provided in the next section. After all features have been processed, the image is rendered in GIMP and saved as an image file. The tile images are stored in a WMTS compatible directory structure that was already introduced in 4.5.

### 5.3.3 Feature Processing

As described in the previous section, the map features are processed in an iteration loop. All geometric features declared for rendering by the zoom level value in the styling file are consecutively processed in the ascending order of their specified `z_order` value. Based on the spatial extent that is calculated for the tile image or given for the full image, the geometry features are queried from the *PostGIS* enabled OSM geodatabase. Simplification of geometry, generalization of polygons and coordinate transformation steps are performed inside the database and are described in the next section in detail. The returned geometry data is then modified applying the estrangement methods which are explained in chapter section 5.2.3. Lines and polygon outlines are altered using jittering and the hatching is computed for the polygon fill if no mask image is selected. Before the feature processing, an image was created in a non-interactive GIMP session that is now accessed from the components of the processing to perform the rendering of the features. Once the feature processing iteration is finished, the image is saved as an image file. An overview of this process for the derivation of tiles is shown in figure 5.9.

The feature processing for the single map is similar to the tile processing except for two aspects. First, the image extent has to be computed as it is not predefined like for tiles. Furthermore no loop is required to calculate the tile bounds that define the image but

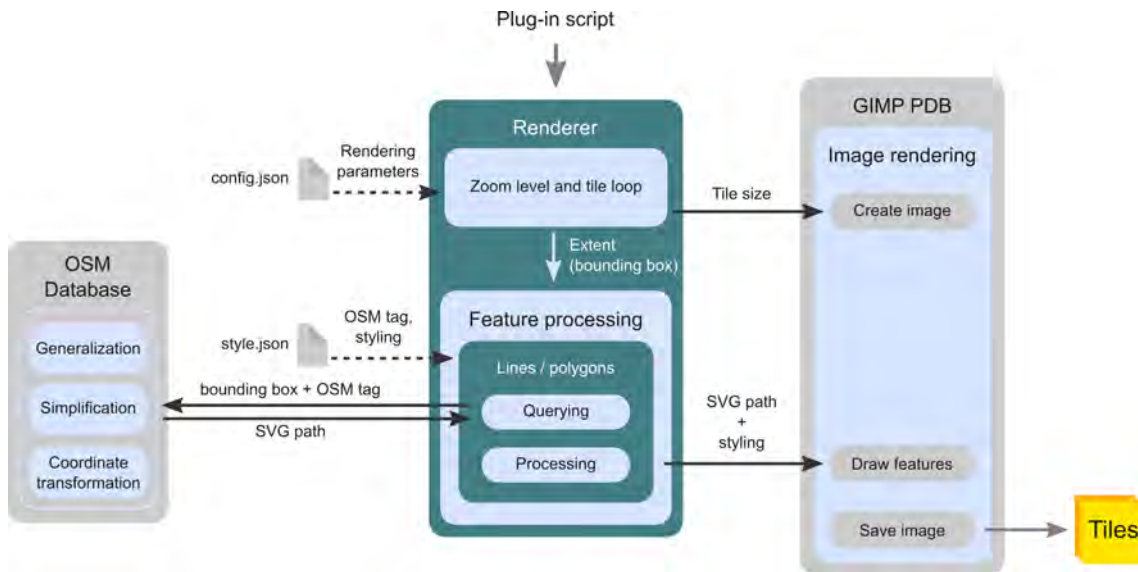


Figure 5.9: Workflow detail tiles

only the print resolution is calculated. Secondly, the rendering of text labels is added which requires another database access that is a different from the line and polygon query. This is only implemented for the map rendering and not for the tile rendering as the determination of text extents and placement is separate field of research that could not be covered within the scope of this work. However, a simple approach was developed that demonstrates the capabilities of GIMP with respect to this topic but it can only be used with polygon features. The label text is taken from the name of the OSM polygon feature in the database. Additionally, the centroid or another point on the surface if the centroid is outside the polygon is computed and returned in the text feature query instead of the actual geometry. Geographic coordinates are also transformed into image coordinates and only the X and Y image coordinates for the point are returned. Figure 5.10 illustrates the processing for a single map with the additional text drawing functionality.

### Database functions

A significant amount of processing takes place directly inside the database and not within the actual automation script in Python. Several stored procedures in the PostgreSQL procedural language PL/pgSQL have been added to the database to execute operations for the bounding box computation, generalization and coordinate transformation. This is also caused by the fact that the GIS-capabilities of the functions available in *PostGIS* enabled database provide some general performance benefits. OSM Key-value tags included in the

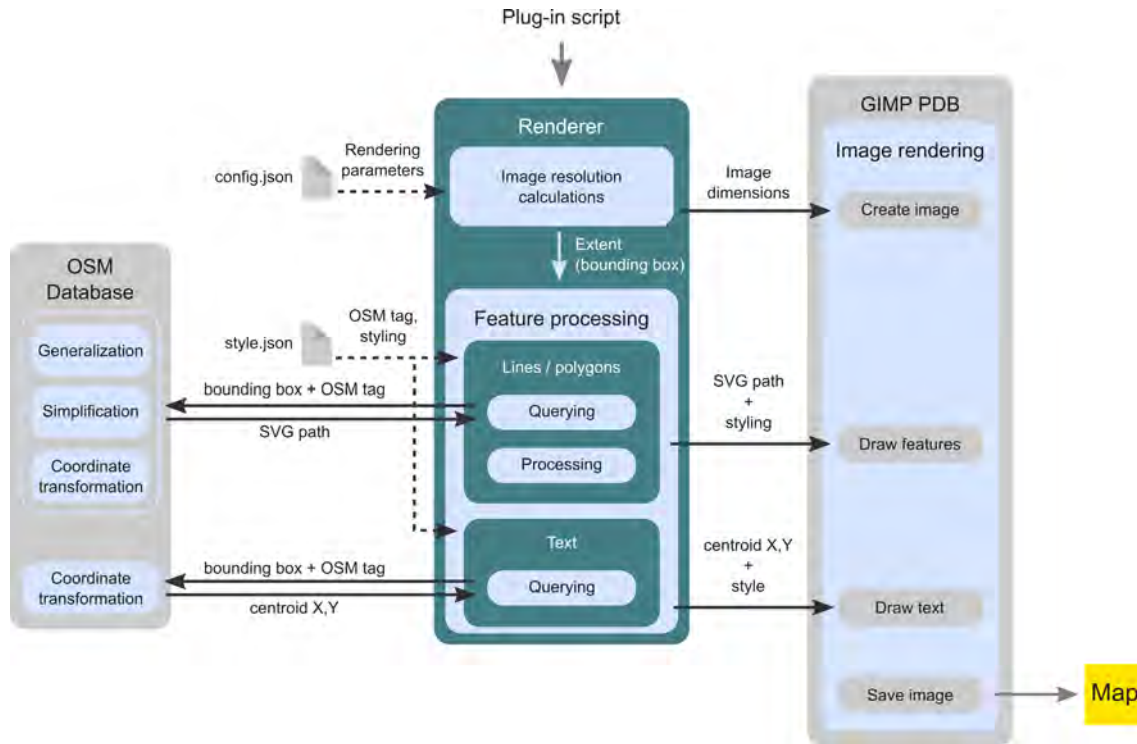


Figure 5.10: Workflow detail map

style file are used as a filter condition in the selection queries performed on the database tables. Based on the spatial extent defined for a map image or calculated for tiles, a buffer depending on the styling is added to the extent which is used for the spatial query. This is important to realize tile mosaicing later as some features might not consistently appear on adjacent tiles. All geometry features that intersect this bounding box area defined by upper left and lower right coordinates are selected in the query for the feature type. The tile or map extent is computed in meters during the first part of the rendering process and then passed to the feature processing. For every feature type, the buffer is then calculated in meter depending on the stroke size that is specified in the style configuration file. It is equivalent to a half of the stroke size in pixel and added to the extent of the tile to get the bounding box for the query. This is necessary as the stroking of the geometry line can also overlap with neighbouring tiles depending on the size of the line stroke which is illustrated in figure 5.11.

The *PostGIS* geometry of polygon features is also processed during the query with a number of generalization steps being executed. Similar to the erosion/dilatation principle in image processing, the *ST\_Buffer* method is applied on every polygon with a buffer radius that depends on the actual size of one image pixel in meter. The function is

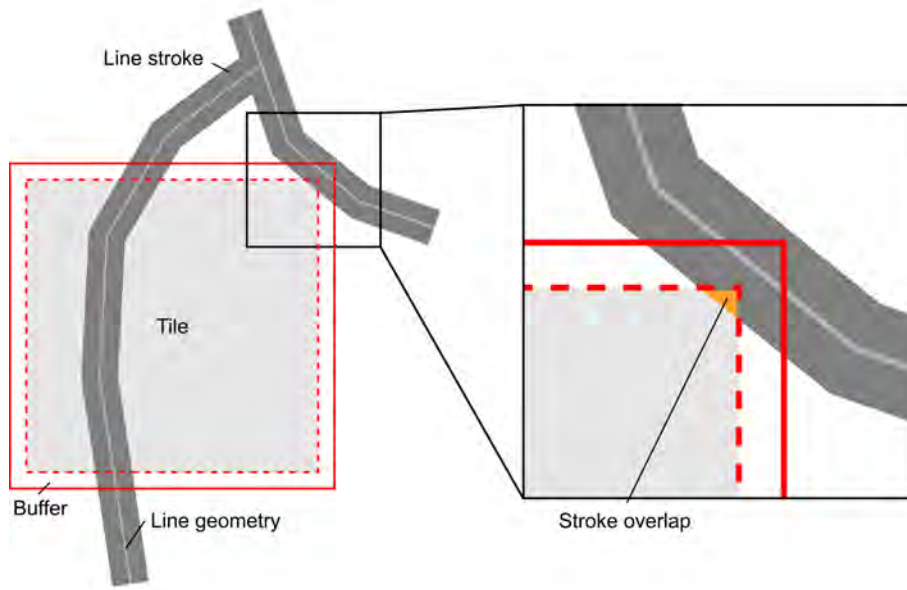


Figure 5.11: Tile bounding box buffering

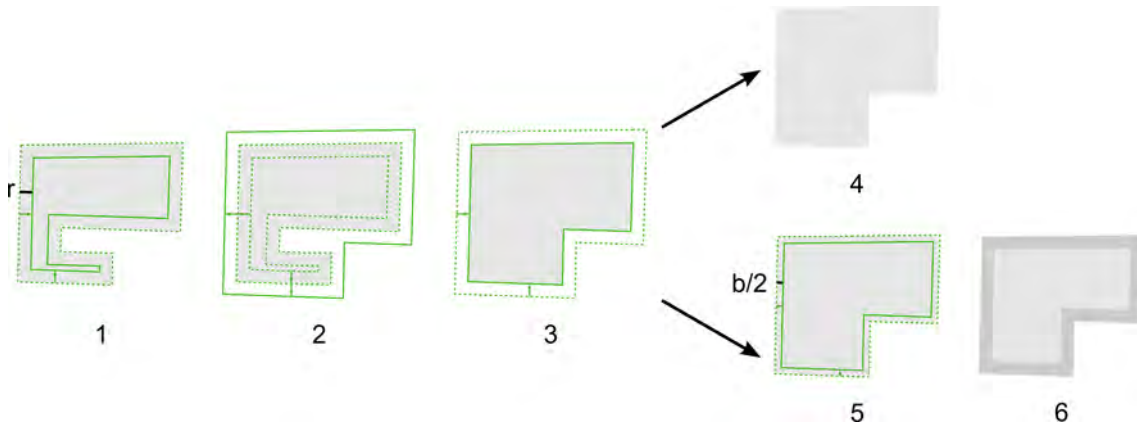


Figure 5.12: Polygon generalization schema

executed four times, at first with a negative radius for shrinking (erosion), then two times with a positive radius (dilatation) for growing and then again with a negative radius (erosion). This removes small holes and outlines with too much detail and the defined mitre option for the joining of corners preserves an angular outline. If an outline stroke is defined, a fifth negative buffering is applied with half of the stroke brush size as the radius. This procedure is illustrated in figure 5.12 with the general steps (1 to 3), the result of these (4) and the alternative additional processing for the negative outline buffer (5,6). An example of the generalization process is shown in figure 5.13 for the inner city of Dresden comparing the OSM geometry before the process and after.



Figure 5.13: Polygon generalization example

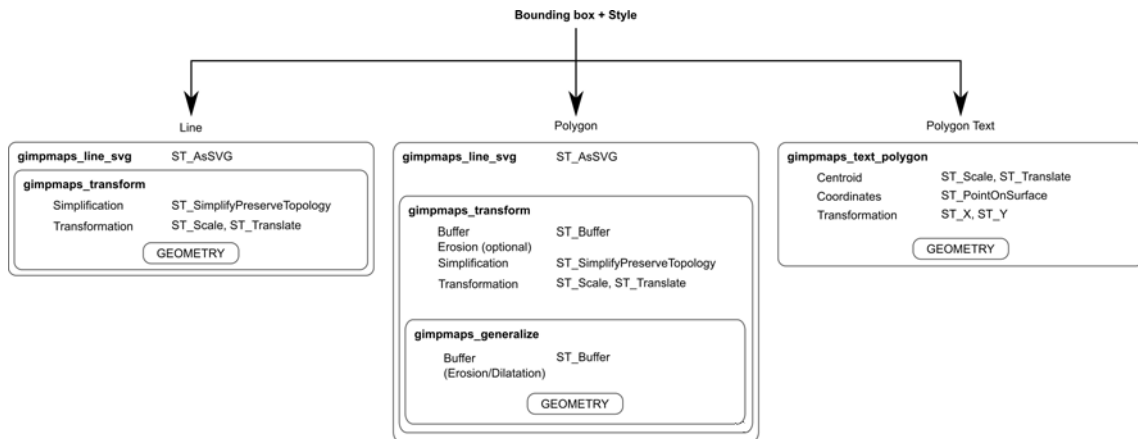


Figure 5.14: Schema database functions

Geometries are also simplified using the *ST\_SimplifyPreserveTopology* function of PostGIS which implements the Douglas-Peucker algorithm. Polygons are simplified after their generalization, whereas lines are only simplified. Following that step, all spatial coordinates of geometry features are transformed from the database coordinate reference system into image coordinates inside the database employing the *ST\_Translate* and subsequently the *ST\_Scale* function. Finally, the *PostGIS* function *ST\_AsSVG* is used to return the geometry in the SVG format which is used for further processing. Figure 5.14 gives an overview of all the database functions that were implemented and the used *PostGIS* functions. They can be found in the file *gimpmaps-functions.sql* which is stored in the directory */sql/functions/* on Github or inside the rendering directory on the compact disc.

### 5.3.4 Image rendering

The experimental rendering is restricted to two feature types for map tiles and three feature types for a map image. These are lines, polygons and text labels for the single map which have been mentioned earlier. In this section the rendering of these objects is described after they have been queried from the database and processed. GIMP functions that are provided via the Python-Fu scripting interface from the GIMP Procedural Database (PDB) are used for this purpose. An overview of the script functions has been given in the previous section 3.2 which is the reason why the focus of this section is on the actual rendering of the final results. The first step is always the creation of an image in a non-interactive GIMP session. After a new image is created, a background image is added to the image first which has either the standard size of the tile (256 pixel) or integral multiples of that (512, 768, 1024 pixel etc.) which requires a image span value in the configuration. This parameter is defined for all images so all images used must have the same integral multiple size. The background image has to be seamless as it is used repeatedly for tiles and also in most cases for single maps. For each feature a layer group is added that serves as a container for the canvas and other drawing objects like masks. After that, first the polygons are drawn, then the lines and finally the text for the single map images, all into separate layers as mentioned before. Geometric data of lines and polygon outlines are imported from the geodatabase using the SVG import and stroking capabilities for paths in GIMP. The drawing order of features is specified in the style configuration file.

#### Lines

Line geometry is the basic feature that can be rendered using SVG formatted data. The SVG path defining the line string is imported as a text string containing the SVG path information. Multiple paths can be imported at once but it is important to notice that all the imported paths are also drawn at the same time. Paths cannot be stored in different layers and have to be removed from the image session after they have been drawn. The drawing of the line paths is taking place applying the style parameters which are set as the so called context of the image. Context information includes the following information:

- Name of the brush

- Name of the paint method (e.g. Pencil, Airbrush, etc.)
- Stroke size (in pixel)
- Foreground and background color
- Name of the brush dynamics to use

Once the context is set to the desired parameters, the paths can be drawn using the vector stroking functionality of GIMP that was already explained in chapter X.

### **Polygons**

Outlines and the computed hatching are rendered in the image in the same way as regular lines which is described in the paragraph before. However, polygons can also be drawn differently than that using masks or just a fill color. A mask can also be defined using a SVG vector path and a PDB function is available to apply the vector outline as a selection of image pixels that provide a mask in a raster format. The mask can then be used on an imported image that has the same characteristics as a background image. Like the background, it has to be seamless for tiling and have an image size that is 256 pixel or an integral multiple of it. The interactive creation of images that meet these requirements using GIMP is described in chapter 3.2.

### **Text**

Text characters are also defined by their vector outline with its fill drawn on the image in a raster format which is similar to a mask selection. These outline vectors can be obtained from every text area that was added to the image by a PDB function. Aside from that, the text drawing basically makes use of the methods that have already been explained in various combinations that are offered for the image rendering. In this experimental project, text can only be added for areas i.e. for polygon features. Its name is queried from the database along with the coordinates of the centroid that have been transformed to coordinates relative to the image dimensions.

Two different methods can be considered to render the text. One option is that it is drawn with a specified fill color, a font and a font size as a text area which is then rendered in a raster on the image canvas. Or alternatively, the outline of the text vectors that are

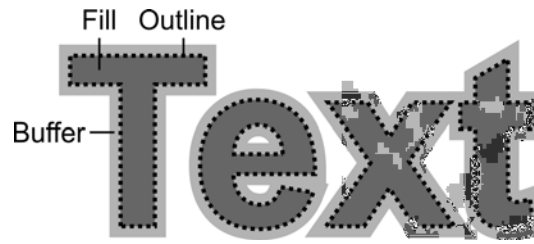


Figure 5.15: Schema text rendering

defined by a font and size can be stroked like a path using a brush and appropriate styling parameters. Of course both methods can also be combined using different colors, for example with the outline being drawn on top of the regular text fill. However, adding solely the text on top of the other feature layers which have been added before makes it hard to distinguish from these features as the density of objects in the map increases. For this reason, the possibility to add a buffer around the text is also added using a PDB function for the growing and shrinking of a pixel selection on the image. The selection is produced utilizing the vector outline as a raster pixel selection. A number of pixels can be specified to perform a growing of this area to get a buffer around the initial text characters. This selected area can then either be filled with a color or used as a mask again. Using the buffer as a mask on the background image for example produces a buffer that appears homogeneously with the background and makes the image appear less cluttered. Figure 5.15 shows the different elements of a text label that can be rendered.

### 5.3.5 Output and WMTS

The final output format of the images that are rendered during the process can be specified in the configuration file as described before. Results can either be stored as a Portable Network Graphics (PNG) image file or a XCF GIMP file which preserves all elements of the rendering e.g. layers and masks and therefore allows supplementary editing with GIMP. An output location can either be defined in the configuration file or left blank which stores the generated files within a results folder along inside the project directory that is named by the process and the system time at the start of the rendering process. However, a few things need to be considered regarding the tile generation. The tile images are already stored in a WMTS compatible directory structure and by default only PNG graphics are created. Inside the defined output directory a tree structure of folders is created based on the tilename scheme for slippy maps that was introduced in section 4.5.



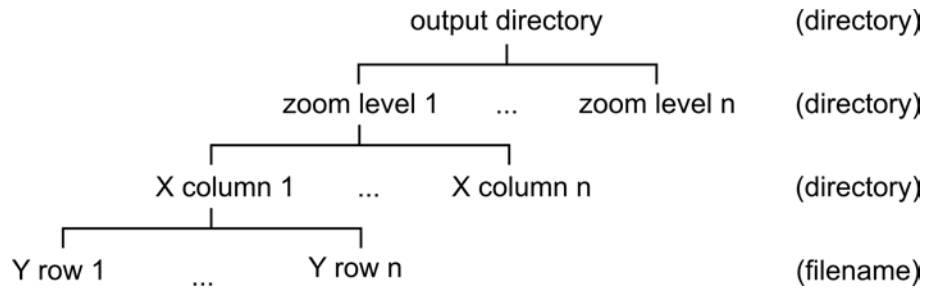


Figure 5.16: Schema tile naming

A directory is created for each zoom level to which subdirectories are then added. These subdirectories are named after the tile position in horizontal direction (column) which is calculated from the geographic bounds. In these folders, the actual image files are stored with the vertical position (row) of the tile as the filename. Figure 5.16 illustrates this structure of the tile images that are georeferenced in this manner.

Tile images that are stored in a directory structure like that can easily be used in a web mapping application. To make the tiles that are created in this rendering process available via the Internet in such an application, the Web Map Tile Service (WMTS) standard is used. It is introduced in section 4.5 and provides a common way to serve map tiles in a client-server architecture. GIS software and mapping libraries like OpenLayers or Leaflet can handle and display this data. For the realization of a WMTS server for pre-rendered tiles in this project, the open-source software *TileServer PHP* was used <sup>1</sup>. It is developed and maintained by Klokantech and makes use of an standard Apache PHP web server that meets certain requirements. These are the support of `.htaccess` and the `mod_rewrite` rules that can be configured for every Apache webserver installation. A single PHP script is placed on a server and all available tile layers are displayed in a browser interface. Tiles are stored within a designated directory and a JSON file with specifications about the extent called `metadata.json` needs to be stored along with the zoom level directories. Additionally, a password protection can be set to restrict the access to the service.

<sup>1</sup><https://github.com/klokantech/tileserver-php>



## 6 Processing results and evaluation

In this chapter, the results of the processing that was implemented as described in the previous chapter are presented and evaluated. This includes map and tile images on the one hand and text rendering on the other. Problems that occurred concerning the rendering parameters are discussed in the evaluation.

### 6.1 Results

#### Maps

Example renderings based on the embroid style map that was developed as a interactive process are presented along with a different style that is based on other painting parameters. Furthermore examples of an additional sketchy or hand-drawn styling are presented. For all examples a single map and a solution with tiles that are conform with the WMTS standard are given. In addition, the capabilities of the text rendering that is possible with GIMP are shown. In the processing, only line and area OSM features were used which provide sufficient content for a sample map. These cover the road network, rivers and major areas of landuse, including buildings. Relevant OSM tags for line and polygon features that provide enough content to fill a map of a urban area are listed in the appendix (table A.3 and A.4).

The aim of the work was to provide an automated solution to the interactive map rendering process that was explained in section 3.3.3. In this context, a styling inspired by Chinese embroidery was developed and as a first step, this style had to be prepared for the automation. At first, the used background and fill images for the area masks had to be created by hand as the interactive processing could not be adequately automated. Furthermore, this technique can be used for other styles as well. Cutouts of the originals



Figure 6.1: Embroid tiles at zoom level 14 (left), 15 (center) and 16 (right)

were transformed to seamless and tileable images for the filling masks as described in section 3.2. Furthermore, the styling parameters had to be stored in the JSON style format that was introduced in section 5.3.1. The files that define the styling can be found in the *style* directory of the *gimpmaps* application and is called *embroid*. It can be found on the compact disc that is handed in with this thesis.

The styling parameters like the colors have been slightly changed due to the adjustments that had to be made. In addition, a different data source was used and therefore the selection tags of the map features are not exactly the same. This results in a different classification of some features but a generally similar categorization is achieved. Based on this styling, the tiles are rendered and a result is shown in figure 6.1 where screenshots of the web application tiles are shown for three different zoom levels. It can be seen that different features are displayed at each zoom level and the building polygons are generalized. Additionally, only buildings of a minimum size are rendered for each zoom level according to the definitions in the style configuration file. And even though the brush that is used has a random component in its brush dynamics, significant visible inconsistencies at tile borders do not occur. This can be contributed to the disturbance effect that is a result of the random brush angle which covers possible visible inconsistencies.

These tiles are provided in a WMTS which can be accessed via the Internet URL ([http://wwwpub.zih.tu-dresden.de/~s4410781/ma/tiles\\_embroid/](http://wwwpub.zih.tu-dresden.de/~s4410781/ma/tiles_embroid/)). Screenshots of the web application and the tiles displayed in an OpenLayers web application and QuantumGIS are displayed in appendix A.2 and A.3. For this example application, an area around the city center of Dresden was rendered for the zoom levels 14 to 18. The resulting 2606 tiles occupy approximately 350 megabytes of disk space and the rendering of these took around



Figure 6.2: Chalk tiles at zoom level 14 (left), 15 (center) and 16 (right)

90 minutes.

Additionally, another style was created which is based on the chalk-like brushes which GIMP has already integrated. The brush Chalk 02 was used for the stroking, polygons are filled with a simple color and a freely available chalkboard like seamless image from the website [patternlibrary](http://thepatternlibrary.com/#chalkboard)<sup>1</sup> is applied as the background. Examples of the web application tiles are displayed in figure 6.2 for three different zoom levels like for the embroid style.

Aside from the tiles, an option to render a single map is provided. In the configuration, the desired scale can be defined along with the extent. For the defined extent, which is 1700 by 1200 meters, with a given scale of 1:1000 and a desired print quality of 300 DPI the resulting image has a resolution of 1295 by 914 pixel.

Finally the results of the hand-drawn emulation have to be presented. Figure 6.4 shows the rendering with sketchy lines, polygon outlines and hachures for different disturbance factors. The GIMP Brush Oils 02 was used for this result along with a hatching angle of 35 degrees and a spacing of 8 pixel.

## Text

To analyze the text rendering capabilities of GIMP, experimental text rendering techniques are implemented. As the placement of labels on maps is an own field of research that requires various parameters, in the scope of this work the focus was reduced on rendering possibilities with GIMP. Based on the available text functions of GIMP, methods were implemented to add labels of areas. The positioning was restricted to a simple point-on-

<sup>1</sup><http://thepatternlibrary.com/#chalkboard>



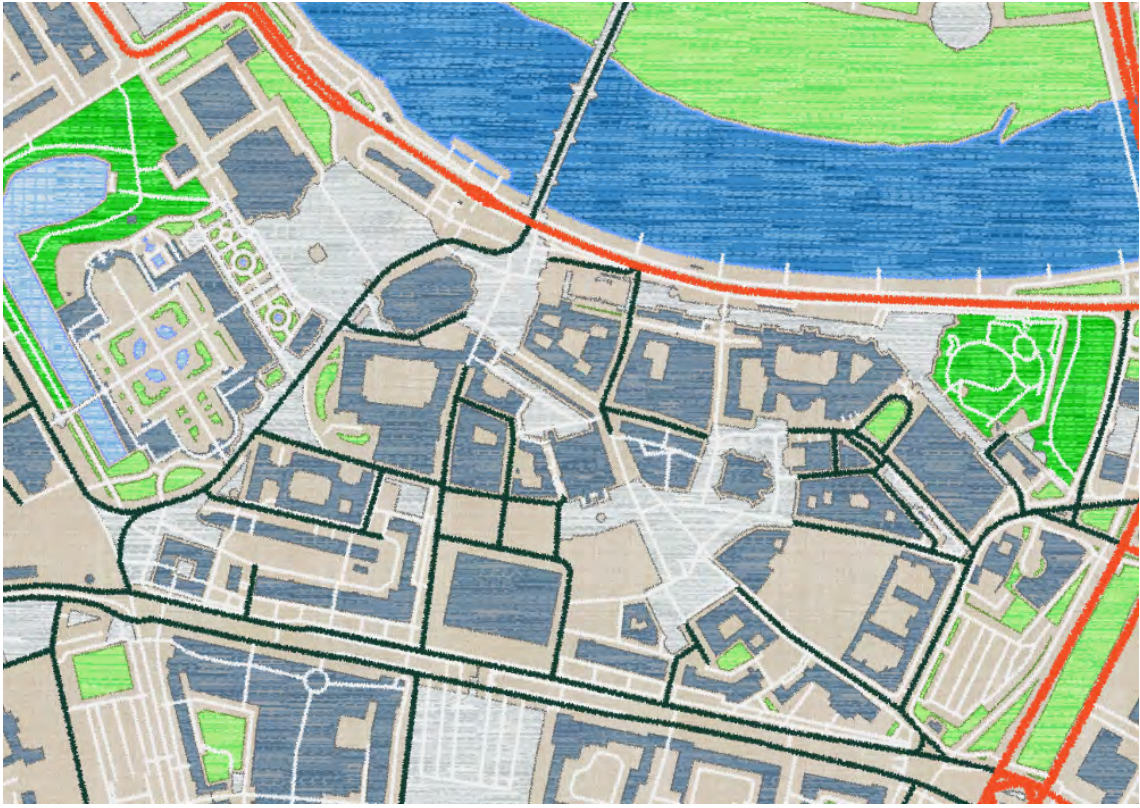


Figure 6.3: Embroid map with polygons as masks plus outlines



Figure 6.4: Sketchy map renderings with factor 1, 2 and 5 (left to right)



Figure 6.5: Text label rendering methods: fill, fill and outline, outline (left to right) and no buffering, buffering with fill color and buffering with background image mask (top to bottom)

polygon approach as it is provided by PostGIS functions. Therefore, the labeling has only been tested for the single maps and not for the tiles. This would require a full meta tiling technique that has to be added to the existing workflow.

With the available functions some example renderings for single map images were achieved. Examples for a cutout of the inner city of Dresden are shown in figure 6.5 for the label 'Altstadt' (German for 'old city'). The label is derived from the OSM value name for polygons which have a value of 9 or 10 for the `admin_level` key. It illustrates the technique which is based on the outline vectors that can be used as a vector path (see section 3.2). The freely available font 'Short Stack' provided by Google Fonts<sup>1</sup> was used for these examples. Figure 6.6 shows an example of the rendered map with the fill-plus-buffer effect (1st column, 2nd row) used for the inner city of Dresden for the OSM key-value pair `tourism=attraction`.

The text rendering capabilities are limited to drawing a text that is rendered in a predefined font and its outlines. This is a basic functionality which can be compared to other DTM tools and GIS software. The additional option to draw and outline employing a custom brush adds some artistic value to the rendering. Using random dynamics on the outline also results in a hand-made appearance. Also, the technique of creating a buffer that is melted with the background image is a technique to create an artistic effect. However, the labeling depends on other influencing factors like the positioning or multiple tile covering that can not be controlled by GIMP and the functions that are provided via the Python-

<sup>1</sup><https://www.google.com/fonts/specimen/Short+Stack>



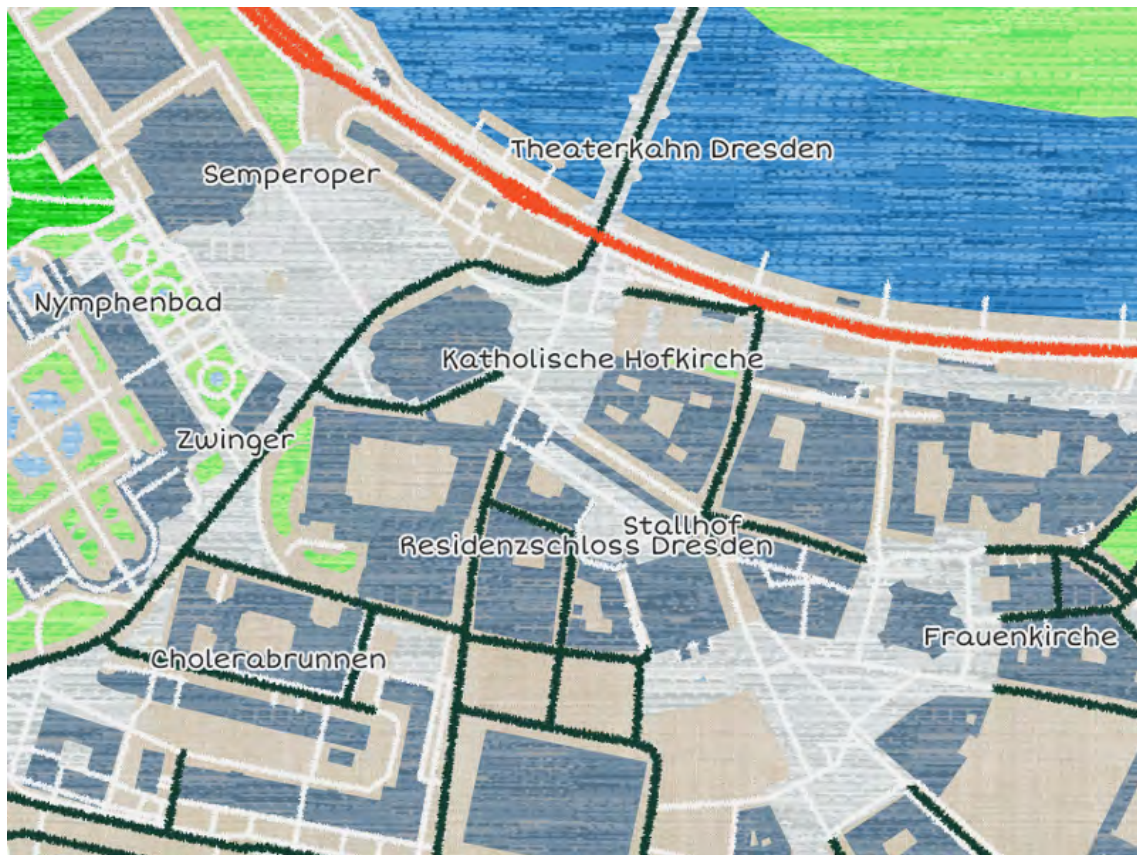


Figure 6.6: Text label rendering example for the OSM KVP `tourism=attraction`



Fu scripting interface. This needs to be implemented using GIS capabilities or custom code.

## 6.2 Limitations and potential

In general, GIMP with its utilities offers a wide range of possible style variations that could be applied. One requirement is however the creation of seamless images by hand if it is desired. Custom brushes or dynamics can be created by the user and linked with the scripted rendering. The use of JSON files enables the user to define the styling in detail but it is necessary to get familiar with the basics of the syntax. Large style files contain much information that can also be confusing on a first glance and requires some familiarity for the user to keep an overview. However, the JSON files have proven as a reliable format to add additional scripting parameters during the continuous development of the software.

One of the major drawbacks of the scripting interface is the lack of an option to emulate brush dynamics. This is an option in the GUI but not available via the scripting interface yet as this feature was first introduced with version 2.6<sup>1</sup> of GIMP. A possible benefit of this function for a more artistic and hand-drawn rendering is shown in figure 6.7. A sample road network is imported as a path in GIMP and drawn using the Stroke Path option. On the Left Side is the rendering result as it can be achieved using scripting on the right side the result with emulated dynamics. These include factors line Pressure, velocity or the direction of the pen and are usually determined by a input device like a computer mouse or graphics tablet.

The coverage of functionality provided via the PDB is in general not fully equivalent with the capabilities of GIMP. The interface is not very well documented and still lacks some functions and has a few bug. For example the function *pdb.gimp\_context\_set\_opacity* to set the opacity of a brush is not working. Another shortcoming is the lack of parameters for random elements like brushes, dynamics or filters in GIMP. An option for specifying a seed value would be a possible solution to this problem. Especially the artistic filters would provide great opportunities for a custom styling but they are not usable for tile creation as they are not deterministic. Satisfactory effects on single map images can be

---

<sup>1</sup><http://docs.gimp.org/2.6/en/gimp-introduction-whats-new.html>

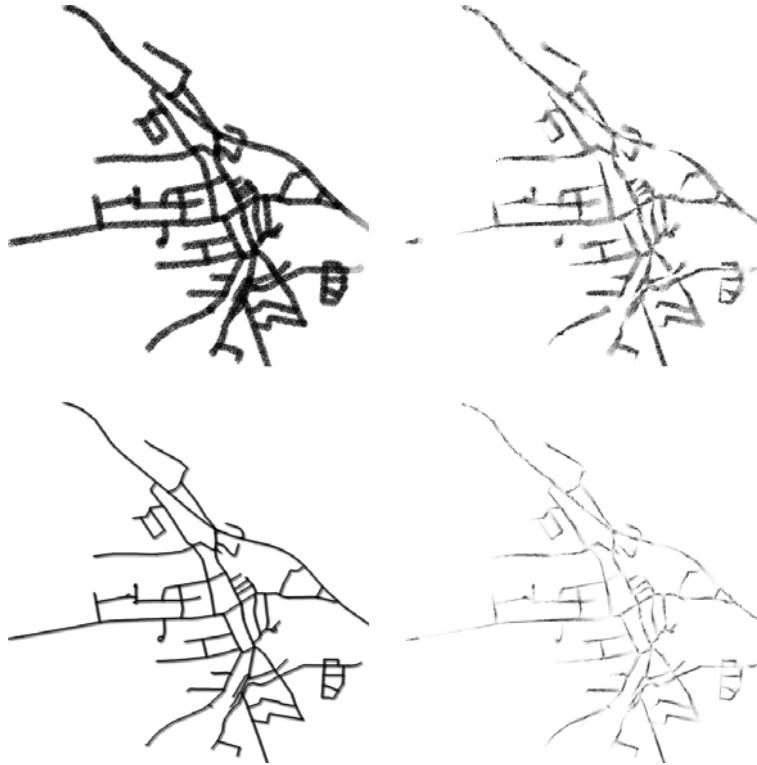


Figure 6.7: Brush comparison: without (left) and with emulated dynamics

achieved though.

A major problem of the hand-drawn emulation with the sketching module occurred with the unification of polygons that are touching, intersecting or contained by others. The unification is carried out during the generalization and is necessary to create more homogeneous polygon areas. If a polygon is on one tile and has a relation to a polygon that is on another tile, the other polygon is not taken into account for further processing. This is sometimes the case, e.g. in the way that OSM treats large polygons like Rivers. They are not stored as a single polygon but of a series of adjacent polygon parts. As the result, different polygon outlines for adjacent polygons are generated in different tiles after the polygons are unified in the last step. This problem illustrated in figure 6.8 and examples is shown in figure 6.9. In consequence of that, outlines and hachures are computed differently for different outlines and are therefore not consistent for all tiles. Due to that, the hand-drawn emulation is only working properly for lines in the map tiles. A possible solution would be a preprocessing of the geodata or the implementation of a recursive polygon union function that aggregates connected features. This would require additional effort in customization of the geodatabase functions and it is not guaranteed

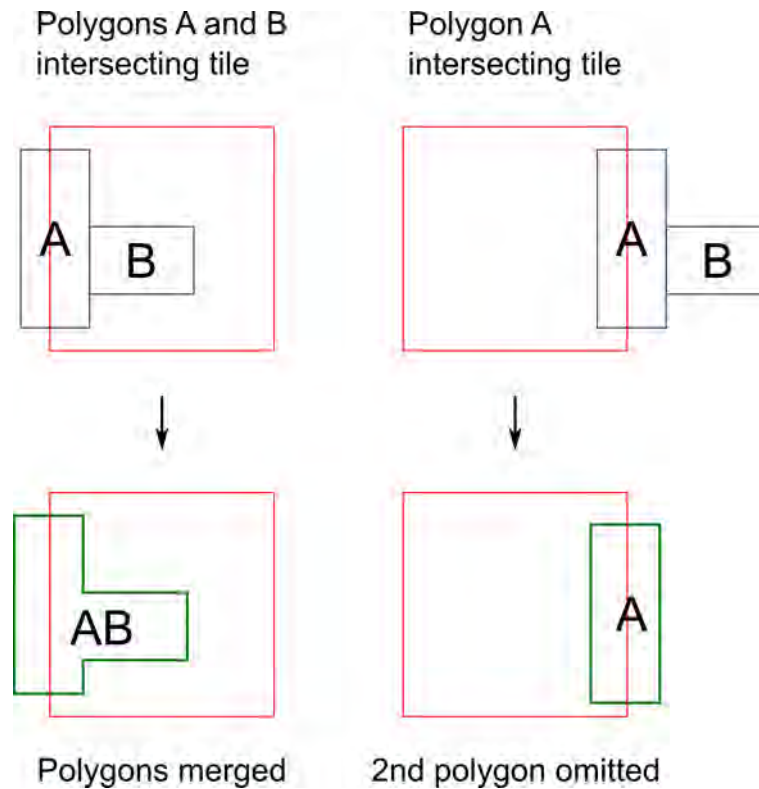


Figure 6.8: Polygon union problem

that PostGIS can provide such a procedure in a way that it can be used as an additional query in this workflow.

Another reason for preprocessing the dataset is the fact that a decrease of processing time for the actual rendering process could be achieved. The rendering involves many different steps and the GIMP interface as well as the database connections for various queries can be considered as time consuming processes. Possible preprocessing could include a zoom-level independent simplification with preserved topology or a common low level generalization progress. This would however make the maintenance of the OSM geodata more complex as the import and updating cannot be carried out in the way it is possible with the existing implementation. A limitation concerning the labeling is the lack of a vectorized text skeleton. Only the outline is available but for a more custom rendering, the possibility to stroke the text skeleton with a custom brush as well would be a valuable additional feature. This could contribute to a more artistic appearance as it would enable the hand-drawn rendering for text as well. However, this would require a complete new approach to the vector-based text rendering that cannot be realized inside GIMP but with additional implemented technology. Research has already been carried



Figure 6.9: Polygon union problem example: The green area north of the river and the river are correctly spanning the tiles towards the center of the image but there is an inconsistency on the right

regarding this field but not with respect to cartographic representation. What needs to be considered as well is the fact that the OSM data does not include world wide data about coastlines and country borders. These are added to the standard OSM Mapnik rendered map using Shapefiles from the Natural Earth website<sup>1</sup>. However, the data is not contained in an OSM dump that was used for this processing and did not appear in any of the renderings because the extents did not cover coastal areas. In general it is not possible to work with other sources than OSM as the entire database querying approach is strongly tailored to OSM. A possibility to handle other data sources as well would be desirable but would require the entire process to be re-structured. Taking the non-photorealistic hand-drawn rendering into account, a huge potential for possible additional techniques exist. More seed-controlled random components could be implemented to achieve different line alterations in addition to the existing ones. Some improvements in the line smoothing would also be desirable and the amount of possible approaches is unlimited. Furthermore, even more artistic painting techniques like multiple line representations of the same line or the splitting of lines into many short lines to simulate single brush marks could be implemented additionally. A possible approach would also be to reduce lines to single points for a punctiform abstracted rendering. Geometric shapes are

<sup>1</sup><http://www.naturalearthdata.com/downloads/>

also not implemented yet. This would require standardized outline rendering techniques for geometric shapes like squares, rectangles, circles or ellipses. A possible application of these primitives would be the generation of hand-drawn looking map symbols that would add more point related information to the maps. Finally, the filling of the polygons could be realized in many different ways. Aside from parallel hatching which is implemented, other hatching techniques could be created like zig-zag or cross hatching. It would also be worth analyzing how hatching could be used as an option to add three-dimensional information to the rendering as stylized shadows for example. Alternatively, completely different fillings are imaginable, consisting of random or curly lines or randomly placed primitives like circles.



## 7 Conclusion

### 7.1 Summary

The aim of this work was to implement an automated artistic rendering workflow which could be realized using the open-source image editing software GIMP. As a result of this work it can be stated that the automation with the GIMP scripting interface has the potential to create adequate results. Example maps were rendered that feature different styles, an experimental text rendering was implemented and an approach to create web map tiles was described. However, the scripting interface lags behind the possibilities that are available for interactive image editing inside the GIMP GUI and therefore it is worth to consider the further development of GIMP by its community in the future. It could be demonstrated that custom styled web application compatible tiles can be generated based on freely available OSM geodata without preprocessing which enables a continuous updating of the data as it keeps changing. These tiles can be displayed in a WMTS application but for some rendering options, there are shortcomings that are impeding the realization of seamless tiles. Techniques of non-photorealistic rendering can be implemented using different randomization approaches with SVG and custom brushes. This offers a wide range of visual abstraction techniques which are supporting an artistic and hand-drawn rendering of geodata. Randomization techniques however require strict rules for the rendering of tiles and have to be implemented with great care regarding deterministic results. To sum it up, GIMP can be adequately employed as a tool for cartographic rendering but there are some restrictions that are disadvantageous compared to other tools from the GIS or DTM field.

## 7.2 Outlook

For the further development of the implemented workflow, it is at first important to see how the evolution of the image editing software GIMP is continuing. Looking back on a twenty year long history of advancement, it is remarkable that it provides a valuable open-source alternative to proprietary software product with such a wide range of functionality. Many features have been implemented recently and it can be assumed that the capabilities will be extended continuously. For this reason, it is likely that the scripting interface will become more comprehensive and more elaborated. This may of course also require some maintenance of the existing code and adaption to possible new standards. Further improvements of GIMP could also contribute to its use as a cartographic tool to create maps based on a raster approach and not vector-based like DTM or GIS applications.

Additionally, other software could be taken into account and its automation potential could be evaluated. Adobe Photoshop for example, an even more powerful proprietary image editing software lends itself to a detailed examination regarding its automation capabilities for map and tile creation.

Another field of interest that offers great potential for hand-drawn rendering is the meaningful use of random components. Some approaches of abstraction were introduced in section 2.2.2 and some have been developed for this work. The different approaches though offer a wide range of combination possibilities and possible applications for different results are as versatile as art itself. Developing these techniques further can be considered as a main goal of non-photorealistic rendering approaches and as an ongoing challenge in this field of research. The meaningful abstraction of data for simplification or artistic reasons is a claim that will require future investigations. In this context, the raster editing capabilities of GIMP could be examined and a workflow could be added for a further raster based processing of the rendered tiles that is not restricted to vector based sketching and brush stroking techniques. These processes have to be deterministic to ensure a valid tiling result but NPR techniques are not always depending on random calculations and so are the GIMP filters.

Taking the latest developments on the market for custom styled web maps into account it is obvious that this is a field on increasing interest. Many people have a desire for a map styling that goes beyond ordinary map purposes like data visualization and naviga-



tion. However, the generation and serving of pre-rendered tiles is a resource consuming process that could be avoided using vector tiles for example. On the other hand, the best way to create real artistic rendering results is probably the raster approach with image processing functionality that works only on existing or pre-rendered tiles. With increasing technological possibilities though, it might be possible to perform a raster based image processing on a web server upon client request. Until then, vector tiles can be considered as the most adequate solution to provide customized styling. For this reason, standards for the serving of geodata with custom styling need to be developed on the base of vector tiles and it would be of general interest to have an official standard for that implemented by the OGC to make it more easy for people to create custom maps, tailored to their needs or satisfying their artistic and creative desire.



## Appendix A Images and Tables

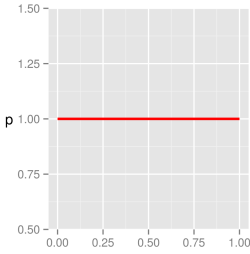
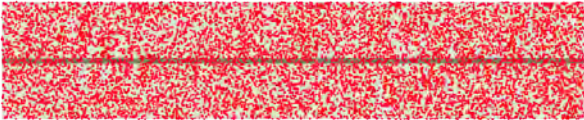
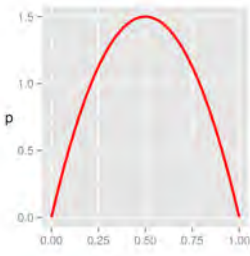
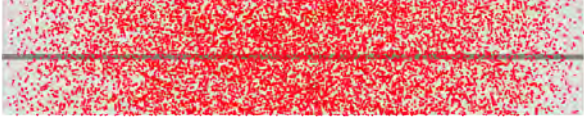
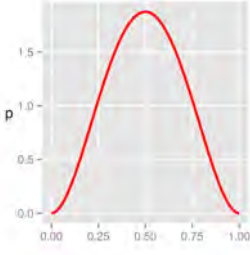
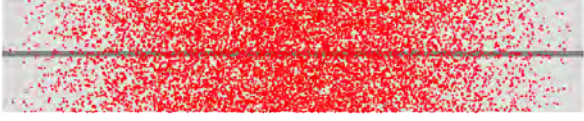
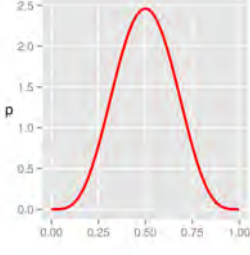

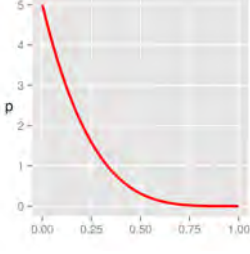
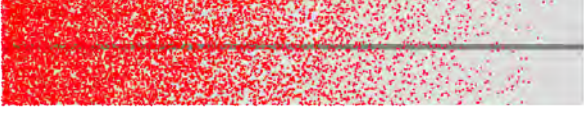
Distribution	Density function	Example
Uniform		
Beta (2,2)		
Beta (3,3)		
Beta (5,5)		
Beta (3,1)		

Table A.1: Random distribution examples

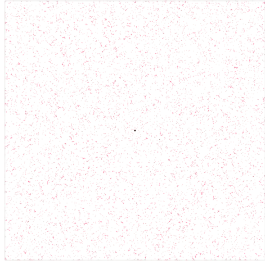
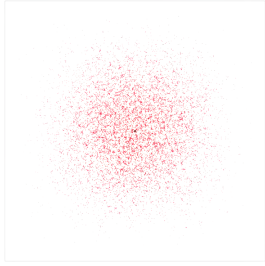
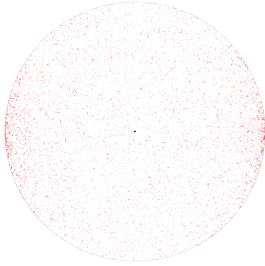
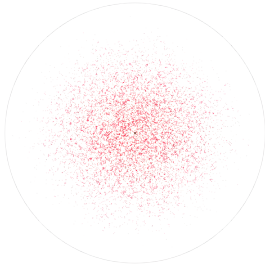
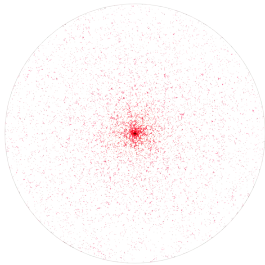
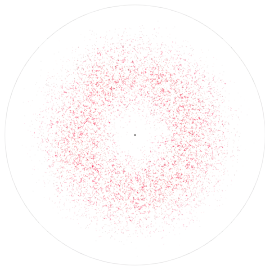
Name & Distribution	Perimeter shape	Example	Computation time (in ms)
<i>square</i> , Uniform	Square with equal sides being twice the length of the given radius		39.294
<i>square_beta</i> , Beta	same as square		191.130
<i>circle</i> , Uniform	circle around original point with given radius using Cartesian coordinates		45.396
<i>circle_beta</i> , Beta	same as circle		194.234
<i>polar</i> , Uniform	circle around original point with given radius using Polar coordinates		35.705
<i>polar_beta</i> , Beta	same as polar		121.374

Table A.2: Random point displacement

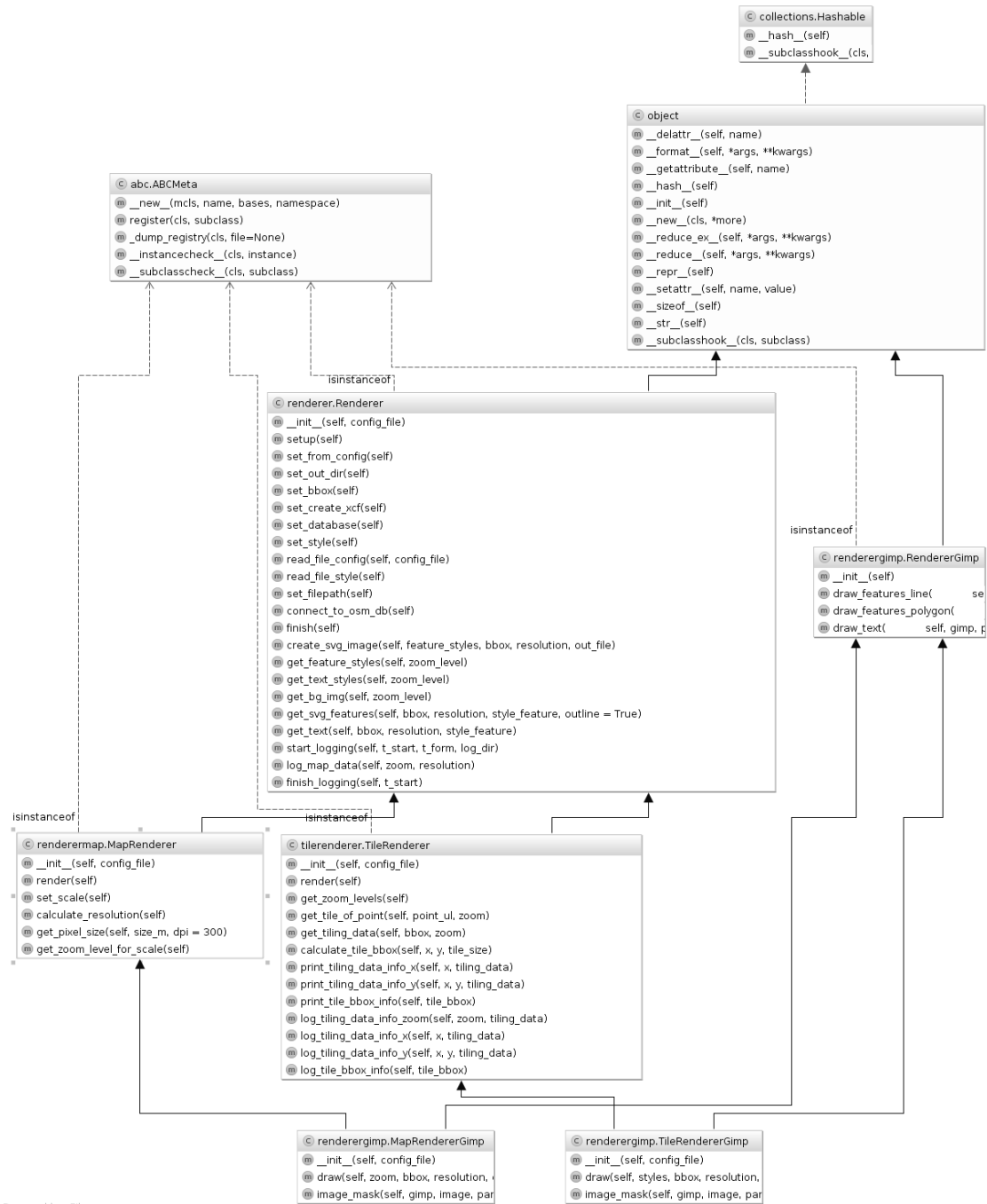


Figure A.1: Full UML diagram of the rendering classes

Type	Tags	Description
Road network	highway='motorway' highway='motorway_link' highway='trunk' highway='trunk_link' highway='primary' highway='primary_link' highway='secondary' highway='secondary_link' highway='unclassified' highway='tertiary' highway='tertiary_link' highway='residential' highway='living_street' highway='service'	Road types in hierarchical order, resembled by different brush sizes
Paths/Tracks	highway='pedestrian' highway='footway'	Major footways
Rivers	waterway='river'	Small rivers and streams

Table A.3: Selected OSM line features

Type	Tags	Description
Landuse	highway='motorway' landuse='forest' landuse='village_green' landuse='grass' landuse='recreation_ground' leisure='park'	Areas with a vegetation land cover
Water	waterway='riverbank' natural='water'	Water areas, e.g. large rivers and lakes
Buildings	building IS NOT NULL	All buildings
Other areas	area='yes'	Areas, e.g. bridges or pedestrian areas

Table A.4: Selected OSM polygon features

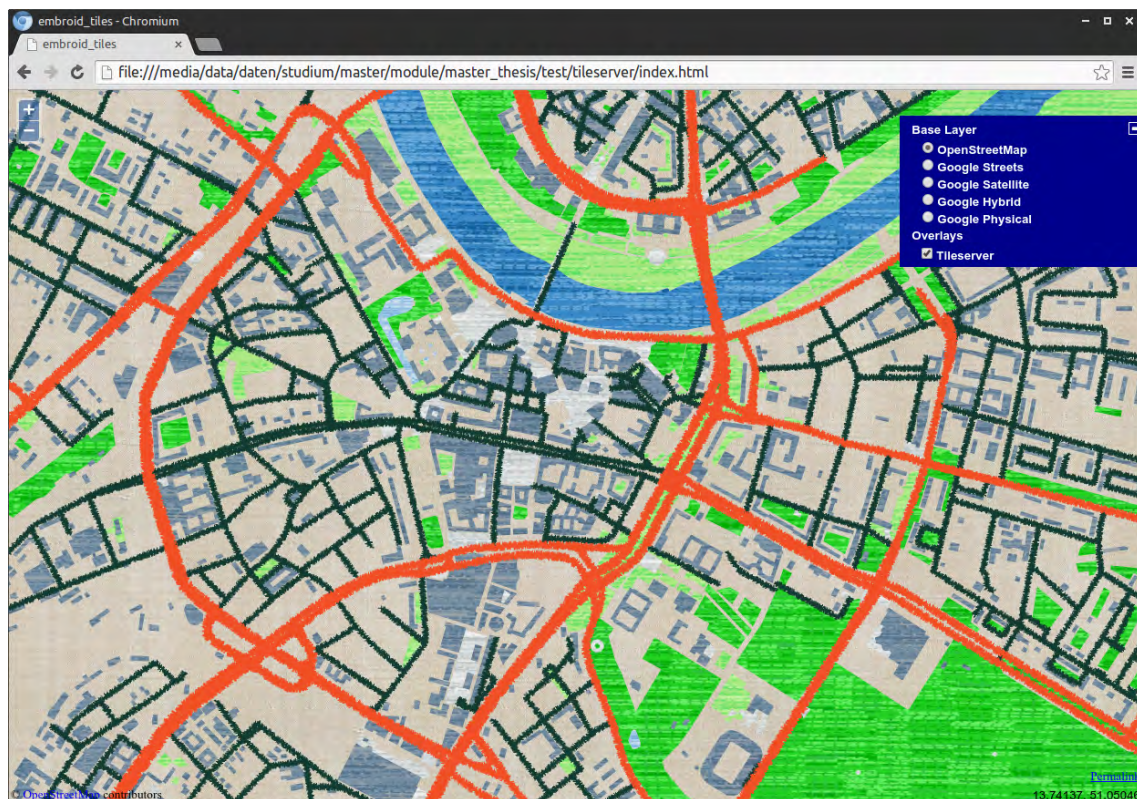


Figure A.2: WMTS application OpenLayers



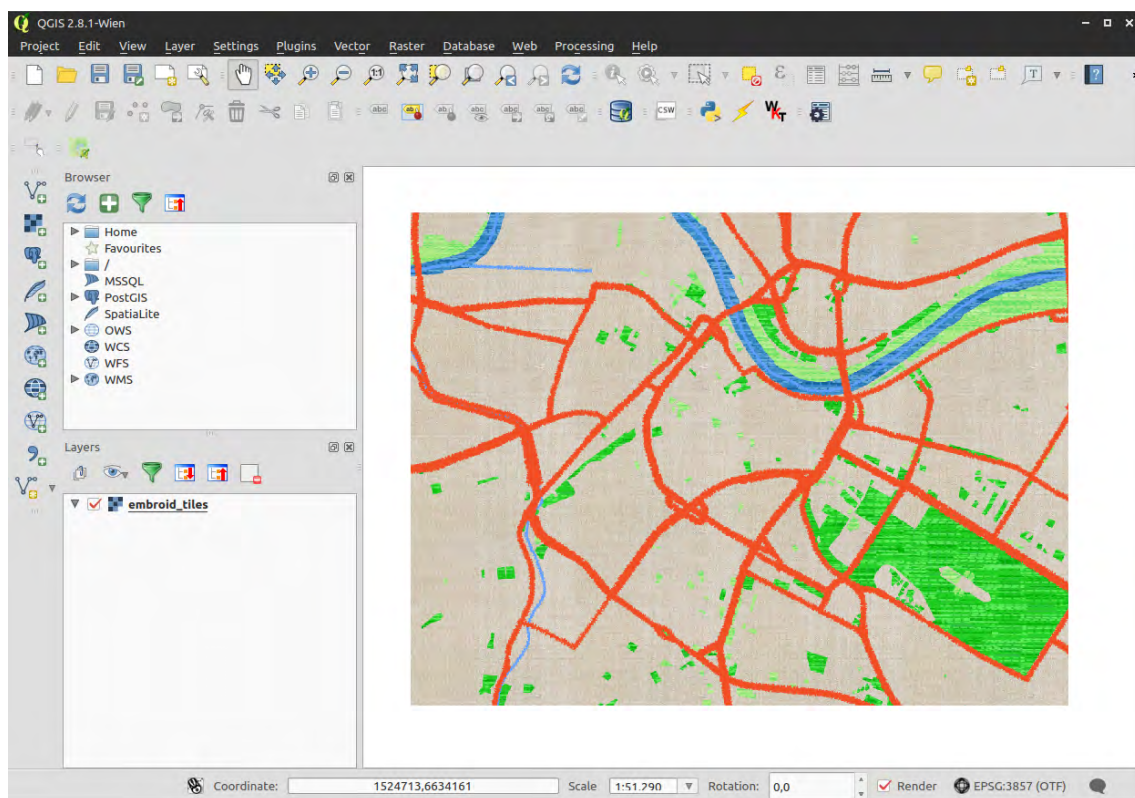


Figure A.3: WMTS application QGIS



## Appendix B Compact disc content

`master_thesis.txt`: The pdf version of the thesis

`data`:

- `database`: A dump of the OSM database for the Dresden area
- `gimp_files`: Created GIMP files (bruhes and dynamics)
- `gimprenderer`: Code for the GIMP map and tile rendering

`gimpmaps`: Tile-/Maprenderer modules

`conf`: Configuration files

`div`: Simple HTML map viewer and JSON schema files

`styles`: Style files

`create_gimpmaps.py`: GIMP Plug-in script

`sketching`: Sketching modules

- `map_examples`: Example map renderings
- `sql`: SQL Script with database functions
- `tileserver`: TileServer PHP example



# Bibliography

## Literary references

- Arsanjani, J. J. et al.** (2015). “An Introduction to OpenStreetMap in Geographic Information Science: Experiences, Research and Applications”. In: *OpenStreetMap in GIScience: Experiments, Research and Applications*. Ed. by J. J. Arsanjani et al. Springer Science and Business Media, New York.
- Ashikhmin, M. et al.** (2009). *Fundamentals of Computer Graphics, Third Edition*. O’Reilly Media, Sebastopol CA, pp. 1–3.
- Bassett, L.** (2015). *Introduction to JavaScript Object Notation: A to-the-point Guide to JSON*. O’Reilly Media, Sebastopol CA, pp. 1–5.
- Bertin, J.** (1983). *Semiology of Graphics: Diagrams Networks Maps*. The University of Wisconsin Press, Madison, pp. 84–86.
- Burdziej, J.** (2011). “Using Web Map Service (WMS) with Styled Layer Descriptor (SLD) for creating dynamic and user oriented maps”. In: *roceedings of the 11th International Symposium on Location-Based Services*. Ed. by G. Gartner and H. Huang. Research Group Cartography, Vienna.
- Cartwright, W.** (2003). “Maps on the Web”. In: *Maps and the Internet*. Ed. by W. Cartwright. Elsevier Science, London.

- Cartwright, W.** (2008). “Delivering geospatial information with Web 2.0”. In: *International Perspectives on Maps and the Internet*. Ed. by M. P. Peterson. Springer-Verlag, Berlin Heidelberg.
- (2009). “Art and Cartographic Communication”. In: *Cartography and Art*. Ed. by W. Cartwright and G. Gartner. Springer-Verlag, Berlin Heidelberg.
- Cartwright, W., Gartner, G., and Lehn, A.** (2009). “Maps and mapping in the Eyes of Artists and Cartographers – Experiences from the International Symposium on Cartography and Art”. In: *Cartography and Art*. Ed. by W. Cartwright, G. Gartner, and A. Lehn. Springer-Verlag, Berlin Heidelberg.
- Cerba, O. and Cepicky, J.** (2012). “Web Services for Thematic Maps”. In: *Online Maps with APIs and WebServices*. Ed. by M. Peterson. Springer-Verlag, Berlin.
- Clouston, A. and Peterson, M. P.** (2014). “Tile-Based Mapping with Opacity”. In: *Developments in the Theory and Practice of Cybercartography: Applications and indigenous mapping, Second Edition*. Ed. by D. R. F. Taylor and T. P. Lauriault. Elsevier, Amsterdam.
- Dong, L.** (2015). *The Derivation of Digital Embroid Styled Maps. TU Dresden, Master thesis.*
- Field, K.** (2009). “Editorial Preface - Art in C’art’ography”. In: *The Cartographic Journal* Vol. 46 No. 4 Art & Cartography Special Issue. The British Cartographic Society.
- Field, K. and Demaj, D.** (2012). “Reasserting Design Relevance in Cartography: Some Concepts”. In: *The Cartographic Journal* Vol. 49 No. 1. The British Cartographic Society.
- Gaffuri, J.** (2012). “Toward web mapping with vector data”. In: *Geographic Information Science: 7th International Conference, GIScience 2012, September 18-21, Proceedings*. Ed. by N. Xiao et al. Columbus, OH, USA.

- Gartner, G.** (2009). "Web mapping 2.0". In: *Rethinking maps: New frontiers in cartographic theory*. Ed. by M. Dogde, R. Kitchin, and C. Perkins. Routledge, New York.
- Gooch, B. and Gooch, A.** (2001). *Non-photorealistic rendering*. A K Peters Ltd, Natick, pp. 1–28.
- Goodchild, M. and Li, L.** (2012). "Assuring the quality of volunteered geographic information". In: *Spatial Statistics* 1.
- Hertzmann, A.** (2010). "Non-Photorealistic Rendering and the Science of Art". In: *Proc. Int'l Symp. Nonphotorealistic Animation and Rendering (NPAR)*.
- Isenberg, T.** (2013). "Visual Abstraction and Stylisation of Maps". In: *The Cartographic Journal* Vol. 50 No. 1. The British Cartographic Society.
- Isenberg, T. et al.** (2006). "Non-photorealistic rendering in context: an observational study". In: *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, June 05-07, 2006, Annecy, France.
- Jones, C.** (2013). *Geographical Information Systems and Computer Cartography*. Routledge, New York, pages.
- Kent, A. J.** (2005). "Aesthetics: A Lost Cause in Cartographic Theory?" In: *The Cartographic Journal* Vol. 42 No. 2. The British Cartographic Society.
- Kresse, W. and Danko, D.** (2012). *Springer Handbook of Geographic Information*. Springer-Verlag, Berlin Heidelberg, pp. 549–556.
- Krygier, J. B.** (2000). "Cartography as an art and science?" In: *The Cartographic Journal* Vol. 32 No. 6. The British Cartographic Society.

- Lechthaler, M.** (2010). "Interactive and Multimedia Atlas Information Systems as a Cartographic Geo-Communication Platform". In: *Cartography in Central and Eastern Europe*. Ed. by G. Gartner and F. Ortig. Springer-Verlag, Berlin Heidelberg.
- Li, S., Dragicevic, S., and Veenendaal, B.** (2011). "Advances, challenges and future directions in web-based GIS, mapping services and applications." In: *Advances in Web-based GIS, Mapping Services and Applications*. Ed. by S. Li, S. Dragicevic, and B. Veenendaal. Taylor & Francis Group, London.
- Lienert, C. et al.** (2012). "Current Trends in Vector-Based Internet Mapping: A Technical Review". In: *Online Maps with APIs and WebServices*. Ed. by M. Peterson. Springer-Verlag, Berlin.
- Lum, E. and Ma, K.-L.** (2001). "Non-Photorealistic Rendering using Watercolor Inspired Textures and Illumination". In: *IEEE Transactions on Visualization and Computer Graphics*, 5(01).
- MacEachran, A. M.** (1995). *How Maps work: Representation, Visualization and Design*. Guilford Publications, New York, pp. 8–12.
- Mauldin, S.** (2015). *Data Visualizations and Infographics*. Rowman & Littlefield, London, pp. 27–29.
- Mooney, P. and Corcoran, P.** (2013). "Understanding the Roles of Communities in Volunteered Geographic Information". In: *Progress in Location-Based Services*. Ed. by J. Krisp. Springer-Verlag, Berlin Heidelberg.
- Obe, R. and Hsu, L.** (2015). *PostGIS in Action*. Manning Publications Co., Shelter Island, pp. 4–9.
- Palazzolo, A. and Turnball, T.** (2012). *Mapping with Drupal*. O'Reilly Media, Sebastopol CA, p. 20.



- Peck, A.** (2008). *Beginning GIMP: From Novice to Professional (Expert's Voice in Open Source) 2nd Edition*. Springer Science and Business Media, New York.
- Peterson, M. P.** (2008). "International Perspectives on Maps and the Internet: An Introduction". In: *International Perspectives on Maps and the Internet*. Ed. by M. P. Peterson. Springer-Verlag, Berlin Heidelberg.
- (2012). "Online Mapping with APIs". In: *Online Maps with APIs and WebServices*. Ed. by M. P. Peterson. Springer-Verlag, Berlin.
- (2014). *Mapping in the Cloud*. The Guilford Press, New York, p. 31.
- (2015). "Evaluating Mapping APIs". In: *Modern Trends in Cartography: Selected Papers of CARTOCON 2014*. Ed. by J. Brus, A. Vondrakova, and V. Vozenilek. International Publishing, Switzerland.
- Ramm, F., Topf, J., and Chilton, S.** (2011). *OpenStreetMap: Using and Enhancing the Free Map of the World*. UIT, Cambridge, 3–5 and 307.
- Reed, C** (2011). "OGC Standards: Enabling the geospatial web". In: *Advances in Web-based GIS, Mapping Services and Applications*. Ed. by S Li, S. Dragicevic, and B. Veenendaal. Taylor & Francis Group, London.
- Rosin, P. L. and Lai, Y.-K.** (2013). "Artistic minimal rendering with lines and blocks". In: *Graphical Models Volume 75, Issue 4*. Elsevier.
- Sample, J. and Ioup, E.** (2010). *Tile-Based Geospatial Information Systems: Principles and Practices*. Springer Science & Business Media, New York, pp. 127–129.
- Sayeed, R. and Howard, T.** (2006). "State of the Art Non-Photorealistic Rendering (NPR) Techniques". In: *Theory and Practice of Computer Graphics* M. McDerby, L. Lever (Editors), EG UK.

- Schmidt, M. and Weiser, P.** (2012). “Web Mapping Services: Developments and Trends”. In: *Online Maps with APIs and WebServices*. Ed. by Peterson M. P. Springer-Verlag, Berlin.
- Semmo, A. et al.** (2015). “Cartography-Oriented Design of 3D Geospatial Information Visualization – Overview and Techniques”. In: *The Cartographic Journal*.
- Strothotte, T. and Schlechtweg, S.** (2002). *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Elsevier Science, San Francisco, p. XVII.
- Tateosian, L. G. and Healey, C. G.** (2004). “NPR: Art Enhancing Computer Graphics. Technical Report TR-2004-17”. In: Knowledge Discovery Lab Department of Computer Science, North Carolina State University.
- Watson, R.** (2009). “Mapping and Contemporary Art”. In: *The Cartographic Journal* Vol. 46 No. 4 Art & Cartography Special Issue. The British Cartographic Society.
- Winkenbach, G. and Salesin, D. H.** (1994). “Computer-Generated Pen-and-Ink Illustration”. In: Department of Computer Science and Engineering, University of Washington.
- Wood, C. H. and Keller, C. P.** (1996). “Design: Its place in Cartography”. In: *Cartographic Design: Theoretical and Practical Perspectives*. Ed. by C. H. Wood and C. P. Keller. John Wiley & Sons Ltd, Chichester.
- Wood, D.** (2006). “Map art”. In: *Cartographic Perspectives*, Number 53.
- Wood, J. et al.** (2012). “Sketchy rendering for information visualization”. In: *IEEE Transactions on Visualization and Computer Graphics*, 18(12).

## Internet references

*Mapbox map styles*. Accessed: 03.08.2015. URL: <https://www.mapbox.com/editor/#style>.

*Mapzen map styles*. Accessed: 03.08.2015. URL: <http://tangrams.github.io/carousel/>.

*OpenGIS WMTS Specification*. Accessed: 05.08.2015. URL: <http://www.opengeospatial.org/standards/wmts>.

*Stamen map styles*. Accessed: 03.08.2015. URL: <http://maps.stamen.com>.

*SVG W3C recommendation*. Accessed: 03.08.2015. URL: <http://www.w3.org/TR/SVG/>.