



NTNU – Trondheim
Norwegian University of
Science and Technology

An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications

Robert Patrick Victor Nordan

Master of Science in Engineering and ICT

Submission date: June 2012

Supervisor: Terje Midtbø, BAT

Norwegian University of Science and Technology
Department of Civil and Transport Engineering



Report Title: An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications	Date: 10.06.2012			
	Number of pages (incl. appendices): 109			
	Master Thesis	X	Project Work	
Name: Robert Patrick Victor Nordan				
Professor in charge/supervisor: Terje Midtbø				
Other external professional contacts/supervisors:				

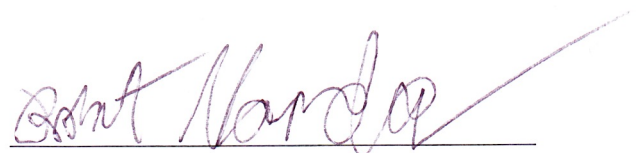
Abstract:

Vector tiling is a new trend that the geospatial industry is likely to explore in coming years, bearing the promise of the advantages in clarity and interactivity afforded by vector data whilst also providing a cacheable and efficient solution akin to raster tiles. An important question is then to ascertain how one might ensure that topological metadata is preserved across tiles; i.e. how does one convey the fact that two lines on adjacent tiles are in fact part of the same road?

This report aims to explore this question by assessing current vector tile solutions, and creating hypothetical solutions for a vector tile system that delivers tiles with topology preserved in line with the Simple Features Access Specification. Some of the most promising of these are selected for prototyping, and the prototypes are tested with regard to speed and functionality. Finally conclusions about suitable methods are drawn based on these tests. Furthermore, the suitability of using vector tiles for a Geographic Information System (GIS) application is discussed.

Keywords:

1. Vector Tiles
2. GIS
3. Web Maps
4. WFS



An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications

Master thesis TBA4925

Stud.techn. Robert Patrick Victor Nordan

Division of Geomatics

Department of Civil and Transport Engineering

Faculty of Engineering Science and Technology

Norwegian University of Technology and Science (NTNU)

Abstract

Vector tiling is a new trend that the geospatial industry is likely to explore in coming years, bearing the promise of the advantages in clarity and interactivity afforded by vector data whilst also providing a cacheable and efficient solution akin to raster tiles. An important question is then to ascertain how one might ensure that topological metadata is preserved across tiles; i.e. how does one convey the fact that two lines on adjacent tiles are in fact part of the same road?

This report aims to explore this question by assessing current vector tile solutions, and creating hypothetical solutions for a vector tile system that delivers tiles with topology preserved in line with the Simple Features Access Specification. Some of the most promising of these are selected for prototyping, and the prototypes are tested with regard to speed and functionality. Finally conclusions about suitable methods are drawn based on these tests. Furthermore, the suitability of using vector tiles for a Geographic Information System (GIS) application is discussed.

Sammendrag

Bruken av vektordata tile å lage kartfliser er en ny trend som det er sannsynlig at geomatikkindustrien vil utforske i de kommende årene, siden det muliggjør den typen fordeler innenfor klarhet og interaktivitet som vektordata tilbyr, samtidig som det gir en effektiv løsning som kan mellomlagres slik som fliser laget med rasterdata. Et viktig spørsmål i så måte er hvordan man sikrer at topologiske metadata bevares mellom fliser, med andre ord: Hvordan formidler man det faktum at to linjer på tilstøtende fliser faktisk er del av det samme veistykket?

Målet med denne rapporten er å utforske dette spørsmålet ved å vurdere eksisterende løsninger med vektorbaserte kartfliser, og skape hypotetiske løsninger for et vektorbasert system som leverer kartfliser med topologien bevart i tråd med SFA-standarden. De mest lovende av disse utvelges for testing med prototyper der hurtighet og funksjonalitet utforskes. Til slutt konkluderes det på grunnlag av disse observasjonene. Videre vil også nytteverdien av å bruke vektorfliser i et geografisk informasjonssystem bli diskutert.

Preface

It's a bit bitter-sweet to write the preface to this thesis, as it marks the end of five great years at NTNU. (As always, the first thing the reader sees is the last thing the author writes.) I've been very happy as a student at the department of Geomatics, and I look forward to applying my skills in the industry. (I'm also looking forward to working nine to five, five days a week, since that will actually be less work than what I've been doing in the last year!)

From that perspective, this thesis has been a fitting final task at NTNU. It combines areas of interest to me, and explores areas that probably will be of interest to the geospatial industry in the coming years. I would like to thank Sverre Wisløff, Rune Aasgard and Harald Jansson of Norkart Geoservice for enlightening discussions that helped shape the research goals for this thesis, Terje Midtbø for his work as my advisor, and Mats Taraldsvik for continuing to be a useful sounding board for ideas whenever I wanted to discuss something. Finally, I wish to thank my parents for giving me a good enough start in life that I've ended up writing these words, and my beloved Christina once again helping to preserve my sanity while I've been working against a deadline.

Trondheim, June 10, 2012

Robert Patrick Victor Nordan
rpn@robvvn.net

Contents

Abstract	v
Preface	vi
Table of Contents	x
List of Figures	xi
List of Tables	xi
1 Background & Related Work	1
1.1 Interactive Browser Based Maps	1
1.1.1 Plug-ins Versus HTML & JavaScript	2
1.1.2 Raster Maps	3
1.1.3 Vector Maps	3
1.2 Tiling	5
1.2.1 Tiling in Vector Maps	7
1.3 Simple Feature Access Specification	8
1.4 Current Browser Based Map Frameworks	8
1.4.1 Plug-in Based Frameworks	9
1.4.2 HTML & JavaScript Based Frameworks	10
1.5 Current Vector Tiling Implementations	12
1.5.1 GIS Cloud	12
1.5.2 Polymaps & TileStache	14
1.5.3 Mapnik Metawriter	16
1.5.4 Nokia Maps 3D	17
2 Project Goals	19
2.1 Motivation	19
2.2 Project Outline	19
2.3 Desired Outcome	20
3 Challenges with Vector Tile Generation	21
3.1 Generalisation	21
3.2 Tile Sectioning & Overlap	22
3.3 Features With Multiple Geometries	22
3.4 Data Completeness	24
3.5 Unique Feature Identification	27
3.6 Rendering & Concatenation Order	27

4	Potential Data Structures & Algorithms	29
4.1	Global Feature Search	29
4.1.1	Tile Specification	29
4.1.2	Construction & Concatenation Algorithms	29
4.1.3	Probable Advantages	31
4.1.4	Probable Disadvantages	31
4.2	Edge Pointers	31
4.2.1	Tile Specification	32
4.2.2	Construction & Concatenation Algorithms	32
4.2.3	Probable Advantages	35
4.2.4	Probable Disadvantages	35
4.3	Central Feature Registry	36
4.3.1	Tile Specification	36
4.3.2	Construction & Concatenation Algorithms	36
4.3.3	Probable Advantages	37
4.3.4	Probable Disadvantages	37
4.4	Probabilistic Matching	38
4.4.1	Tile Specification	38
4.4.2	Construction & Concatenation Algorithms	38
4.4.3	Probable Advantages	39
4.4.4	Probable Disadvantages	39
4.5	Combined Approaches	40
5	Application Architecture	41
5.1	Server Side Application	42
5.1.1	Extensions	42
5.1.2	Tile Transfer Format	43
5.1.3	Cache	45
5.2	Client Side Application	45
5.2.1	Modifications & Supporting Code	46
5.2.2	Rendering & Concatenation Order	47
5.2.3	Integrated Web Server	47
5.2.4	Issues With Polygon Union	47
5.2.5	Automatic Tester	51
5.2.6	Demonstration Applications	53
6	Experimental Design	55
6.1	Data Structures & Algorithms Selected for Testing	55
6.2	Test Cases	56
6.2.1	General Case - Multiple Geometries	56
6.2.2	Special Case - Single Geometries	56

6.3	Testing Patterns	59
7	Results & Discussion	60
7.1	Algorithmic Execution Speed Analysis	60
7.1.1	Generalised Global Search	60
7.1.2	Specialised Global Search	60
7.1.3	Edge Pointers	61
7.1.4	Summary	61
7.2	Timing Results	61
7.2.1	Download Times	62
7.2.2	Concatenation Times	65
7.3	Non-Normality of Tile Loading & Concatenation Time Distributions	68
7.3.1	Implications	70
7.4	Feasibility of GIS Operations	71
7.4.1	Data Completeness	71
7.4.2	Projection & Distortions	72
7.4.3	Preservation of Data Integrity	73
7.4.4	Data Upload	73
8	Conclusions	75
8.1	Observations	75
8.1.1	Generalised Versus Specialised Methods	75
8.1.2	Importance of Added Data	75
8.1.3	Importance of Caching	75
8.2	Recommendations	76
8.2.1	Choice of Methods	76
8.2.2	Utility for GIS Operations	76
9	Future Work	77
9.1	Further Development, Speed & Compactness	77
9.1.1	Polygon Union in JavaScript	77
9.1.2	GIS Enhancements	78
9.2	Tile Load Distributions	78
	References	86
	Appendix	87
	A Project Assignment	87

B	Trying Out the Prototype	91
B.1	Online Demonstration	91
B.2	Running the Prototype Locally	91
B.2.1	Trying the Experimental Union	92
C	Electronic Attachments	93
D	Tilestache Modifications	94
E	Polymaps Modifications	95
F	Test Computer Specifications	96
G	Licences	97
G.1	Application	97
G.2	Project Report	97

List of Figures

1.1	Visualisation of a raster data structure	4
1.2	Visualisation of a vector data structure	5
1.3	Visualisation of a tile pyramid	6
1.4	Illustration of GISCloud.com interface	13
1.5	Demonstration of Polymaps Statehood example	14
1.6	Illustration of visible edges in Polymaps	15
1.7	Demonstration of Mapnik Metawriter	16
1.8	Illustration of tile loading in Nokia Maps 3D	18
3.1	Illustration of raster tile sectioning	23
3.2	Illustration of vector tile sectioning	23
3.3	Illustration of multiple geometries in The United Kingdom.	25
3.4	Illustration of multiple geometries in Portugal.	26
4.1	Illustration of edge pointer generation algorithm	33
5.1	Illustration of server-client interaction	41
5.2	Illustration of tile removal errors	46
5.3	Illustration of degenerate vertex edges	49
5.4	Illustration of errors in experimental union	51
5.5	Illustration of pseudo-union in practice	52
5.6	Illustration of a combined raster & vector tile application	54
5.7	Illustration of a pure vector tile application	54
6.1	Illustration of the general case data set	57
6.2	Illustration of the special case data set	58
7.1	Histogram of all 3998 download times for special data set, cached.	63
7.2	Histogram of all 1999 concatenation times for special data set, cached.	66
7.3	Histogram of 999 concatenation times for special data set, cached.	66
7.4	Estimated fit of Gamma distribution to the special case, experimental union data.	70

List of Tables

1.1	Summary of browser based map frameworks.	9
7.1	Number of usable samples from all test runs.	62
7.2	Mean download time from all test runs.	62
7.3	Download time variance from all test runs.	62
7.4	Standard deviation of download time from all test runs.	62
7.5	Mean concatenation time from all test runs.	65
7.6	Concatenation time variance from all test runs.	65
7.7	Standard deviation of concatenation time from all test runs.	65

1 Background & Related Work

1.1 Interactive Browser Based Maps

Interactive browser based maps are nearly as old as the internet itself, with the first attempts made as early as 1993.[1] This was less than two years after Tim Berners-Lee introduced the World Wide Web[2], but already then the basic concepts of a map window with tools for panning and zooming were in place. Seen with the eyes of today, they seem primitive and slow, for every movement entailed a full refresh of the web page. Map images had to be generated on the server and downloaded for every panning or zooming action.

In 1996, the commercial service called MapQuest was introduced. It made available features like routing to determine driving routes between places, and quickly became very popular. In fact, it held the largest market share of online map services until 2009[3]. While MapQuest certainly was a step up from previous efforts, it still performed panning and zooming operations with full page reloads. In this period, many map services moved from redrawing the map for each request to caching it, since the limited navigation options meant that only specific bounding boxes could be requested. (The implication being that those images could be drawn up in advance.)[4]

In the professional GIS field, internet map applications were also being developed, culminating in the Web Map Server (WMS) standard that was introduced in the year 2000.[5] It defined clear and interoperable means to fetch map data, but still required the data to be rendered upon each request.

In 2005, Google introduced their Maps service. Google Maps introduced a new paradigm, the so called "slippy map", where users can pan and zoom fluidly without reloading the page, and without having to install any additional plug-ins in their browser. This was accomplished through the use of raster tiles (see section 1.2 for more on tiling) and "Asynchronous JavaScript And XML", or Ajax as it is commonly known.[4] Ajax requests allow the browser to add new items to the web page by manipulating the HTML, and to initiate downloads of new data in the background.[6]

Another new concept introduced by Google Maps is the use of Application Programming Interfaces (APIs) to give other developers access to their map tools. As well as meaning that Google Maps could easily be integrated in any website, this inspired a blooming of neogeographic "mash-ups", where outside developers used Google Maps in conjunction with their own data to create new products.[7]

The Google Maps model quickly became popular, with all the other major map websites adopting it as well the then nascent OpenStreetMap (OSM) organisation. (OSM is a freely available map where all the background data is contributed by volunteers and also freely available.[8]) All of the HTML and JavaScript based web map frameworks listed in section 1.4 depend on Ajax for fetching data regardless of whether working with tiles or with WMS, something that proves its utility.

In the last few years web maps have had more functionality added to them, such as Google Street View [9] and similar services inspired by it, the ability for users to add their own points of interest, and support for mobile devices and geolocation. (Geolocation is a term applied when the user's device determines its position and supplies that to the map service.)[10] However, the basic principles behind map delivery do not seem to have changed that much. With the arrival and broad implementation of HTML 5 new possibilities have opened up, and new map frameworks have started arriving which make use of those features. (See section 1.4 for more information.) As a part of that, the possibility of making more use of vector tiles has appeared. This is something that will be explored more in this report.

1.1.1 Plug-ins Versus HTML & JavaScript

There are two main ways of delivering a browser based map: One is to use only technology available in the typical browser environment, and the other is to make use of extensions to the browser known as plug-ins. These plug-ins are hooked into the browser and provide facilities like use of a different programming language, rich libraries for development, connections to the underlying operating system or access to hardware components. Examples of plug-ins include Adobe Flash, Microsoft Silverlight and Oracle Java.[11]

Typically, while plug-ins have offered greater power and flexibility to the developer, they have also burdened the user with increased maintenance issues related to installing and updating them.[11, 12] Therefore, many users like to avoid them where possible, and one of the motivations behind the coming HTML 5 standard is to reduce the reliance on plug-ins.[13] It is the opinion of the author that HTML 5 will grow to be the dominant method of delivering map data on the web.

See section 1.4.1 for more details on the plug-ins currently used for mapping applications.

1.1.2 Raster Maps

Raster data is one of the two major ways of modelling spatial data. In a raster data model, geography is described as a grid of cells, where each cell has a value denoting the kind of feature that occupies the space represented by the cell.[14] (See figure 1.1 for an illustration of how raster data is constructed.) It can be used for data storage and analysis, but also for visualising data of any format. A typical example of this in a web context is data (that may originally be in a vector format) rendered to an image file for purposes of display. In other words a visual representation of the map data, which is how most casual map users interact with it.

Raster maps are completely dependent on the resolution of the data file where the source is a raster data file, or the resolution of the rendered image where it has been rendered from another type of data source. If one only has one resolution available, zooming in will result in the map data becoming steadily more block-like and harder to comprehend. Many systems for visualisation, such as WMS or tiling systems render the map data in many different resolutions as required so that the user is always presented with relatively smooth images.

1.1.3 Vector Maps

The other major way of representing spatial data is vector data, which is built up of points with given coordinates. The points can be connected to form lines, and lines can connect to form polygons, and through those three structures one can represent every kind of geographic feature.[14] (See figure 1.2 for an illustration of how vector data is constructed.) This data format is often used for analytic work, but also for visualisation. One major advantage of this approach is that one has easy access to all the relevant data of an object, allowing it to be queried for additional information if needed.

Vector data is independent of resolution, as for every zooming action taken the map image can be redrawn using the source data. The disadvantage is that the map image must be redrawn by the client for every action, and that the entire data set to be must be transmitted, even if it is more detailed then what is needed.

The Open Geospatial Consortium (OGC) has standardised a method for transmitting vector data over the web, called the Web Feature Service specification (WFS).[16] It allows the query of and transmission of vector data specified by a bounding box, using the Geographic Markup Language (GML). The Transactional

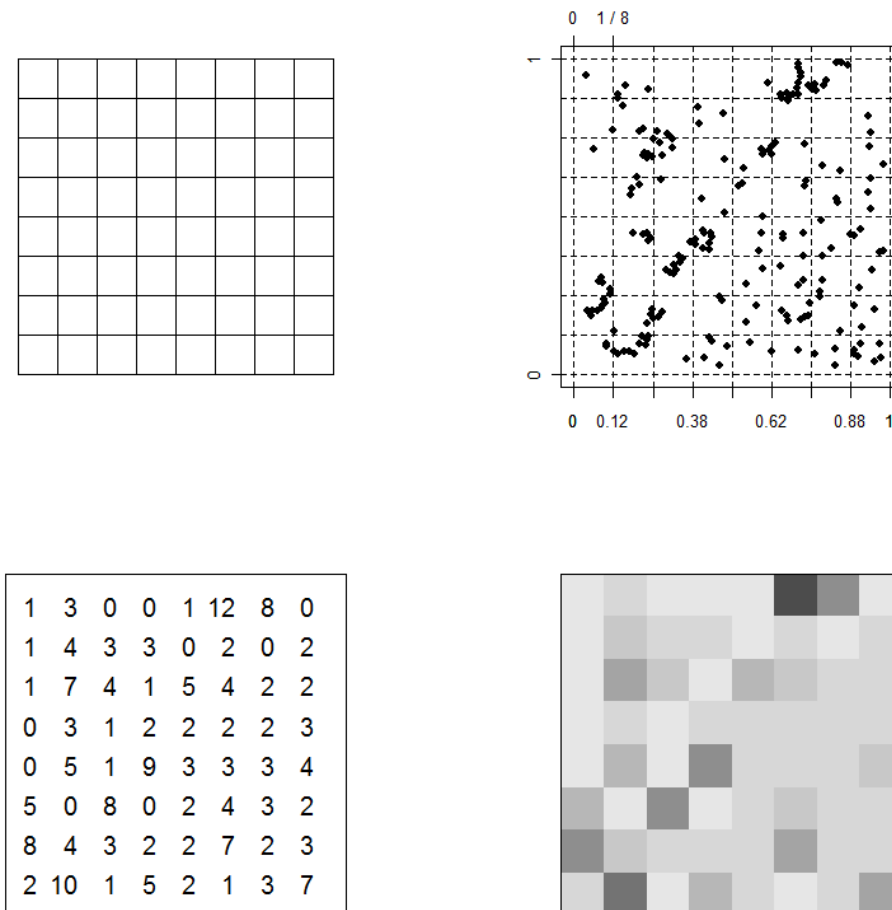


Figure 1.1: Visualisation of the the stages of converting a collection of points to a raster grid. 1) Define a grid size. 2) Superimpose grid on data points. 3) Count the number of points in each grid section. 4) Visualise the the grid using grey levels. (Illustration from [15])

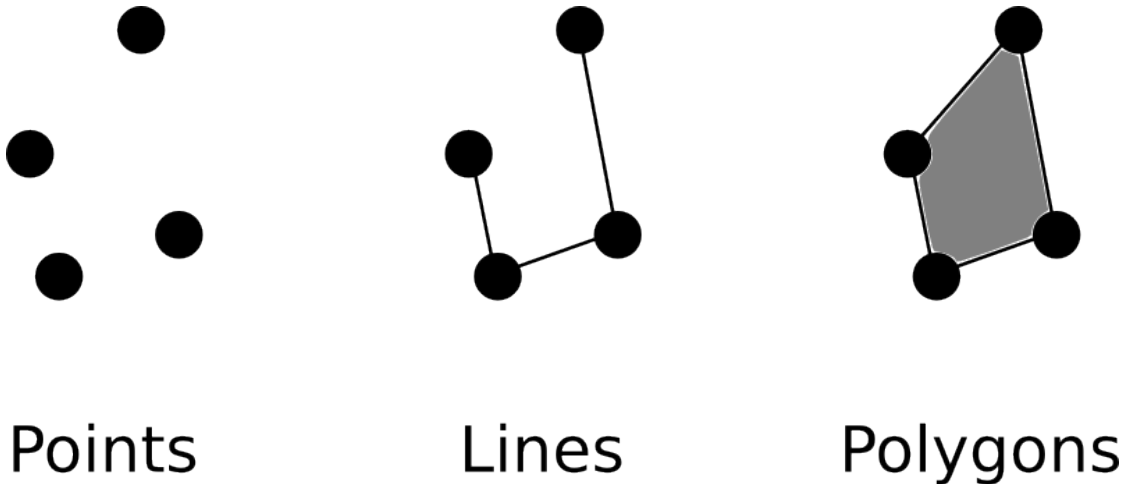


Figure 1.2: *Visualisation of the components that make up a vector data structure. Each data type is composed of elements of the previous data type.*

variant (WFS-T) also allows the user to upload data to the server for editing purposes in a GIS application. WFS, much like WMS, makes no reference to tiling implementations.

1.2 Tiling

Tiling is the name applied to a method for delivery of map data where the map client is sent many small pieces of a larger whole, which are then reassembled to be presented to the user. (The name being derived from the similarity it has to using many ceramic tiles to make up a larger surface like a floor.)

As mentioned previously, raster tiling techniques for web maps were popularised by Google Maps, and raster tiles are now among the most common forms of delivering maps on the web. There are a lot of non-standardised implementations of raster tile solutions available, but all of them share certain common traits. They are broadly compatible with each other, but may differ in details such as the position of the coordinate system origin.

Drawing on the experiences of the industry in developing tile solutions, the OGC coalesced the common features of the various tiling applications into the Web Map Tile Services (WMTS) standard. It defines common behaviour for creating interoperable tile server systems, and is a sign of maturity for this kind of solution. Noticeably, while the WMTS standard is based mainly on experiences

with raster tile systems, it has been future-proofed by not placing any restrictions on what kind of data is delivered in each tile. (It is instead up to implementing products to announce the kinds of data being delivered through methods defined by the WMTS, and connecting clients to handle that correctly.)

In order to section the rendering up into tiles, one takes as a starting point a single tile encompassing the entire world as projected in the chosen projection.[17] (This is known as zoom level zero.) Thereafter, one divides the tile up into four equal parts, where each resulting section equals a tile at zoom level one. Each of these tiles renders an area on quarter the size of the previous zoom layer onto an area the same size as the previous rendering, which means each geographic feature is magnified accordingly. This process is repeated as many times as needed to get to the desired zoom level. (See figure 1.3 for a visualisation.)

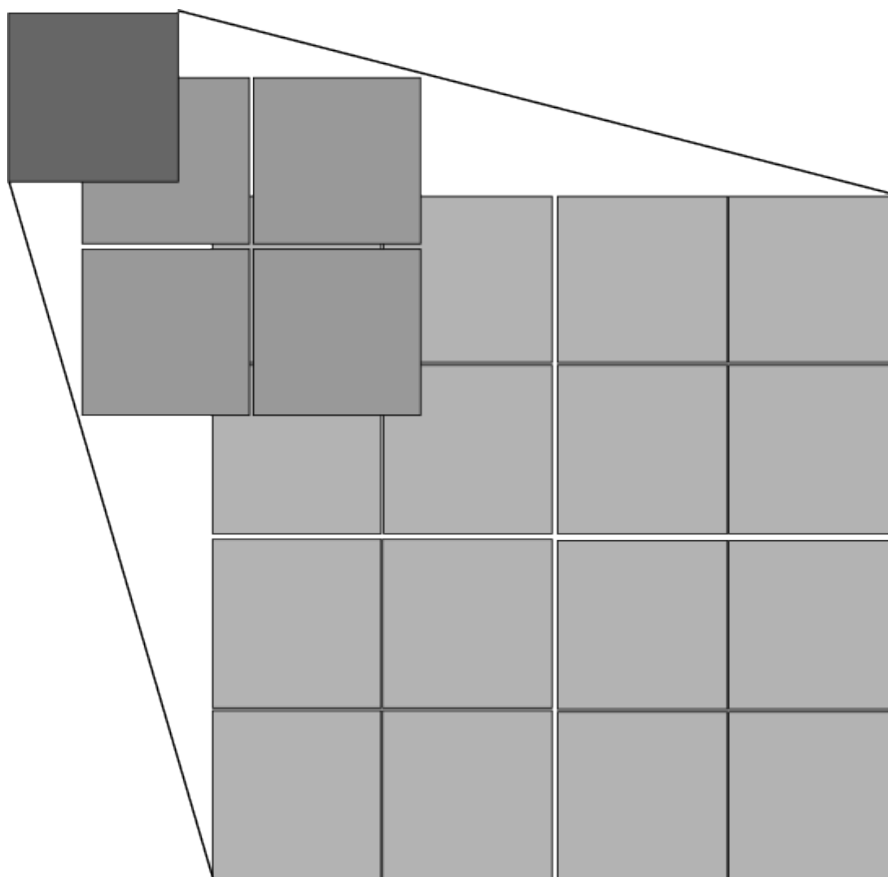


Figure 1.3: *Visualisation of a tile pyramid, showing how the contents of one tile at a low zoom level is split into many tiles at a higher zoom level.*

This way of sectioning up the map area is called an image pyramid, or more technically a quad tree index system.[18] The tile addressing system is a logical consequence of this way of sectioning tiles. Every tile is identified by a zoom number relating to the level of the tree, and a pair of row and column numbers within that zoom level. Together the three numbers make up a unique key for the tile. This key is easy to use for fetching tiles, as they can be stored in a folder system so that the tile Uniform Resource Locator (URL) becomes something along the lines of "http://example.com/maptiles/zoom/row/col.png".

Most raster tile systems use a spherical Mercator projection made popular by major commercial internet map providers such as Google and Microsoft, as well as the OpenStreetMap project. [19, 20] It has also been defined as a well-known scale set in the WMTS standard. (But it must be noted that this is by no means the only possible projection.) This projection, known as the EPSG:3857 "WGS 84 / Pseudo-Mercator"[21] uses metres as its unit of measurement. With the correct boundaries set, this projection will render a tile containing the world at zoom level zero as a square, which propagates down to all other zoom levels.

1.2.1 Tiling in Vector Maps

While using vector data for web maps has been the subject of a fair amount of scientific research, the question of creating a tiling solution with vector data so far has only been approached by a few researchers.[22, 23] On the other hand, members of the geospatial industry have recently expressed significant interest in this. [24, 25, 26] At least four organisations have deployed solutions using some variation of vector tiles, and one can assume that other companies are working on the same issues and will probably be following suit by deploying similar solutions in the near future.

The major potential advantage of using tiled vector maps is the ability to combine the extra information capacity of vector data with the speed of raster tiles. For example, it could be used to create a fast loading slippy map that also allows the user to highlight features of interest, query extra information and change map styles on the fly, without downloading any additional data.

As the development and deployment of vector tile solutions is at such an early stage, there are no formal standards for implementations. Furthermore there have not yet evolved any de facto standards as deployed by market leaders. A credible assumption would be that vector tiling will develop in much the same way as first Web Map Services and then Web Map Tile Services have: First diverse innovation,

then widespread adoption which leads to informal or de facto standardising, and finally a formal standardisation process. [27, 5, 17] Currently we are experiencing the first of these phases, and the work contained in this thesis is intended to be a part of that diverse innovation. (As a side note, there are also elements of the last phase in play due to the WMTS standard being format-agnostic, so there is a possibility that it may guide the further development of vector tiles.)

The most significant academic work on vector tile solutions that the author has been able to locate is [22], which explored many of the same areas as this report. Hopefully, this report will provide a more thorough examination of some of the aspects covered there, as well as shed light on some aspects not covered in that report.

1.3 Simple Feature Access Specification

The Simple Features Access Specification (SFA) is an OGC standard for accessing geographic information in a predictable and interoperable fashion.[28] It defines a baseline set of common geographic constructs that all conforming implementations must provide, with extensions for specific situations such as access through Structured Query Language (SQL).[29]. All OGC standards regarding data storage and vector data refer to the SFA, and most spatial databases and data formats also base themselves on this specification. (In general, the constructs contained in the SFA are constructs that were widely agreed upon and implemented in most geographic system even before the standards process started, as is often the case in a standardisation process.)

Since the current standard methods of dealing with vector data, WFS and GML, comply with the SFA it is of interest to establish in what ways vector tile solutions could be made to be compliant as well. This is particularly important if one was to consider using such solutions for GIS analysis work in addition to presentation.

1.4 Current Browser Based Map Frameworks

The following is an overview of current browser based map frameworks in common use, along with some of their defining characteristics. Table 1.1 summarises them for easy reference.

Name	Type	Map types
ArcGIS Silverlight	Silverlight, Proprietary	Raster & vector
ArcGIS Flex	Flash, Proprietary	Raster & vector
OpenScales	Flash	Raster & vector
OpenLayers	HTML4, JavaScript	Mainly raster, some vector
Leaflet	HTML5, JavaScript	Mainly raster, some vector & tiles
Cartagen	HTML5, JavaScript	Vector only
Polymaps	HTML5, JavaScript	Mainly raster, some vector & tiles
Tile5	HTML5, JavaScript	Raster
Ovi Maps 3D	HTML5, JS, WebGL, prop.	Vector

Table 1.1: Summary of browser based map frameworks. Frameworks are available under an open source licence except where noted.

1.4.1 Plug-in Based Frameworks

These frameworks are dependent upon the user having external plug-ins installed in their browser. Currently, the two plug-ins that are used for mapping applications as identified in the following section are Adobe Flash and Microsoft Silverlight, both of which are proprietary but freely available for installation in browsers. They provide general purpose programming environments which allow for many different types of interactivity, where mapping is just one. The Flash plug-in is available for Windows, Mac OS X and Linux operating systems as well as several mobile phone operating systems such as Android. (But not iOS, the second most popular smartphone operating system.)[30, 31] The Silverlight plug-in is only available for Windows, Mac OS and Windows Phone 7 operating systems.[32]

ESRI ArcGIS API for Microsoft Silverlight A commercial product from Environmental Systems Research Institute (ESRI), the world’s largest vendor of commercial geographical informations systems. [33] The ESRI ArcGIS API allows the creation of rich internet applications with mapping, and allows the use of both raster and vector data sources. The framework is closely tied to ESRI’s ArcGIS Server products and is designed to utilise them as much as possible, and while using the ArcGIS API for Silverlight is free of charge deploying a ArcGIS Server can be very expensive. The product depends on the Microsoft Silverlight plug-in, and cross platform support is therefore somewhat limited. [34, 35]

Vector tiles are not directly supported, but can probably be approximated through custom code. This would be accomplished by creating a special class that

requests and receives vector tiles in response to map movements, concatenates them and then inserts them into the map as a standard vector data type.

ESRI ArcGIS API for Adobe Flex Functionally more or less equivalent to the ArcGIS API for Silverlight, this product uses the Adobe Flash plug-in and is therefore supported on more platforms. According to ESRI, the choice between Flash and Silverlight platforms should depend entirely on what the developer is most comfortable with.[34]

OpenScales An open source framework using the Adobe Flash plug-in that can utilise both raster and vector data sources, with emphasis on compatibility with specifications set forth by the Open Geospatial Consortium. Originally began as a part of the OpenLayers project (see the next section) from JavaScript to ActionScript, it has evolved to take advantage of the capabilities of the Flash platform.[36] As a result it has vector capabilities on par with the ArcGIS API products.

Vector tiles are not directly supported, but can probably be approximated through custom code in the same way as previously noted.

1.4.2 HTML & JavaScript Based Frameworks

These frameworks only depend on features present in the browser to function. When a framework is said to depend on HTML 4, it will work on all current browsers as well as older browsers such as Internet Explorer 6. (Released in 2001.[37]) If the framework depends on HTML 5, it requires a newer browser. HTML 5 support has been added piece by piece to browser releases in the last few years, and can be said in general to be supported by the current versions of all major browsers. [38]

OpenLayers Currently the most popular open source web map framework[20], OpenLayers is JavaScript based and designed to function in older browsers such as Microsoft Internet Explorer 6.[39] It depends on HTML 4 and therefore does not make use of the newer features available in HTML 5, which is what allows it to support older browsers but also precludes it from making use of more powerful features in modern browsers. Historically it has not been very strong on mobile devices, but there has been some recent development to improve the support in that area.

While it does support vectors, it is known to have somewhat poor performance if many vectors are displayed simultaneously. [39] Vector tiles are not directly

supported, but can supported with custom code as has been done in [40]. (See section 1.5.3 for more information.)

Leaflet Leaflet is a relatively recent mapping framework which has already acquired a fair amount of traction. Using HTML 5 to gain speed and features at the expense of backwards compatibility, it focuses on a solid implementations of core features. The architecture is meant to be easy to extend via third party plug-ins for new functionality. Furthermore, equal weight is placed on compatibility with both desktop and mobile browsers.[41]

As mentioned in section 1.5.1, GisCloud uses Leaflet with some custom code and a proprietary back-end to allow for the use of vector tiles. This implementation appears to preserve topology between objects, but also makes compromises such as delivering objects that at the given scale would be visualised as single pixels as single pixel values without any other information. That reduces the possibilities for geographic analysis somewhat. Since it has been implemented there can be no doubt that vector tiling is possible with this framework, and a topologically coherent system should be achievable with a certain amount of custom code.

Cartagen Cartagen is an open source HTML 5 framework that focuses purely on vector data. Users can style the map using Geographic Style Sheets (GSS), and it allows great interactivity. Cartagen is closely tied to OpenStreetMap data, using a data format known as OSM-JSON for importing and exporting vector data.[42]

Cartagen is focused on vectors, but currently operates by reading in big vector data files. All generalisation and similar processing is done in the client, which can cause rather long processing times when drawing data. There is no support for vector tiles, but support could probably be approximated with some amount of custom code. In contrast to previous frameworks, this one has innate support for vectors but no support for tiles. Therefore the custom code would have to handle tile logic, download and concatenation before appending new vector info to the map.

Polymaps Polymaps is a relatively new mapping framework that aims to allow fast and easy rendering of maps, while using Scalable Vector Graphics (SVG) for displaying vector data. It also allows styling of map data with ordinary Cascading Style Sheets (CSS).[43] It makes use of features in HTML 5, thereby sacrificing backwards compatibility for a greater feature set. Polymaps is designed to be simple and easy to use for both cartographers and designers.

As mentioned in section 1.5.2, Polymaps supports vector tiles out of the box. They are however not topologically coherent, with visible seams between the tiles in the tiling demonstration.[44] Building a topologically coherent vector tile system based on this framework would probably be fairly straight-forward with some custom code that handles concatenation.

Tile5 An open source framework focusing on mobile devices and providing the same API regardless of data source, using HTML 5.[45] Has reached maturity, but active development has been reduced and the product is only being maintained.[46]

Supports vector data sources like GeoJSON, but not tiling directly. Could probably be approximated with custom code in a similar fashion to OpenLayers or Leaflet.

1.5 Current Vector Tiling Implementations

The following paragraphs detail four current vector tile solutions. Not all information pertaining to these solutions is publicly available, and they may depend on proprietary components on the server side. Information about them has been collected from public statements by the makers and by examination of client side behaviour in web browsers.

1.5.1 GIS Cloud

GIS Cloud is a company that offers GIS tools for use in an online, browser-based environment with data storage on their servers. (So called "cloud computing".)[47] As a part of these services, they have developed a mapping client that can utilise tiled vector data (among many data sources) for faster delivery and display.[48]

The tiles are highly optimised for speed and are generated on demand by a proprietary back-end. The tile addressing scheme seems to be similar to the common approach of establishing a pyramid as detailed in section 1.2. The features are pre-generalised and all coordinates are converted to screen coordinates, and every object that is calculated to have a display size of under a pixel is dropped and replaced by a single pixel value.[49] The tiles do not contain metadata, but every object (that is displayed as more than one pixel in size) is identified with a unique identification number. When a user clicks on a tile for more information, a POST request[50] with that identification number is sent to the server asking for information which is then received and displayed.

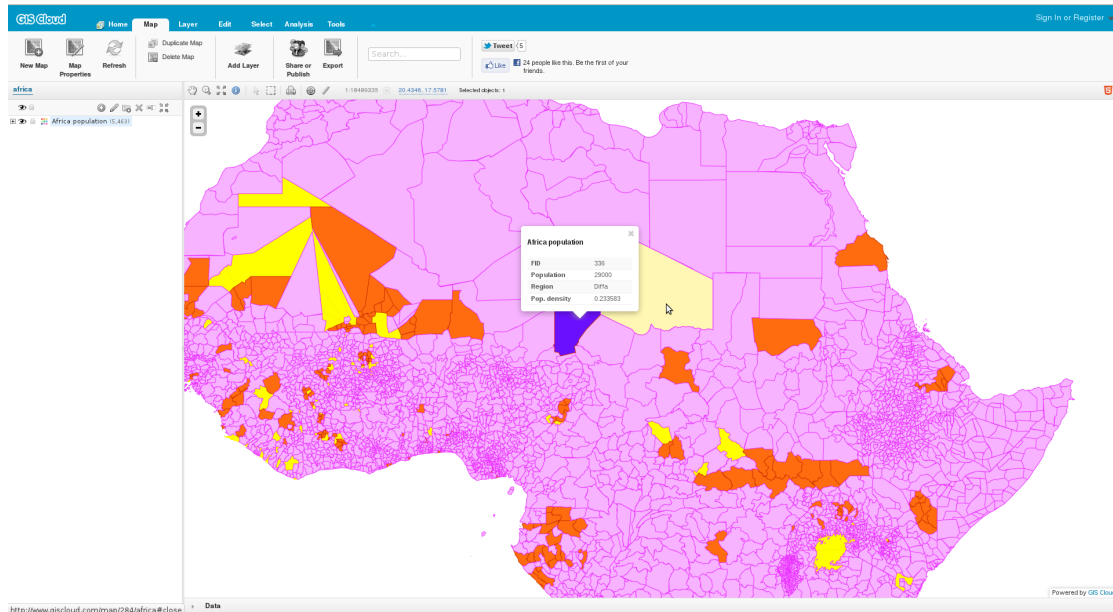


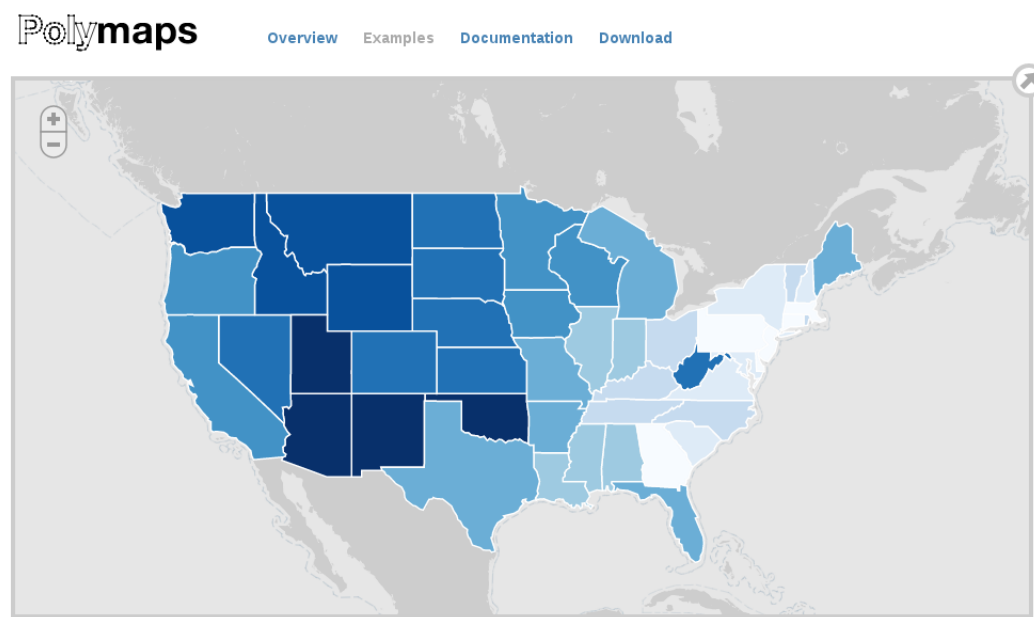
Figure 1.4: Illustration of GISCloud.com interface, showing a vector tile presentation of population in Africa.

It appears to be through this identification number that topology is preserved, as the individual tiles do not carry any other indications of objects in neighbouring tiles. Therefore the concatenation of tiles is probably accomplished by looking for features in neighbouring tiles with matching ID numbers and then creating a union of them, in a fashion akin to the method described in section 4.1. Unlike the Polymaps case, it is quite evident that the contents of the tiles are concatenated in some fashion, as hovering the mouse over a polygon will highlight the whole polygon with accentuated edges. If features on different tiles were not concatenated, the accentuated edges would not be along tile boundaries rather than borders.

The client side of GIS Cloud utilises a lot of open source code, like the Leaflet library, but the back-end is completely proprietary and tailored for their purposes.[49] As such it should be possible to examine the client code and build a similar client side solution based on the same tools. However, the most important part of any tiling solution is the server side back-end, and since the GIS Cloud back-end is proprietary not much can be learned about it other than publicly disclosed details. There is no way to directly build a vector tile solution on the GIS Cloud back-end, but the public details can be taken into consideration when designing a similar solution.

1.5.2 Polymaps & TileStache

Polymaps is a web mapping framework designed by SimpleGeo Inc. and Stamen Design, designed to take advantage of HTML 5 capabilities. One of the features of this framework that is displayed as an example [44] (See figure 1.5) is the ability to use vector tiles as a data source. There do not appear to be any production deployments using this yet.



Polymaps [Overview](#) [Examples](#) [Documentation](#) [Download](#)

© 2010 [CloudMade](#),
[OpenStreetMap](#) contributors,
CCBYSA. Colors by [Cynthia Brewer](#).

Statehood

This simple visualization uses color to encode the date each of the fifty United States joined the union; lighter states joined earlier, and darker states joined later. [Choropleth maps](#) are easy with vector tiles and CSS styling. And, unlike static images, you can [zoom in](#) for higher resolution. You can also experiment with different [color scales](#), such as [reds](#), [greens](#) and [blues](#).

The map background is a monochrome [image](#) layer from [CloudMade](#). Register a [developer account](#) with CloudMade for your own API key. We're hosting GeoJSON tiles for states and counties on [Google App Engine](#), and as with image tiles, you can always roll your own!

Source Code

```
var po = org.polymaps;  
  
// Compute noniles.  
var quantile = pv.Scale.quantile()  
  .quantiles(9)  
  .domain(pv.values(states))  
  .range(0, 8);  
  
// Data format
```

This example also uses a data visualization library called [Protovis](#) for computing quantiles and formatting dates. Protovis is optional and not required to use Polymaps.

Note that we're not using Protovis to render anything—just data processing.

Polymaps is a project from [SimpleGeo](#) and [Stamen](#).

Figure 1.5: *Demonstration of the Polymaps Statehood example, using vector tiles.*

The tiles are generated as individual GeoJSON files[51, 52], and since they are compliant with the GeoJSON standard they appear to not contain any special optimisations beyond what is allowed by the standard. Each tile has a number of self-contained features that would still be usable if every other tile was missing. The features do however have unique identification numbers which identify them even when they are split across several tiles, like with the GIS Cloud example. These ID numbers are checked against a table to retrieve information on the features. (In principle it could also have led to a query over the network for more information.)

Unlike the GIS Cloud example, the Polymaps example does not appear to concatenate the tiles and their features in any way. There are visible gaps between polygons, and there is no highlighting when the mouse is held over a feature. (See figure 1.6 for an illustration.)

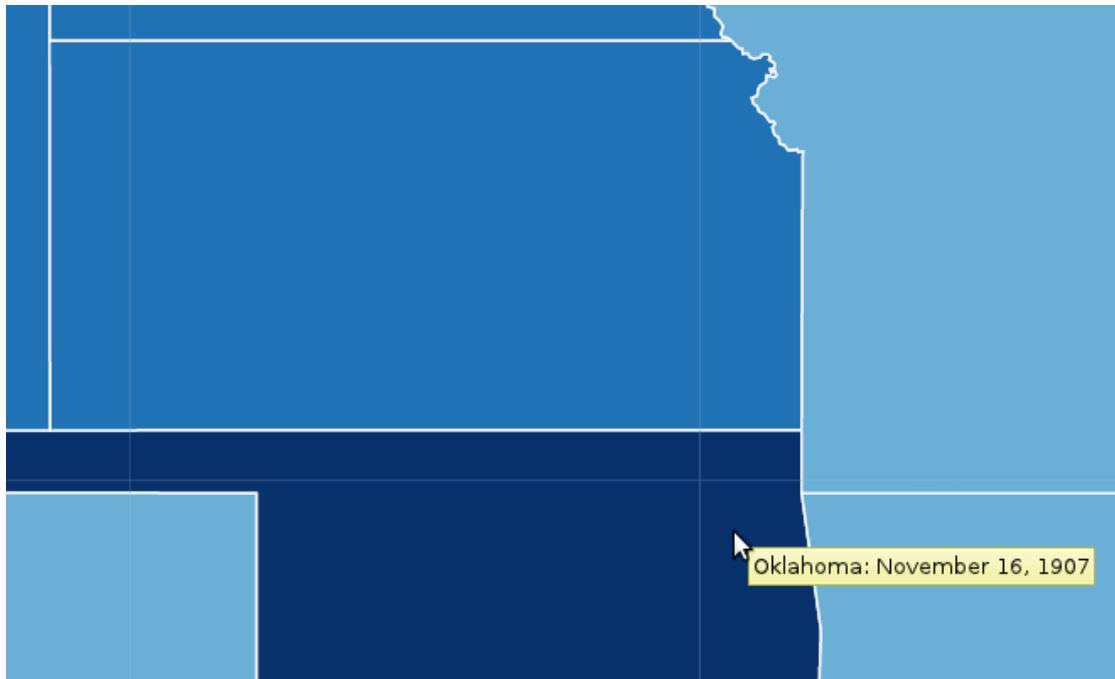


Figure 1.6: *Illustration of visible edges and lack of highlighting in Polymaps*

Both the client side and the server side of this example are fully open source. The back-end used is called TileStache, and is designed to be easily extensible to handle both new types of input and new types of output. Therefore there are great opportunities for learning from the set-up, and also for building a vector tile solution based on it. (It is possible to deploy an exact clone of the existing

example.) The inherent extensibility in TileStache makes it a good candidate for building upon.

1.5.3 Mapnik Metawriter

The Mapnik rendering engine is a popular open source tile renderer, which is used by the OpenStreetMap project. As a part of the Google Summer of Code 2010 (An initiative sponsored by Google that sponsors open source development[53]), a tile metadata renderer was implemented. This renderer outputs JSON tiles that contain coordinates and information regarding features as they are rendered on the raster tiles, and is intended as a complement to the raster tiles. Figure 1.7 shows a demonstration of how it is used to add extra information and interactivity. It could probably be successfully used for a pure vector tile implementation as well, if one does not deliver the raster tiles.

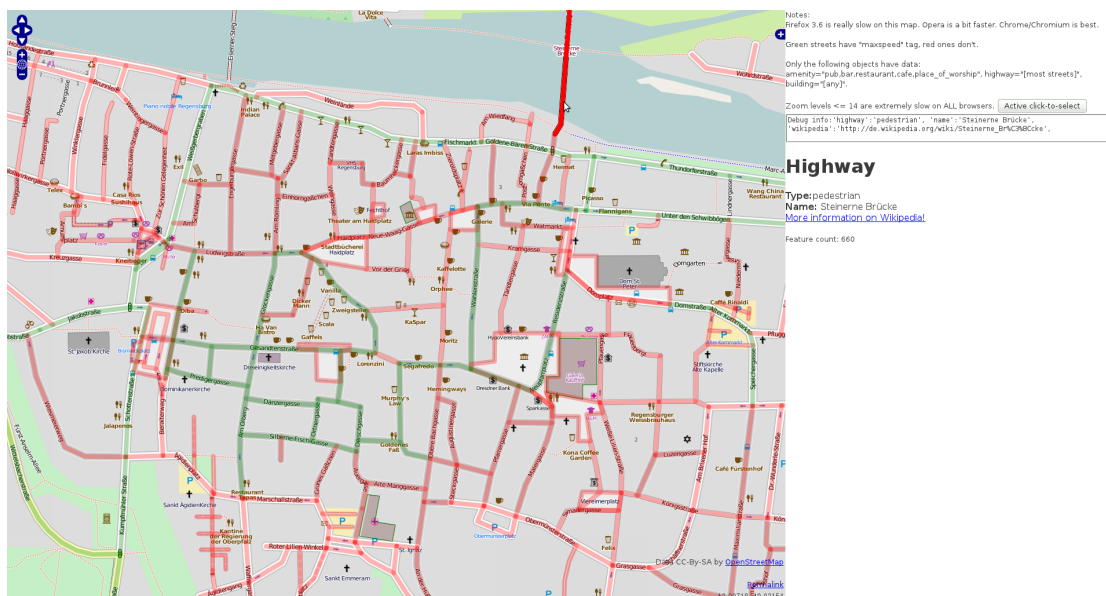


Figure 1.7: Demonstration of Mapnik Metawriter, showing a bridge section highlighted.

The client side is implemented as a plug-in for OpenLayers, where the vectors from the tiles are loaded in as ordinary vector objects in OpenLayers. This gets a bit slow and inefficient at lower zoom levels when more objects are on screen at once. There is no concatenation of feature segments, as each vector tile instead carries the entire feature that intersects it, not just the intersection. While being easier to handle (it simply checks whether the feature in question has already been

added), this would appear to create a fair bit of unnecessary data transfer.

Both the server side (Mapnik) and the client side (OpenLayers) are fully open source, making it possible to work with them. However, given that this implementation is tightly coupled to the raster visualisations provided by Mapnik and OpenStreetMap it might be wiser to build upon a vector tile solution that is more flexible, such as Tilestache.

1.5.4 Nokia Maps 3D

While not using vector map data per se, the Nokia Maps 3D WebGL client uses vector data to construct 3D models of buildings that aerial photography is draped over, providing a richly detailed model of major cities such as New York.[54] The client uses WebGL, a rendering context for the HTML 5 Canvas element for displaying 3D graphics in the browser, which is normally hardware accelerated for greater speed.[55] All of this data is delivered as tiles, with each tile having vector data detailing the model and a corresponding raster tile with image data to drape over the buildings.[56] The city models appear to have been pre-generated and then sliced into tiles according to a grid system, with the tile format being specifically engineered to correspond to OpenGL graphics vertex arrays. This allows for optimal rendering speed.

However, there is no preservation of topology in this approach. When a building straddles the divide between two tiles, each tile contains half a building with an open side facing the tile border, and no knowledge of what it is. For the display to look correct the other half must be rendered as well, and because the building halves line up it looks correct. (As can be seen in figure 1.8.)

The whole tool chain (both server side and client side) for Nokia Maps 3D is proprietary, so the amount of knowledge that can be gleaned is limited and there is no possibility of building a new vector tile solution on top of this technology.

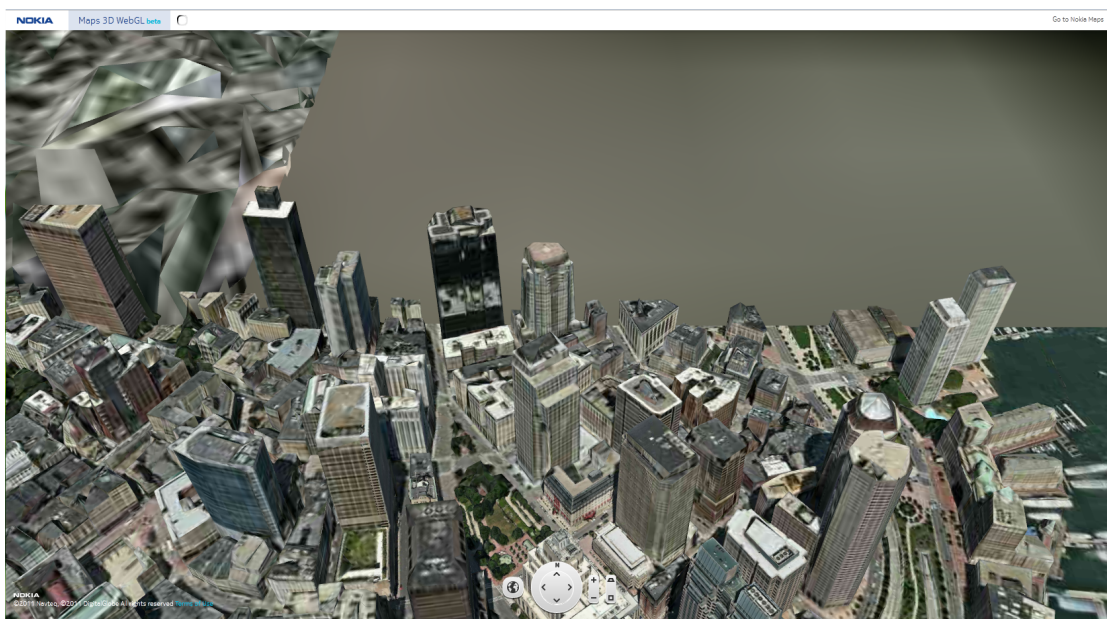


Figure 1.8: *Illustration of tile loading in Nokia Maps 3D. When the models are half loaded, the structure of the tiles becomes visible.*

2 Project Goals

2.1 Motivation

The motivation for this project is to explore aspects of vector tiling that have not been thoroughly examined before. It is the author's firm belief that vector tiling is a concept that is gathering a growing amount of enthusiasm in the geospatial industry, and that we will be seeing more and more of it in the years to come. Some products have already been deployed, and it seems reasonable to assume that others could be in development. Academically, there are still a lot of areas within vector tiling that can be explored in greater detail. Therefore there is a significant opportunity to experiment and codify knowledge on the subject that may be of use to both future research and practical applications.

Of particular interest to the author is how vector tiles can bridge the divide between the purely presentational function of raster tiles and the analytical opportunities of vector data. This includes looking at how topological information can be preserved, along with data completeness, data accuracy and similar issues.

2.2 Project Outline

In the course of this project the following goals are to be obtained:

Analyse current vector tile research & implementations Establish known developments in the area of vector tiles and look at what can be learned from them.

Theorise on ways to ensure topological consistency Examine possible strengths and weaknesses of different approaches.

Build prototypes & automated tests Implement the most promising methods for testing.

Perform tests & analyse results Collect data and work with it to obtain insights into whether theorised strengths and weaknesses were correct.

Draw conclusions regarding vector tile strengths, weaknesses & opportunities Based on the experiments performed as well as literature studied, make recommendations that can be of use to future vector tile projects.

2.3 Desired Outcome

The desired result of this project is to have gathered useful information about vector tile systems that can be referenced for future development of similar applications. A set of technology demonstrators shall have been produced, and been tested in a reproducible fashion. Specifically, the demonstrators shall implement various methods of preserving topological information for vector tiles. The experimental data that will have been generated will be used to make inferences about possible future applications.

3 Challenges with Vector Tile Generation

When considering the question of making vector based tiles, one must inevitably look to raster based tiles and draw on the experiences and research made in that area. Beyond that, one can also expect that certain traits of raster tile solutions will be transferred to vector tile solutions, creating a hybrid with characteristics from both raster tile and traditional vector solutions. On the other hand, some challenges are unique to vector tiles (especially those pertaining to preservation of topology), which will affect vector tile solutions in their own way.

3.1 Generalisation

In a raster solution the geographic data is rendered for each zoom level, with all the relevant generalisation and styling rules applied. On the other hand, in a classic vector solution all the data is transferred and the local client renderer does the job of generalisation. Arguments can be made both that this is an advantage for vector solutions because it guarantees that the accuracy of the rendered map is only limited by the accuracy of the data, and that it is a disadvantage because of the large data transmissions and processing time required. Imagine, for example, that one was to render an entire country using vector data. If one was zoomed out to see the entire country, extremely high resolution data is only an extra burden.

The vector tile generator in Tilestache (and therefore the prototype) does not pre-generalise, opting instead to transmit the full data set of each tile. For a low zoom level with a highly detailed data set, this can be problematic and slow down the transfer and display of tiles. In the tests executed, this problem has been avoided by limiting them to only use zoom levels that correspond with the detail level of the data set.

Previous research in progressive vector transmission[57, 58] has demonstrated the possibility of pre-generalising vector data, caching it and transferring more and more detailed vector sets as the user zooms in. From there it is not hard to imagine doing the same for each zoom level in a vector tile solution. In fact it would probably be easier as one has a very specific set of zoom levels to prepare, and one could use the same kind of generalisation procedures as one does for raster tiles. (One could check how the contents of the vector tile is rendered as a raster and drop any features that are rendered to be less than one pixel in size.) Of course, this is probably best done ahead of time and cached.

Another solution for pre-generalisation is to pre-generalise the entire data set

to several levels of resolution and store them in a multi-scale database. One can then source the tiles from the appropriate generalised data set depending on zoom level. This method has a lot in common with a common method of creating generalised data sets for raster data.[59]

3.2 Tile Sectioning & Overlap

Tilestache adopts the popular system for tile sectioning and addressing described in section 1.2 for vector operations mainly because it uses the same system for raster operations, and therefore the prototype uses it too.

One intuitive reason to adopt a similar addressing approach to that commonly used by raster tiles like Tilestache has done is compatibility. It would be natural to combine vector and raster based tiles for many types of applications, whether it is to add a simple layer of interactivity on a raster map, or to add map details to aerial imagery. Making sure that both types of tiles have the same addressing and equal tile sizes at each zoom level makes the job a lot easier.

A side effect of the chosen sectioning method, which is likely to be noticed for every vector tile solution, is that every vector tile generated has edges that overlap with the edge of the neighbouring tile. When you draw a line and declare that everything east of the line is in tile A while everything west is in tile B, the binary nature of rasters will ensure that that line will slice neatly in the infinitesimal gap between two pixels, as seen in figure 3.1. (The actual size of the gap depends on the resolution of the raster data.) In the meantime, when dividing up into vector tiles there are no neat gaps to be exploited in that way, and so the intersection of the bounding line and the geographical data must contain the bounding line itself. (As seen in figure 3.2.)

In itself this is not very dramatic, but it does raise some interesting questions when it comes to polygon clipping operations, which are further detailed in section 5.2.4.

3.3 Features With Multiple Geometries

When thinking about features being sliced apart by the intersections between tiles, one might generally imagine a large polygon being divided up among several tiles. Detecting if this has happened is simple enough, one only needs to check if a feature segment on a tile intersects with the edge of the tile. However, a much more difficult to detect situation arises when one employs features with multiple

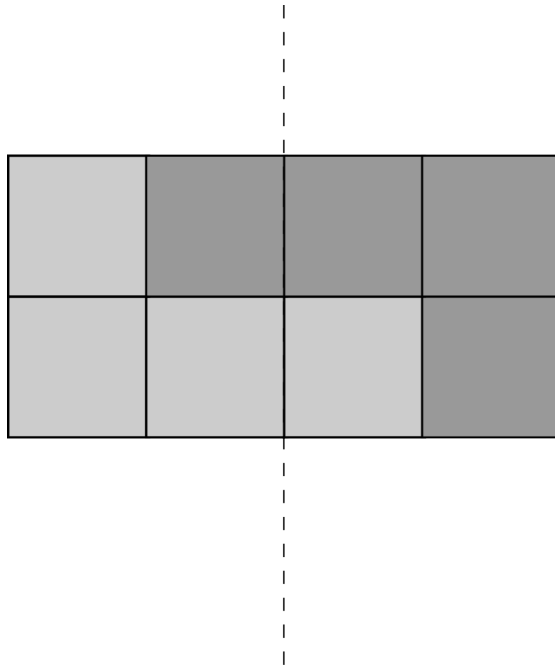


Figure 3.1: *Illustration of raster tile sectioning. Notice how the stapled edge neatly aligns with the gap between raster cells.*

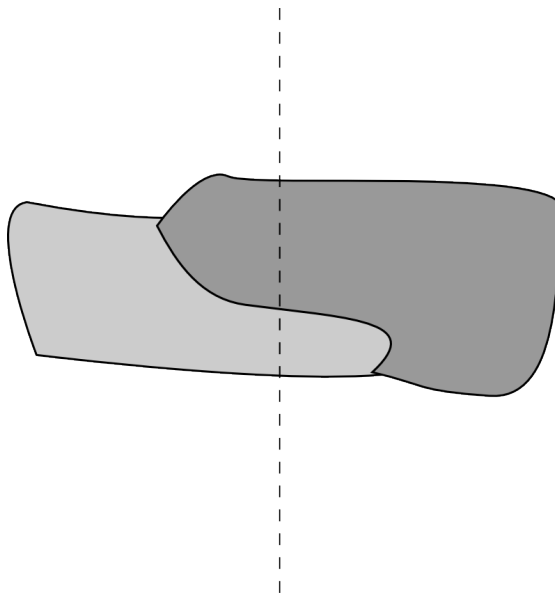


Figure 3.2: *Illustration of vector tile sectioning. Notice how the stapled edge intersects with data content.*

geometries. This leads to a need for more complicated algorithms.

As defined in the SFA standard, any feature can have multiple geometries represented in a GeometryCollection. These geometries can for example be MultiSurfaces/MultiPolygons, MultiCurves/MultiLineStrings or MultiPoints, which each are a collection of surfaces, curves and points that make up a feature. Using a world map as an example, many countries would be represented as MultiPolygons, for example the United Kingdom of Great Britain and Northern Ireland. This country consists of at least two distinct parts separated by sea no matter what level of generalisation is applied, in addition to numerous smaller islands that appear when the map is displayed with sufficient detail. (See figure 3.3.)

When the tiles are sized so that all of the polygons are in the same tile, there is no problem, and similarly if one or more the multiple polygons intersects with the border between tiles any algorithm developed for single polygons should work well with only small adjustments. However, in cases where the polygons are spread over multiple tiles without intersecting the boundaries (or even have multiple unrelated tiles between them), algorithms assuming single polygons will start to break down. One such example would be Portugal and the Azores Islands, where both are part of the same Multipolygon collection but can be rendered several tiles apart. (See figure 3.4.)

Addressing this issue will require more complicated algorithms and will also lead to a reduction in efficiency when compared to the more optimal situation of only having single polygons. However, it is necessary both to comply with the SFA specification and because a lot of real world cases require multiple geometries. As such, the algorithms in section 4 are considered for both the general case of multiple geometries and special cases where that requirement can be dropped.

3.4 Data Completeness

One particular challenge connected to using tiles for data transmission is ensuring that the complete features are assembled. For example, when one has loaded and concatenated all the feature segments of a particular feature that are available in the currently loaded tiles, one may still not have assembled the entire feature. (That is, parts of the feature are outside the currently visible map window and therefore the currently loaded tiles as well.) This is of significance if one wishes to do GIS operations such as calculating total area, creating buffer areas, and more.

In the context of the SFA standard, this is also problematic. The standard states[28]: “All Geometry classes described in this standard are defined so that

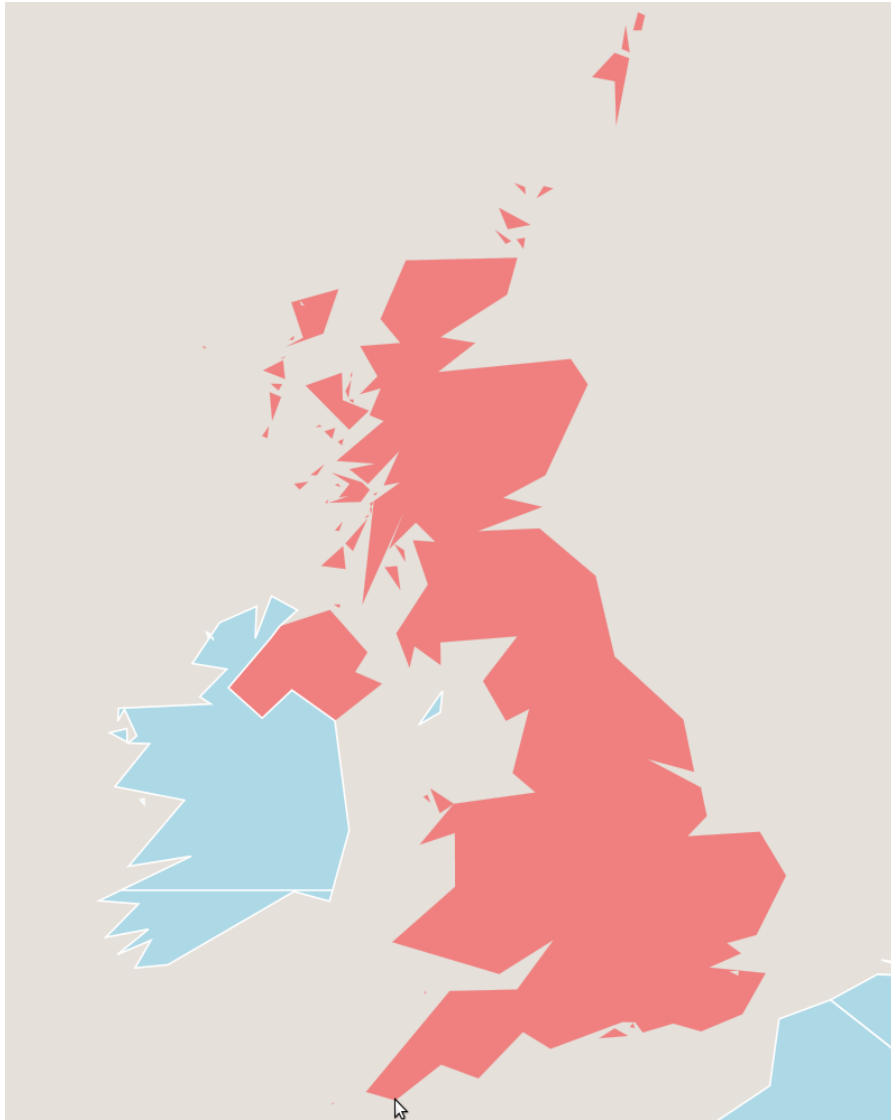


Figure 3.3: *Illustration of The United Kingdom, notice how it is composed of many different geometries.*

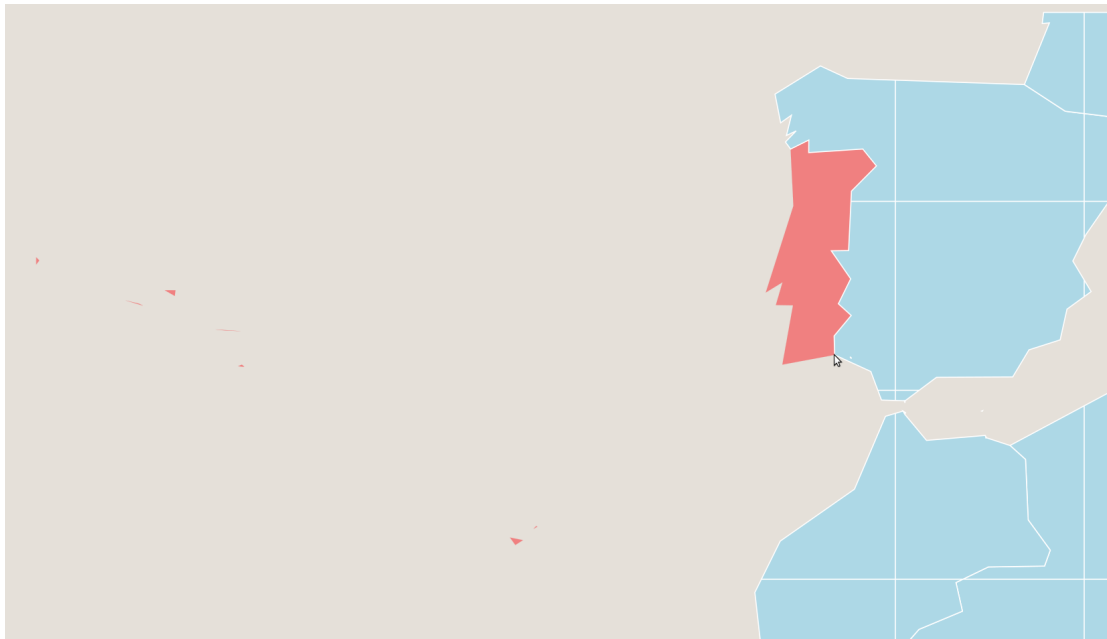


Figure 3.4: *Illustration of Portugal, notice how it is composed of many different geometries which are more than one tile apart. (See the tile structure outlined in neighbouring countries for comparison.)*

instances of Geometry are topologically closed, i.e. all represented geometries include their boundary as point sets”. This means that (as one might naturally assume) that a feature and the connected geometry must include all parts of the relevant geometry. So if one doesn’t have all the relevant tiles loaded and used for concatenating features, then the assembled feature will not be identical to the transmitted feature and will also violate the SFA standard.

However, it is debatable whether data completeness is a goal in and of itself. For GIS operations, it is probably vital to have the complete data set. On the other hand, if all that is needed is displaying of the data, with for example highlighting and name display of areas the user hovers the mouse over, then data completeness is not necessary for a good user experience. In fact, it might make the user experience worse by spending a lot of time pulling in tiles and stitching together feature segments that will not be seen by the user. For example, if the user was zoomed in to look at the border between Russia and Finland, the considerable time and computing power required to download and assemble the entire outline of Russia at that zoom level would be a complete waste.

A possible intermediate solution would be to implement a variant of “lazy

loading” where geometries are initially assembled only from current tiles, and are reassembled with information from additional tiles if the user tries to perform a GIS operation on the feature.

Some of the algorithms suggested in section 4 are only applicable in the special case where data completeness does not need to be guaranteed, whilst others can be used in both cases. For those which can be employed in both cases, a comparison will be made.

3.5 Unique Feature Identification

The algorithms described in section 4 require each original feature to have a unique identification assigned to it, as the easiest way for a computer to determine whether two segments belong together is if they have matching IDs. This ID could be a number, some sort of hash or any other uniquely identifying characteristic. (Possibly a combination of several characteristics that can be used to uniquely identify it.)

However, such IDs are not required to comply with the Simple Features Access standard (Part 1, Common Architecture)[28], while the SFA standard Part 2 (SQL Option) does require unique keys for every feature. Therefore, all compliant spatial databases will have unique IDs available[29], but not all data storage types will. For example, GeoJSON files are not required to have unique keys. (Although obviously they can have if desired.)

In other words, one cannot expect the data that is to be tiled to have unique IDs, although commonly it will have. As such, the tiler must have a mechanism for assigning unique IDs to features (through for example a numbering or hashing scheme) but should respect any existing unique IDs as they will likely have more semantic meaning. For reasons of efficiency, it would be preferable if the source data had been prepared by assigning sensible IDs before handing it to the tiling program.

3.6 Rendering & Concatenation Order

One issue related to the questions of multiple geometries and data completeness is when to start rendering the data in the browser. For traditional raster tile, this is not an issue: Because they have no topological considerations, each tile can be rendered as soon as it has been received. The same is true of simple vector

tile implementations that do not consider topology. But when you are stitching together feature segments from several different tiles, you need to wait until they are all received before the final feature can be assembled. In the meantime, users are getting impatient because they want to see results fast. Again, this depends somewhat on the use case. Users waiting to do GIS tasks might have more patience than users looking for a quick overview.[60]

There are several options for how one could handle the question for rendering time:

Wait for all tiles The simplest solution is simply to wait for all tiles to be loaded and then concatenate and render their contents. This option ensures the best data completeness but may cause the user long periods of waiting.

Progressive Concatenate and render features as soon as possible from the currently available tiles, and when new tiles arrive with more feature sections, re-render the feature. Will give the map the appearance of “growing” while the user watches, leading to less impatience. [61] However, this approach is somewhat inefficient, as features have to be concatenated and re-rendered several times as new tiles come in. This may again adversely affect the time it takes to reach a completed map.

Quick render & re-render As a kind of middle ground, it could be useful to render each tile as soon as it arrives without doing any concatenation, and when all tiles are loaded proceed with concatenating and re-rendering features. In this fashion, users quickly get to see something on the screen, which is then improved upon once all tiles are loaded. (Whilst avoiding the inefficiencies of the progressive approach.)

The order of concatenation and rendering may affect the performance of the algorithms discussed in section 4, but does not require changes to the actual algorithms.

4 Potential Data Structures & Algorithms

In the following section, a number of different potential data structures are presented with basic descriptions and algorithms for implementing them. Where possible, they will be considered for both the general case of features with multiple geometries and the special case where features only have continuous geometries. (For example, road networks or grid overlays.) Possible strengths and weaknesses such as ease of implementation, algorithmic efficiency and data completeness are considered and presented.

4.1 Global Feature Search

A simple and intuitive solution is to iterate over every tile, and every feature, and then searching every other tile for matching feature segments. They can be matched by comparing feature IDs (as mentioned in section 3.5) and creating unions of those features that have identical IDs. One could probably describe this as a naive solution, but being naive in the algorithmic sense of the term does not have to mean it has poor performance, simply that it is the most immediately obvious solution.

It would appear that [22] uses this method for finding concatenation targets, so it obviously works in practice. They did however not mention alternative methods, and used it on a three by three tile group, which is rather small. Therefore, one can not claim that [22] makes a thorough examination of the tile concatenation aspect and it is still valuable to explore this method and compare it with other possible methods.

4.1.1 Tile Specification

- Each feature must have a unique identifying ID. Typically, and this is also preferable, a unique ID will already exist in the geographic data being used as the source for tiling. If the source does not carry unique IDs, an ID must be assigned to each feature by the tile generation program.
- That ID must be provided to every section of the feature rendered on a tile.

4.1.2 Construction & Concatenation Algorithms

The construction algorithm is extremely simple, and is the same for both the special and general cases:

- If IDs are not pre-assigned: Generate a unique ID for each feature. (May be generated by combining several attributes to create a unique key or by assigning a number.)
- Include the ID in the feature representation on each tile created.

In the general case, the concatenation algorithm is quite simple:

Create a list F of completed features. (That is, features that have been completely reconstructed from their constituent tiles.)

```
for each available tile do
  for each feature segment on the tile do
    if the feature segment's ID is listed in  $F$  then Move to the next feature
    segment.
    else
      Check every other tile for matching segments and add them to a list.
      Create a union of the feature segments when all available tiles have
      been checked.
      Draw the feature and add it to  $F$ .
    end if
  end for
end for
```

Concatenating the tiles in the special case can be solved in several ways, for example the general case algorithm would work correctly in the special case. However, there may be more efficient ways of solving the problem in the special case. A algorithm which is presumably more efficient is presented here:

Create a list F of completed features. (That is, features that have been completely reconstructed from their constituent tiles.)

```
for each available tile do
  for each feature segment on the tile do
    if The feature segment's ID is listed in  $F$  then Move to the next feature
    segment.
    else
      Create a list  $S$  for feature segments to be assembled and add the first
      known segment.
      if The feature segment intersects any of the tile boundaries then
        Recursively check all tiles that share a boundary intersecting the
        feature, and add all feature segments with matching IDs to  $S$ .
      end if
    end if
  end for
end for
```


Assemble all the feature segments to a feature by creating a spatial union of the segments in S .

Draw the feature and add it to F .

end if

end for

end for

Both of these algorithms assume all the needed tiles having been completely downloaded and would therefore need to be rerun several times for new tiles in a progressive rendering style such as the one discussed in section 3.6.

4.1.3 Probable Advantages

Simple implementation This system is simple to implement on both the server and the client.

Little extra server work If IDs have been preassigned, there is no extra work for the server, otherwise only a minor amount of extra work.

Little extra data transmission Again, if IDs have been preassigned there is no extra data being transmitted and otherwise only very little extra.

Should perform well in the special case With the special case constraints on the data, the algorithm can be made to be quite efficient.

4.1.4 Probable Disadvantages

Can't guarantee data completeness This method does not have any information on whether there are more tiles with relevant feature sections that have not been loaded yet.

Inefficient in the general case Examining every feature on every tile many times over does not scale well.

4.2 Edge Pointers

A possible solution is to equip every produced feature segment with additional information on which other tiles the other feature segments reside. Then when a new feature segment is encountered, all of the segments can be pulled together and assembled straight away by following these pointers. However, including pointers to every single other tile which contains a feature segment could in some cases lead to the pointer information being many times larger than the actual feature

information. (For example, if you look at a large country at a high level of zoom there will be very many tiles to point to.) Therefore it is probably more sensible to include pointers to the nearest tiles with corresponding feature segments and to recursively follow the trail of pointers. That way it is also possible to stop following the trail if continuing requires downloading new tiles and that is not desirable, without the overhead of including pointers to every single tile.

4.2.1 Tile Specification

- Each feature must have a unique identifying ID. Typically, and this is also preferable, a unique ID will already exist in the geographic data being used as the source for tiling. If the source does not carry unique IDs, an ID must be assigned to each feature by the tile generation program.
- That ID must be provided to every section of the feature rendered on a tile.
- For each feature section there must also be four pointers to the nearest tiles in each direction (North/Up, East/Right, South/Down, West/Left)
 - The pointer consists of the tile address relative to the current tile address. For example: "N : -1, 1". (See section 3.2 for more on tile addressing.)
 - When the feature section is the outermost in some direction (there are no more tiles to point to in that direction), the pointer has a null value.

4.2.2 Construction & Concatenation Algorithms

In order to utilise edge pointers, an extra step must be added to the construction of each tile. This algorithm is a method of finding neighbouring tiles containing parts of the same feature, and is based on the concept of an equilateral cross with arms bent at right angles. In this cross, the arms are iteratively lengthened until the tile area at the end of an arm in a particular direction either strikes a part of the feature or exits the bounding box for the feature. See figure 4.1 for an illustration of the concept.

In practice, most searches will result in a hit or a bounding box exit on the very first tile checked, but the algorithm is equipped to handle any amount of distance between component geometries in a feature. For special cases with only single geometry features, the algorithm will always result in an immediate hit or exit without any particular performance penalties when compared to a naive check

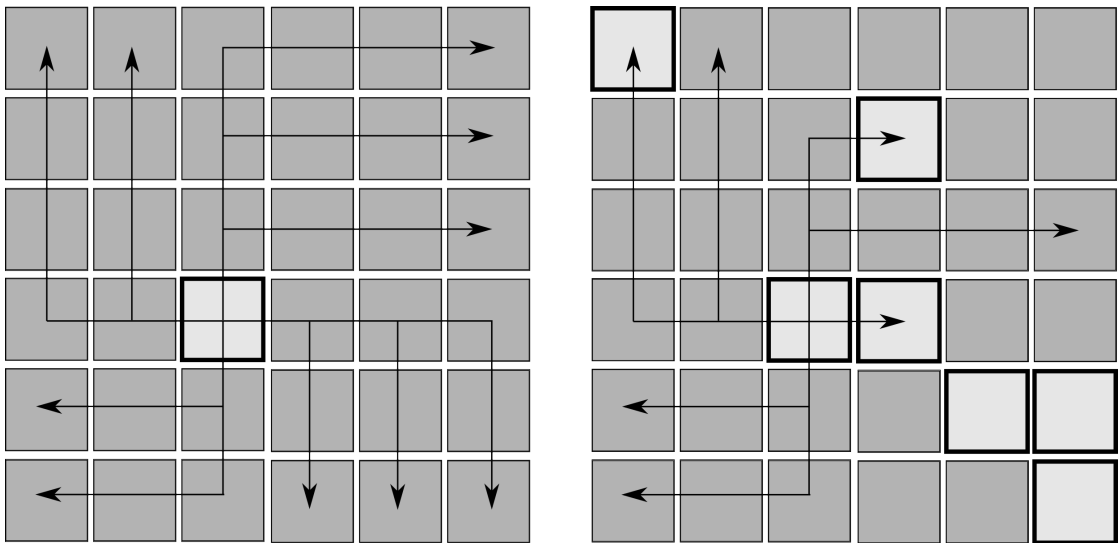


Figure 4.1: *Illustration of the edge pointer generation algorithm. The image on the left shows how the algorithm moves through the potential tiles within the bounding box of a feature. The image on the right gives an example of how the algorithm stops when it encounters a tile containing a feature section. Note how the SW pointer does not encounter any suitable tiles (resulting in a null pointer), and the fact that there are several more feature segment tiles within the bounding box that are not discovered when searching from this tile. (They are instead discovered when searching from the tiles that were discovered in this round of searches.)*

of the nearest neighbour tiles.

```
for each tile do
  for each feature do
    for each direction N,E,S,W do
      Create boolean variables continue_col and continue_row and set them
      to be true.
      create column and row counters X and Y, initialised to 0.
      while continue_col is true do
        Increment the column counter X and set continue_row to be true.
        Create a new search tile bounding box with a centre that is the
        original tile's position plus X and Y times a tile width.
        if the search tile's bounding box is within the feature bounding
        box then
          Set continue_col to false.
        end if
        if The search tile intersects the feature's geometry then
          Return X and Y
        end if
        while continue_row is true do
          Increment the row counter Y
          Create a new search tile bounding box with a centre that is
          the original tile's position plus X and Y times a tile width.
          if The search tile's bounding box is within the feature bound-
          ing box then
            Set continue_row to false.
          end if
          if The search tile intersects the feature's geometry then
            Return X and Y
          end if
        end while
      end while
    end for
    Return (no hits have been made).null
  end for
end for
```

The concatenation algorithm is predictably simple, using a recursive method that can simply be described as following the trail of edge pointers.

Create a list *F* of completed features. (That is, features that have been com-

pletely reconstructed from their constituent tiles.)
for each available tile **do**
 for each feature segment on the tile **do**
 if the feature segment's ID is listed in F **then** Move to the next feature segment.
 else
 if the edge pointers are *null* **then** Move to the next feature segment.
 end if
 Create a list S of feature segments
 for each edge pointer in the feature segment **do**
 if the edge pointers is *null* **then** Move to the next edge pointer.
 else if the linked segment is not in S **then**
 Add the linked segment to S .
 Check all of its edge pointers and follow them in the same way.
 end if
 end for
 Concatenate all the segments in S .
 Draw the feature and add it to F .
 end if
end for
end for

4.2.3 Probable Advantages

Can guarantee data completeness This method is able to guarantee data completeness by collecting every relevant tile, even if it requires that tile to be downloaded.

Equal efficiency in the general case and special cases The algorithm is identical for both cases and therefore gives the same efficiency.

4.2.4 Probable Disadvantages

More work on the server The server has to do more work to generate the tiles than with other solutions, although this can be negated through the use of caching.

More data to be transmitted Each feature has more data added to it, and this adds up when all the features are taken into account.

4.3 Central Feature Registry

Instead of providing the needed information to assemble the tiles with the tiles, as with the edge pointer concept, one could use a central registry that informs the client of every constituent tile belonging to a feature at every zoom level. For example, the client could, when encountering a new feature, look up the entry for that feature and learn which tiles this feature is divided among. It could then go directly to those tiles to collect the feature segments and create a union of them.

In this way one avoids the need for relative tile addressing algebra like with the edge pointers, but one must find a way to create and maintain the registry as well as allow access to it from the client. As transferring a large file with all this information in it before sending the actual tiles would rather defeat the purpose of a tile-based solution, it would probably be implemented as a database that the client application can make calls to for more information.

4.3.1 Tile Specification

The tile specification is simple and exactly the same as in the Global Search approach.

- Each feature must have a unique identifying ID. Typically, and this is also preferable, a unique ID will already exist in the geographic data being used as the source for tiling. If the source does not carry unique IDs, an ID must be assigned to each feature by the tile generation program.
- That ID must be provided to every section of the feature rendered on a tile.

4.3.2 Construction & Concatenation Algorithms

The algorithm to construct this system is simple but probably time consuming, however since it must be done in advance that disadvantage can be negated by caching.

```
Create a database of features and their tiles
for each tile do
  for each feature do
    if The feature exists in the database then Continue to the next feature.
    end if
    Get the bounding box for the feature
    Calculate every tile that intersects with he bounding box of the feature
```

Add the tile addresses to the database.
 end for
end for

The algorithm for concatenation is even simpler:

Create a list F of completed features. (That is, features that have been completely reconstructed from their constituent tiles.)

```
for each available tile do
  for each feature segment on the tile do
    if The feature segment's ID is listed in  $F$  then Move to the next feature
    segment.
    else
      Check the database for any other tiles containing segments
      Check all the listed tiles and create a list of segments
      Create a union of all the segments.
      Draw the feature and add it to  $F$ .
    end if
  end for
end for
```

4.3.3 Probable Advantages

Can guarantee data completeness This method is able to guarantee data completeness by collecting every relevant tile, even if it requires that tile to be downloaded.

Equal efficiency in the general case and special cases The algorithm is identical for both cases and therefore gives the same efficiency.

4.3.4 Probable Disadvantages

More work on the server The server has to do a fair amount of work to generate the feature database.

Mandatory caching of all tiles Because the file has to know the locations and tile allocation of every feature, every tile must be pre-rendered and cached before the application can go live. This means that the common practice of rendering only higher levels of the tile pyramid and rendering the lower levels on demand can not be used.

More communication overhead When the client needs to communicate with a server to look up every feature in the registry, there will be a lot of com-

munication overhead. (Especially when using HTTP requests.) This can probably be mitigated somewhat by using the WebSockets technology in HTML 5. [62]

Unnecessary in the special case In the special case this method can safely be considered overkill.

4.4 Probabilistic Matching

Previous methods have relied on some sort of authoritative source of information to determine the relations between feature segments. However, it is also possible to make a radical departure from the assumption that relations must be properly verified. Could it perhaps be enough for some use cases that the segments of feature probably fit together? In the special case, when there are no multiple geometries, it might well be possible to simply estimate whether to features in neighbouring tiles belong together.

By checking which tile edges intersect a feature and then checking which feature intersects the opposing tile edge in the same places, one can establish a likely match between feature segments.

4.4.1 Tile Specification

The tile specification is the simplest of them all, as no extra data is required at all.

4.4.2 Construction & Concatenation Algorithms

As the tile specification is so simple, it follows that there are no extra steps in the construction process.

The concatenation algorithm is more complex, requiring more calculation and estimation than any other alternatives.

Create a list F of completed features. (That is, features that have been completely reconstructed from their constituent tiles.)

for each available tile **do**

for each feature segment on the tile **do**

if The feature segment's ID is listed in F **then** Move to the next feature segment.


```
    else
      Create a list  $S$  for feature segments to be assembled and add the first
      known segment.
      if The feature segment intersects any of the tile boundaries then
        if The opposing tile edge is intersected in the same place then
          Create a separate list that contains tile addresses that have
          been visited
          Recursively check all tiles that share a boundary intersecting
          the feature to see if they have similar intersections and add segments & tile
          addresses to the lists
        end if
      end if
      Assemble all the feature segments to a feature by creating a spatial
      union of the segments in  $S$ .
      Draw the feature and add it to  $F$ .
    end if
  end for
end for
```

4.4.3 Probable Advantages

Requires no extra data transmission There is no extra data transmission. In fact, the data does not even require a unique ID.

Requires no extra work on the server The server does not have any extra tasks to complete when sending the data, making a suitable for a situation without caching.

4.4.4 Probable Disadvantages

Will only work in the special case Spatially continuous lines or polygons are required for this method to work, so features with multiple geometries are excluded. Many real world scenarios such as the geography of nations with major islands would not work. On the other hand, simpler scenarios may work well.

Can not guarantee data completeness or correctness Since knowledge of the features is limited to the already downloaded tiles, one can not know which tiles may contain further feature segments. Furthermore, since the matching is based on estimation and not checked against IDs, the matching might not even be correct.

Places a higher workload on the client With this method, all the work involved in matching feature segments is left to the client. It is probable that the calculations involved in performing the matching might be more resource intensive than those used on the client in the previous three methods.

4.5 Combined Approaches

It could also be possible to develop combined approaches where two or more of the proposed methods are combined in some fashion. The easiest way would be to implement for example the specialised global search algorithm and the edge pointer algorithm in one client, and set some sort of configuration signal that tells the client which algorithm to use for a given vector tile set.

Another method could be to provide ways for the client to detect what kind of data it was operating on. For example, one could combine the specialised and generalised global search algorithms by keeping track of not just which features had been assembled but also which tiles they were assembled from. That way, if the program came over a new tile with that feature that hadn't been used as a source earlier, it would know to start using the general algorithm for completeness.

5 Application Architecture

The prototype application, like most internet map applications, consists of two central components. One is the server side component which generates tiles and provides complementing services such as caching, while the other is the client side component which allows the client to reassemble the tiles and use them in a slippy map.[63] It too will typically provide complementing services such as compositing of several layers, annotations and so on. See figure 5.1 for an illustration of the application structure.

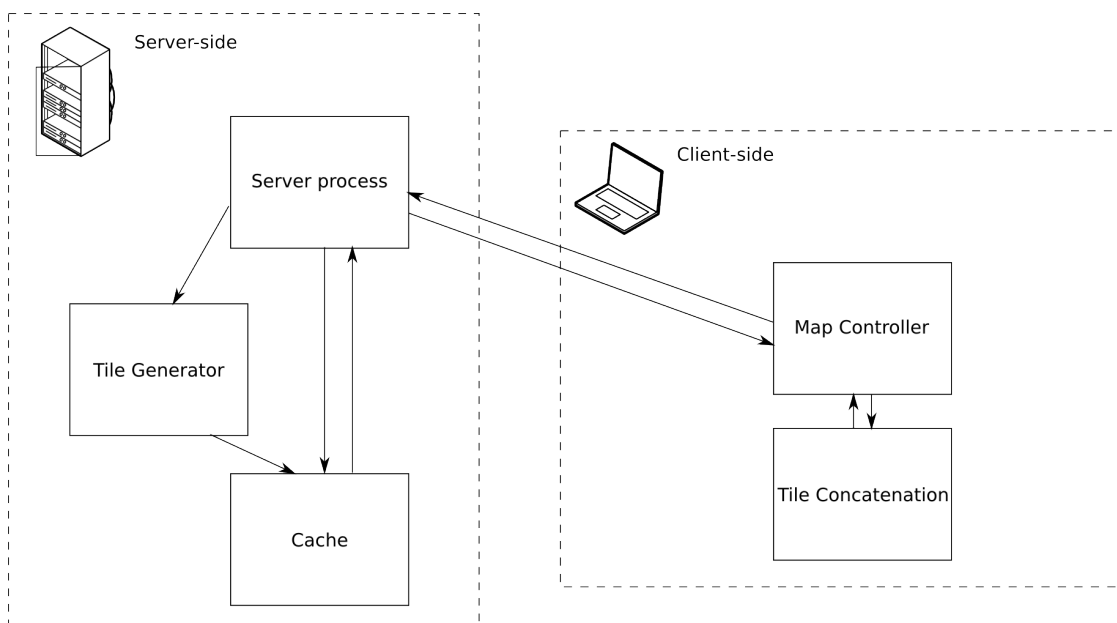


Figure 5.1: *Illustration of server-client interaction. The map controller makes tile requests to the server, which then either fetches the tile from the cache or orders the tile provider to produce the tile and cache it. The returned tiles are then processed through the tile concatenator and the results are placed on the map.*

For reasons of expedience, it was decided to build upon existing applications so that time could be spent researching and implementing methods of tile topology preservation rather than "reinventing the wheel" by spending time implementing all the basic aspects of tile servers and slippy map applications. Based on the research described in section 1.4 the most suitable candidates were found and used.

Polymaps and Tilestache (see section 1.5.2) were chosen as the basis for the prototype, primarily for three reasons: Firstly, together they make a well functioning application for simple vector tiles without topological consistency, and

secondly they are both open source, making it easy to work with and alter them where necessary. Thirdly, Tilestache was seen as easier to extend and less tightly coupled to a specific data source. They were enhanced for the purposes of this project both by pairing them with new components and by making small changes to them, which are detailed in the next sections.

5.1 Server Side Application

Tilestache is a relatively new tile server application, intended to create a modern alternative to the popular web map caching application Tilecache. (The naming is a play on words referring to Tilecache.) [64, 65] It provides both tile generation and caching in one, directly from the relevant data source by employing various libraries to handle data access. It does not provide any other services like WMS or WFS.

It is written in the Python programming language, so once all the dependencies are installed it should run on any major operating system. [64] It is licensed under the permissive 3-clause BSD licence, which allows modification, use and redistribution for any purpose. [64, 66] As such, it is well suited to be adapted for academic purposes because it can be both modified and freely used without any further restrictions. [67]

During the course of the work conducted extending Tilestache, the author of this report discovered three bugs, all of which have been reported to the upstream project. One of them was reported along with a patch fixing the bug which has since been accepted and incorporated into the upstream Tilestache source code. [68]

5.1.1 Extensions

Tilestache is built to be extendable by simply creating new components, and then telling Tilestache to use that new component by referencing it in the configuration file. Since Python is not a compiled language it is sufficient to write a text file with Python code that implements all the methods that are expected of the kind of component being replaced. This is so called "duck typing", named after the old adage that if something swims like a duck, and quacks like a duck, it probably is a duck. [52, 69] In this context, if a component implements the methods expected of a duck component, it will be treated as a duck component just like if it was part of the core program. (Regardless of whether it actually is a duck or a cleverly disguised dragon making the appropriate noises.) In that fashion the program does not have to be recompiled, and one can avoid making any changes

to the program itself when trying out new features that may potentially be broken.

In this case the component being added is a new tile provider that calculates edge pointers for each feature on a tile, and appends them to the relevant feature data being transmitted for the tile. It is essentially an extended version of Tilestache’s existing vector tile provider, in other words all the code from there was copied and then changed to fit the new purpose. (See appendices C and D for more about which files have been modified.) The implemented algorithm is described in section 4.2.

5.1.2 Tile Transfer Format

The file format used for transferring tiles from the standard vector tile provider in Tilestache is GeoJSON, a simple format that is natively understood by Polymaps and many other pieces of software.[51, 43] As such it was logical to use it for the tiles generated by the edge pointer extension as well. The following is an example of how a typical GeoJSON tile might look when enhanced with edge pointer information for each feature. This tile represents a small part of the north-western-most tip of Iceland.

```
{
  "type": "FeatureCollection",
  "features": [{
    "geometry": {
      "type": "Polygon",
      "coordinates": [[
        [
          -16.549679877363268,
          66.51326044169886
        ],
        [
          -16.527779,
          66.64828036827288
        ],
        [
          -16.100330428882376,
          66.51326044169886
        ],
        [
          -16.17187499845066,
```

```
        66.51326044169886
        ],
        [
            -16.549679877363268,
            66.51326044169886
        ]
    ]
},
"type": "Feature",
"properties": {
    "SUBREGION": 154,
    "NAME": "Iceland",
    "AREA": 10025,
    "REGION": 150,
    "LON": -18.480000,
    "ISO3": "ISL",
    "ISO2": "IS",
    "FIPS": "IC",
    "UN": 352,
    "LAT": 64.764000,
    "POP2005": 295732
},
"edgepointer": [
    [null, null],
    [1, -1],
    [0, -1],
    [null, null]
]
}]
}
```

However, it's worth noting that while GeoJSON is a simple and human-readable format, it is not necessarily the only possible way of encoding vector tile information. Since this thesis work has been concerned with exploring relative advantages and disadvantages of different approaches, absolute speed has not been a concern and therefore possible optimisations or more compressed formats have not been explored. For more information on creating vector tile solutions optimised for greater speed, please see [70], a master thesis project conducted by another NTNU student at the same time as this one.

5.1.3 Cache

Tilestache includes a variety of different cache options that can be configured through the settings file. (And of course, new cache types can be implemented if desirable.) For the purposes of the tests performed here the cache used is the most simple cache, the hard disk cache. (Cache data is stored in a plain folder structure on the disk.) Considering that the test scenarios were executed with only one user and the fact that the test computer used a high performance Solid State Disk (SSD), this cache can be considered to be plenty fast enough. (See appendix F for more information on the test computer.)

5.2 Client Side Application

As described in section 1.4.2, Polymaps is a relatively new JavaScript framework for web mapping applications, making use of SVG and HTML 5 elements. One of its core creators and maintainers is also the creator and maintainer of Tilestache, so naturally there is good compatibility between the two. It is also licensed under the same BSD licence, which brings with it the same advantages for academic purposes.[71]

Unlike Tilestache, Polymaps is not a stand-alone application but a library designed to be used within a web browser environment. As such, even when you want to use it completely without any additions you would still at the very least need to create a HTML page that invokes the library to create a map window. For the tests described in this report, a number of different HTML pages were created that invoke the library with different input, as well as the modifications and additions described in the next section.

It is important to note that this client side application is merely a prototype, and is built upon a library framework that was not originally intended to handle features spanning multiple tiles. Therefore there can be some visual errors, especially when panning and zooming the map because the underlying code operates purely on a tile-by-tile basis and does not recognise when a feature has been extended to span several tiles. (When the first tile of the feature is moved off-screen, the whole feature may be removed as is seen in figure 5.2.) This error is acceptable in the prototype context as it is purely visual and does not affect the mechanics of finding matching segments and concatenating them, which is the focus of this research. In any future implementations of a concatenating vector tile system, however, the framework should be designed from the ground up in order to avoid this issue as well as the issues described in section 5.2.4.

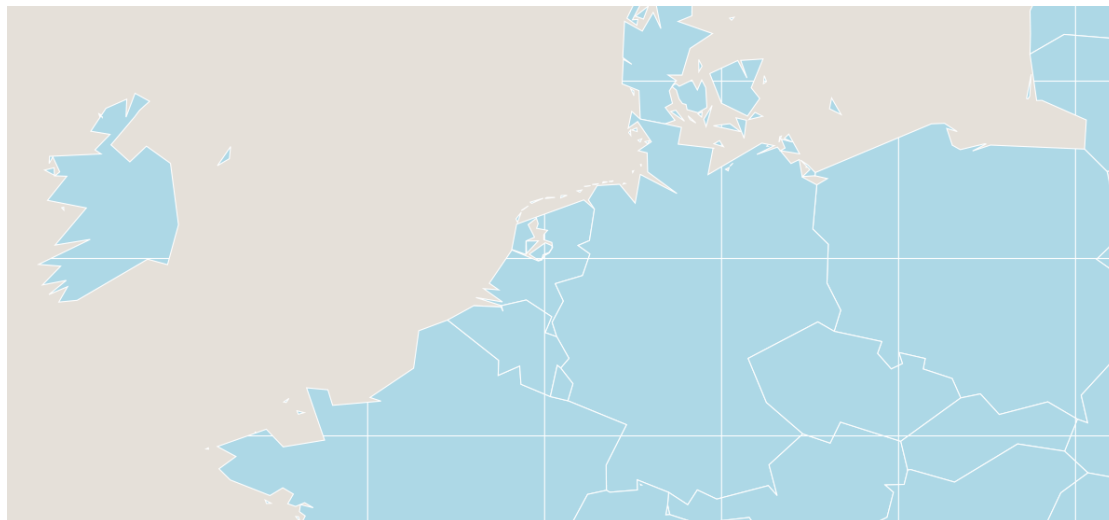


Figure 5.2: *Illustration of tile removal errors. In this example the UK, Sweden and Kaliningrad (Russia) have been removed from the map because the first tile they were concatenated from has been moved out of the map view. (The view was originally centred further north.)*

5.2.1 Modifications & Supporting Code

Unlike Tilestache, Polymaps does not have any mechanism for easily inserting extensions and new features. Therefore it was necessary to make some modifications to the core code, although they were kept as small as possible. Mainly all the new code and functionality for tile concatenations was written in separate JavaScript files (two files in common for all the tests, and one file for each test) which then communicated with the Polymaps code through exposed interfaces, some of which were created by the changes to Polymaps. See appendix E for more information on which files have been modified.

The changes to the Polymaps code were designed to expose information to the outside code that otherwise would have been hidden. Previously the Polymaps code only exposed an event that was triggered when a tile completes loading, but with the changes it also exposes events for when tiles are requested for download and for the eventuality that a download request is cancelled. The supporting code uses these to keep a tally of how many tiles have been requested and how many tiles have been delivered. When the two numbers are equal, it calls to the relevant concatenation algorithm for the test. The concatenation function then uses and modifies the SVG information created by Polymaps to sew the tiles together.

Another modification to Polymaps is the addition of row and column numbers

to every SVG tile that is generated. This is so that tile concatenation algorithms like the edge pointer algorithm can work by calculating the tile and row numbers of the next tile to add feature segments from, and then look it up.

5.2.2 Rendering & Concatenation Order

As explored in section 3.6, there are several ways the order of rendering and concatenating tiles can be ordered. In the prototype, the "quick render & re-render" method has been chosen. This is partly because it has good performance, and partly because it is simple to implement on top of the rendering methods used by Polymaps. Finally, it also gives the user visual feedback on which tiles and features are changed by the concatenation.

5.2.3 Integrated Web Server

By default, Tilestache comes with an integrated Web Server Gateway Interface (WSGI) web server named Werkzeug[64] that handles tile requests from clients. WSGI is a common Python interface for allowing programs to handle requests from web servers, where the WSGI server passes requests through to the relevant program. Since Werkzeug also can handle simple file serving, the configuration has been modified for this project. It has been set up so that all requests starting with a specified folder name are diverted and handled as simple file requests, while the rest are passed through to Tilestache. In this fashion one can serve both the test web pages using Tilestache and the tiles from Tilestache from the same server, making the prototype easier to set up and maintain.

For any evolution from prototype to a full production-ready application, it might be advantageous to split these areas of responsibility and deploy separate, full-featured web application servers.

5.2.4 Issues With Polygon Union

When one has gathered together all of the constituent feature segments that make up a feature, a new polygon needs to be created as the union of all the segments. This new polygon will then replace all of the segments. However, as it turns out, there does not appear to be any JavaScript libraries dealing with the issue of polygon clipping, of which polygon union is a subcategory.

The authors of [22] give a rough outline their method for creating polygon unions, but do not provide any concrete algorithm examples or implementation

code. Therefore it could only be used as a guideline in the author's attempts to explore polygon union, rather than provide a ready-made solution.

Polygon clipping is an area that has seen a fair amount of research and has a number of well-known algorithms. [72, 73, 74] However, these algorithms can be quite complex to implement and therefore most users tend to make use of one of a number of popular libraries implementing one of these algorithms.[75] None of these implementations, as far as the author of this report has been able to ascertain, are made in JavaScript.

The State of Polygon Clipping in the Browser Attempts have been made by the author of this report to use automatic source code translation of two implementations of well-known algorithms ([76, 77]) to JavaScript using suitable tooling[78, 79], but as they use constructs and value types that are not immediately translatable to JavaScript, these attempts failed. Therefore a robust polygon clipping library will have to be implemented from scratch in JavaScript.

The reasons why nobody yet has implemented this functionality in JavaScript can only be speculated on, but might be as simple as that no one has had a large enough need for polygon clipping inside the browser to make it worth the rather considerable effort required to implement it. Another reason may be that one is afraid the required calculations would be too expensive to perform in a web browser environment, or a combination of both. For example, the ESRI ArcGIS API for Microsoft Silverlight described in section 1.4 allows for polygon clipping operations by submitting them to the ArcGIS Server application as a REST call to be calculated using existing ArcGIS functionality, and then receiving the results. But considering the expected continuing increase in processing power in computers[80] as well as the continual improvement of JavaScript Virtual Machines in web browsers[81], it would be natural to assume that the question of expensive calculations could resolve itself.

Another possible reason for the lack of JavaScript implementations of union operations might ironically enough be that the need for them has been identified. The proposal for the next version of the SVG standard (version 2) calls for union and intersection operations to be available as standard SVG features[82], meaning that they would have to be implemented by the browser makers. Being implemented in native code by the browser makers would probably result in better execution speed than JavaScript, so between that and the aforementioned large amounts of effort to implement polygon clipping it might well be more tempting to

wait for the next SVG standard to be implemented. On the other hand, the SVG2 recommendation is not expected to be finalised before August 2013[83], so there is a considerable intermediate period where implementing a JavaScript version would be useful.

Degenerate Cases and How They Relate to Vector Tiles Another complicating issue that is unique to the vector tile question, is the so called degenerate case. A degenerate case occurs when the vertex of one polygon lies exactly on the edge of another polygon, or when two vertexes lie on another edge such that the edges from two polygons exactly overlap. (As seen in figure 5.3.) Since all our vector tiles have an overlap exactly on the edge between tiles as a result of the way they've been divided up, every single case is a so called degenerate case. Far from being an exception, it is the rule.

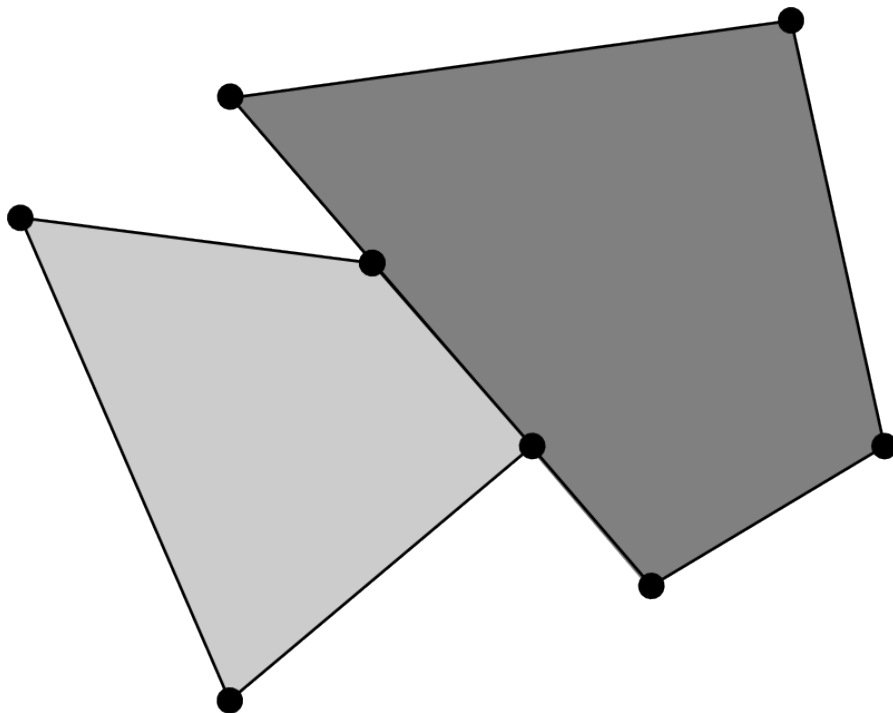


Figure 5.3: *Illustration of degenerate vertex edges. Note how the two polygons have overlapping edges.*

The problem is that many of the simpler and more well known polygon clipping algorithms treat degenerate cases as rare exceptions, which are treated in non-deterministic ways. (Typically by moving the vertex a marginal distance in

some direction.)[73, 74] Care must therefore be taken to implement an algorithm which handles degenerate edges in a deterministic fashion, such as [75] or [84].

Pseudo-Union Solutions In summary, implementing a general polygon clipping solution in JavaScript that suits the needs of this project is a fairly large challenge which probably could be the subject of an academic examination in itself. Therefore, due to the time constraints placed on this project implementing a true union operation is not feasible. Instead, one has turned to "pseudo-union" solutions which are described in the following section. Three possible solutions were considered, of which one was selected.

One quick approach was to appropriate a piece of open source code designed to click together pieces of a jigsaw puzzle[85], and adopt it to work with the vector tiles. It met with moderate success, where a lot of tiles were properly clipped, but unfortunately there were a lot of artefacts and small changes to polygon outlines as seen in figure 5.4. Because these problems were an inherent result of the different nature of jigsaw puzzles and geographic data, further development on this approach was abandoned. However, it was not a total waste as this method creates approximate results and spends a presumably similar amount of time on processing as a real solution, and is therefore of use for testing purposes.

The second approach attempted was to take lessons from the general polygon clipping algorithms of [75] and [74] along with the approach outlined by [22], and implement a simplified, non-general polygon union algorithm by making use of the known conditions in the tile solution. By limiting the algorithm to only have to deal with tile edge cases, one might be able to make do with less complex processing. Again, this met with only partial success. There were some correct unions, but also a lot of artefacts and problems that occurred whenever geographic entities had overlapping edges that were parallel or near parallel with a tile edge. (Using a set of polygons for every country in the world to test with, this became especially obvious with international borders that had been drawn along geographic lines of constant latitude or longitude.) It is quite possible that this algorithm eventually could be developed into a functional solution, and when implementing a vector tile application it would probably be wise to consider if a specialised algorithm could grant performance benefits compared to a general algorithm. However, within the time frame of the project, it did not seem viable to make a working solution with this approach so it was abandoned.

The final approach, which has been implemented in the prototype, is the sim-

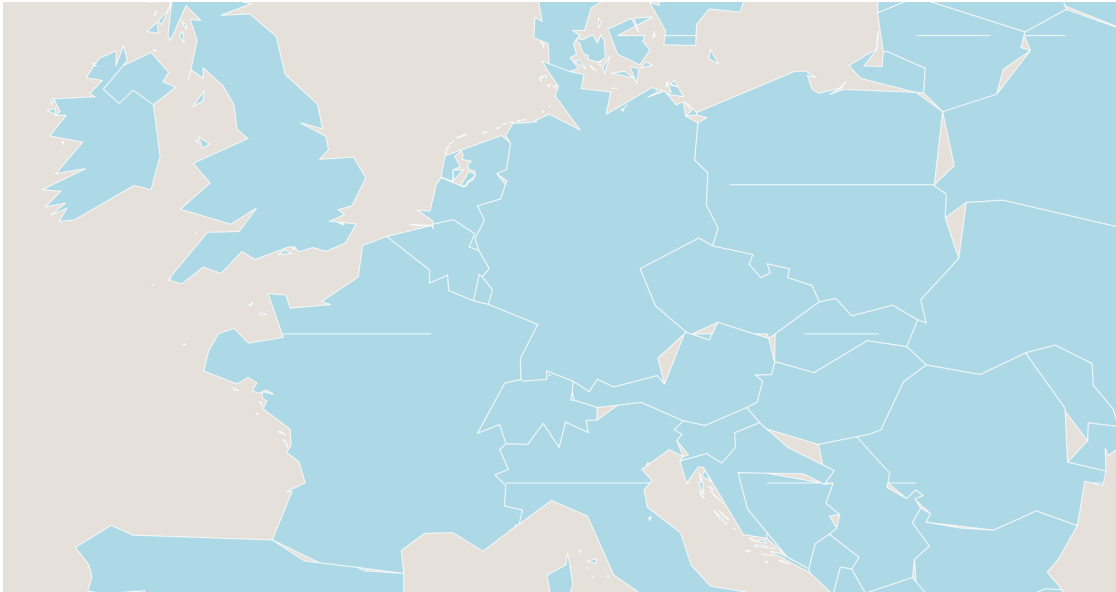


Figure 5.4: *Illustration of errors in experimental union. Some tile edges have not been properly removed, while a lot of small changes to polygons have destroyed the geographic integrity of the data.*

plest solution of them all. In SVG, geometries are defined by text-based drawing path instructions in which a sequence of control characters and coordinates describe how the theoretical "brush" painting on the screen should move and draw.[86] Since these instructions can include several independent polygons, joining together the paths of two different SVG objects is as simple as adding the drawing instructions from them both together. The major disadvantage is that this is not a true union, and the tile edge lines are still drawn. But if one sets the polygon fill and edge colours to be the same, they will visually appear as one single feature. (As seen in figure 5.5.)

While it is far from ideal to use this kind of pseudo-union, the union operation itself is merely the result of the concatenation algorithms which are the focus of this report doing their job. Therefore working to achieve a proper union was considered less of a priority, and this solution is good enough to visually verify the results of the concatenation algorithms.

5.2.5 Automatic Tester

In order to perform the speed tests discussed in section 6, a simple automatic testing system was implemented. The pages that require testing make a call to the

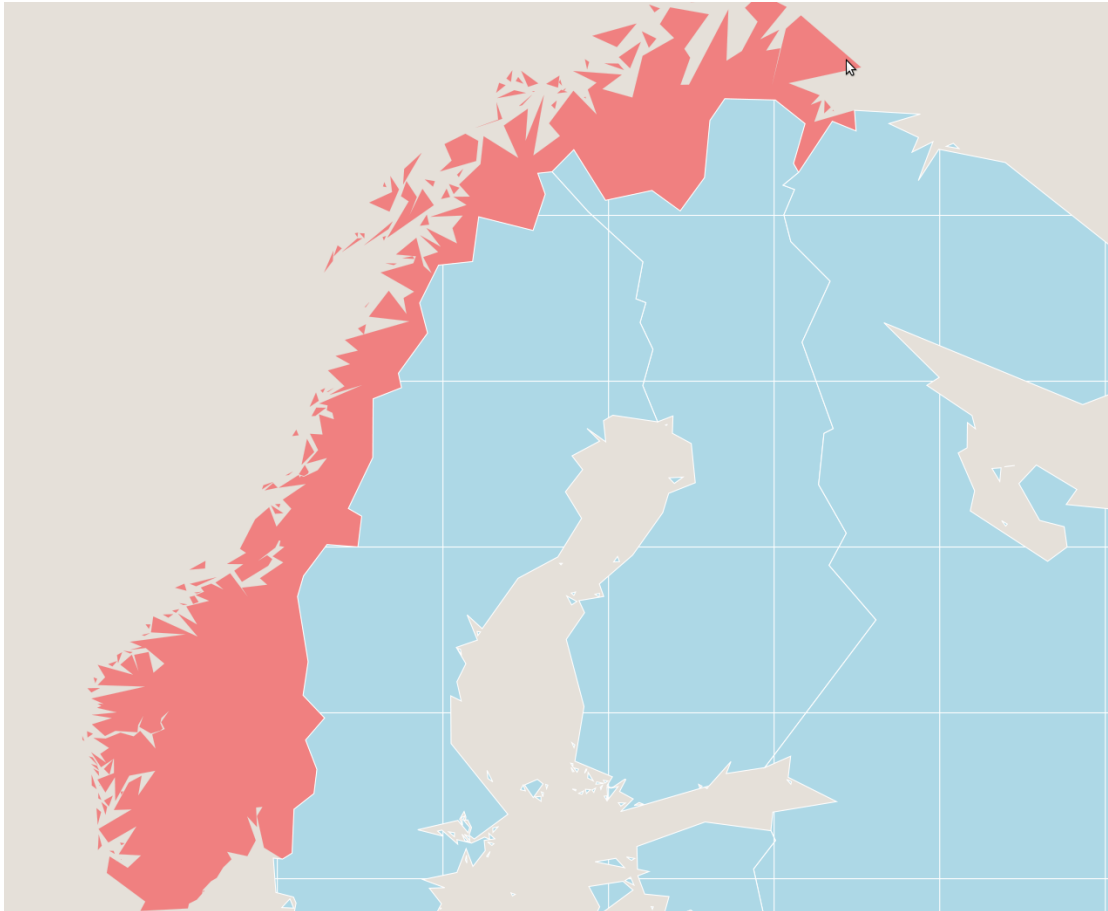


Figure 5.5: *Illustration of pseudo-union in practice. When outlines and fills are different colours, tile edges are visible, as with Sweden. When outlines and fills are the same colour, there are no visible tile edges within the feature, as with Norway.*

testing set-up function, which launches an alert window asking the user to enter the number of iterations to run. It will then, for as many iterations as required, select a random location and zoom level on the map to move to, measure the time it takes to load the tiles and then the time it takes to concatenate the tiles. After all the iterations are completed, it will present the timing results as a plain text report in the browser that the user can save to disk.

The testing set-up function takes as input a bounding box for which area the random movements are to be performed in, the zoom levels to move between, and the total number of iterations. It actually performs one more iteration than the user requests, because the first set of movements is not recorded due to it potentially being comprised by the extra processing required when setting up the client.

5.2.6 Demonstration Applications

In addition to the principle demonstrations and the testing applications, two demonstrations of vector tile possibilities have been designed. One of them shows the possibilities of using vector tile layers on top of traditional raster tile layers as a means of conveying additional information. In this case by creating highlightable polygons with extra information over roads and buildings, as seen in figure 5.6. The other displays a purely vector tile based map, showing how raster tiles can be done away with entirely. (As seen in figure 5.7.)

The data set used for this has been fetched from OpenStreetMap. The user may notice that roads being highlighted don't seem to cover the entire length of the road, but more typically a piece of road extending to the next intersection. This is a result of the underlying data structure in OpenStreetMap, as can be seen by observing the differing OSM id numbers on each section.

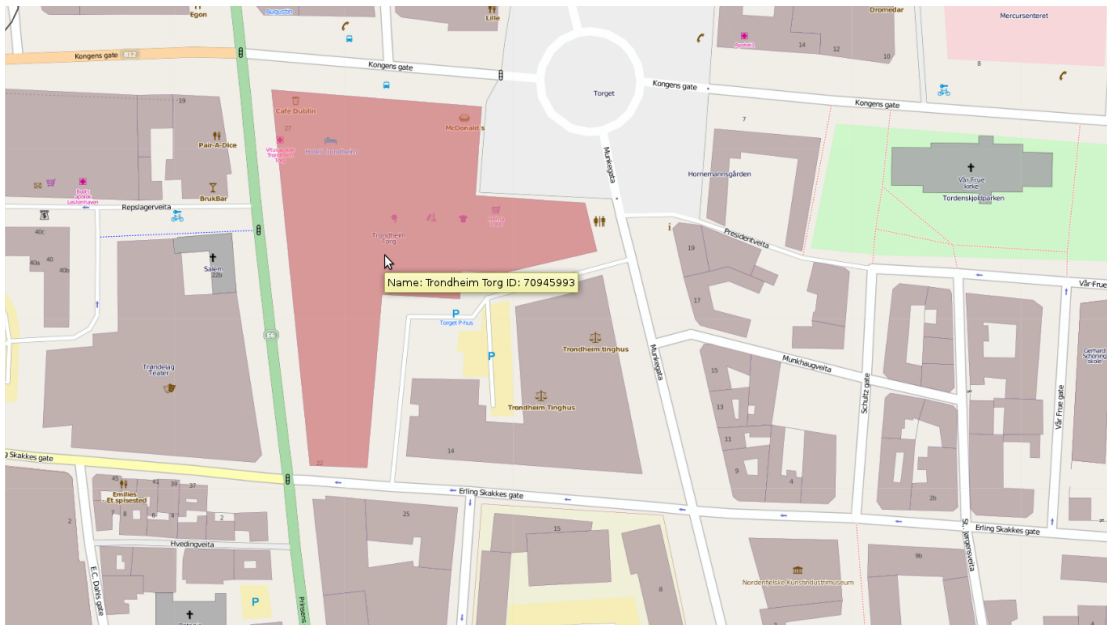


Figure 5.6: Illustration of a combined raster & vector application. Seen here is an area from the centre of town in Trondheim.



Figure 5.7: Illustration of a pure vector tile application. Seen here is an area from the centre of town in Trondheim, with roads, buildings and railway lines drawn on the map.

6 Experimental Design

The objective of the experimentation is to develop a set of timing data that can be used to compare relative characteristics of the proposed solutions against each other. This data will be useful when considering whether a priori assumptions about the strengths and weaknesses of the solutions were correct or not.

It is important to stress that the results of the experiment are strictly relative, and as such can not be used for an absolute determination of execution speeds. Absolute speeds depend on such factors as the data being used, the machine the tests are being run on, and the maturity of the code being run.[87] Since the code being run is only a prototype with emphasis placed on ease of implementation rather than speed and optimisations, there is a definite room for improvement in absolute speed. Furthermore, the author cannot guarantee that the implementation of the algorithms is completely optimal or error free. This will also affect absolute speed, but assuming that inadequacies are evenly distributed the relative speed will still give valuable indications.

6.1 Data Structures & Algorithms Selected for Testing

The following solutions have been selected for testing:

- Global Search, general case
- Global Search, special case
- Edge Pointers

These solutions were selected based on the author's judgement of them likely being the best solutions, with regards to the weaknesses and strengths discussed in section 4. The selected solutions will also give an indication of whether choosing one solution for all situations or customising based on situation is wisest.

The Central Feature Registry concept was decided against because the complex implementation coupled with the large amounts of extra requests meant that it, in the author's opinion, probably was a less efficient solution than the ones selected.

The Probabilistic Matching solution was left out because of the experiences earned from creating a prototype implementation of it. It follows from the algorithm that the concatenation has at least as many steps as the special case Global

Search algorithm, and the same constraints. Using a probabilistic determination instead of a simple ID check essentially guarantees that the concatenation is less efficient. Coupled with the fact that the extra communication overhead from transmitting a feature ID is small and that developers quite often will have use of it anyway (such as for lazy loading or exposing feature information to the user), implementing Probabilistic Matching was not worth the effort. The implementation was therefore abandoned before it was completed.

6.2 Test Cases

Two different data sets were chosen for test runs in order to investigate performance in both the general case of multiple geometries per feature and the special case of single geometry features. Both are included in the attached files for this report, see appendix C for more details.

6.2.1 General Case - Multiple Geometries

As all the examples in section 3.3 demonstrating the challenges of multiple geometries use the geography of nations, it is natural to use a data set with countries for testing multiple geometries. The selected data source is a shapefile of the world in the typical web map pseudo-Mercator projection, showing the whole world except the poles. (As seen in figure 6.1.) It is included in Tilestache as the data source for a demonstration of the raster tile layer provider, so it was simple to re-purpose it.

The map data has a low resolution which means that the map is best viewed at relatively low zoom levels. This is perfectly suitable for the purpose of visualising nations, and causes no particular problems for multi-geometry concatenation trials.

6.2.2 Special Case - Single Geometries

For the single geometry case a source file had to be obtained. While roads have been used as an example of single-geometry data, the choice fell on building outlines because they are polygons, which raises more interesting challenges. The data used is taken from a shapefile extract of OpenStreetMap building outline data for Norway[88], which was then edited in Quantum GIS to restrict the data to a specific region with a high density of buildings. The area selected is the historic centre of Trondheim, which has a high density of buildings as well as a

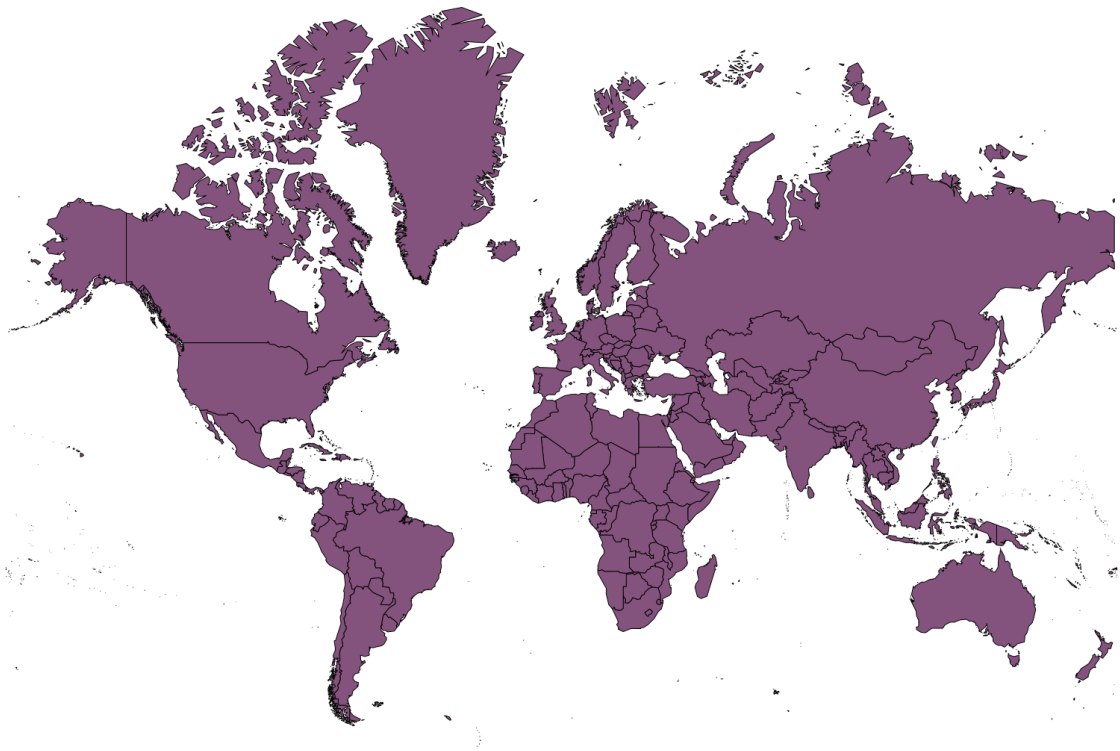


Figure 6.1: *Illustration of the general case data set, as rendered in Quantum GIS.*

useful mix of small and large ones. (As seen in figure 6.2.) The data model used by OpenStreetMap for building outlines[89] along with the nature of buildings ensures that all polygons are simple, with one set of geometry per feature. (Note that the OpenStreetMap data model does not actually adhere to the Simple Features specification, having only the simpler constructs.[90])



Figure 6.2: *Illustration of the special case data set, as rendered in Quantum GIS.*

In contrast to the general case data set, this data set has a high resolution and is therefore best viewed at a high level of zoom. Again, the zoom factor should not cause any problems, as the size of tiles relative to each other is the same.

6.3 Testing Patterns

With three algorithms to test on two data sets, of which one pattern can only be used on one of the data sets, there are in total five combinations of algorithms and data sets to test. In addition, there are three different implementation variations to test so that they can be contrasted against each other:

Simple pseudo-union, not cached Running a test completely without cache will highlight differences in tile production time between methods.

Simple pseudo-union, cached A test with caching will give indications as to how much of the tile delivery time is construction and how much is transport time. For the cached tests all tiles are "seeded", which is to say they are generated previously and placed in the cache.

Jigsaw experimental union, cached While the experimental union currently gives incorrect union results (see section 5.2.4), it does exact a construction time penalty that probably lies close to that experience by a proper union operation. Running this test will give indications as to how large tile union processing time is compared to the collecting of feature components.

In total that adds up to fifteen different tests to be run. Each of them will be run for a thousand iterations in order to achieve a sound statistical foundation for conclusions.

7 Results & Discussion

7.1 Algorithmic Execution Speed Analysis

The following is an asymptotic analysis of the algorithms used for concatenation, based on their descriptions from section 4. There are four variables that affect the runtime performance of these algorithms: The first is n , denoting the number of tiles involved in the concatenation operation. The second is s , which is a measure of how many feature segments there are per feature to be concatenated. This number is obviously not possible to define exactly, instead one might use a average number. Likewise, the third and fourth variables f and k , denoting the number of features available to concatenate and the number of points in each feature segment, are only estimations of the average value.

Since s , f and k are sizes that do not grow towards ∞ , they can be disregarded when describing the asymptotic growth of the function as n goes towards ∞ . They are included, however, because it may be useful information in a typical usage scenario such as where one needs to concatenate a screen full of tiles. In that case n might only be around 30, and s , f and k may have an effect. Note that even then, s and f will be smaller than n , meaning that n is likely still the dominant factor, although k may begin to rival it.

7.1.1 Generalised Global Search

Since the generalised global search algorithm will check every single available tile, the value of s does not actually matter. The algorithm goes through every tile, checking for features that have not yet been concatenated. For each of those, another full traversal of tiles is performed. This gives a growth function of $\Theta(n \cdot f \cdot n) = \Theta(fn^2)$. Generalised for when n moves towards ∞ , it becomes simply $\Theta(n^2)$.

7.1.2 Specialised Global Search

The specialised algorithm shares the first step, but does not perform a full traversal of all tiles for each new feature it starts assembling. Instead, it investigates neighbours for connections, which depends on the complexity of the feature segment being investigated. This results in a growth function of $\Theta(n \cdot f \cdot 4 \cdot s \cdot k) = \Theta(4fks)$. Generalised for when n moves towards ∞ , it becomes simply $\Theta(n)$. This proves that theoretically, the specialised version of the algorithm should be more efficient.

7.1.3 Edge Pointers

Like the specialised global search algorithm, this algorithm does all full traversal of all tiles for unconcatenated features but restricts the secondary search. In this case it is done through preprepared edge pointers, eliminating the question of the feature segment's complexity. This results in a growth function of $\Theta(n \cdot f \cdot 4 \cdot s) = \Theta(4fns)$. Generalised for when n moves towards ∞ , it also becomes simply $\Theta(n)$.

7.1.4 Summary

We see that all the algorithms share the common first step of traversing every tile looking for unassembled features to work with, resulting in a common lower bound of $\Omega(n)$. We also see that the generalised global search in theory will perform worse than the two others, who are equivalent. However, this is only when n goes towards ∞ , which is not particularly relevant in a user scenario concatenating one screen full of tiles at a time. Since the other factors come into play at this level, performing timing tests to ascertain their behaviour is still necessary.

7.2 Timing Results

The following tables present timing results from all of the executed tests. The time spent running tests was about a half an hour for each test, totalling seven and a half hours not counting time spent on stalls. Please see appendix C for the full data files containing more detailed information from each test run. Before the tests were run, the computer was unplugged from the network and restarted, and all extraneous applications were shut down to ensure that as few factors as possible could affect the timing results. (See appendix F for more details on the computer used for the tests.)

Sometimes the test runs would stall due to the random movement (being truly random) requesting a position that resulted in the same tiles as previously downloaded being requested. Since they were already in place, no downloads would be made and the concatenation procedure would not be triggered, thus preventing the test from moving on. This was rectified by manual observation and intervention, when the screen was seen to have no movement for an extended period of time a manual movement was performed so that the test runner would recover from the stall. The timing measurements that were adversely affected by these stalls have been removed from the collected data. Please refer to table 7.1 for information on how many usable timing results were recorded for each test run.

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	1000	1000	1000
General data, edge pointers	998	999	999
Special data, general global search	999	999	1000
Special data, special global search	1000	999	1000
Special data, edge pointers	998	999	999

Table 7.1: *Number of usable samples from all test runs.*

7.2.1 Download Times

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	1886.506	342.757	323.664
General data, edge pointers	3344.753	269.642	314.751
Special data, general global search	463.933	259.825	302.563
Special data, special global search	497.536	300.790	354.355
Special data, edge pointers	604.595	370.071	423.282

Table 7.2: *Mean download time from all test runs.*

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	4573971.922	537211.982	28735.845
General data, edge pointers	21044065.587	30669.565	23414.019
Special data, general global search	160109.760	16986.289	20923.510
Special data, special global search	1514362.221	28689.280	38400.984
Special data, edge pointers	402006.378	48011.691	58571.772

Table 7.3: *Download time variance from all test runs.*

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	2138.685	732.948	169.517
General data, edge pointers	4587.381	175.127	153.0164
Special data, general global search	400.137	130.332	144.650
Special data, special global search	1230.594	169.379	195.962
Special data, edge pointers	634.040	219.116	242.016

Table 7.4: *Standard deviation of download time from all test runs.*

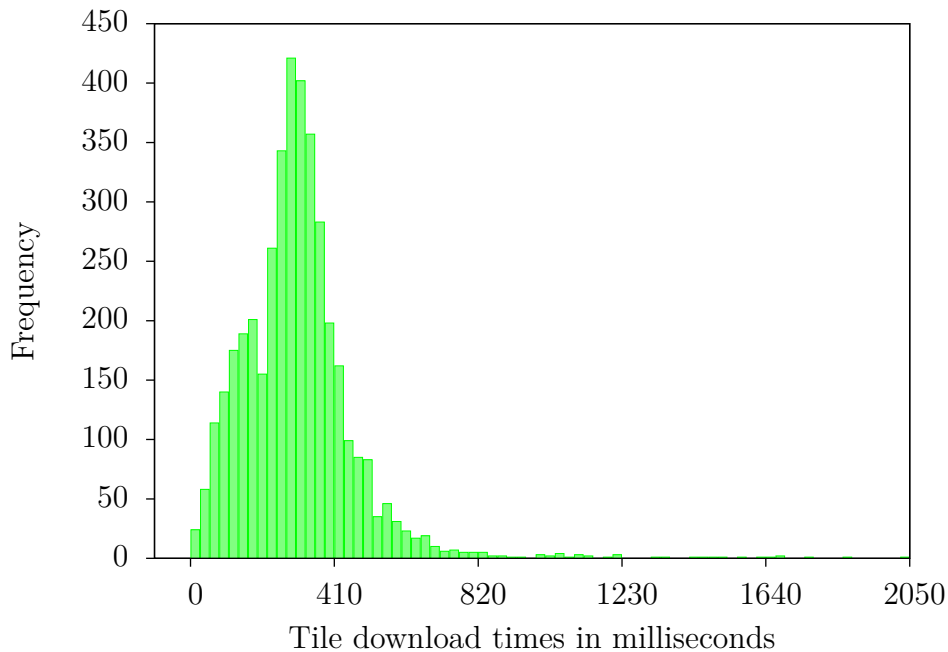


Figure 7.1: *Histogram of all 3998 download times for special data set, cached. Collected from four test runs with identical circumstances.*

The astute observer will notice that the histogram in figure 7.1 shows signs of not being normally distributed, with a pronounced positive skew resulting in a large concentration to the left and a long tail on the right.[91] In order to test that assertion, we can perform Geary’s test, a statistical test designed to test for normality that is simple and works well on large sets of data. [92, 93]

$$U = \frac{\sqrt{\frac{\pi}{2} \frac{\sum_{i=1}^n |X_i - \bar{X}|}{n}}}{\sqrt{\frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n}}} \quad (7.1)$$

Geary’s test (equation 7.1) examines the ratio between two common estimators for the standard deviation, σ . The denominator is a reasonable estimator regardless of if the data set is normal or not, whilst the numerator is only good when the distribution is normal. If the distribution is not normal, the numerator will often underestimate or overestimate σ causing the value of U to change. As such, if the value of U differs considerably from 1.0 this indicates a non-normal distribution.

$$U = \frac{\sqrt{\frac{\pi}{2} \frac{441591.8649}{3998}}}{\sqrt{\frac{109346243.5878}{3998}}} = 0.8371$$

Referring to the table in [94], it would indeed appear that this distribution is not normally distributed. (For full calculations, see the attached spreadsheet files listed in appendix C.) This is likely a consequence of the simple fact that the act of downloading tile data has a natural limit: Download time can not be less than zero. It is possible that the distribution might have been normal if tile download time uniformly was so large that the mean was much further away from zero and near-zero download times never occurred. Implications of the non-normal distribution are further discussed in section 7.3

Looking at the relevant tables (7.2 to 7.4) we can begin to discern some patterns. As expected, the time for download is in all cases larger when it is uncached then when it is cached. Similarly, uncached download times for the edge pointer solution are always larger than uncached download times for the unmodified vector tile constructor, which is to be expected due to the additional processing on the server.

An initially more surprising result is that the edge pointer solution when cached appears to have a smaller download time for the general case than the unmodified vector tiles, as this is contrary to the logical conclusion that they must be slightly larger due to the extra data they carry. The same trait is not visible on the download data for the special case. However, examining the variance and standard deviation shows that in all cases of cached downloads, the means are within one standard deviation of each other. This could indicate that the difference is in fact so small that a much larger sample size is needed to be able to detect it properly.

The two sets of cached download time for the two variations of union should in theory have identical means and variances, yet the data shows they have not. Again, each mean is within one standard deviation of the other means. It would seem that because of the non-normal distribution of download time results, a few results along the heavy tail of the distribution have a great influence on the means. This is indicated by the high variance of results. Increasing the sample size of the tests would probably narrow down the discrepancy between the recorded means. (When n goes towards ∞ , the means should converge.)

At the sample sizes used in this study it is hard to discern a statistically significant difference in cached download times between the edge pointer solution and the unmodified vector tile solution. However, this is in fact a result in itself: One can expect the average user to make a lot fewer than one thousand map movements, so from the user perspective there would be no perceptible difference in download times. This is a characteristic that is worth noting.

7.2.2 Concatenation Times

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	52.392	52.148	316.831
General data, edge pointers	57.902	59.188	328.904
Special data, general global search	180.538	51.283	92.384
Special data, special global search	49.752	42.230	77.412
Special data, edge pointers	35.014	35.686	67.889

Table 7.5: Mean concatenation time from all test runs.

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	3401.496	5035.972	348302.225
General data, edge pointers	3497.092	5970.868	344861.598
Special data, general global search	53360.756	6355.470	15506.761
Special data, special global search	4818.931	2058.191	8346.677
Special data, edge pointers	1428.762	1443.065	7557.478

Table 7.6: Concatenation time variance from all test runs.

Test Run	No Cache	Cache	Exp. Union, Cache
General data, global search	58.322	70.965	590.171
General data, edge pointers	59.136	77.271	587.249
Special data, general global search	231.000	79.721	124.526
Special data, special global search	69.419	45.367	91.360
Special data, edge pointers	37.799	37.988	86.934

Table 7.7: Standard deviation of concatenation time from all test runs.

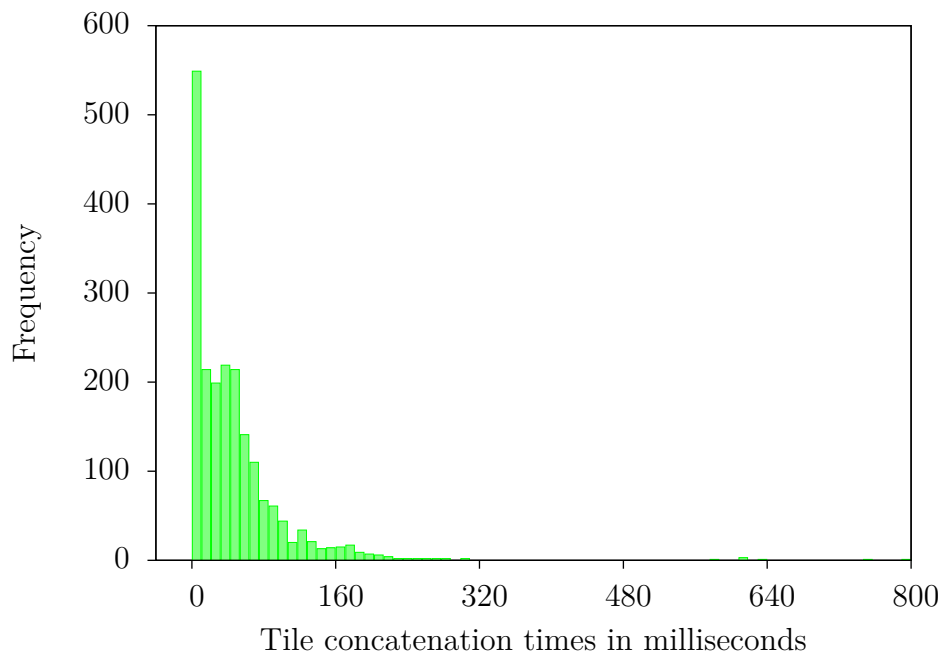


Figure 7.2: *Histogram of all 1999 concatenation times for special data set, cached, concatenated with specialised global search algorithm and pseudo-union. Collected from two test runs with identical circumstances.*

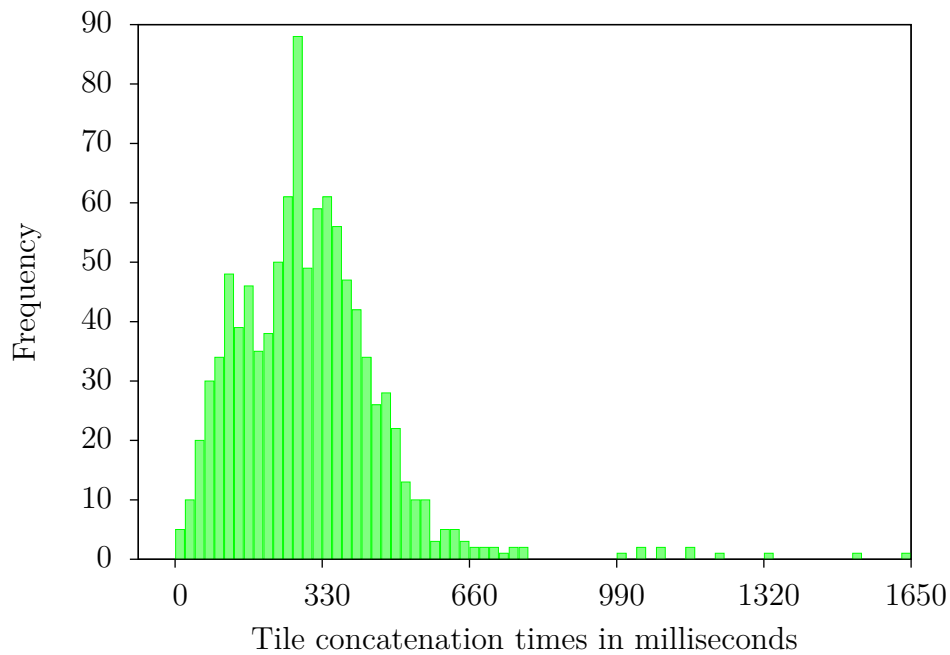


Figure 7.3: *Histogram of 999 concatenation times for special data set, cached, concatenated with specialised global search algorithm and experimental union. Collected from a single test run.*

Once again, we observe apparently non-normal distributions. Checking with Geary's test like before we find that this interpretation is correct and that they indeed are non-normal. (Figure 7.2 has a U of 0.7662, and figure 7.3 has a U of 0.8883)

Figure 7.2 illustrates concatenation time with the pseudo-union method, and is therefore as close an estimate to the cost of the actual feature reassembly algorithm as we can get. It shows that the actual discovery of related tiles is fairly quick. By contrast, notice how figure 7.3 not only shows a lot larger use of time, but also bears a close resemblance to the download time histogram in figure 7.1. This is a possible indication of covariance between the tile's complexity (and therefore its download size) and the time needed to create a union.

The time expenditure for the experimental union is generally higher than for the pseudo-union. Time spent on concatenation with the experimental union is also noticeably larger on the general data set compared to the special data set, as might be expected considering that the general data set is more complex. It is worth noticing the high variance and standard deviation here, indicating that tiles with complex content require a processing time that is so large that it profoundly affects the mean time.

A lesson to take from the contrast in timing results between pseudo-union and the experimental proper union is that for application usage it is the actual merging of the tiles that takes the most time, not finding which tiles are to be used as sources for the merging.

The timing numbers indicate that the specialised global search is more efficient than the general global search on the special data set, which is as expected. Furthermore, the edge pointer solution outperforms even the specialised global search. This is also in line with expectations. However, for the general data set the edge pointers appear to have performed slightly worse than the global search solution. This result was not as expected, but might be related to the longer chains of edge pointers in a multiple geometry environment and the way the chains are investigated. Again, both means are well within one standard deviation of each other, indicating that the differences might be down to chance.

Like with the download times, at the number of samples used it is somewhat

difficult to make accurate statements about the differences between the different methods due to the large variance induced by the non-normal distribution. Still, it would appear that it is not possible to point out any solution that is better under any circumstance. Beyond that it seems that from a user perspective it would be difficult to perceive any marked difference between the global search and edge pointer solutions in terms of execution speed. (Especially when both are dwarfed by the time spent on proper merging in the general case.) This is also a valuable lesson insofar as it means that the choice of method can be informed by other considerations than execution speed.

7.3 Non-Normality of Tile Loading & Concatenation Time Distributions

As it was discovered in section 7.2, vector tile generation, download and concatenation operations do not follow a normally distributed probability curve. This has implications for any attempts to predict the behaviour of such a solution, for example when trying to determine characteristics of a system's behaviour under load. One aspect is that when one needs to compare logged timing results, one can not apply many of the standard statistical tests that engineers often resort to, and must therefore determine other tests that may be suitable. Another aspect is that when modelling future behaviour, a suitable kind of distribution must be found.

The normal distribution is the most common distribution for naturally occurring variations, and is widely used in statistical modelling for engineering purposes. It provides a pleasingly elegant model, and there a lot of common statistical tests available for working with such distributions. However, there is no law of nature demanding that phenomena adhere to a normal distribution, and in this case there are factors that make other distributions a better fit.

All of the tile loading and concatenation time data has a natural lower limit of zero, since tiles cannot be delivered before they are requested. In itself this does not have to be incompatible with normal distributions, but only if the process takes so much longer than zero that it never comes into the question. This is clearly not the case for vector tile solutions, where in fact paring down times as close to zero as possible is the goal. The collected data also reveals an interesting fact about vector tiles, namely that the majority of them are either empty or contain a single feature segment. (Typically a square segment from a larger feature at high zoom levels.) These simple tiles take an accordingly small amount of time to process.

Only a few tiles are actually highly complex, taking much longer time to deliver.

Several related distributions were considered as possible fits for the collected data, including gamma, log-normal and Weibull distributions. Using a distribution-fitting library[95] for the R programming language (an language designed for statistical tasks[96]), maximum likelihood estimations were performed against the measured data sets for each of the candidate distributions. The estimated fit with the lowest standard error was for a gamma distribution, see figure 7.4 for a graphical representation of how well the fitted distribution corresponds with the data set.

$$f(x; \alpha, \beta) = \begin{cases} \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}, & x > 0 \\ 0, & \text{elsewhere} \end{cases} \quad (7.2)$$

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} e^{-x}, dx \quad (7.3)$$

The gamma distribution is often applied to queuing theory and reliability problems[92] where x must be larger than zero. It is often used for modelling time-to-arrival problems, and as such should be a good fit for modelling tile download times. The values for α and β , (see equations 7.2 and 7.3 for the gamma distribution density function), have been determined by the program to be 3.17522 and 0.01056, with standard errors of 0.130781 and 0.000467, respectively.

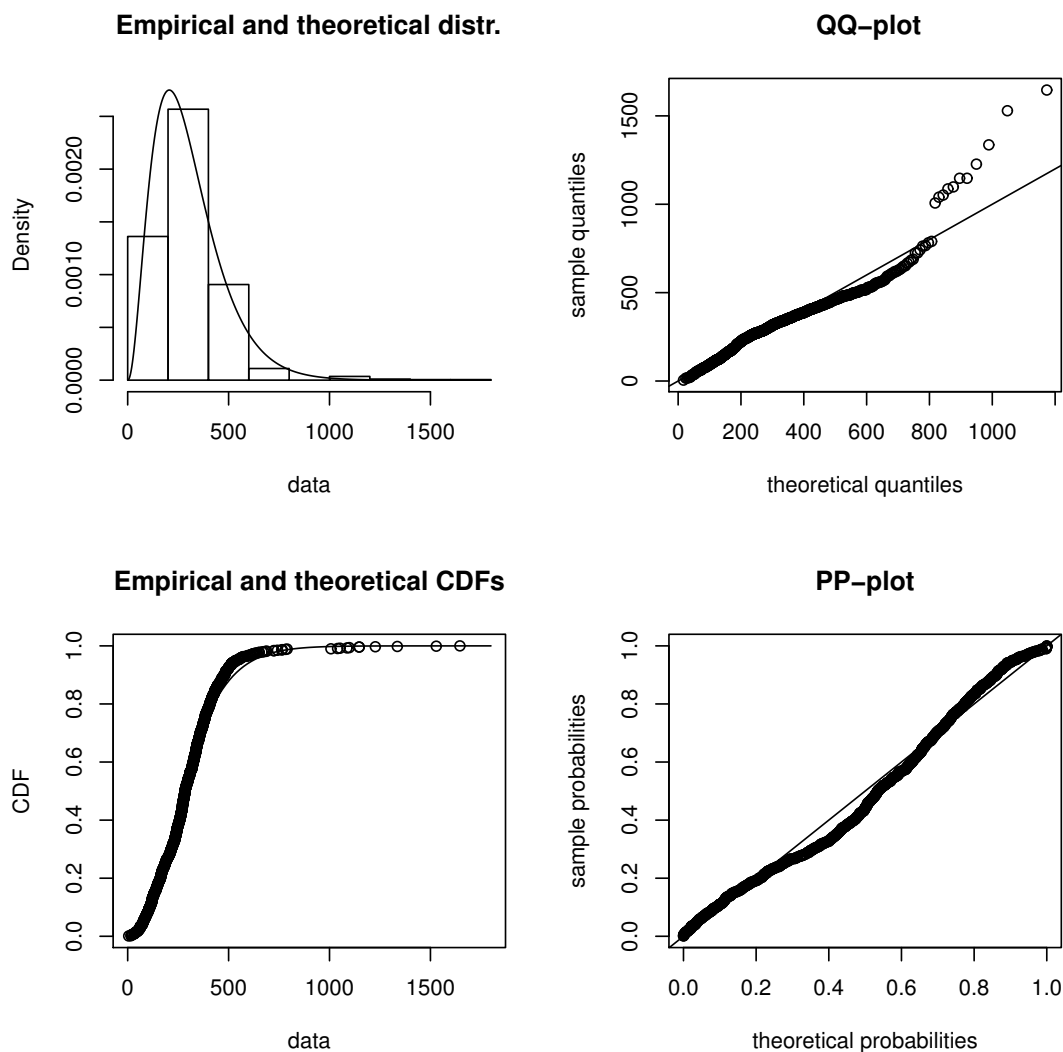


Figure 7.4: Estimated fit of Gamma distribution to the special case, experimental union data.

7.3.1 Implications

The gamma distribution does not have any implications other than that commonly used statistical tests intended for normal distributions cannot be used. It is an important observation precisely because using statistical methods meant for normal distributions would have produced erroneous predictions that could have adversely affected decision making. Apart from that, gamma distributions are perfectly functional for making predictions about probabilities of tile delivery time. (There are some statistical tests that can be applied to non-normal distributions,

but they are uncommon and belong to advanced subjects in statistics which the author, quite frankly, does not have a good enough knowledge of to effectively apply.)

One important question regarding the collected data and fitted distribution is this: Are the results really applicable for a real-life production application? No actual user will be moving completely randomly around the map, making navigational jumps that are not possible when using a standard slippery map interface. Instead, their use might typically be clustered around a single area within a limited amount of zoom levels. (Meaning that all requests would be for tiles of similar complexity.)

For predicting the tile load incurred by a single user, the answer is most likely that the collected data is not particularly accurate. However, for predicting the total load coming from a large number of users, it is quite likely relevant. A large number of users will be looking at different things, creating a spread of tile requests that more closely resemble a random distribution. Therefore the data is useful for planning the capacity of a multi-user application.

One hypothesis worth noting, which requires further research to verify, is that actual users will probably make fewer requests for sparsely detailed areas such as the middle of the ocean than a random process does. (Likewise, they might make more requests for densely detailed areas such as population centres.) In such a case, the distribution of results would probably be less positively skewed with a smaller concentration on the left and possibly a more pronounced right hand tail. It would however still likely adhere to a gamma distribution.

7.4 Feasibility of GIS Operations

As explained in section 1.1.3, there already exists a standardised system for doing GIS work on vector data transmitted over the internet, namely WFS and WFS-T. If one was to consider using a vector tile solution for similar tasks, as opposed to simply viewing the data, there are a number of issues to consider. The prototype work that has been performed addresses some of them, while others to greater or lesser extent are left for future research to consider.

7.4.1 Data Completeness

If one is to be able to perform GIS analysis on geographic features, one must be sure that one has acquired the entire feature first. The prototype has demon-

strated that using edge pointers is a viable method of ensuring that all the data needed to completely reassemble a feature is collected before assembly is done. The prototype has not actually implemented the downloading action required to acquire new sections, but it does recognise when there are missing pieces of a feature.

If one was to implement such forced downloading of tiles with missing feature segments, it would likely introduce a significant delay for the tile concatenation operations. This is because one would have to wait for needed tiles to finish downloading before one could finish assembly. (Provided none of the downloaded tiles point to further tiles that need to be downloaded!) It is in other words perfectly feasible, but may incur a reduction in performance as seen from the user's perspective.

7.4.2 Projection & Distortions

As noted in section 1.2, the EPSG:3857 "WGS 84 / Pseudo-Mercator" projection is the most commonly used projection in tiling systems, and is also the one used in the prototype. This is for reasons of expediency and compatibility with raster tile providers like OpenStreetMap. EPSG:3857 is quite well suited for visualisation of geographic data in a user context where absolute precision is not required. This is primarily because it is able to project all areas of the world (except the poles) with relatively good accuracy, and is easy to use in a quadratic addressing system like tiling systems use.

The downside of using this projection is the compromises that have to be made. In order to easily encompass the entire world, EPSG:3857 sees the world as a perfect sphere, something it is not. While this is an acceptable assumption at small scales, it is not as well suited for larger scales. Because of this distortions of up to 800 meters in position and 0.7 % in scale may arise.[21] For proper GIS work one needs a projection that allows a minimal amount of distortion in the area one is interested in.

In Norway, the Universal Transverse Mercator (UTM) system is commonly used. UTM is divided into zones where each zone has an acceptably low distortion for a given geographic area. (For example UTM zone 32 works well in western and central Norway.) But outside of this zone distortion increases to an unacceptable level, which is why a single UTM zone projection can not be used for a global map.

Therefore, if one is to use a vector tile system for GIS work, one must abandon the EPSG:3857 projection and select a projection that is suitable for the area of

interest. Per the WMTS, any projection can be used for a tiling service as long as the server provides the client with a definition of the scale set used for addressing with that projection. As such, a tile server can specify for example the UTM32 projection for a set of tiles, thus providing good accuracy within a relatively small area. The corresponding downside is the inability to use said tile server for a global application, and incompatibility with tile servers such as OpenStreetMap meaning they can not be used as a background layer.

7.4.3 Preservation of Data Integrity

For GIS work it is paramount that the data is correct, and this becomes more of a challenge when the data has been assembled from many different sections. How does one know that the union operations have performed flawlessly and recreated the exact feature that was the source for all the segments transmitted through tiles? The prototype does not do any work on such verification due to the incomplete implementation of unions, but it is a question that must be considered.

One possible method is to implement some sort of hashing or checksum operation that creates a short and unique fingerprint of the feature. That fingerprint could then be transmitted with the fragments on each tile, and after the union operation a new fingerprint could be created and compared to the original. The extra data transmission would be minimal, and the hashing operation on the server would be inconsequential due to caching. However, depending on how complex the hashing operation is, it might cause a noticeable delay in the union operations.

Another issue is what to do if there is a mismatch between the provided and the generated fingerprint. Potentially, one could try rerunning the union operation with altered parameters (such as starting from a different tile), or one could download a fresh copy of the entire feature in a manner akin to requests from a WFS server. (One could simply provide a WFS interface to the same data set.)

7.4.4 Data Upload

Provided the issues in the previous sections have been taken care of, the data can be manipulated using various GIS operations. How does one then upload the changed data to the server? One option is to once again cut the data into tiles and send them up to the server for reassembly. However, this certainly seems like an unnecessarily complex method, involving replication of server functionality on the client as well as client functionality on the server and creating time consuming

intermediary steps.

Continuing along that line of thought, it would be more sensible to upload the entire feature in one chunk. In other words, exactly like the way WFS-T handles data upload. It might well be a good idea to provide a WFS-T interface to the data store and make use of that for data upload in order to avoid reinventing the wheel.

8 Conclusions

8.1 Observations

Through the tests that have been run, insights have been gained about the performance of the tested solutions. This has provided useful knowledge that can be contrasted with the a priori assumptions made about them.

8.1.1 Generalised Versus Specialised Methods

The testing has demonstrated that specialised methods can indeed have advantages over a general method. (The specialised global search performed better than the generalised global search on a special data set.) However, it turned out that using the edge pointer method provided even better results on the special data set. This indicates that implementing different algorithms for special and general data sets may not be worthwhile.

8.1.2 Importance of Added Data

The timing results indicate that adding a small amount of extra data, such as for edge pointers, is inconsequential compared to the amounts of data transmitted for a feature. The extra transmission burden disappears against the "background noise" of the varying sizes of feature data being transmitted, meaning there is no discernible difference between edge pointer tiles and unmodified tiles when examining transmission time. This is of course dependent on the data being cached, as generating edge pointer tiles still requires a large premium of time. With that requirement fulfilled, the developer can disregard small differences in data transmission size as a factor when choosing between topology preservation methods.

8.1.3 Importance of Caching

It can be concluded that using a cache to deliver tiles has a major impact on on tile delivery time in all cases. Furthermore, in those cases where extra processing is performed on the server, such as for edge pointers, caching has an even greater effect as regards delivery time reduction. With caching enabled, the difference in delivery time between methods is for all intents and purposes eliminated. This leaves the developer free to choose between topology preservation methods based on other merits than generation time.

8.2 Recommendations

Based on the knowledge gathered during this project, some recommendations can be made for future reference.

8.2.1 Choice of Methods

There is no definite winner among the proposed methods for topology preservation, as the results show that performance varies depending on the circumstances. In general, the most important factor determining performance is caching, not which of the tested methods is employed. Because of this, other factors can be taken into consideration.

Therefore, the author sees using edge pointers as the most promising method for preservation of topology in vector tile applications. (With the important proviso that caching must be employed in order to negate the increased tile generation time.) This is because using edge pointers allows the application to guarantee data completeness, while having broadly similar performance to a global search method. By using the edge pointer method and employing a correct union method (which is left for future work, see section 9) it should be quite possible to create a vector tile solution that complies with the SFA.

8.2.2 Utility for GIS Operations

It is quite possible to create a GIS tool using vector tiles, taking into account the issues raised and remedies proposed in section 7.4. However, the added complexity that entails, along with the probable need to run a WFS server in parallel, makes it an impractical suggestion. In the author's opinion, using vector tiles for GIS applications is less suitable than using a plain WFS server and client. Vector tile solutions are best suited for visualisation, interactivity and providing added information about features.

9 Future Work

There are several avenues of future work which may be pursued in relation to the subjects covered in this thesis, some of which are mentioned here. They consist both of further development of the concepts explored in this thesis as well as concepts that have been omitted as a result of prioritisation (such as speed) and concepts that have appeared during the course of the thesis work (such as tile load distributions).

9.1 Further Development, Speed & Compactness

As this report has concerned itself with topology and data integrity rather than execution speed, there are likely large gains to be made in speed and efficiency. This applies equally to the server side and the client side applications, as they both were purely prototype implementations. The prototype is not well-suited for further development as it essentially is a modification of an existing product, bringing with it inherent flaws. Therefore developing a new vector tile application from the ground up, taking into account the experience gained from this prototype, would be a suitable goal for further academic work. (See also the next two subsections.)

At the same time as this work was performed, another master thesis study has been performed at NTNU, looking into the potential for increasing speed and efficiency in web maps through using vector tiles.[70] It would probably be very helpful to combine the experience gained from both reports when designing a new vector tile application.

9.1.1 Polygon Union in JavaScript

One major challenge left unsolved by this report is the issue of polygon union, as detailed in section 5.2.4. Therefore, a suitable extension of the work done here would be to continue working on methods of performing accurate polygon unions in JavaScript. It is probable that these methods will be implemented in browsers when the SVG 2 standard has been finalised, but until then something is needed to bridge the gap. Another viable academic project might be to get involved with the standards process and work on implementing union features for SVG in the major open source browsers ahead of the standard finalisation. In either case, the major academic challenge would be working on creating the most optimised approach to polygon clipping in a possibly resource-constrained environment. (For web users,

the execution speed is the most important factor.)

One could also go for a more specialised route, and work on a union method that was specifically adapted for vector tile applications. Both the work performed for this report and the work done in [22] would be good starting points for such an approach.

9.1.2 GIS Enhancements

As noted in section 7.4, making a GIS solution using vector tiles is probably feasible but might not be practical. An avenue of research could be to investigate these conclusions more closely, to see if it really is impractical or not. At any rate, subjecting issues like upload methods, access to underlying data, integrity checks and so on to a more rigorous investigation might yield valuable insights that can be applied in other settings as well.

9.2 Tile Load Distributions

As discovered through the testing performed during this project, tile loading times have a gamma distribution. However, this was during synthetic tests with random movements. An interesting angle that could be investigated further is looking at real-life tile load situations with human users. Both tile load times from individual users and tile load times aggregated over many users would be helpful in pointing out usage patterns, and would be helpful to confirm or deny the distribution patterns discovered during the synthetic testing.

Probably the greatest challenge with such a project is the scale of the experiment. One would need many different users, and they would also have to be simply using the system for self-defined tasks. Inviting volunteers to complete a series of goals would not work, as it would effectively predetermine their usage patterns. One possible solution would be to approach some larger entity offering map services and request a cooperation for the purpose of collecting anonymous usage data.

References

- [1] S. Putz, “Interactive information services using world-wide web hypertext,” *Computer Networks and ISDN Systems*, vol. 27, no. 2, pp. 273 – 280, 1994. Selected Papers of the First World-Wide Web Conference.
- [2] T. Berners-Lee, “Www: past, present, and future,” *Computer*, vol. 29, pp. 69 –77, oct 1996.
- [3] M. Peterson, “Twenty years of the world wide web: Perspectives on the internet transition in cartography,” in *Proceedings of the 25th International Cartographic Conference, Paris, France, July 3-8, 2011.*, 2011.
- [4] J. Sample and E. Ioup, *Tile-based geospatial information systems: principles and practices*. Springer-Verlag New York Inc, 2010.
- [5] Open Geospatial Consortium, “OpenGIS Web Map Server Interface Implementation Specification 1.3.0,” 2006.
- [6] J. Garrett, “Ajax: A new approach to web applications,” 2005.
- [7] S. Liu and L. Palen, “The new cartographers: Crisis map mashups and the emergence of neogeographic practice,” *Cartography and Geographic Information Science*, vol. 37, no. 1, pp. 69–90, 2010.
- [8] M. Haklay and P. Weber, “Openstreetmap: User-generated street maps,” *Pervasive Computing, IEEE*, vol. 7, no. 4, pp. 12–18, 2008.
- [9] L. Vincent, “Taking online maps down to street level,” *Computer*, vol. 40, pp. 118 –120, dec. 2007.
- [10] M. Rost, H. Cramer, N. Belloni, and L. Holmquist, “Geolocation in the mobile web browser,” in *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing*, pp. 423–424, ACM, 2010.
- [11] C. Grier, S. T. King, and D. S. Wallach, “How i learned to stop worrying and love plugins,” in *In Web 2.0 Security and Privacy*, 2009.
- [12] A. Taivalsaari and T. Mikkonen, “The web as an application platform: The saga continues,” in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pp. 170–174, IEEE, 2011.
- [13] P. Lubbers, B. Albers, and F. Salim, *Pro HTML5 Programming*. Springer, 2011.

- [14] I. Heywood, S. Cornelius, and S. Carver, *An Introduction to Geographical Information Systems*. Essex, UK: Pearson Education Limited, 3. ed., 2006.
- [15] L. D. Cola, “The use of a raster data structure to summarize a point pattern.” (available online at https://en.wikipedia.org/wiki/File:The_use_of_a_raster_data_structure_to_summarize_a_point_pattern.gif), 2011.
- [16] Open Geospatial Consortium, “OpenGIS Feature Service Implementation Specification 1.3.0,” 2010.
- [17] Open Geospatial Consortium, “OpenGIS Web Map Tile Service Implementation Specification 1.0.0,” 2010.
- [18] M. Worboys and M. Duckham, *GIS - A Computing Perspective*. CRC Press, 2. ed., 2004.
- [19] P. Přidal and P. Žabička, “Tiles as an approach to on-line publishing of scanned old maps, vedute and other historical documents,” *e-Perimtron*, vol. 3, no. 1, pp. 10–21, 2008.
- [20] E. Hazzard, *OpenLayers 2.10 Beginner’s Guide*. Packt Publishing, 1. ed., 2011.
- [21] H. Butler, C. Schmidt, D. Springmeyer, and J. Livni, “EPSG:3857.” (available online at <http://www.spatialreference.org/ref/sr-org/6864/>), 2012.
- [22] V. Antoniou, J. Morley, and M. Haklay, “Tiled vectors: A method for vector transmission over the web,” in *Web and Wireless Geographical Information Systems* (J. Carswell, A. Fotheringham, and G. McArdle, eds.), vol. 5886 of *Lecture Notes in Computer Science*, pp. 56–71, Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-10601-9_5.
- [23] B. Campin, “Use of vector and raster tiles for middle-size scalable vector graphics’ mapping applications,” in *SVGOpen 2005*, 2005.
- [24] Stackexchange.com, “How to create vector polygons at the same amazing speeds giscloud is able to render them?.” (available online at <http://gis.stackexchange.com/questions/15240/how-to-create-vector-polygons-at-the-same-amazing%2DsPEEDS-giscloud-is-able-to-render>), 2012.
- [25] Robert Nordan and others, “Conversations between the author and employees of Norkart Geoservice AS,” January 2012.

-
- [26] OpenLayers user mailing list, “Giscloud showing tons of vectors features on web browser.” (available online at <http://osgeo-org.1560.n6.nabble.com/GisCloud-showing-tons-of-vectors-features-on-Web-Browser%2Dtd3913621.html>), 2012.
- [27] Open Geospatial Consortium, “Leaflet features.” (available online at <http://www.opengeospatial.org/ogc/process>), 2012.
- [28] Open Geospatial Consortium, “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture,” 2010.
- [29] Open Geospatial Consortium, “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option,” 2010.
- [30] Adobe Inc., “Adobe flash player 11 - tech specs.” (available online at <https://www.adobe.com/products/flashplayer/tech-specs.html>), 2012.
- [31] Techcrunch.com, “ios market share up from 26% in q3 to 43% in oct/nov 2011.” (available online at <http://techcrunch.com/2012/01/09/ios-marketshare-up-from-26-in-q3-to-43-in-octnov-2011/>), 2012.
- [32] Microsoft Inc., “Silverlight overview.” (available online at <http://msdn.microsoft.com/en-us/library/bb404700%28v=vs.95%29.aspx>), 2012.
- [33] ESRI, “Cots gis: The value of a commercial geographic information system,” white paper, ESRI, 2002.
- [34] ESRI, “Which api should i use: Javascript, flex, or silverlight?.” (available online at <http://events.esri.com/uc/QandA/index.cfm?fuseaction=answer&conferenceId=DD02CFE7-1422-2418-7F271831F47A7A31&questionId=3992>), 2012.
- [35] ESRI, “Arcgis api for silverlight.” (available online at <http://help.arcgis.com/en/webapi/silverlight/>), 2012.
- [36] OpenScales team, “Openscales - documentation.” (available online at <http://openscales.org/documentation/index.html>), 2012.
- [37] Microsoft Inc., “Microsoft support lifecycle - internet explorer 6.” (available online at <http://support.microsoft.com/lifecycle/?LN=en-us&x=5&y=6&p1=2073>), 2012.
- [38] Caniuse.com, “When can i use...” (available online at <http://caniuse.com/#cats=HTML5>), 2012.

- [39] OpenLayers team, “Frequently asked questions about the openlayers project.” (available online at <http://trac.osgeo.org/openlayers/wiki/FrequentlyAskedQuestions>), 2012.
- [40] H. Kraus, “Mapnik metawriter.” (available online at <https://github.com/mapnik/mapnik/wiki/MetaWriter>), 2012.
- [41] CloudMade, “Leaflet features.” (available online at <http://leaflet.cloudmade.com/features.html>), 2012.
- [42] Cartagen team, “Cartagen wiki home.” (available online at <https://github.com/jywarren/cartagen/wiki>), 2012.
- [43] SimpleGeo Inc., “Polymaps documentation.” (available online at <http://polymaps.org/docs/>), 2012.
- [44] SimpleGeo Inc., “Polymaps statehood example.” (available online at <http://polymaps.org/ex/statehood.html>), 2012.
- [45] D. Oehlman, “Tile5 - html5 mobile mapping.” (available online at <http://www.tile5.org/>), 2012.
- [46] D. Oehlman, “The future of tile5.” (available online at <http://www.tile5.org/news/the-future-of-tile5.html>), 2012.
- [47] GIS Cloud Ltd., “GIS Cloud Features.” (available online at <http://www.giscloud.com/features/>), 2012.
- [48] GIS Cloud Ltd., “GIS Cloud’s HTML5 Vector Map Engine Demo.” (available online at <http://www.giscloud.com/map/284/africa>), 2012.
- [49] D. Ravnic, “Re: GisCloud showing tons of vectors features on Web Browser .” (available online at <http://osgeo-org.1560.n6.nabble.com/GisCloud-showing-tons-of-vectors-features-on-Web-Browser%2Dtp3913621p3913639.html>), 2011.
- [50] R. Fielding and J. Gettys and J. Mogul and H. Frystyk and L. Masinter and P. Leach and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1.” (available online at <https://tools.ietf.org/html/rfc2616#section-9.5>), 1999.
- [51] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and C. Schmidt, “The GeoJSON Format Specification,” 2008.

-
- [52] M. Migurski, “TileStache API.” (available online at <http://tilestache.org/doc/>), 2012.
- [53] Google Inc., “Google summer of code.” (available online at <https://code.google.com/soc/>), 2012.
- [54] Nokia Inc., “Nokia maps 3d webgl (beta).” (available online at <http://maps3d.svc.nokia.com/webgl/index.html>), 2012.
- [55] Khronos Group, “WebGL Specification Version 1.0,” 2011.
- [56] F. A. Krueger, “Nokia 3d map tiles.” (available online at <http://idiocode.com/2012/02/01/nokia-3d-map-tiles/>), 2012.
- [57] M. Bertolotto and M. J. Egenhofer, “Progressive transmission of vector map data over the world wide web,” *Geoinformatica*, vol. 5, pp. 345–373, Dec. 2001.
- [58] P. Corcoran, P. Mooney, A. Winstanley, and M. Bertolotto, “Effective vector data transmission and visualization using html5,” in *Proceedings of the GISRUUK 2011 Portsmouth, England*, April 2011.
- [59] R. Weibel, “Generalization of spatial data: Principles and selected algorithms,” *Algorithmic foundations of geographic information systems*, pp. 99–152, 1997.
- [60] M. Fiedler, C. Eliasson, P. Arlos, S. Eriksén, and A. Ekelin, “Quality of experience and quality of service in the context of an internet-based map service,” technical report, Blekinge Institute of Technology, 2008.
- [61] M. Fiedler, C. Eliasson, P. Arlos, S. Eriksén, and A. Ekelin, “Mapping service quality – comparing quality of experience and quality of service for internet-based map services,” in *Proceedings of the 30th Information Systems Research Seminar in Scandinavia IRIS 2007*, 2007.
- [62] M. Taraldsvik, “Exploring the future: is html5 the solution for gis applications on the world wide web?,” technical report, NTNU, 2011.
- [63] Z. Liu, M. Pierce, G. Fox, and N. Devadasan, “Implementing a caching and tiling map server: a web 2.0 case study,” in *Collaborative Technologies and Systems, 2007. CTS 2007. International Symposium on*, pp. 247–256, may 2007.
- [64] M. Migurski, “TileStache Readme.” (available online at <https://github.com/migurski/TileStache>), 2012.

- [65] J.-G. Sanz-Salinas and M. Montesinos-Lajara, “Current panorama of the foss4g ecosystem,” *Novatica*, vol. X, no. 2, pp. 43–51, 2009.
- [66] Open Source Initiative, “List of Approved Open Source Licenses.” (available online at <http://www.opensource.org/licenses/alphabetical>), 2011.
- [67] J. Willinsky, “The unacknowledged convergence of open source, open access, and open science,” *First Monday [Online]*, vol. 10, August 2005.
- [68] R. Nordan and M. Migurski, “Closed pull request: Fixed topology errors in sample data (world_merc.shp).” (available online at <https://github.com/migurski/TileStache/pull/39>), 2012.
- [69] J. Spolsky and B. Eckel, “Strong typing vs. strong testing,” in *The Best Software Writing I*, pp. 67–77, Apress, 2005. 10.1007/978-1-4302-0038-3_11.
- [70] M. Taraldsvik, “The future of web-based maps: can vector tiles and HTML5 solve the need for high-performance delivery of maps on the web?.” (available online at <https://github.com/meastp/efficientvectortiles>), 2012.
- [71] Michael Migurski, “Polymaps license.” (available online at <https://github.com/simplegeo/polymaps/blob/master/LICENSE>), 2012.
- [72] M. K. Agoston, *Clipping*, ch. 3. Springer, 2005.
- [73] B. R. Vatti, “A generic solution to polygon clipping,” *Commun. ACM*, vol. 35, pp. 56–63, July 1992.
- [74] G. Greiner and K. Hormann, “Efficient clipping of arbitrary polygons,” *ACM Trans. Graph.*, vol. 17, pp. 71–83, Apr. 1998.
- [75] F. Martínez, A. J. Rueda, and F. R. Feito, “A new algorithm for computing boolean operations on polygons,” *Computers & Geosciences*, vol. 35, no. 6, pp. 1177 – 1185, 2009.
- [76] A. Johnson, “Clipper.” (available online at <http://sourceforge.net/projects/polyclipping/>), 2012.
- [77] F. Martínez, A. J. Rueda, and F. R. Feito, “An algorithm for computing Boolean operations on polygons.” (available online at http://wwwdi.ujaen.es/~fmartin/bool_op.html), 2012.
- [78] A. Zakai, “Emscripten.” (available online at <http://emscripten.org/>), 2012.
- [79] K. Gadd, “JSIL - .Net to Javascript Compiler.” (available online at <http://jsil.org/>), 2012.

-
- [80] M. Bohr, “Moore’s law in the innovation era,” vol. 7974, p. 797402, SPIE, 2011.
- [81] C. Severance, “Javascript: Designing a language in 10 days,” *Computer*, vol. 45, pp. 7–8, feb. 2012.
- [82] W. S. W. Group, “SVG2 Resolutions.” (available online at http://www.w3.org/Graphics/SVG/WG/wiki/SVG2_Resolutions#Keep_constructive_geometry_operations_in_Vector_Effects_and_see_if_it.27s_possible), 2012.
- [83] W. S. W. Group, “SVG2 Roadmap.” (available online at <http://www.w3.org/Graphics/SVG/WG/wiki/Roadmap>), 2012.
- [84] D. H. Kim and M.-J. Kim, “An extension of polygon clipping to resolve degenerate cases,” *Computer-Aided Design & Applications*, vol. 3, no. 1-4, pp. 447–456, 2006.
- [85] R. Hill, “Jigsaw Puzzle by Raymond Hill: A HTML5 canvas based jigsaw puzzle.” (available online at <http://www.raymondhill.net/puzzle-rhill/jigsawpuzzle-rhill.php>), 2012.
- [86] Dahlström, E., et al. (eds), “Scalable Vector Graphics (SVG) 1.1 (Second edition). World Wide Web Consortium Recommendation.” (available online at <http://www.w3.org/TR/SVG11/>), 2011.
- [87] E. Weyuker and F. Vokolos, “Experience with performance testing of software systems: issues, an approach, and case study,” *Software Engineering, IEEE Transactions on*, vol. 26, pp. 1147–1156, dec 2000.
- [88] Geofabrik GmbH, “Download OpenStreetMap Extracts - Europe.” (available online at <http://download.geofabrik.de/osm/europe/>), 2012.
- [89] OpenStreetMap Contributors, “Area.” (available online at <https://wiki.openstreetmap.org/wiki/Area>), 2012.
- [90] OpenStreetMap Contributors, “The Future of Areas/Simple Features.” (available online at https://wiki.openstreetmap.org/wiki/The_Future_of_Areas/Simple_Features), 2012.
- [91] NIST/SEMATECH, “Nist/sematech e-handbook of statistical methods.” (available online at <http://www.itl.nist.gov/div898/handbook/>), 2012.

- [92] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability & Statistics for Engineers & Scientists*. Upper Saddle River, NJ, USA: Pearson Education International, eighth ed., 2007.
- [93] R. D’Agostino, “Simple compact portable test of normality: Geary’s test revisited.,” *Psychological Bulletin*, vol. 74, no. 2, p. 138, 1970.
- [94] R. Geary, “Moments of the ratio of the mean deviation to the standard deviation for normal samples,” *Biometrika*, vol. 28, no. 3/4, pp. 295–307, 1936.
- [95] M. L. Delignette-Muller, R. Pouillot, J.-B. Denis, and C. Dutang, *fitdistrplus: help to fit of a parametric distribution to non-censored or censored data*, 2012. R package version 0.3-4.
- [96] R. Ihaka and R. Gentleman, “R: A language for data analysis and graphics,” *Journal of Computational and Graphical Statistics*, vol. 5, no. 3, pp. pp. 299–314, 1996.
- [97] Creative Commons, “Attribution-noncommercial-sharealike 2.5 canada (cc by-nc-sa 2.5).” (available online at <https://creativecommons.org/licenses/by-nc-sa/2.5/ca/>), 2012.

APPENDIX

A Project Assignment

MASTER DEGREE THESIS

Spring 2012
for

Student: Robert P. V. Nordan

An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications

BACKGROUND

Vector tiling is a new trend that the geospatial industry is likely to explore in coming years, bearing the promise of the advantages in clarity and interactivity afforded by vector data whilst also providing a cacheable and efficient solution akin to raster tiles. An important question is then to ascertain how one might ensure that metadata is preserved across tiles; i.e. how does one convey the fact that two lines on adjacent tiles are in fact part of the same road?

TASK DESCRIPTION

- Assess current vector tile solutions.
- Create hypothetical solutions for topology preservation.
- Create prototypes and test with regard to speed and functionality.
- Draw conclusions about topology preservation.
- Discuss possibility of using vector tiles for GIS tasks

General about content, work and presentation

The text for the master thesis is meant as a framework for the work of the candidate. Adjustments might be done as the work progresses. Tentative changes must be done in cooperation and agreement with the professor in charge at the Department.

In the evaluation thoroughness in the work will be emphasized, as will be documentation of independence in assessments and conclusions. Furthermore the presentation (report) should be well organized and edited; providing clear, precise and orderly descriptions without being unnecessary voluminous.

The report shall include:

- Standard report front page (from DAIM, <http://daim.idi.ntnu.no/>)

- Title page with abstract and keywords.(template on: <http://www.ntnu.no/bat/skjemabank>)
- Preface
- Summary and acknowledgement. The summary shall include the objectives of the work, explain how the work has been conducted, present the main results achieved and give the main conclusions of the work.
- Table of content including list of figures, tables, enclosures and appendices.
- If useful and applicable a list explaining important terms and abbreviations should be included.
- The main text.
 - Clear and complete references to material used, both in text and figures/tables. This also applies for personal and/or oral communication and information.
 - Text of the Thesis (these pages) signed by professor in charge as Attachment 1..
- The report must have a complete page numbering.

Advice and guidelines for writing of the report is given in: "Writing Reports" by Øivind Arntsen. Additional information on report writing is found in "Råd og retningslinjer for rapportskrivning ved prosjekt og masteroppgave ved Institutt for bygg, anlegg og transport" (In Norwegian). Both are posted on <http://www.ntnu.no/bat/skjemabank>

Submission procedure

Procedures relating to the submission of the thesis are described in DAIM (<http://daim.idi.ntnu.no/>). Printing of the thesis is ordered through DAIM directly to Skipnes Printing delivering the printed paper to the department office 2-4 days later. The department will pay for 3 copies, of which the institute retains two copies. Additional copies must be paid for by the candidate / external partner.

On submission of the thesis the candidate shall submit a CD with the paper in digital form in pdf and Word version, the underlying material (such as data collection) in digital form (eg. Excel). Students must submit the submission form (from DAIM) where both the Ark-Bibl in SBI and Public Services (Building Safety) of SB II has signed the form. The submission form including the appropriate signatures must be signed by the department office before the form is delivered Faculty Office.

Documentation collected during the work, with support from the Department, shall be handed in to the Department together with the report.

According to the current laws and regulations at NTNU, the report is the property of NTNU. The report and associated results can only be used following approval from NTNU (and external cooperation partner if applicable). The Department has the right to make use of the results from the work as if conducted by a Department employee, as long as other arrangements are not agreed upon beforehand.

Tentative agreement on external supervision, work outside NTNU, economic support etc.

Separate description to be developed, if and when applicable. See <http://www.ntnu.no/bat/skjemabank> for agreement forms.

Health, environment and safety (HSE) <http://www.ntnu.edu/hse>

NTNU emphasizes the safety for the individual employee and student. The individual safety shall be in the forefront and no one shall take unnecessary chances in carrying out the work. In particular, if the student is to participate in field work, visits, field courses, excursions etc. during the Master Thesis work, he/she shall make himself/herself familiar with “ Fieldwork HSE Guidelines”. The document is found on the NTNU HMS-pages at <http://www.ntnu.no/hms/retningslinjer/HMSR07E.pdf>

The students do not have a full insurance coverage as a student at NTNU. If you as a student want the same insurance coverage as the employees at the university, you must take out individual travel and personal injury insurance.

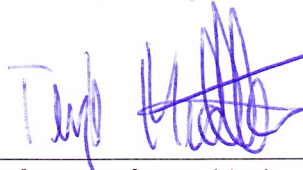
Start and submission deadlines

The work on the Master Thesis starts on January 16, 2012

The thesis report as described above shall be submitted digitally in DAIM at the latest at 3pm June 11, 2012

Professor in charge: Terje Midtbø

Trondheim, January 16, 2012. (revised: 07.06.2012)



Professor in charge (sign)

B Trying Out the Prototype

There are two methods available for trying out the prototype, of which the online demo is the easiest. However, should one wish to try out different data sets or other such changes, then one must run the demo locally.

B.1 Online Demonstration

The prototype application has been set up to run on a server belonging to the Department of Geomatics, where it is accessible to the general public. The address to visit it is <http://geomatikk.hopto.org:8080/static/index.html>, where the user is presented with a index page showing all the available demonstrations and tests.

The author can not guarantee the uptime of the server, as it may be subject to service interruptions and outages outside of his control. (This being a master thesis, the author will be leaving the university by the middle of June 2012.) However, if the server is not responding you can contact the author and he will attempt to rectify the situation to the best of his abilities.

B.2 Running the Prototype Locally

In order to run the prototype you must first obtain the program code either from the electronic attachment to this report, or from the public source code repository at <https://github.com/robpvn/Vector-Tile-Research>. The prototype has been set up to be run on a Unix-like system, but should in theory be possible to set up on a Windows system with a bit more effort. (This has not been tested.) In addition there are a number of dependencies that must be installed on your computer for Tilestache to work. These are detailed in the Tilestache readme, and are repeated here for simplicity.

pip Python module installer Installed through a method detailed in the Tilestache readme, used for installing other modules.

ModestMaps A Slippy Map library, installed through pip.

Python Imaging Library (PIL) A helper library for image operations, installed through pip.

Werkzeug A compact application server for running the prototype, installed through pip.

Mapnik A raster tile rendering engine, installable following instructions from <http://mapnik.org/download>

GDAL with Python bindings Used for data access and manipulation in the vector tile provider, installable following instructions from <http://trac.osgeo.org/gdal/wiki/DownloadingGdalBinaries>

In order to run the prototype, navigate to the root directory of the prototype and run this command:

```
PYTHONPATH="[The full path to the root directory]" \  
./scripts/tilestache-server.py
```

Then you should be able to point your web browser to <http://127.0.0.1:8080/static/index.html> and access the same index page as in the online demonstration.

B.2.1 Trying the Experimental Union

If you wish to try out the experimental union mode using jigsaw code, you must have the git source control program installed and then run the following command in the root folder of the prototype:

```
git checkout improved_union_jigsawcode
```

Then you can start the server as usual. In order to change back to the normal pseudo-union, run this command:

```
git checkout master
```

C Electronic Attachments

This is a list of files that are included in the electronic attachment to the report, which is available from NTNU's DAIM system along with the PDF version of the report.

"Prototype Source Code" folder This is the complete source code for the prototype, with contributions by the author as described in appendices D and E. See appendix B for more information on how to use the prototype.

"Timing Results" folder This folder contains one subfolder for each test variation, which each contain raw text files with test result data as well as spreadsheets used for organising the data and extracting statistics.

"R Calculations" folder This folder contains the data sources and scripts used to calculate the statistical distribution fitting that was used to determine the kind of distribution the test data followed.

"Plots" folder This folder contains the data sources and scripts used to generate the histogram plots used in the report.

"Report Source" folder This folder contains the \LaTeX source files used to generate the report.

D Tilestache Modifications

The Tilestache set-up is, as mentioned in section 5.1, basically an unaltered Tilestache set-up with some extensions added in. The following is a list of exactly which files have been changed, with addresses relative to the root folder of the prototype.

The "static" folder This folder, and all of its subfolders, have been added in order to integrate test applications and demonstrations with the Werkzeug web server. In this way all of the demo applications could be served from the same server application as the tiles.

scripts/tilestache-server.py The script that starts the Werkzeug server, modified to allow serving of files from the "static" folder.

TileStache/Goodies/Providers/EdgePointerGeoJSON.py & Arc.py These extension files have been added in order to provide the edge pointer tile functionality.

tilestache.cfg The Tilestache configuration file, changed to include the example data providers used in testing.

E Polymaps Modifications

As outlined in section 5.2, Polymaps was not as easily modifiable as Tilestache. Therefore, changes had to be made to the core code of Polymaps in addition to the inclusion of accompanying code. The following list shows the changes made to Polymaps and new files that were included. In the prototype, all of these files are located in the "static" folder.

polymaps_modified.js This is the original Polymaps, with the necessary changes for signalling integrated. All changes are marked in the source code with "RPVN".

common_functions.js This file contains all the common functions used by every example to manage things like triggering concatenation, and is tightly coupled to polymaps_modified.

worldtiles.js (in every subfolder) Every example has its own file containing the actual concatenation algorithm, which is unique to each type of solution.

tester.js This file contains the code for driving all the tests, which each individual test calls into.

raphael-min.js RaphaelJS is an external library used by common_functions to simplify some SVG manipulations.

jigsawpuzzle-rhill-3.js (When the experimental union features are checked out) This is the external code used for jigsaw puzzle solving, adapted to polygon union.

F Test Computer Specifications

These are the specifications of the computer used in the tests described in section 7.2. It is the author's personal computer, a five year old Dell D830 laptop computer with some upgraded components. The disk storage and Random Access memory (RAM) are competitive with a modern computer, but the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU) are not. This means that caching performance is completely comparable to a modern computer, but tile generation, concatenation and drawing might not be. As mentioned in section 6, one must only consider the relative performance of algorithms against each other, not the absolute execution time.

CPU Intel Core 2 Duo T7100 @ 1.80GHz

RAM 4 GB DDR-RAM

GPU Nvidia Quadro NVS 140M, 512 MB integrated video memory

Storage Intel SSD 320 Series, 120GB

Operating System 64-bit OpenSUSE Linux, version 11.4

G Licences

G.1 Application

The prototype is based on, and makes use of, a number of different open source projects. These licences are listed here, and can be viewed in their entirety at the referenced web sites.

Tilestache 3-Clause BSD Licence[66]

Polymaps 3-Clause BSD Licence[66]

RaphaelJS MIT Licence[66]

Jigsawpuzzle-rhill Creative Commons Attribution-Noncommercial-Share Alike 2.5 Canada License[97] The software is free to use for non-commercial projects, provided a link is provided to the author's website. (<http://www.raymondhill.net/>)

All changes and additions beyond those listed are the author's own work. They are hereby licenced under the 3-Clause BSD licence, in the same fashion as Tilestache and Polymaps.

G.2 Project Report

Unless otherwise agreed (as with the delivery of the report to the university for evaluation), the report is licenced under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported Licence, as available from <https://creativecommons.org/licenses/by-nc-sa/3.0/>. The work can be shared and built upon, but only non-commercially and with attribution like this:

Robert Nordan

rpvn@robpvn.net

<http://www.robpvn.net>