



NTNU – Trondheim
Norwegian University of
Science and Technology

The future of web-based maps: can vector tiles and HTML5 solve the need for high-performance delivery of maps on the web?

Mats Taraldsvik

Master of Science in Engineering and ICT

Submission date: June 2012

Supervisor: Terje Midtbø, BAT

Co-supervisor: Dr. Ing. Rune Aasgaard, Norkart Geoservice AS

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



Report Title: The future of web-based maps: can vector tiles and HTML5 solve the need for high-performance delivery of maps on the web?	Date: 10.06.2012		
	Number of pages (incl. appendices): 136		
	Master Thesis	X	Project Work
Name: Mats Taraldsvik			
Professor in charge/supervisor: Terje Midtbø			
Other external professional contacts/supervisors: Dr. Ing. Rune Aasgaard, Norkart Geoservice AS			

Abstract:

The majority of the current maps available on the web with a standard browser are raster-based, which impose a couple of limitations with regard to functionality as well as performance. Most work is done on the server, and the raster image tiles that are rendered on the client can not be interacted with directly. Some solutions are available that solve this partially, but in the majority of cases, they depend on technology that is not standardised, and rely on third-party extensions that are only available on certain platforms.

Creating map tiles and implementing efficient caching are crucial in high availability web maps, whether they are raster-based or vector-based. Due to the different storage models of raster images and vector data structures, there are also techniques that differ in their application (such as compression, client rendering), and the increased exposure of raw information with vector data needs attention. The data formats chosen for representing spatial data in vector-based maps, have lots of implications for the efficiency and usability of the map application itself.

The accelerated development and standardisation of the open web -- namely HTML5 -- are giving developers better tools to meet the functionality and performance requirements for vector maps on the web, without resorting to third-party software that is not supported across platforms. Previously, with the technology available, creating a functional, efficient vector map on the web was hard or impossible using only open technology, but with the recent advancements, it is interesting to see to what extent this can be accomplished.

By developing an implementation of a vector map client and server, with multiple vector data structures in both binary and text formats for measuring efficiency between server and client, as well as assessing the impact of techniques such as generalisation, tiling and caching, the potential for future vector-based maps on the web have been analysed. Testing revealed interesting results, which suggest that it is possible to achieve performance with vector based maps on the web that either matches or exceeds the current raster based maps.

Keywords:

1. efficient, web-based maps
2. HTML5, open standards
3. vector tiles, vector data
4. binary and text data formats

Abstract

The majority of the current maps available on the web with a standard browser are raster-based, which impose a couple of limitations with regard to functionality as well as performance. Most work is done on the server, and the raster image tiles that are rendered on the client can not be interacted with directly. Some solutions are available that solve this partially, but in the majority of cases, they depend on technology that is not standardised, and rely on third-party extensions that are only available on certain platforms.

Creating *map tiles* and implementing efficient *caching* are crucial in high availability web maps, whether they are raster-based or vector-based. Due to the different storage models of raster images and vector data structures, there are also techniques that differ in their application (such as compression, client rendering and generalisation), and the increased exposure of raw information with vector data needs attention. The data formats chosen for representing spatial data in vector-based maps, have lots of implications for the efficiency and usability of the map application itself.

The accelerated development and standardisation of the open web – namely HTML5 – are giving developers better tools to meet the functionality and performance requirements for vector maps on the web, without resorting to third-party software that is not supported across platforms. Previously, with the technology available, creating a functional, efficient vector map on the web was hard or impossible using only open technology, but with the recent advancements, it is interesting to see to what extent this can be accomplished.

By developing an implementation of a vector map client and server, with multiple vector data structures in both binary and text formats for measuring efficiency between server and client, as well as assessing the impact of techniques such as generalisation, tiling and caching, the potential for future vector-based maps on the web have been analysed. Testing revealed interesting results, which suggest that it is possible to achieve performance with vector-based maps on the web that either matches or exceeds the current raster-based maps.

Sammendrag

Det store flertallet av dagens kart på internett er basert på rasterteknologi, noe som setter begrensninger på hva som kan oppnås, både når det gjelder funksjonalitet og ytelse. Med rasterbaserte kart gjøres det meste av arbeidet på tjeneren, og produktet av denne prosessen er kartfliser (tiles), der det ikke er mulig å ha direkte interaksjon med de underliggende geometriske objektene. Det finnes hybridløsninger som bøter på problemet, men få eller ingen som bruker åpen, standardisert, plattformuavhengig teknologi, og som ikke avhenger av tredjeparts programvare.

Et av de store problemene med kart på web er skalering – tjenesten må takle et stort antall samtidige brukere. Mellomlagring av data og kartfliser (tiles), er helt nødvendige teknikker som må brukes dersom systemet skal skalere, uavhengig om underliggende data er på vektorformat eller rasterformat. Forskjellene mellom kart på rasterformat og vektorformat er tidvis store, med vidt forskjellige muligheter for optimalisering mot høyere ytelse. Måten vektorene blir representert har avgjørende betydning for størrelsen og ytelsen, og det er viktig å undersøke hva som lønner seg.

Generalisering har vært viktig for kartografien lenge, og er en essensiell teknikk for å skape et godt kart. Tradisjonelt har generalisering nesten utelukkende blitt brukt til å endre utseendet på kartet, men med digitale data påvirker det også størrelsen på dataene. Avhengig av hvordan dataene representeres, vil både datastørrelsen og dermed hvor raskt selve dataene kan prosesseres kunne komprimeres i stor grad. Mens rasterdata i noen grad påvirkes av generalisering, vil generalisering ha stor betydning for romlige data på vektorform.

Utviklingen av åpne standarder for bruk i nettapplikasjoner har i lengre tid hengt etter den funksjonaliteten som ønskes, og dette har begrenset mulighetene for å utvikle plattformuavhengige nettapplikasjoner uten bruk av tredjeparts programvare. Dette er i ferd med å endre seg, og med HTML5 har teknologi som bedre tilfredsstillende kravene til funksjonalitet og ytelse, samtidig som det kun avhenger av åpne standarder. Det er derfor først nå det er mulig å lage effektive og funksjonelle vektorbaserte kart på nett, med høy tilgjengelighet og plattformstøtte.

Oppgaven presenterer en implementasjon av vektorkart på klient- og tjenersiden, der det er lagt spesiell vekt på dataformater i tekst- og binærformat som brukes til kommunikasjon, og hvordan valg av dataformat påvirker ytelsen. I tillegg vurderes betydningen av generalisering, oppdeling av kartet i fliser (tiles), samt mellomlagring. Implementasjonen testes grundig, og resultatene som presenteres, viser at det vil være mulig å lage kart på nett basert på vektordata, som yter tilsvarende eller bedre enn dagens rasterbaserte kart.

Preface

This paper, *The future of web-based maps: can vector tiles and HTML5 solve the need for high-performance delivery of maps on the web?*, and the accompanying software, data for tests and results (see Section 1.1), is the result of the master thesis assignment in the course TBA4925 at the division of Geomatics at the Norwegian University of Science and Technology (NTNU), with a timespan constrained to twenty weeks in the spring of 2012.

My primary advisers during the project, which I would like to thank especially for their valuable contributions, patience and assistance, were Terje Midtbø at the division of Geomatics and Rune Aasgaard at Norkart Geoservice AS.

I would also like to thank Gunstein Vatnar, Harald Jansson and Sverre Wisløff at Norkart Geoservice AS and Alexander Nossun at the division of Geomatics for valuable feedback.

June 10, 2012
Mats Taraldsvik

License

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License[1]. You may share and distribute this work freely under the same or similar license, only if you attribute the work to the author, who is Mats Taraldsvik.

Contents

1. Introduction	6
1.1. Source code for the examples	6
I. Delivering maps on the web	7
2. History and Motivation	7
3. Tiling	9
4. Caching	10
5. Generalisation	12
6. Raster-based maps	13
6.1. Pre-generation of tiles	14
6.2. Client load	14
7. Vector-based maps	15
7.1. Progressive vector transmission	15
7.2. Compression	18
7.3. Real-time styling	21
7.4. Client load	21
7.5. Vector tiles	22
7.6. Security	23
II. How new technology will improve the web map experience	24
8. What is HTML5?	24
8.1. Background	24
8.2. Motivation	24
8.3. JavaScript	25
9. Native support for inline SVG	26
10. Binary processing with TypedArray	27
11. Replacing HTTP with Web Sockets	29
12. Implementing non-blocking behaviour with Web Workers	30
13. Data formats	30
13.1. Considerations when choosing a data format	31

13.2. Geography Markup Language	32
13.3. GeoJSON	34
13.4. BSON	34
13.5. WKB and WKT	35
13.6. ESRI Shapefile	36
13.7. Non-standard formats	38
14. Optimisation	38
III. Performance comparison of new and existing web map solutions	39
15. What is Performance?	39
16. Existing maps using mature technology	41
16.1. The current map standard	41
16.2. Reference web map experience	42
17. A vector map using modern HTML5 technologies	45
17.1. Server Architecture	45
17.1.1. PostGIS	45
17.1.2. Python and Shapely	47
17.1.3. Web Sockets	47
17.2. Data Transmission	48
17.2.1. Binary data	48
17.2.2. Text data	49
17.2.3. Data Formats	50
17.3. Client Architecture	51
17.3.1. SVG	51
17.3.2. Choosing a Library	52
17.3.3. User interaction	52
17.3.4. Native application comparisons	54
17.4. Implementation performance	55
17.4.1. Data size	56
17.4.2. Storage	60
17.4.3. Latency	63
IV. Conclusion	73
18. Future work	73
Appendices	76

A. Attachment 1: Master Thesis Assignment	77
B. The implementation and source code accompanying this paper	81
C. HTML5 Support in Web Browsers	83
D. Data Formats Example	84
E. Large versions of the visualisations of the performance test results	85
List of figures	109
List of tables	114
List of code examples	115
References	116

1. Introduction

The current web maps are mostly based on raster images, which makes interaction hard or impossible. Functionality such as custom styling can not be implemented to scale, because all processing is done on the server, and the amount of variables directly affects the speed of the map implementation. This is why we have started exploring vector map solutions for use on the web.

The goal in this paper is to examine new, open, standardised web technology, and whether this technology is mature and efficient enough to handle vector data equally or better than the current raster images, making headway for more interaction and functionality (such as individual styling and direct queries on features). The goal is also to explore how generalisation, tiling and caching affect the end performance and efficiency of a vector web map.

1.1. Source code for the examples

The implementations and examples that were developed by the author for this paper is publicly available on the collaboration site GitHub[2]. The URL to the source code repository, is <http://www.github.com/meastp/efficientvectortiles/>

Part I.

Delivering maps on the web

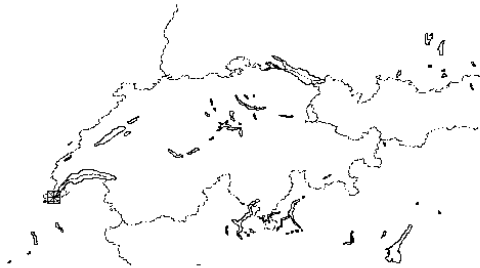
2. History and Motivation

The first map server – Xerox PARC Map Viewer – was released as early as 1993, just four years after the World Wide Web was invented as a document exchange system at CERN[3]. It allowed the use of interactive maps on the web by linking GIF raster image files with CGI and Perl[4]. This sparked creativity in the community, and The World Wide Earthquake Locator was made, based on the Xerox PARC Map Viewer[5]. Following this, major progress was made with the creation of new services, such as an online atlas (The Atlas of Canada[6]), and a geographical database with interactive mapping (The Gazetteer for Scotland[7])[8].

Example Showing a Map of Switzerland

The links on this page connect directly to the Map Viewer at Xerox PARC.
(border=1/grid=0/ht=2.81/lat=46.92/lon=8.34/mark=46.26.1/wd=5.62).

Map Viewer: world 46.92N 8.34E (64.1X)



Select a point on the map to zoom in (by 2), or select an option below. Please read [About the Map Viewer](#), [FAQ](#) and [Details](#).

Options:

- Zoom In: ([2](#)), ([5](#)), ([10](#)), ([25](#)); Zoom Out: ([1/2](#)), ([1/5](#)), ([1/10](#)), ([1/25](#))
- Features: [Default](#), [All](#), [-borders](#), [+rivers](#)
- Display: [color](#); Projection: [elliptical](#), [rectangular](#), [sinusoidal](#); [Narrow](#), [Square](#)
- Change Database to [USA only \(more detail\)](#)
- [Hide Map Image](#), [No Zoom on Select](#), [Reset All Options](#)

Options can also be typed in as search keywords (e.g. "lon=-100", see [details](#)). Current region is 5.62 deg. wide by 2.81 deg. (193.89 miles) high.

Preset Coordinates:

- [Globe](#), [USA](#), [Alaska](#), [Hawaii](#), [San Francisco Bay](#), [United Kingdom](#)

Figure 1: The Xerox PARC Map Viewer with a map of Switzerland

Most of the web map solutions were proprietary, which meant that the vendors had their own implementation that they did not share. The main problem, however, was the lack of open standards for web mapping. Without open standards, it is very hard to collaborate across different implementations, to be able to consume or use parts of a competitors technology, or for users to migrate data between two incompatible implementations. The Open

Geospatial Consortium (OGC)[9], an international voluntary standards organisation, was created in 1994 to promote open solutions and standards in the Geographic Information Systems (GIS) domain[10]. The organisation became involved with web mapping in 1997, after Allan Doyle published a paper describing a "WWW Mapping Framework"[11], and started a task force to outline a strategy.

The next major milestone – at least for everyday use – was in 1996, when MapQuest emerged with the first popular online map service with address lookup and a routing service[12]. MapQuest was aimed at consumers, and especially the turn-by-turn driving directions which it provided, became very popular since it was the only one offering these services (some dictionaries even list mapquest as a verb[13], similar to "just google it" today). Several new mapping services aimed at consumers emerged following MapQuest's success, and GIS companies such as ESRI[14] and MapInfo[15] released server software, acknowledging the potential of the internet for online mapping[16].

The maps were mostly static, and greater performance and interactivity was achieved by employing technologies such as Dynamic HTML, Java and ActiveX. However, when Google Maps launched in 2005, they used asynchronous JavaScript and XML (Ajax) – a new technology – to create a "slippy map"[17] which resembled desktop applications more closely than previous attempts. This familiarity might partly explain why Google Maps became so popular, but the ease of creating so-called "mashups" – a mixture of custom gathered data on top of Google Maps – was probably more important because it helped to create possibilities in the GIS field for people who were not GIS professionals[18].

The work of the Open Geospatial Consortium have also had major impact on the usefulness of today's different map solutions. The first version of the Web Map Service (WMS)[19], which is an open standard specification for collecting data from a spatial database, and serving the resulting raster maps over the web to multiple clients, was released in 2000. Today, lots of maps employ this technology, which means that it is easy to consume the maps without becoming reliant on a specific vendor. This standard has also made interoperability between vendors easier, and users have a better experience.

The amount of online maps have grown tremendously since MapQuest, and today there are lots of web maps to choose from (Google Maps[20], OpenStreetMap[21], Bing Maps[22], Yahoo! Maps[23], MapQuest[12]), that have become a natural part of everyday life for the average consumer because the web makes them more available. In fact, maps are no longer exclusive to large map sites – since integrating maps have become such a simple task, lots of sites where it makes sense to have a map, has one. Examples are Walmart's Store Finder[24] and FixMyStreet[25]. Location-based services and the demand for maps on the web have grown, and are still growing, and continued improvement of maps on the web is more important than ever before[26][27][28].

3. Tiling

A map contains a lot of information, and delivering this information all at once through the internet is either not possible due to bandwidth restrictions, or unnecessary, since the viewer may only be interested in a subset of the map. Tiling solves both of these issues by dividing the map into smaller sections, and deliver these sections as they are needed, instead of transferring the whole map at once[29][30].

Online map services capable of zooming employ tiling recursively (i.e. each tile is further divided at larger scales). The division is usually done in a quad tree-like manner, i.e each tile is divided into four tiles when the zoom level is increased (and the other way when zoom level is decreased)[31].

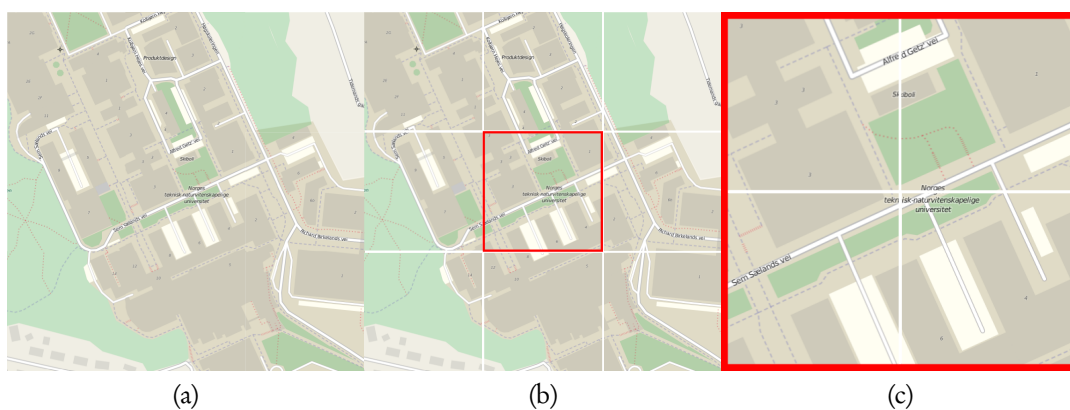


Figure 2: Tiling works by dividing the map (a) into tiles (b) that are loaded independently on demand. For maps that support zooming, the tiling process is done for every map scale (c).

In practice, tiling is an optimisation at the cost of flexibility. Although it is a technique that can be used to render tiles at custom scales, on-demand, it is rarely done – tiling is used to enable delivery of maps in a scaling manner, when there are a high number of users. Also, as long as the server has to regenerate common parts of the map (i.e tiles) on every user’s request, the limit on the number of users the server can handle, remains fairly low. Therefore, tiles are very often used with caching (see Section 4) to create an efficient web map.

A Web Map Service (WMS)[19] – an Open Geospatial Consortium (OGC) standard – is highly configurable and flexible for the user, and the server processes each user’s request, with all custom parameters, and responds with an image tailored for the user.

Because a Web Map Service does not impose restrictions on the user, for example in terms of map scale and style, a new image needs to be generated on every user request, which is every time a user changes the scale, style or another available parameter. Essentially, this also

means that there is no way to limit the generation of images on the server, and the result is a (disappointing) limit on the number of users the server can handle[32].

The Open Geospatial Consortium (OGC) have since complemented the Web Map Service with another standard; the Web Map Tile Service[33] is designed especially to scale when there are lots of users, and – as the name suggests – it uses tiles to accomplish this. The tile map has a predefined set of scales, which reduces the possible number of tiles from almost indefinite, to an amount of tiles – and resulting data size – which modern servers can handle[34].

Modern maps where the user is able to pan and zoom, are called ”slippy maps“[17]. Since the user can – in theory – view all map tiles during a single session, and transferring all tiles at once is not feasible (it would both be too slow, and consume too much storage on the client), the tiles are loaded in an on-demand process using JavaScript and AJAX[35].

4. Caching

To generate tiles (see Section 3) suitable for delivery to a client, they need some form of processing, which puts a certain demand on the server, and, subsequently, increased delay for the user of the map. For map services that responds to thousands – and even millions[36][37] – of requests per day, the performance hit of processing every request directly would be significant, and the server therefore use a cache in which to store generated tiles, after they are first requested. The client often use a cache to pre-fetch tiles that are close in distance to the tiles that are shown on the user’s screen. As long as the user behaves according to this pattern, the result is a smoother experience.

The idea of a cache is to reduce access time on frequently used data. The cache is typically a lot smaller in size than the resource which is cached, because a small subset of the data is typically accessed a lot more often than the rest (also known as the principle of locality), or that an accessed resource’s neighbours (in e.g. time or distance) are likely to be requested (and therefore should be cached). Unfortunately, the subset of map tiles that are accessed most frequently in map services, tend to be large, while the time a user is willing to wait for a tile is typically low and the storage cost to cache all frequently used tiles is too high to be viable[35][38].

The amount of processing power that is saved by using a cache, varies greatly, and depends on the amount of requests (i.e users) the service has at any time, how expensive the generation of tiles is, the amount of cache storage available, whether some tiles are more frequently accessed than others and what the requirements are with regards to recency (i.e. how often the cache needs to be updated)[39].

The generation of map tiles is either a complex process, or a simple one – this depends greatly on the requirements of the resulting map. Depending on the data format in which the spatial data is stored, a conversion from vector to raster, or raster to vector, needs to be performed. To avoid making the map overly complex at small scales, different degrees of

generalisation might be required as well, which further increases the latency if the tiles were to be delivered in "real-time". The obvious downside of a cache is that the cached resource is static, and thus not appropriate for map services where the tile(s) need to be updated frequently, since in that case it would be cheaper to generate the tiles on demand instead, avoiding the latency associated with traversing the cache [40].

A simple optimisation technique for a raster tile cache when there are multiple identical tiles is to maintain references to tiles and store it once (see Figure 3). This optimisation is for example done with the MBTiles data format (which is essentially a SQLite database with tiles stored as binary blobs)[41].

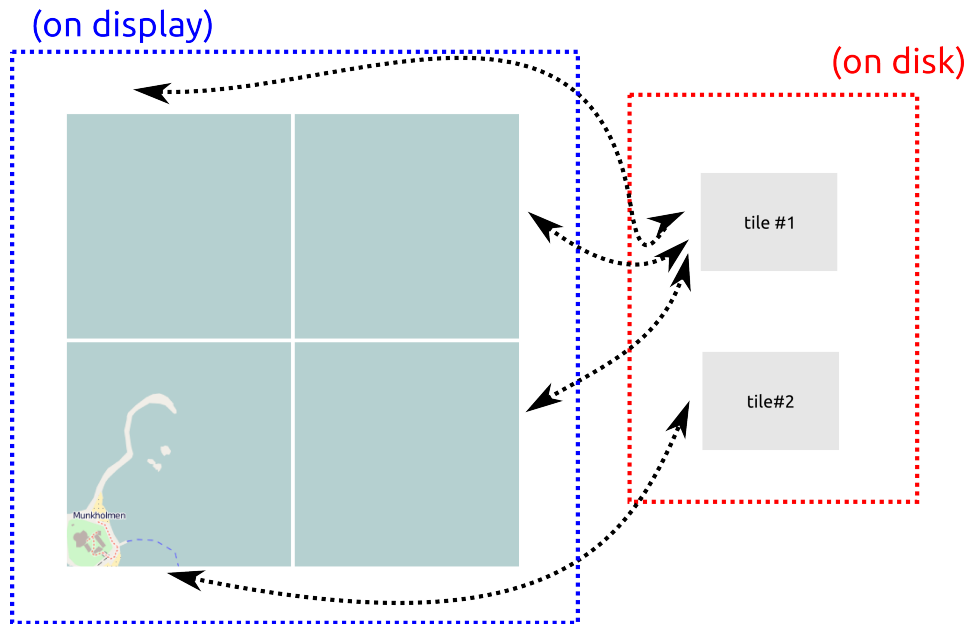


Figure 3: Identical tiles are stored once on disk, and referenced when displayed, to avoid redundant data and to save space. For water tiles, which there are many of in some datasets, this technique can save a lot of disk space.

It is also possible to cache tiles or map data on the client, which would shorten the latency and decrease the wait time even more for the user[42]. However, the allowed cache size on the client is often more limited than the server (if it is not, one can use e.g. MBTiles locally), and can not keep every unique tile in the cache at once. This is a common problem, and there are multiple cache eviction algorithms developed to solve this. To be able to choose between the algorithms, the usage pattern of the map needs to be examined.

A common cache eviction algorithm is LRU (least recently used), which removes items based on the last time they were accessed, but there are lots of alternative policies for removing items, like LFU (least frequently used) (removes items that are most rarely accessed), a queue (FIFO) (remove the oldest retrieved items to make room for new items), a stack (LIFO) (remove the last retrieved items to make room for new items) or even random removal of items[43, 44, 45].

5. Generalisation

”Our human and natural environments are complex and full of detail. Maps work by strategically reducing detail and grouping phenomena together. Driven by your intent, maps emphasise and enhance a few aspects of our world and de-emphasise everything else.“[46]

Map generalisation was relevant even before there were digital maps, and then as a manual process, where cartographers needed to modify maps when they were published in different scales, to ensure that the map’s information were as consumable and usable as possible, rather than completely accurate[47].

The motivation is simple: too high accuracy leads to more complexity, and may not increase the usability of the map, which is the end goal, and should be pursued. In other words, generalisation is not merely a way to make a less accurate map, but a means for the cartographer decrease detail to better communicate the information encoded in the map[48]. In that regard, the lesser detail might – and ideally should – convey overall patterns and trends otherwise hidden, while with too much detail, the cartographic message may drown[49].

With regards to the goal of the best possible usability and readability of the map, the transition to digital maps brings nothing new. However, the need to formalise and specialise the methods to make them suitable for automation emerged, and much work has been done in this field to date. Early research in automated map generalisation focused on individual map objects, which resulted in a large number of operations on individual points, lines and areas, and the view that a solution should only perform manipulations on single objects at a time[50].

Since map generalisation has a lot to do with behaviours and interactions between objects – this ”layered view“, which focused on objects individually in isolation, resulted in ”computer assisted cartography“[51] rather than a fully automated solution. As soon as the value of a feature’s context on the map was realised, and the major flaw of a ”layered view“ acknowledged, researchers tried to capture the context of objects in their map generalisation algorithms. Examples are automatic displacement of point and line symbols[52][53], using Delaunay triangulation for constraint-based generalisation (see Figure 4)[54] and solving complex decisions regarding displacement using Artificial Intelligence[55].

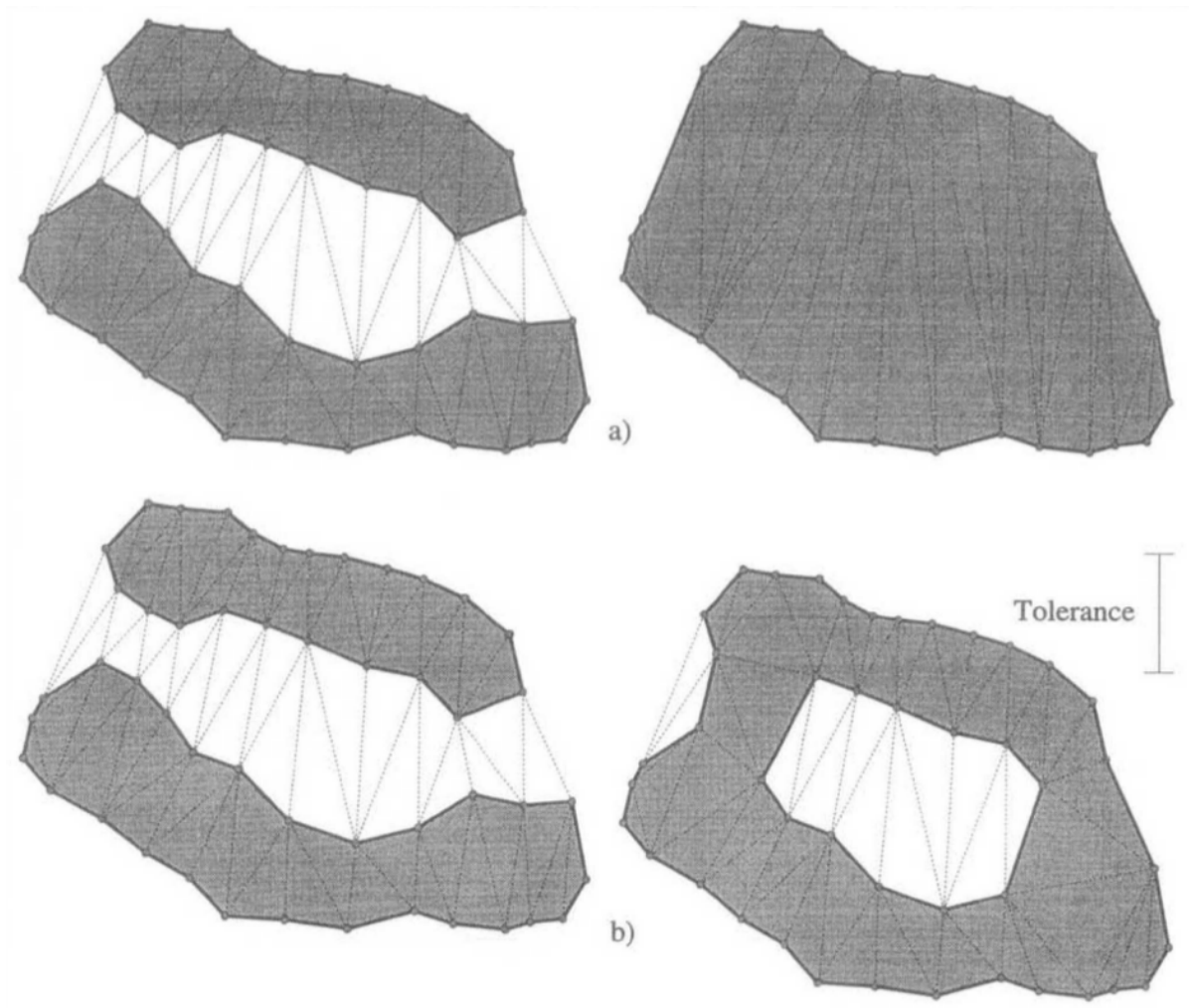


Figure 4: Usage of Delaunay triangulation to merge two areas, with a specified triangle size tolerance (b) – and without this constraint (a). [54]

A prominent challenge when delivering maps on the web, is to perform the actual transfer of data as efficient as possible. By generalising the map data, the result is a more compact and simpler data set, depending on the grade of generalisation, which may improve efficiency of the data transfer as a positive side effect.

When transferring vector data sets, various techniques can be used in combination with map generalisation to improve efficiency as well (see Section 7.1).

6. Raster-based maps

Raster-based maps are essentially just images, and before the transition to digital map production, they were scans of paper maps. Images have a long history on the web, and they

are therefore both supported by all browsers, and hence predictable and easy to work with. The concepts of tiling and caching (see Section 3 and 4) fits rather well with raster maps, and is not very hard to implement, which is probably why raster-based maps have been used throughout mainstream web mapping services since maps became popular on the web.

However, what makes a raster map simple – that it is static – is also a weakness. Interactivity is important in modern map services, and since a raster map is a static image, this is an expensive task to solve, since the image will need to be updated for every interaction to simulate feedback to the user (most implementations add an independent layer on top of the raster image for placing markers, routing etc. – this is not part of the raster map)[56][57].

Furthermore, overlapping lines are a major problem with raster images[58]. If there are multiple, overlapping lines or rectangles in an area, that area becomes blurred in a raster image, which makes it difficult to read accurately.

6.1. Pre-generation of tiles

A full-quality raster image, where as much detail as possible is retained from the capture device, consumes a lot of disk space. For purposes such as transferring data between a server and one or several clients over the internet, the data consumption of a full-quality raster image is too large. Hence, the raster images are compressed, preferably with a lossless compression algorithm, if it is able to compress the image enough. If a lossy compression algorithm is chosen, the result is loss of information, lower quality and lower resolution raster images[59].

Raster tiles are expensive to generate, especially when the source data is in vector format, and too time consuming for on demand rendering[60]. Therefore, it is common to pre-render all tiles in a predefined style, and store them in a disk cache. This leads to obvious constraints on the use of raster tiles, since the generation of tiles is too resource heavy for on-demand service, and on-demand generation is the only way for raster tiles or images to provide (native) interactivity.

6.2. Client load

Raster maps do not require additional processing on the client. The native support in web browsers also means that no plugin(s) are required for the map to work – sites that require third-party plugins often suffer from a significant abandonment rate[61][62]. The light client is one of the strongest points of the raster map, especially historically, where clients were not very powerful.

Most of the processing work, like generating the raster map (and creating tiles), is done on the server prior to the client's request. Raster maps are therefore especially efficient if the client does not have powerful hardware (which might be the case in mobile devices).

7. Vector-based maps

The most important difference between a vector object and a raster object, is that the vector is stored as geometrical coordinates defining points, lines and polygons (object-based), while the raster object is limited to the images' resolution (field-based). If we know the projection, and have the coordinates, vector data gives us the possibility to re-project the geometry on the client, which is not possible with static raster images. Each vector object is thus only dependent on the precision – and amount of detail – of its own coordinates, and having objects with variable precision will result in consequent reduction in data size – which is not the case with raster objects where the image resolution defines the data set's precision.

Since vectors are objects, and can be tied to events and properties easily, they are very suitable for user interaction and direct manipulation. Work has been done to provide vector maps on the web, often using SVG and GML for visualisation and data transfer, respectively[59]. However, a major issue has been the requirement of third-party plugins to render the data, or verbose vector data structures, that results in excessive bandwidth consumption.

7.1. Progressive vector transmission

A major hurdle when working with vector data, is the resulting size of the data. With raster data, an image is only as large as its resolution and the number of pixels, but a vector data structure can provide almost an infinite amount of detail, even if the details are too tiny to be rendered on the screen, resulting in an unnecessarily large data footprint.

With raster data, the generalisation process affects the appearance of the map, and the size of the resulting image file. However, the size of a raster image is not always affected as much as a vector image, and a re-transfer of the whole image is required, instead of incremental progressive transfer (see Figure 5). The most important role of cartographic generalisation has always been to more effectively communicate the message of the map – and this is still extremely important.



Figure 5: Progressive transmission with raster images works by starting with a coarse image (a), and gradually transfer images with higher resolution which are replaced with the coarser image (b), finally resulting in a high-resolution, full-detail image (c).

When using vector data, however, generalisation significantly affects the size of the data sent to the client as well, giving it a major role in the quest of providing vector maps that are as efficient as existing, raster-based maps – which the user expects for solutions that provide the same functionality.

To solve the challenge of data with almost unlimited detail – and thus an equally large data footprint – the amount of detail delivered to the client is dependent on the map scale. In other words, the vector data needs to go through generalisation at multiple stages, and create maps appropriate for different scales, striking a balance between data consumption, detail and the data (and message) presented to the user. Although the data size is smaller and will perform better at small scales, the amount of data is still high when the map is viewed at larger scales[63].

The idea of quickly providing a coarse model of the map, and then iteratively improving it by requesting more detailed information, has been subject to research for years, and is not unique to vector data. Raster data, for example, also gain from this technique, by progressively enhancing the image[64], and by using compression[65].

With progressive transmission using vector data, most early research was done with Triangulated Irregular Networks[66, 67, 68], which is not directly applicable to web maps. The complexity of maintaining topology and generalising vector data is high, which may explain why there has not been much research on this topic until quite recently. Additionally, the requirements depend on display size, as well as system and network performance – there is a large difference between desktop computers and tablets, for example. Whether the data is subject to further analysis, or just for rapid visualisation also affects the process[69].

The main principle of progressive vector transmission (see also Figure 6) is to initially provide a coarse map, and then dynamically reconstruct the map to appropriate detail, based on the map scale, in the background (i.e. without freezing the interface)[70]. This pattern is

especially well made for a typical user that begins with the smallest scale possible, gradually zooming to the desired location. The client is then given more time to iteratively construct the map, saving time when rendering at the largest map scales.

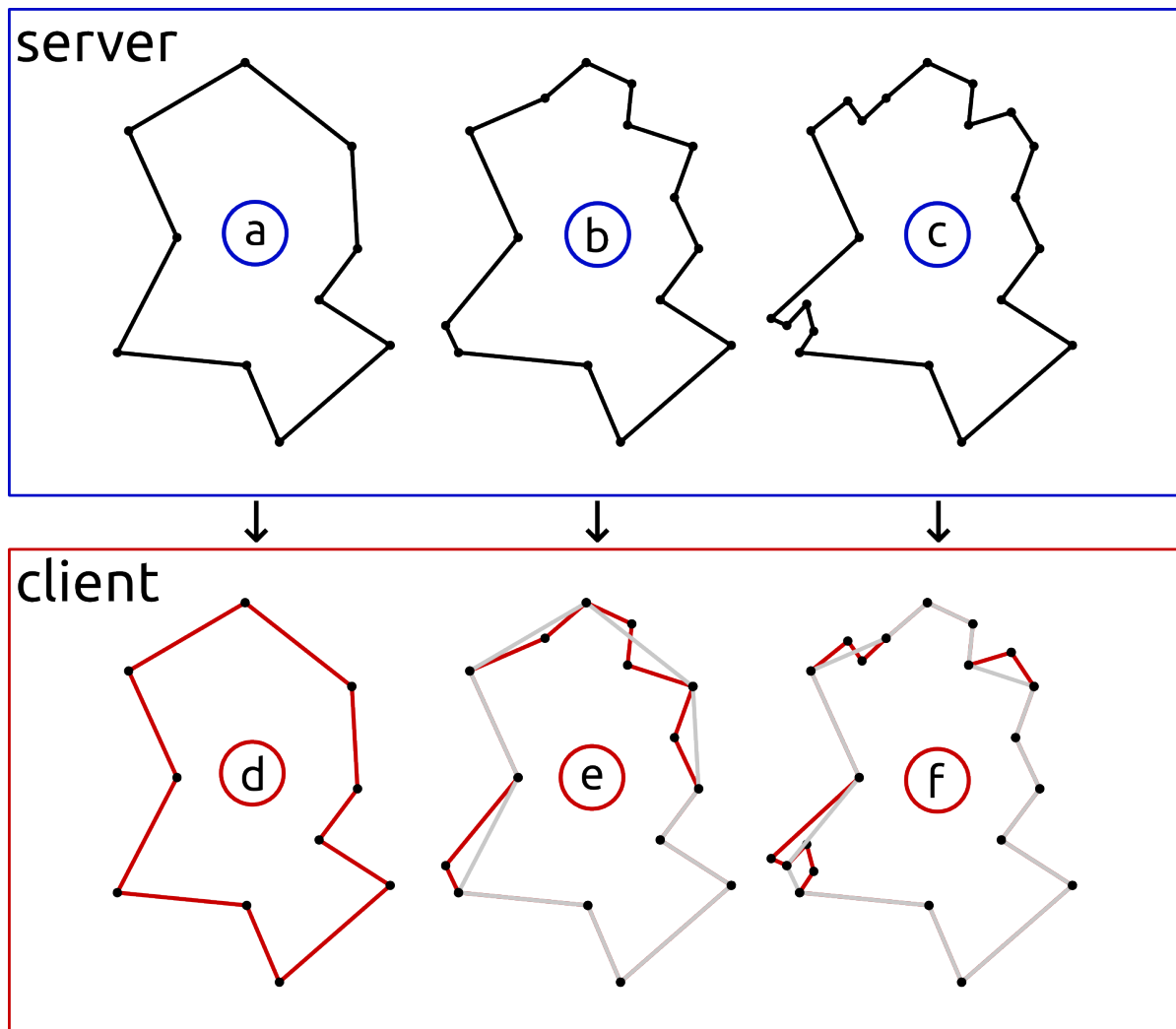


Figure 6: On the server, the original object(s) of full detail is generalised into multiple incremental models, from coarse to full detail ((a), (b) and (c)). The initial object(s), which are coarse and provide limited detail (a), are sent to the client (d). Either automatically, or as the user increase the map scale, the object(s) are upgraded in a number of steps (b), transferring only additional points that increase detail (e), until the original detailed object (c) is transferred to the client (f), providing equal amount of detail on both server and client.

There has been some research on the subject of progressive vector transmission on individual components, such as the Douglas-Peucker algorithm[71], the "bendsimplification" algorithm[72] and the Visvalingam algorithm[73]. Unfortunately, these do not take the ob-

ject's context into the equation, which is important to maintain topology and preserve the meaning of the map even at coarse detail levels.

To solve this, an implementation was made, which used ordering and area of all objects to keep the context and maintain the shape characteristics and topology of the map objects. By recording the coordinate sequences in an object, the overhead when adding more detail is lowered[69].

Even though the research is not extensive, when the number of vector web maps grows, the author predicts an increase in this effort, as progressive vector transmission will be a key factor to delivering acceptably performing vector web maps, together with generalisation these methods decide the data size, processing cost, and, in the end, user's satisfaction with vector maps. Vector maps' developers should have a solid interest in this scientific field and its progression.

7.2. Compression

Data compression reduces the data consumption of a file, object or stream, and is widely used in a number of domains[74]. The algorithm that is used to compress and decompress is either lossless, or lossy. A lossless algorithm does not lose information (i.e. the original object can be reconstructed fully), while a lossy algorithm will have some information loss in the sense that the reconstructed object is not identical.

Compression of spatial data structures has been researched for both raster and vector data[75], and this has been important given the limited resources available for serving maps to computers (and people) across the world – and the rise of mobile devices that can not handle complex, full-scale data.

Generalisation is a process to simplify the geometry, where the primary goal is to remove features that are unnecessary (despite correct) for the reader to understand the map. Cartographers have used generalisation for decades – even before maps were processed digitally – and it is an established and thoroughly researched field. Since features are removed and simplified, generalisation is a form of lossy compression, useful in reducing the data size (see Figure 7).

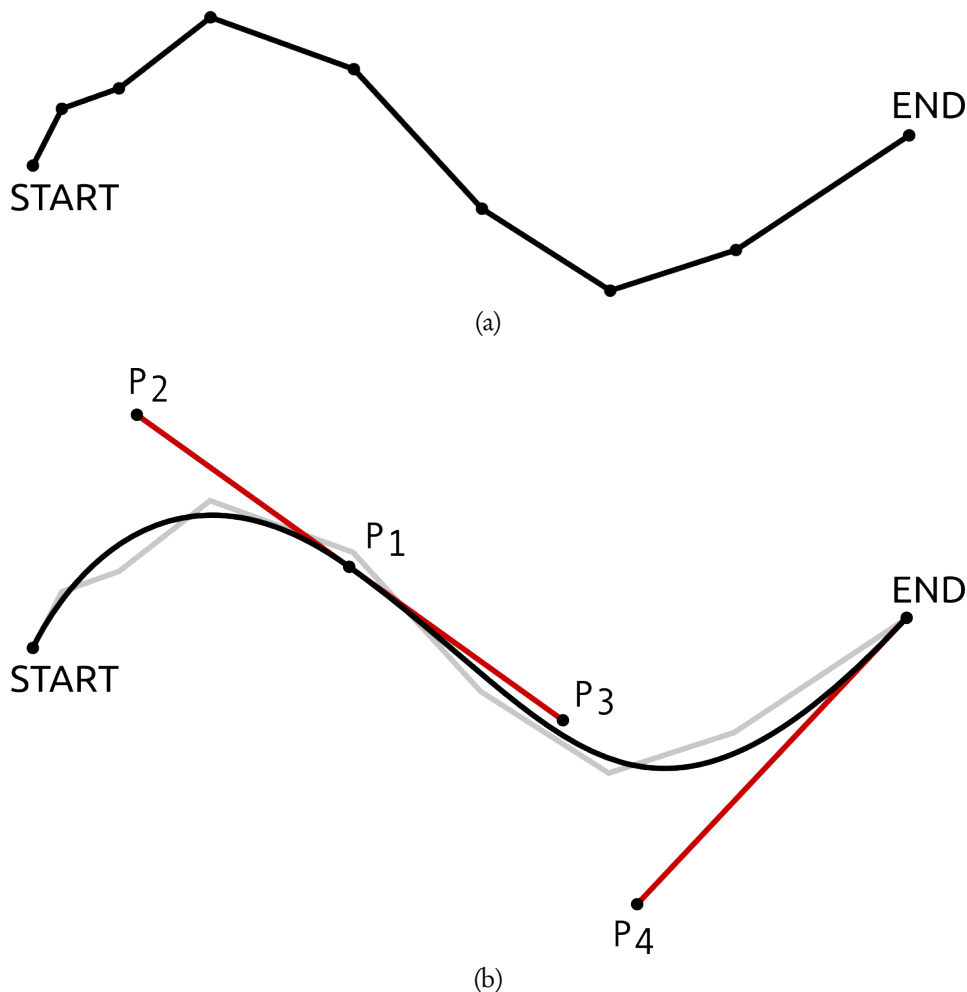


Figure 7: Example of a line (a) and the line converted to a Bézier curve (b). The points in (a) are compressed to a smooth curve (b), which consists of start, middle (P1) and end points, with three control points (P2, P3 and P4) (the red lines are not visible, only for illustrating the control points' influence on the curve). Depending on the data structure, the Bézier curve could take less space than the original line.

For example, a line with 20 points can be approximated with a mathematical function, resulting in less space consumption at the cost of accuracy (i.e. lossy compression). Bézier curves, like the B-spline can be used to create a lossy representation of lines[76, 77].

Another way of compressing data with loss, is data type conversion. A coordinate might have more precision than it needs – for example, the display contains a finite amount of pixels and the usefulness of overly precise numbers is therefore limited – and the type can therefore be converted to a less precise type, with a smaller data footprint.

Lossless data compression is a way to decrease the size of our data, without changing the geometry or topology of the spatial data structure[78]. This problem is not as domain-specific as lossy compression, since one need to be able to reconstruct the exact file when decom-

pressing. Lossless compression algorithms reduce repeating patterns in the data, either text or binary, and the goal is to end up with a compressed file that is smaller than the original.

Dictionary coders and entropy encoding (see Figure 8) are the most common lossless compression techniques[79]. The former identifies identical sequences, stores one copy of every sequence in a dictionary, and replaces the original sequences with references to the dictionary's copy. Byte pair encoding is an example of this, where each byte that is not used, is assigned to the most common two (and three, four if there are enough unused bytes) byte sequence. There are also multiple variants of the Lempel-Ziv algorithms (LZ77, LZ78, LZW, LZMA ...), which starts in a predetermined state that changes during the coding process, and reuses the already encoded information while encoding.

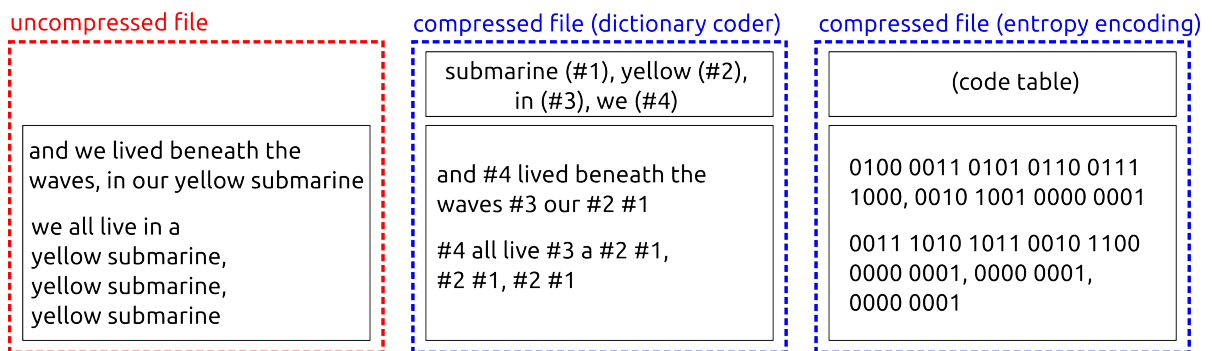


Figure 8: Examples of file compression with a dictionary coder and entropy encoding.

The most common use of entropy encoding is to assign a unique fixed-length code to each symbol in the input, and then replace each code, giving the symbol that most often occur the shortest code (see Table 1). Examples of algorithms that utilise entropy encoding are arithmetic coding and Huffman coding[80].

word	frequency	code
yellow	4	0000
submarine	4	0001
in	2	0010
we	2	0011
and	1	0100
lived	1	0101
beneath	1	0110
the	1	0111
waves	1	1000
our	1	1001
all	1	1010
live	1	1011
a	1	1100

Table 1: The resulting code table for the entropy coding in Figure 8, using 4 bit codes to represent the symbols.

Deflate is a lossless compression algorithm that combines Huffman coding and LZ77 to obtain reduced data size[81]. The gzip algorithm[82], which is based on deflate, is supported as a way to transfer compressed files on the web. By using the HTTP 1.1 header *Content-Encoding*, browser clients may send and receive compressed HTTP requests and responses, respectively, resulting in notably smaller file sizes and bandwidth savings[83].

7.3. Real-time styling

Handing the responsibility of rendering and drawing vector data to the client opens many possibilities with regards to styling and interaction. The client may for example receive a style sheet from the server that dictates the rendering of the features – and once received – the client may change the styling according to own preferences and needs, much like HTML and Cascading Style Sheet (CSS)[84] works in web browsers today.

The big difference worth emphasising, is that this style change can happen without any data exchange to the server, since the whole process is client controlled. Since the styling is decoupled from the spatial data, redundancy is reduced since the server does not have to maintain multiple versions of a data set. For raster maps this is completely different – every style needs to be rendered and kept on the server ready for delivery to client requests.

7.4. Client load

In contrast to raster maps, where tiles are processed on the server and rendered on the client, handling vector tiles offloads some of the processing responsibility to the client. The geometry is transferred as vector data, and needs to be processed and drawn correctly on a surface by the client. This is more complicated and resource consuming than rendering an image (due partly to the maturity of image renderers) – the data is not delivered in a ready-to-render byte stream and needs to be processed before a vector graphics engine can draw the spatial data to the screen.

There are multiple considerations that needs to be addressed because of this increase in resource consumption. Since every object has its own handle, large amounts of objects might be hard for the client to cope with, since it may require greater than trivial amounts of CPU and/or memory. While this may not be very relevant for modern desktop computers, the mobile device and smart phone market has grown large enough to be very relevant for general development – including web mapping applications with high accessibility requirements.

7.5. Vector tiles

The fundamental idea of tiling is easy to understand – it splits the map into parts that are downloaded and viewed as they are needed, saving space and time for the user, who does not have to download the whole map at once. The map also loads incrementally, which should give the user a feeling of higher performance, because there is something happening continuously.

With vector data, there is an additional advantage with tiles that the traditional raster map tiles do not have. The coordinates are the largest part of most vector data structures, and coordinates are numbers. In native binary format, numbers consume a fixed amount of space, depending on their data type. This means that data type A, which is a two byte data type, can store a greater number than data type B, which is a single byte data type, but A always consume twice as much space as B.

By creating local coordinate systems inside each tile, the coordinates become relative to the tile. Depending on the size of the tile, the coordinates then become small enough to fit in data type A or data type B (see Figure 9). In the end, the optimal size of the tiles and data type depend on the size of the tiles relative to the screen, and how much extra information the user has to download, that goes outside the screens boundary.

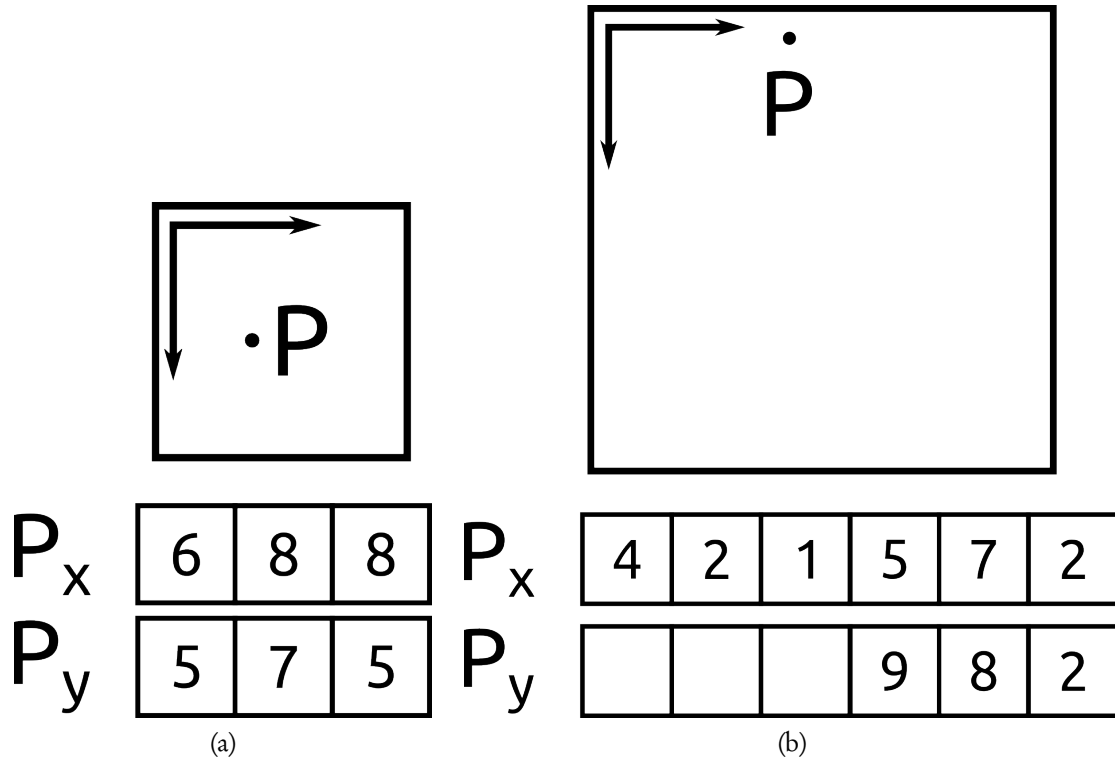


Figure 9: The tile in (b) is twice as large as the tile in (a). The data type in (b) therefore also has to be twice as large as the data type in (a), to be able to store coordinates for the entire tile. Note that P_x in (b) is twice as large as P_x in (a) since the size depends on the data type – not the number stored in it.

7.6. Security

While in some countries, the public have licence-free access to the map data collected by the state map agency, other countries does not have this privilege. The map's underlying data might have licensing that includes restrictions in use, distribution and it might not be free.

With raster maps this is not as problematic, since the underlying vector data is not exposed to the client – only an image is sent from the server – but when the map data's original data structures is sent to the client, which is done to a degree in vector maps, there is always a risk that someone will try to capture that data directly.

Enforcing copy protection by implementing *Digital Rights Management* (DRM)[85], and other schemes, has been tried extensively in the music, movie and computer games industry. It has, however, not proven very successful, and has been inconvenient for customers[86].

When the data is in the hands of the user, there will always be a way to get to it. Raster data has a very high grade of information loss, because the information is translated to pixels, while with vector data, the objects are retained, which might make the information easier

to extract. It might therefore be better to modify the data that is sent to the client, instead of creating third-party plugins for copy protection (and thus decreasing wide platform support).

Part II.

How new technology will improve the web map experience

8. What is HTML5?

8.1. Background

HTML was invented by Tim Berners-Lee in 1989 as a way for scientists to exchange data electronically, to overcome issues with globally distributed science projects and the difficulty of cooperating across great geographic distances[87].

The last major update to the HTML standard – done by the Web Hypertext Application Technology Working Group (WHATWG)[88] and the World Wide Web Consortium (W3C)[3] – was tagged HTML5[89], and is a major effort to update the open web and make it a viable platform for development of modern applications[90].

8.2. Motivation

HTML was designed for exchanging simple documents, and for many years, that is what the internet was – a collection of "pages". However, a web site today often behaves more like a traditional desktop application, with interaction, animation and dynamic behaviour. As a consequence, since HTML did not have the technology to support such scenarios, various vendor-specific and third party extensions, such as Adobe Flash[91], Microsoft ActiveX[92] and Microsoft Silverlight[93] were developed and used.

The original intent of the web was collaboration, in which it is important to maintain compatibility and open standards, to be able to collaborate at all. In this context, where the web represented a common ground of compatibility between different platforms and systems, the introduction of third-party extensions was unfortunate. HTML5 is an investment in closing the gap between the previous HTML4 standard, and the technology demand of modern web applications, with open standards and open technology, creating a common platform for every system with a modern, standards-compliant web browser.

In the last couple of years, the adoption of various mobile devices has grown tremendously[94],

along with the adoption of alternative operative systems to Microsoft Windows[95], such as Apple OS X[96] and Ubuntu Linux[97], which has resulted in a very heterogeneous computing environment[98]. There are multiple devices and operative systems widely used, which are incompatible at the software (e.g OS X and iOS[99], Linux and Android[100]) or CPU architecture level (e.g. x86[101] and ARM[102]).

One of HTML5's primary motivations and selling points is to unify these platforms through the open web. The prospect of developing an application once, and for a single environment (the web), is highly attractive for companies, because of the huge savings in time and effort compared to developing multiple applications for each device/operative system. Since there are so many parties, however, the support for the individual modules of HTML5 may vary, although many have good support, and can be used today (see Appendix C)

8.3. JavaScript

JavaScript (or ECMAScript, which is its vendor neutral name)[103] is a simple programming language that started as a handy tool to enrich a web page with a few "bits and bobs", and has since become such a fundamental part of the modern web, that it is difficult to manage a web page – let alone a web application – without it. JavaScript is an object-oriented language, but differs in some fundamental areas compared to other languages taught at university and in wide use (C[104], C++[105], Java[106], C#[107], Python[108]), which sets a certain barrier of adoption[109].

JavaScript was invented as early as 1995 by Netscape, and it is the only cross-platform programming language natively supported by all major browsers, thus it managed to establish itself as the de-facto standard early on, and has a lot of headway on other languages. With the HTML5 standard, most APIs are exposed in JavaScript, which means that in order to use the new technology, one has to use JavaScript. The entry barrier for new or existing languages is higher than ever, and even though JavaScript was not designed to be used to create complex applications, it remains as the only supported (client browser) web language.

Most media attention is on the HTML5 specification, but established technology like JavaScript is just as important. HTML5 itself does not cover the GIS domain, for example, and it is not intended to. HTML5 standardises the most common technology in an open way, and one must rely on JavaScript to implement the rest.

The high reliance on JavaScript for any web application, translates into a certain demand for performance. Traditionally, JavaScript has not been a very efficient language because it did not have to – it was not designed to handle large, resource-hungry tasks. Today, however, that requirement has changed, and every browser vendor has put great efforts into optimisation of its JavaScript engine, to the extent that JavaScript is not too far behind its traditional desktop counterparts, such as C++, C# and Java [110, 111, 112, 113, 114, 115].

9. Native support for inline SVG

Scalable Vector Graphics (SVG) is a quite mature XML data format for two-dimensional graphics [116]. However, it has not yet been adopted on a large scale, partly due to the lack of native support across all major browsers. This has changed with HTML5, which also introduces inline SVG, where SVG objects are integrated into the HTML Document Object Model (DOM) – which contains all the HTML elements – instead of keeping the SVG structure in a separate file [117].

Integration of the SVG objects in the HTML DOM is an important step for easier web application development, and is an additional advantage compared to external, plugin-based objects. All objects in the HTML DOM can interact with each other, which makes it possible to create the application menu as HTML elements, and the rest of the application as SVG, for example. This is not possible with external, plugin-based objects (such as Adobe Flash, Microsoft Silverlight), because the HTML elements have no way of accessing the information contained in the proprietary object.

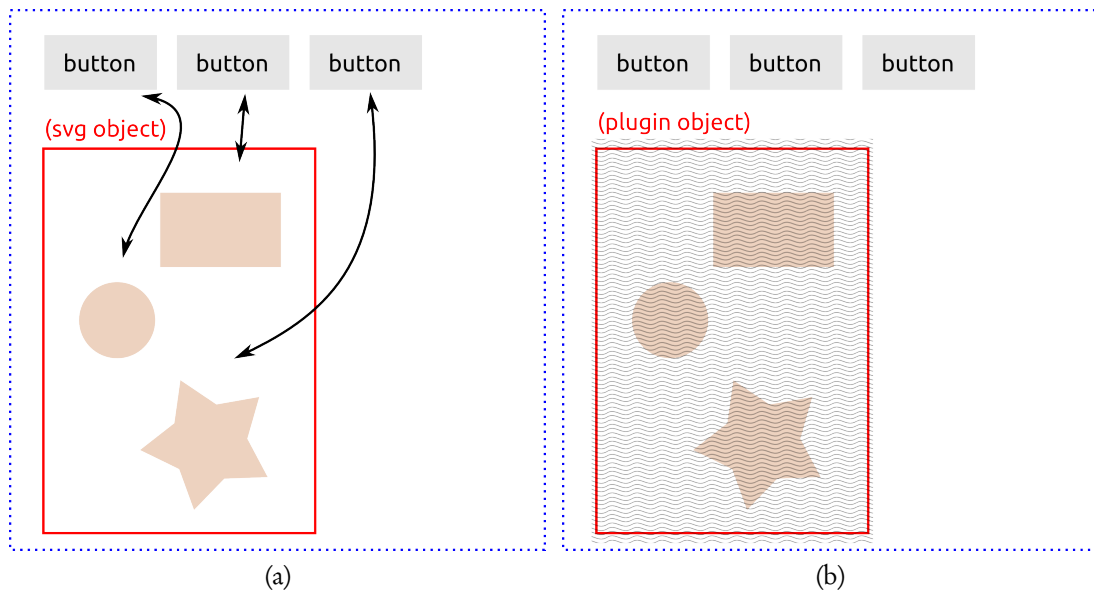


Figure 10: By using SVG, the HTML elements can integrate with individual objects in the SVG element (a), and an application can be created by combining the technologies. This is not possible with proprietary plugins, like Adobe Flash, where the HTML elements can not “see” inside the external object (b).

Since lots of map data is stored in a spatial database in a vector format, the potential processing requirements for conversion between two vector formats, is smaller than a conversion from vector to raster.

Data conversion introduces data redundancy and consistency problems (often due to differences in precision), as well as a non-trivial performance cost and is best avoided if possible.

SVG is an XML data format, and can therefore be hard to work with directly (see Code Example 1). The SVG element is very mature outside of a web context, and multiple vector graphics suites such as Inkscape and Adobe Illustrator exists for creating and editing SVG objects.

Code Example 1: SVG (see Appendix D for complete file contents)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<svg (...) >

  (...)

  <g id="layer1">
    <path
      style="fill:none;stroke:#000000;stroke-width:1px"
      d="M 590053.5096435546875 -6645619.33984375 l
        uuuuuuuuuuuu (...) -35.0400390625 -7.08984375 z"
      id="pol1" />

    (...)

    <path
      style="fill:none;stroke:#FF0000;stroke-width:1px"
      d="M 589990.1103515625 -6645544.4296875 l
        uuuuuuuuuuuu (...) 3.23046875 65.0205078125 "
      id="linestr2" />
  </g>
</svg>
```

XML is a very verbose data format, and might not always be suited for transferring large amounts of data over the web. Libraries solve this, since only vector instructions are needed.

10. Binary processing with TypedArray

It has been difficult to implement very demanding JavaScript applications, because there was no way to work with binary data (see Section 8.3). Media (images, video and audio) are often standardised in binary formats, and for real-time communication, binary formats are used to exchange messages at the rates necessary to be able to deliver the expected performance.[118]

The HTML5 canvas element, being an image, can work with binary data, and has been used as a workaround for interfacing with binary data. When WebGL, a technology built with the canvas element that exposes an OpenGL API, was invented, the demand for manipulation of graphics, and its underlying binary data directly grew even further.

The Typed Array specification for ECMAScript defines a limited interface for working directly with binary data [119]. It defines an ArrayBuffer type, which is essentially raw binary data, and multiple TypedArray and DataView types, for working with ArrayBuffers (see Code Example 2).

Code Example 2: TypedArray Example

```
// TypedArrays, size in bytes per item
//(U means unsigned, i.e. only positive values)

// Int8Array,    1
// Uint8Array,   1
// Int16Array,   2
// Uint16Array,  2
// Int32Array,   4
// Uint32Array,  4
// Float32Array, 4
// Float64Array, 8

// create an array of single byte values for two items
var singlebytearray = new Uint8Array(2);

singlebytearray[0] = 97; // a
singlebytearray[1] = 98; // b

var string_from_array =
    String.fromCharCode.apply(null, singlebytearray);
// string_from_array : "ab"

var singlebyte = new Uint8Array(singlebytearray.buffer, 0, 1);
// singlebyte : 97

singlebyte = singlebytearray.subarray(1, 2);
// singlebyte : 98

var twobytearray = new Uint16Array(singlebytearray.buffer)
// the buffer property is the raw binary buffer
// and can be used to view the data in different ways
```

A TypedArray is a fixed-length array that makes it possible to read and manipulate an ArrayBuffer's values as an integral or floating point type. Often when working with data structures, using TypedArrays is not ideal, because the data might not be ordered according to type (i.e. heterogeneous collection of data).

The DataView type makes it possible to read binary data as a stream, reading and writing data as a stream. The efficiency of native manipulation of binary data is crucial for e.g. real-time or demanding applications[120]. Unfortunately, at the time of writing, the support for DataView is not good enough to be able to use it reliably (see Appendix C).

11. Replacing HTTP with Web Sockets

One of the absolutely most important parts of creating a vector map, is the transfer of data between the client and the server. Existing tile-based raster maps are reliable and efficient, and users are not expecting to wait very long for the map to render. A map using vector data need to perform as well or better, to be adopted, as users will not care about technical implementation details.

There are multiple ways to handle data transfer between client and server with web applications, but none of them are both efficient and easy to use.[121] The traditional way of transferring data, is through HTTP. HTTP has been part of the web since the beginning, and was never designed to handle transfer of massive amounts of data, and especially not with a tight time constraint. Other methods were developed to avoid the performance penalty of HTTP, but they were never properly standardised, and were implemented using dated technology[122]. There is ongoing work to replace the dated HTTP protocol, to minimise or eliminate the shortcomings, and meet the demands of modern web applications[123, 124, 125]. However, it is not yet a standard, and the proposed reference specification, Google's SPDY[126, 127], has met some resistance. Therefore, the next version of HTTP could be years ahead, and a solution is needed now.

HTML5 introduces Web Sockets[128], and gives applications the ability to provide web developers with a socket-based, native, full-duplex communication channel, that does not have the latency and overhead of prior solutions. Tests show that the technique allows sufficient latency even for real-time web applications – which is very relevant, e.g. for temporal maps[129, 122].

The Web Sockets protocol supports both text and binary formats as means of data transportation, which means that it does not limit the data format one may use to transfer data between the server and the client. For image data and 3D operations, there are benefits with avoiding conversion between text and binary just to transport the data, and this applies to coordinate-heavy spatial data formats as well, and also in a general sense[130]. However, binary data formats are more difficult to work with and debug, so the obtained performance benefit needs to outweigh the longer development time[131].

A problem with persistent connections over the web, is that the connection needs to traverse proxies and firewalls, and prevent closing of the connection by these (routers and firewalls). Web Sockets was developed with this in mind, and is perfectly able to handle proxies and firewalls, and prevent these from interrupting the connection by using a persistent tunnel for connection.[121]

12. Implementing non-blocking behaviour with Web Workers

For complex geometries, the processing and rendering of the map may often be bound by the CPU. Processes that take a non-trivial amount of time to complete, will freeze the whole web application until it has completed. Desktop applications solved this by spawning multiple threads, to avoid freezing the user interface during heavy work.

Web Workers is part of HTML5, and provides similar tools for web application developers. By spawning multiple Workers, CPU-heavy algorithms can run separately from the main process, and communicate through a Messaging API. Like with Web Sockets (see Section 11), Web Workers are allowed to transfer binary data[132].

However, this is still a copy operation, and a disadvantage that could actually hurt performance more than the gain of multiple processes, which is the reason for direct transferring with the HTML5 Transferable objects specification[133, 134]. It is direct transfer that avoids copying data, transferring it directly instead. Currently, this is not supported in all browsers yet (see Appendix C).

13. Data formats

Map applications and geodata-heavy applications consume a lot of data – and are essentially useless without data – so it follows that the data formats that are used between the server and the client, are a very important aspect of the application.

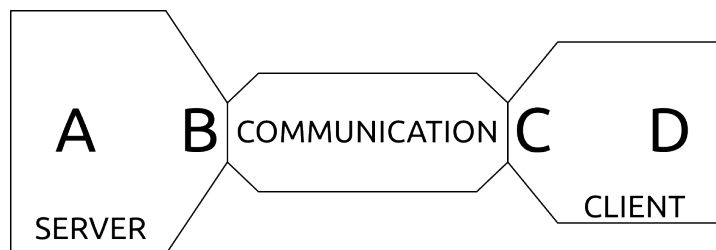


Figure 11: The flow of data in a vector map implementation is moving data from the server's spatial database (A) into a data format that is tiny and fast enough to pass through the limited communication channel (B). The data is received by the client, and, if the data structure does not map to the target representation, it is processed to the desired data structure (C), and rendered (D).

With HTML5 and web applications, the limiting technologies are JavaScript (on the client) and transport protocols between server and client.

The web application has become a "first class citizen" with HTML5, that closes important gaps in technology compared to regular desktop applications.

13.1. Considerations when choosing a data format

Open data formats are popular for very good reasons, such as better interoperability with other software, resistance from vendor lock-in (i.e. being able to migrate the data to other applications in the future), and lots of existing tools and documentation to work with the particular data format. It is therefore natural to employ open data formats where possible.

Depending on the project's main priorities, it is necessary to settle on a data format that is either text or binary. A text format is usually chosen because it is friendly to process and edit by hand, if necessary. A text based data format is typically not as apt to vendor lock-in as a binary format, because a custom parser is easier to implement if the contents can be read directly, but at the same time, the strength of being human readable may introduce a performance penalty compared to a binary data format.

In most cases, the main content of a spatial data structure are point coordinates, as polylines and areas are implemented with points and implicit vertices (i.e. the data structure only contains points in ordered sequences). Coordinates need to be parsed and converted in a text data format, since only strings are stored. A popular text encoding standard is UCS Transformation Format – 8-bit (UTF-8)[135], where each character consumes either one, two, three or four bytes. UTF-8 characters that are part of American Standard Code for Information Interchange (ASCII)[136], consume one byte each, which includes numbers.

A double precision float value, such as the coordinates $x = 584000.12345$ and $y = 6644000.123456$, will then consume (including decimal marks) $8 \text{ bytes} \cdot 12 = 96 \text{ bytes}$ (x) and $8 \text{ bytes} \cdot 14 = 112 \text{ bytes}$ (y), whereas a binary format could represent each double precision float value in 8 bytes, for a total of 16 bytes – a huge saving over the UTF-8 text representation.

With binary data formats, one has extensive control over how the data is stored, and this can be exploited to achieve better performance[137]. When data is stored in memory (i.e. read from permanent storage into memory), the native data type is written to multiple addresses or blocks of a specific size (see Figure 12). If most coordinates are doubles (which consume eight bytes), it makes sense to store the data in eight byte blocks.

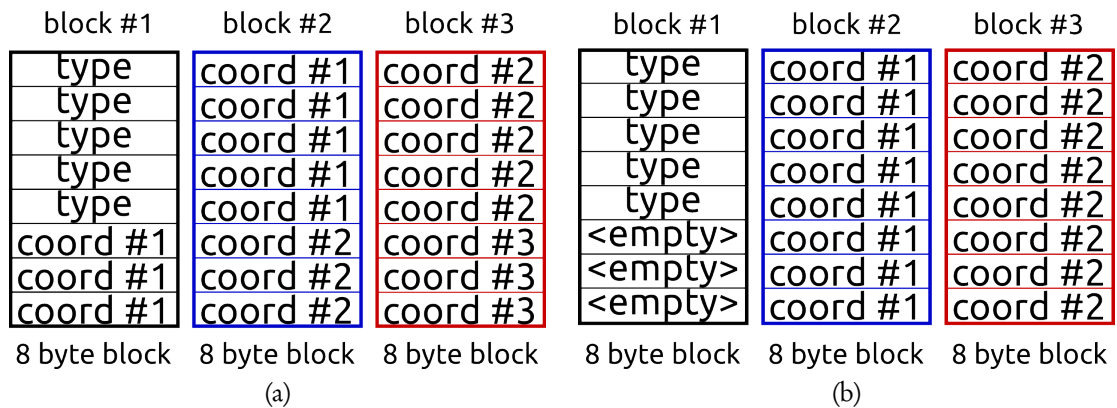


Figure 12: The data in (a) and (b) has a type header, and lots of coordinates consuming 8 bytes each. In (a), the data is misaligned, while in (b), parts of the first block is kept empty to align the coordinates to single blocks.

The point of this is to only access each block once for every coordinate, and extract every double directly from each block. However, if the data is misaligned, one can not do this, but instead need to access the blocks that each coordinate is split up into, and might even have to combine them in a temporary block, before extracting the double value. To avoid this, one needs to use a binary data format, as it can not be done without knowing the storage model of the data format.

The byte-order of data must also be considered when choosing a data format, and especially when that data format is transferred between computers. The difference between the two byte-orders in use is the order that the data is read for multi-byte data types (i.e. data types that consume more than a single byte such as floating point numbers). The data can either be stored with the least significant byte first and most significant byte last – called little-endian – or most significant byte first and least significant byte last – called big endian [138, 139]. Since the big and little endian are different, multi-byte data types will yield different values, so they are not interchangeable. To solve this, one can agree on using the big endian byte-order – which is the standard "network order" – or one can begin every data sequence with a multi-byte number (which will be different in the two byte-orders) to differentiate between big and little endian.

13.2. Geography Markup Language

The most descriptive and feature-rich data format for geometry, and also a popular exchange format for geodata on the web, is the Geography Markup Language [140]. Currently at version 3.3, GML is an XML encoding standard for geographic information, standardised and developed by OpenGIS Consortium [9] – an organisation that develop and maintain publicly available open standards in the geomatics field.

The Geography Markup Language is a very descriptive, text-based data format (see Code Example 3). It is – like XML – intended to be both human-readable and human-editable, and therefore contains a lot of markup and whitespace to ease readability. This greatly reduces the performance of the data format with regard to computer processing, which, in an open web context, is already limited to the performance of the JavaScript programming language.

Code Example 3: GML (see Appendix D for complete file contents)

```
<ogr:FeatureCollection (...)>

  (...)

  <gml:GeometryCollection srsName="EPSG:25832">
    <gml:geometryMember>
      <gml:Polygon>
        <gml:outerBoundaryIs>
          <gml:LinearRing>
            <gml:coordinates>
              589979.040000015171245,6645609.06999983638525,0 (...)
            </gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:Polygon>
    </gml:geometryMember>

    <gml:geometryMember>
      <gml:LineString>
        <gml:coordinates>
          589990.110000015120022,6645544.429999835789204,0 (...)
        </gml:coordinates>
      </gml:LineString>
    </gml:geometryMember>
  </gml:GeometryCollection>

  (...)

</ogr:FeatureCollection>
```

Using GML, a smooth experience might be hard or impossible to achieve, especially on mobile devices and web browsers with limited resources, simply because the parsing of the data format is too time consuming.

GML is made for others to build upon, e.g. by adding national conventions and standards to the namespace. One example of this is CityGML [140], which is an application schema (i.e. a dialect) of GML intended for modelling virtual 3D city models.

13.3. GeoJSON

GeoJSON is a geometry specific JavaScript Object Notation (JSON) format, and as JSON was designed to be easily parsable by Javascript, and more lightweight than XML, GeoJSON is the equivalent opponent to GML for spatial data exchange purposes[141][142].

Like GML, GeoJSON is a text format (see Code Example 4), and suffers from the same disadvantages with machine parsing text data, and markup redundancy. However, the JSON/-GeoJSON markup is easier to parse into native data types than XML/GML, and the data size is a bit smaller[143].

Code Example 4: GeoJSON (see Appendix D for complete file contents)

```
{
  "type": "GeometryCollection",
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::25832"
    }
  },
  "geometries": [
    { "type": "Polygon",
      "coordinates": [[[590053.51,6645619.34], (...)]]
    },
    { "type": "LineString",
      "coordinates": [[590053.51,6645619.34], (...)]
    }
    (...)
  ]
}
```

To further optimise the GeoJSON format for data transfer purposes with minimal data footprint, compression schemes have been developed to minimise redundancy and whitespace. CJSON[144] and json.hpack[145] compress (Geo)JSON strings, resulting in strings that consume significantly less space in common cases[146], although the strings are no longer considered to be human-readable.

13.4. BSON

The BSON specification[147] is the binary counterpart of JSON – on which GeoJSON is based – and stands for Binary JSON. Compared to its text based counterpart, BSON is

designed for efficiency, and the data types are a superset of JSON. BSON was created as part of the MongoDB NoSQL database[148], and use BSON as the storage format for documents – both on-disk storage and network transportation.

BSON is primarily designed to be superior compared to JSON in terms of storage space and parsing speed, however, since arrays have explicit indices, and large objects are prefixed with a length field for scanning performance, BSON will in some cases use more space than JSON[149]. For spatial data structures, the integer and floating point data types are probably the most interesting advantage, compared to regular JSON, since numbers are stored natively.

However, with (1) byte-sized tiles (see Section 3), the BSON specification only has a "raw" binary type without further imposed structure, which frankly will not be different from a custom binary format – nor will it have the advantages of a standard format, since the object is a binary blob anyway. If storing numbers outside the binary type, the forced indices of the array type might create a large space overhead as well.

13.5. WKB and WKT

The Well-known Text (WKT) and Well-known Binary (WKB) are defined as part of the Simple Feature Access open specification[150] by the Open Geospatial Consortium (OGC). While Well-known Text is human readable, and very much appropriate for text-representation of spatial features (see Code Example 5), the Well-Known Binary format's purpose is to be read by computers in an efficient manner, and geometries are therefore represented as a stream of bytes (see Figure 6 for a string representation of the binary data).

Code Example 5: Well-known Text (see Appendix D for complete file contents)

```
"POLYGON ((590053.509643555 6645619.33984375 ,  
          590133.869995117 6645636.04003906 , (...)) )"  
  
"LINESTRING (590053.509765625 6645619.33984375 , (...))"
```

Code Example 6: Well-known Binary (see Appendix D for complete file contents)

(POLYGON)

```
"\x01030000001000000070000000000f00 (...)  
          f004cb0122410000c0d5dc595941"
```

(LINESTRING)

```
"\x01020000007000000000000005cb01 (...)  
          00005cb0122410000c0d5dc595941"
```

A limitation with these formats, compared to GML, is that they only represent geometry objects, and that the Simple Feature Access specification does not include all geometries that are common (for example, it does not include circular lines). A web map with a minimum amount of interaction will need a persistent reference to each geometry object – a feature id – which needs to be linked to the geometry in the data structure. Therefore this information will have to be stored in a different format, or use WKB/WKT as a part of the data structure.

Each point coordinate's size is also specified in the standard to be of the type double precision, which consumes eight bytes per coordinate. When striving to create a small data footprint by for example using local coordinate systems and tiles (see Section 3), this is unfortunate.

The PostGIS spatial extensions[151] for the PostgreSQL database[152] maintains extensions to WKT and WKB (in addition to the OGC standard formats), because the Simple Feature Access specification is limited in the number of geometries it defines. EWKT and EWKT[153] extends WKT and WKB, and provide geometry types that are useful, but not part of the Simple Feature Access specification (such as circular lines), and support for embedding spatial reference systems.

13.6. ESRI Shapefile

ESRI Shapefile (see Figure 13) is a specification that includes multiple file formats to store spatial vector data[154]. It is a partly¹ open specification, and is notable because it is a popular and widespread *binary* data format.

¹Parts of the specification, such as the *Shapefile spatial index format* (suffix sbn), is not documented by ESRI. However, it has been reverse-engineered and documented independently by the open source community

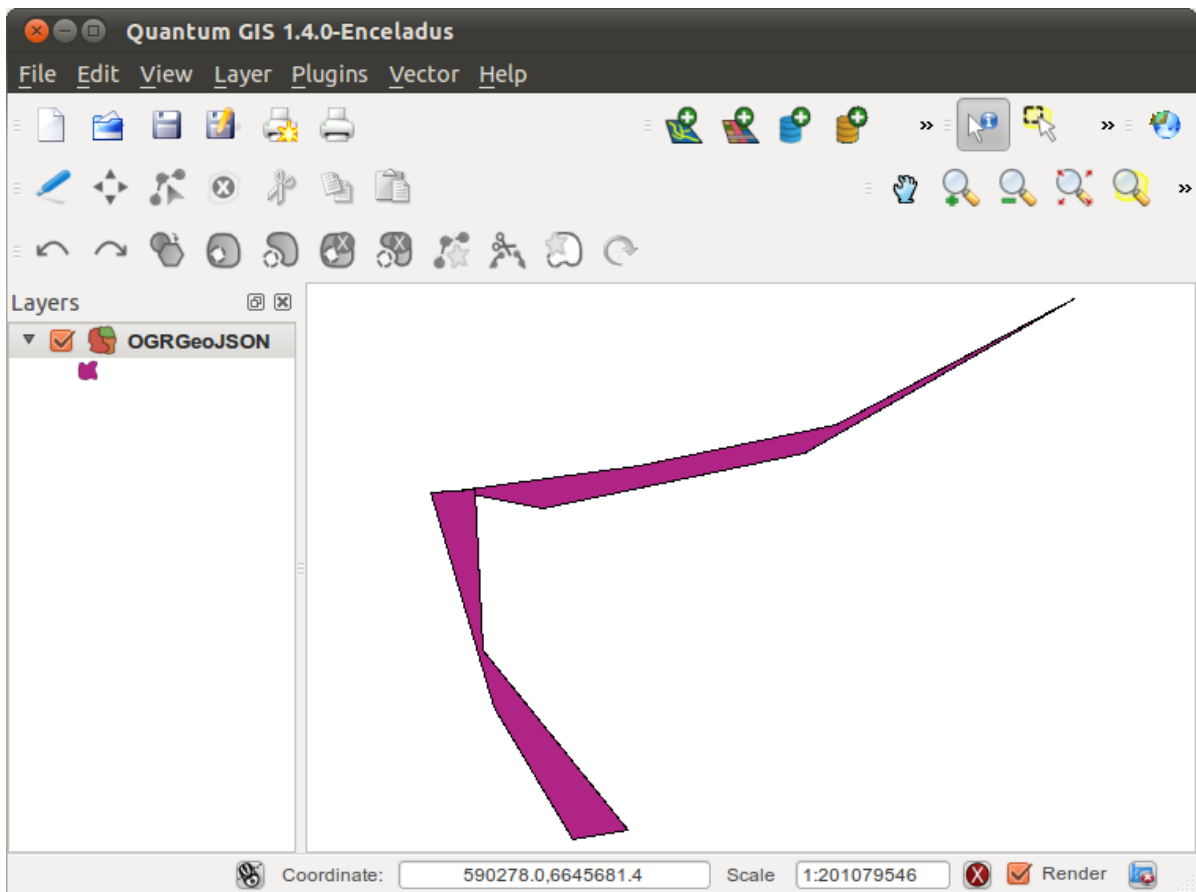


Figure 13: The example shapefile used in Code Example 3 and 4 shown in Quantum GIS. Since it is a binary format, the file contents can not be read directly.

There are 15 file types in total that together comprise the shapefile specification, but only the shape format (shp), which contain the geometry; the shape index format (shx), which is an index to improve performance; and the attribute format database (dbf), which contains each shape's attributes; are mandatory.

The geometry file (shp) is a binary file, and has a fixed, 100 byte header with some meta-data prior to the actual spatial content. Following is a number of records, each with a record header (geometry type, bounding box etc.), and the record's content (coordinates). All coordinates are double precision values, consuming eight bytes per coordinate, like Well-known Binary (WKB).

Shapefiles do not maintain any topology in the data structure. To be able to store topological information, one would need to store pointers geometry and possibly store the geometry multiple times, which would reduce efficiency (with pointers) or increase storage requirements (with redundant data). The specification prohibits storing a mixture of different shapes, even though the data structure makes this possible by having a record header for every record.

13.7. Non-standard formats

There are situations when it is more sensible to invent a new data format, than to use an existing open data format. A common reason to pursue proprietary data formats is often performance related. By tuning the data format to suit its particular purpose, the overhead when processing the data, and also the size of the data, will remain as small as possible.

There could also be a gap between current technology and standardisation work, because standardisation is a very time consuming process, and a standard is built as the technology matures, which is why standards are rarely valid options for emerging technologies. When adapting new technology, a situation may arise, where the only way to consume the new technology is by inventing a data format.

14. Optimisation

More often than not, the standardisation work lags behind what current technology is capable of. While unfortunate, this is natural, as standardisation is both time consuming and hard to do properly. Therefore, there is often a gap between what is available with current standards, and what is possible with current technology.

”First and foremost, we believe that speed is more than a feature. Speed is the most important feature. If your application is slow, people won’t use it. I see this more with mainstream users than I do with power users.

I think that power users sometimes have a bit of sympathetic eye to the challenges of building really fast web applications, and maybe they’re willing to live with it, but when I look at my wife and kids, they’re my mainstream view of the world. If something is slow, they’re just gone.“ *Fred Wilson’s 10 Golden Principles of Successful Web Apps*[155]

Optimisations are likely specific, and expect certain conditions to be true (e.g. CPU architecture, amount of memory, software versions etc.), thus improving part of the application, and possibly reducing compatibility. If this is the case, more complexity is introduced into the application, so one should evaluate if the increased performance in a branch of the application justifies the cost of complexity or reduced compatibility.

Part III.

Performance comparison of new and existing web map solutions

15. What is Performance?

There are multiple parameters that affect the performance of a system, but they are all about timing in some way. Essentially, the system needs to react to an event (user requests, interrupts, messages etc.) in a timely manner[156]. The fact that makes performance so complicated, is that the number of possible origins to trigger the event, and subsequent response patterns, are so numerous.

Formally, the management of resources is central to performance. Any resource that affects the response time, is part of and contributes (positively or negatively) to the end result. Obviously, the amount a certain resource affects performance, depends on the application itself, and its available resources.

A GIS system gets most of its requests from the user – or users, if one is able to operate concurrently – but the requests themselves vary. The user might query for information about an object, and expect the information back as fast as possible; she might edit a feature, expecting the result to be reflected both on screen and in the database (the database might be very important if it is a concurrent multi-user system) or the user might just want to look at the map, and expect a fast response.

For any user, (lack of) performance is exposed through (longer) wait time, which means that a possible metric for performance is the wait time of the user, who, after all, will be using the system. However, when using such an imprecise metric as a user, it is important to be aware that the perceived wait time is sometimes different from the actual wait time (see Figure 14), and there are multiple possibilities to improve the performance without actually making the system faster.

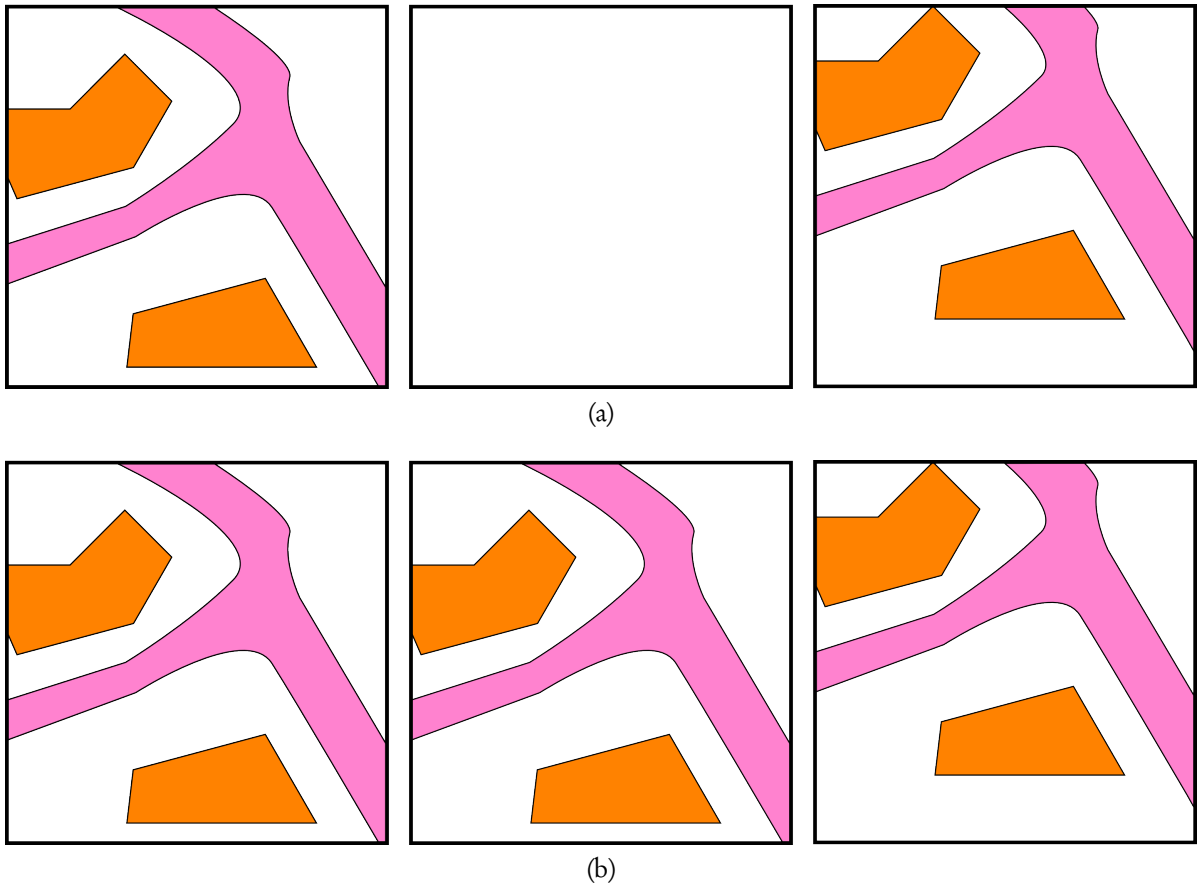


Figure 14: An example where the wait time can be perceived as slower when removing existing data immediately after the request, resulting in a blank map until the new data is completely processed and rendered (a), instead of waiting to replace the existing data until the new data is completely processed and ready to be rendered (b)

To improve performance, one needs to decrease the response time[157]. By improving an algorithm or a process (a), the consumption of a resource – e.g the CPU, GPU or RAM – becomes smaller. The consequence being that some strain on the system can be relaxed – the process could e.g. complete faster, consume less power or free resources to other tasks. Alternatively, every process that depends on (a), and in some scenarios are blocked, waiting for the result of (a), will now have increased performance.

16. Existing maps using mature technology

16.1. The current map standard

The current standard web map, measured by popularity, is composed of raster images, and is often based on cached tiles, to be able to scale to a high number of concurrent users. Mostly, these maps consume the Web Map Service (WMS) or Web Map Tile Service (WMTS) standards, created and maintained by the Open Geospatial Consortium (OGC)[9], enabling cross-compatibility with similar services.

Accounting for a large portion of the increase in use of web maps, is the increased availability of map services that implements standard interfaces such as WMS/WMTS – which makes it easy to aggregate maps from different sources – and the emergence of tools targeted towards non-professional users (i.e. users without a formal degree in GIS or GIS-related subjects) for creating maps from these map services. The demand for fast access to web maps for a lot of users concurrently, has probably caused the increased use of tile-based models (see Section 3).

With de-facto use of WMS and WMTS and other raster-based standards, users have come to expect a certain performance level and especially that maps are quickly delivered and rendered to the display. Moving from raster maps to vector maps, regular users – that neither care, nor understand the reasons for moving from raster to vector-based maps – expect at the very minimum equal or better performance.

The Open Geospatial Consortium also maintains the Web Feature Service (WFS)[158] – a vector data interface based on the Geography Markup Language (see Section 13.2) data format, where interaction is done with the HTTP protocol in a request/response-pattern. The WFS specification has been touted as a solution for providing vector maps on the web, however, the specification has a couple of limitations that limits its performance throughput.

Since WFS uses the HTTP protocol and hence a request/response model[159], and not a more modern technology, retrieval of spatial data is necessarily with higher latency, and a lower bandwidth compared to a more modern approach, such as using HTML5's Web Sockets[122].

The recommended exchange data format – the Geography Markup Language – also contributes negatively to the performance of a vector map solution based on WFS, because of its large footprint (see Section 13.2)[59], the cost of lexically scanning the data (i.e. converting the text to machine-readable tokens) and the overhead of converting text coordinate values (which there are a lot of in a typical spatial data structure)[160]. One is not required to use GML for these reasons, but it will hurt interoperability with other WFS services if one chooses a different format.

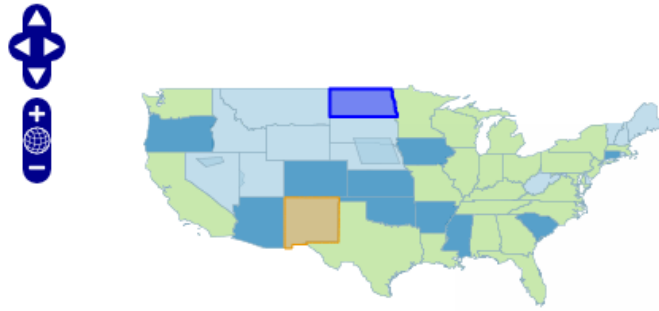


Figure 15: Example of an interactive WFS layer, where one is able to select and query individual features. Unfortunately, each interaction event, which triggers a request to the server, is very noticeable, and might not be acceptable enough in terms of performance for normal use.[161]

There are multiple users of WFS, but they almost exclusively use vector data as a secondary layer[161], with a WMS layer for the majority of the spatial data[162]. The author's opinion is that the lacking adoption of the Web Feature Service specification for complete maps – as opposed to merely being used as a supplement – is based on performance concerns, and that it does not represent the best solution for delivering highly responsive vector maps on the web.

16.2. Reference web map experience

To be able to compare the current user experience – strictly performance-wise – with current raster maps, which a vector map solution would have to match were it to replace today's solutions, a WMS implementation was created and tested.

The PostgreSQL with PostGIS open source spatial database was used to store the spatial data, with the popular open source GeoServer suite[163] providing the WMS interface. GeoServer is the reference implementation of various of Open Geospatial Consortium's standards, including, but not limited to, the Web Map Service. It is written in Java, and is a healthy, community-driven project, and popular enough to provide a sensible performance baseline for comparison with competing web map implementations.

On the client, where the limitations of raster image map solutions result in simple requirements, the open source map library Leaflet[164] was used to consume the (tiled) WMS service and render the map (see Figure 16). Leaflet tries to be as lightweight as possible, and employ responsive map display (i.e. suited for devices with smaller screens), while at the same time being quick and easy to use.

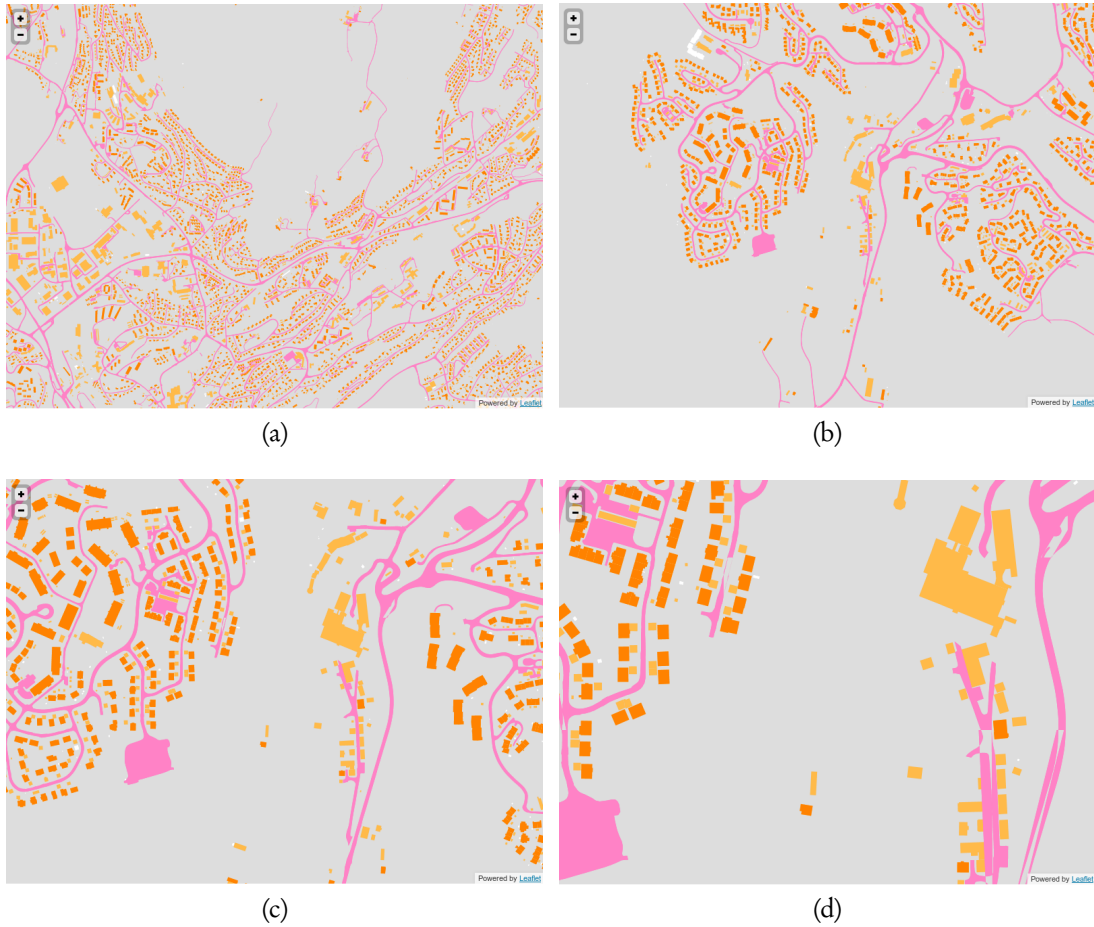
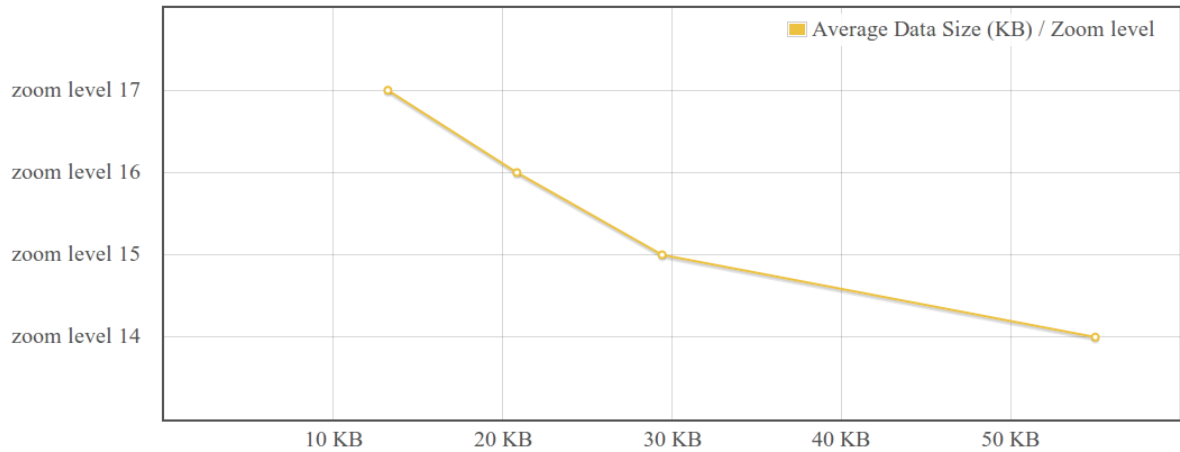


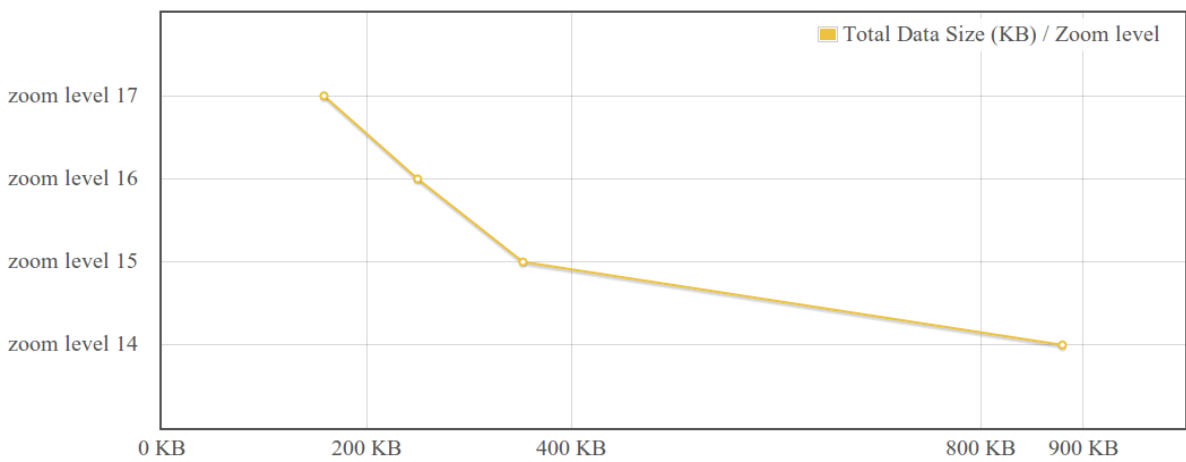
Figure 16: The tile sizes and map loading time were tested at different zoom levels, where the tiles have varying amounts of spatial data.

The implementation acts as a baseline for the current technology, and was tested on average tile size, total data size and total latency for the map. Data was collected through the Google Chromium Inspection/Developer tools interface[165], and written to a JSON data file. It was then processed and plotted with the flot JavaScript framework[166]. The complete source code for the data processing and the resulting graphs are available (see Section 1.1 and Appendix B) for anyone to re-run the tests and verify the end results.

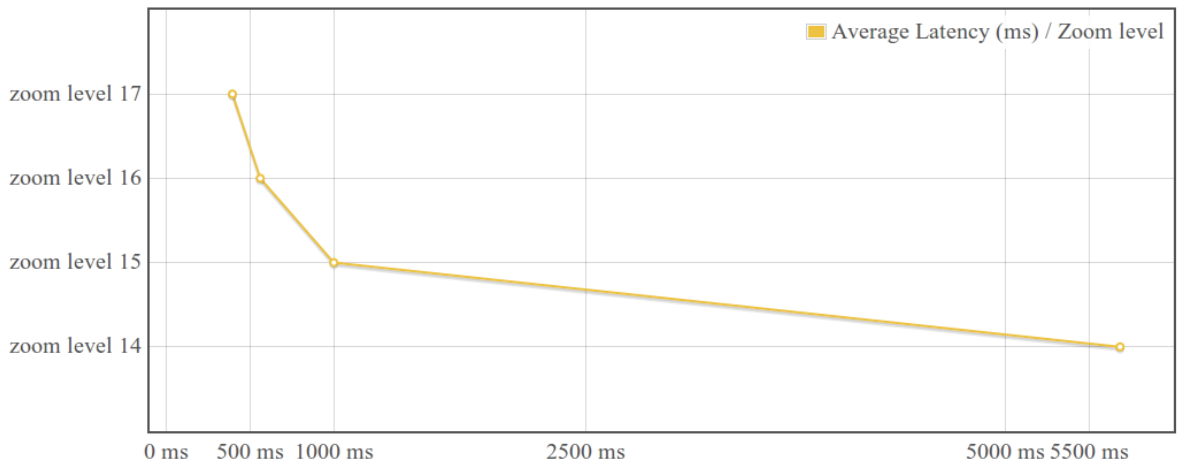
As we can see in the results (see Figure 17), there are large differences in the performance of the map at different zoom levels. All of the tiles (see Section 3) are equal in size (256 pixels x 256 pixels) [167], and there is no size difference in a raw image texture itself what colour the pixel is, so the difference in size is probably due to the compression algorithm used on the image (i.e "empty" pixels or equal colours are compressed (see Section 7.2) to reduce space consumption).



(a)



(b)



(c)

Figure 17: The test results for different zoom levels: (a) the average tile data size, (b) the total data size for the map and (c) the average map latency. (See Appendix E for larger versions of the plots.)

A competing implementation, in this case using vector data, should match or improve on the test results presented here to be considered a viable option in the user space. However, it should be noted that comparing a raster solution with a vector solution is in many ways comparing apples to oranges, since there are a lot of differences in how the solutions work, and how they affect the functionality, client and server (see Section 6 and 7). With that in mind, the comparison is still interesting to get an idea of the consequences when replacing one solution with the other.

17. A vector map using modern HTML5 technologies

In light of recent advancements in modern web technology, it is very interesting to visit the topic of vector maps as a viable way of presenting map data on the web, as vector data has a lot of potential that is either hard or impossible to achieve with existing raster maps.

With vector maps, the client needs to process the vector data, and turn it into an image. This means that there is a lot of potential for optimising data structures and algorithms, much more so than in a raster environment, where no flexibility is given to the client.

The solution presented is an attempt to create a vector map for the web, based on new HTML5 technologies, general improvements in JavaScript and web browsers. Lots of time and work has been put into programming a realistic and flexible architecture to measure the performance of different data formats, and enlighten the different aspects of vector map systems that are different from raster-based map systems.

17.1. Server Architecture

The server is fairly modest compared to a full-scale deployment, a virtual machine running Ubuntu Linux 11.10 with 3GB RAM and two Intel Xeon X5667 (dual core, 3.07 GHz). As the heavy processing is done in the database, the end performance relies mainly on its configuration and performance.

The default configuration of PostGIS is very conservative, and therefore, the theory is that in practice, a very powerful server would only affect the end performance moderately – at least in this implementation with a single or few concurrent users. Hence, for this implementation, and to ease comparison of the results for others, PostGIS' default configuration was kept.

17.1.1. PostGIS

The open source PostgreSQL[152] with PostGIS[151] spatial extensions was used to store and generate spatial data, based on its performance[168], its availability – compared to its competitors (Microsoft SQL[169], Oracle Spatial[170]), it is easy to get hold of and free to

use – and the fact that the author was familiar with the software beforehand. Both PostgreSQL and PostGIS are present in the main repositories of the Ubuntu package system, and thus very easy to install.

PostGIS supports a number of open, standardised output formats as results from, or input to, an SQL query. Among the supported formats are Well-known Text, Well-known Binary, SVG, GML and GeoJSON[171, 172]. This gives the developer a lot of choice when choosing a format, perhaps eliminating an additional format conversion between PostGIS and the target application (by outputting the required format immediately), and also gives opportunity for choosing an appropriate data format.

The spatial extensions includes some methods for generalisation (see Section 5), such as *ST_Simplify*[173] and *ST_SnapToGrid*[174]. The *ST_SnapToGrid* algorithm is used to approximate a geometry to a grid system, such that there is only a single Point in each cell, thus simplifying the geometry and reducing its data footprint.

A well-known algorithm for generalisation is Douglas-Peucker[71], a split-and-merge algorithm which works by simplifying a curve into fewer segments, to a given threshold. A polygon is a closed curve, which means the Douglas-Peucker algorithm will work on both curves and polygons. PostGIS' *ST_Simplify* applies the Douglas-Peucker algorithm to a geometry or a collection of geometries.

In the implementation, to illustrate the impact of even simple generalisation, *ST_Simplify* is used on the geometry. The results (see Section 17.4) indicate that generalisation is crucial to a performing vector map, and that it affects performance in a major way, as it directly correlates with the data amount transferred. It does in no way exhaust the possibilities for further generalisation, of which there are many (see Section 5).

A common technique to minimise data consumption, is to deliver the data in tiles (see Section 3). Tiles were tested both on-demand and cached, with the conclusion that on-demand tiles were too slow to be viable (ten-fifteen seconds wait time from the request to a fully rendered map with 10 tiles).

With raster tiles, the amount of processing required to render tiles makes it unfit for on-demand consumption, and this is also the case with this particular implementation of vector tiles. Therefore, tiles used in the implementation were pre-generated in a separate table (see Table 2), with each tile consuming a single row, with a GeometryCollection containing the tile's spatial data.

original data	tiles	generalised tiles
veg	map-veg-tiles	map-veg-tiles-simple1
bygning	map-bygning-tiles	map-bygning-tiles-simple1
annen-bygning	map-annen-bygning-tiles	map-annen-bygning-tiles-simple1

Table 2: The relationship and layout of the PostGIS database tables in the implementation.

17.1.2. Python and Shapely

The Java Topology Suite (JTS)[175] is an open source package of spatial algorithms and predicates, for manipulating and working with spatial data. JTS' design is mostly shaped by the Open Geospatial Consortium's Simple Feature Access specification[150], which it supports fully. It is able to interact with the standard formats Well-known Text and Well-known Binary, also defined by the Open Geospatial Consortium, and is well-documented and widely used.

GEOS[176] and Shapely[177] are ports of the Java Topology Suite, for C/C++ and Python, respectively. GEOS is the geometry engine that powers PostGIS, which is a testament to the wide usage of JTS/GEOS/Shapely. Python is an open source, interpreted dynamic language, and it is very efficient for prototyping and development time, compared to native, compiled languages like C and C++.

For these reasons, the authors extensive familiarity with Python, and the fact that at the time of writing, the Web Sockets server Autobahn[178] – which has an extensive standards-conforming test suite[179] and was well documented[180] – had a Python implementation, Python was chosen as the server-side development language.

Although Python might not perform as well as a native language, a working theory was that the greatest bottleneck would be the spatial database, and this was confirmed in testing later – from the on-screen log, one could see that the Python program did not affect performance in a major (negative) way, compared to other parts of the implementation, like the database and the SVG renderer.

17.1.3. Web Sockets

The Web Sockets protocol[128] was chosen for its superiority in bandwidth, latency and ease of use, compared to HTTP-based solutions[122]. The most important functionality in a web map, is data transfer, because a map without data is an empty canvas.

If the data is stored locally on the client, it is much easier to maintain high bandwidth and low latency than with data stored in a remote location, simply because there is higher risk associated with separating the spatial data from the representation.

The implementation, however, like many others, are separated into a client-server infrastructure, and has to transfer the map data across a network. As this connection is the main bottleneck in a web map system, it is also the most interesting part to research with regards to performance of the resulting application.

Autobahn is an open source Web Sockets implementation, both client and server side, provided in JavaScript, Python, PHP, Ruby and Java (for Android)[178]. It has an extensive test suite, built to ensure proper functionality and standard compliance[179]. Conforming with the standard is important, and essentially means that one can use native browser im-

plementations where possible, instead of being locked in to using both the client and server – which, in a way, defeats the purpose of using the open Web Sockets standard.

The Autobahn Python Web Sockets server library is used to implement a Web Sockets server to deliver vector data to clients, along with the rest of the server side Python technologies used. The clients use their native browser implementations of the Web Sockets client where possible, to get realistic results for the current support level in browsers.

Future implementations could employ Web Sockets servers with less overhead, such as the WebSocket++[181] library, written in C++ using Boost.Asio[182], which is used as a performance test baseline in the tool wspanf[183]. However, this would require more development time, and might be more appropriate for deployment in highly scalable environments, i.e. when the server should be able to handle hundreds or thousands of users concurrently, which is not possible to test with the implementation presented here.

However, the HTML5 Web Sockets technology has proven, in the implementation tested here, to be a robust and efficient solution for performance critical data transfer of spatial vector data, by employing techniques for immediate delivery and binary data transmission. For data transmission purposes, HTML5 has made impressive advances – what was impossible to achieve a couple of years ago on the open web, is now both easy and straight forward.

17.2. Data Transmission

17.2.1. Binary data

In an efficient spatial data (i.e. points, curves and areas) structure, with minimal metadata and space consumption, points are dominant, which means coordinates – regardless of the number of dimensions – outnumbers everything else by a large margin. A coordinate is a number type, either a float or an (signed or unsigned) integral, which means that an efficient storage format would need to be able to store numbers efficiently, and it should also be efficient to process (encode and decode) the data structure and its coordinates. By using network order, the data is consistent between the server and the client (see Section 13.1).

A binary data structure is ideal for this purpose, as it is a binary string that can store native data formats, and also process them efficiently. The largest difference – apart from the fact that a binary format can not be read directly by a human – is the space that is consumed by integrals and floats. Since the binary number system is used, a number is represented in a predefined amount of bits. One byte is eight bits, and an unsigned number can not be negative, while a signed number need to be able to represent equally large positive and negative numbers (i.e. $\langle -a, b \rangle$ where $a = b - 1$). The difference between the positive and negative number is due to the highest bit being used for negative numbers, and 0 in this context is a positive value (see Table 3).

octal value	binary value	octal value	binary value
0	0000	-8	1000
1	0001	-7	1001
2	0010	-6	1010
3	0011	-5	1011
4	0100	-4	1100
5	0101	-3	1101
6	0110	-2	1110
7	0111	-1	1111

Table 3: The relationship between binary and octal values for a single byte signed integer.

Note that the high bit marks a negative number, and that -1 has the highest binary value.

So, in sixteen bytes (one byte = eight bits), one can for example – this is not an extensive list – store sixteen unsigned numbers between 0 and (including) 255 (one byte is eight bits, $2^8 = 256$, sixteen signed one byte numbers between -128 and 127, eight unsigned two byte numbers between 0 and $2^{16} - 1 = 65535$ (signed between -32768 and 32767) or two eight byte double precision float numbers (their exact representation is more complex, and unfortunately not accurate).

Compared to a text format, this is a lot more efficient, resulting in a data structure that is more compact, and does not require as much processing (i.e. converting numbers between text and binary) as a text format. Support for both integrals and floating precision numbers have been implemented, to compare the benefits of using binary representation. By using tiles with local origos, one is able to use integrals of different sizes (one byte = a 256x256 tile, two bytes = a 65536x65536 tile etc.) for coordinate representation, which further lowers the data footprint.

It becomes clear that larger integral sizes, and even float precision numbers, represent an area that is much larger than a normal display at a zoom level where all details are visible. Smaller, byte-sized tiles are also suited for caching (see Section 4).

17.2.2. Text data

Text data formats are generally more popular than binary formats, because they can be edited and created only with the help of a simple text editor. With binary formats, one either needs software to handle (i.e. create and edit) the data format, or develop custom software, which is not especially user friendly. Data formats that are text-based, are somewhat self-documenting, meaning that it will often be possible to infer the data structure by looking at it – this is much harder or impossible to do with a binary format.

The GeoJSON data format (see Section 13.3), is an open standard text based geospatial data format, and widely supported. The support by numerous applications – both web applications and native applications – gives GeoJSON a solid foundation and it is a great choice for

any new application, as the format choice translates to a certain form of compatibility with other geospatial software.

GeoJSON data is sent in the UTF-8 encoding, which employs a variable length encoding scheme, and any value is encoded using between 1 and 3 bytes. However, only 7 of the eight bits in each byte are used to store data – the first bit is used to encode metadata about the encoding[184]. Compared to binary format, one loses one bit for every byte, which accounts to a certain amount. Also, a number encoded as a string will usually consume more space than the binary equivalent, and will also have to be parsed from string to integral. The UTF-8 encoding stores the text as a single byte sequence, and is therefore not affected with byte-order issues (see Section 13.1).

17.2.3. Data Formats

Multiple data formats were implemented, to be able to test and observe the differences and consequences of choosing the most and least efficient format. By implementing formats with widely different characteristics, such as binary and text, and tile-based and on demand, the results give a result that reflects the choices one has in a production environment.

FORMAT BINARY 1B CACHED TILES A custom binary data format where each coordinate consumes a single byte (0-255), and uses cached tiles that are of size 256x256 units (which correspond to the coordinate range). Each tile has a local coordinate system, which needs conversion from the spatial database (where the coordinates are global), and on the client when creating SVG (where the coordinates need to be global).

FORMAT BINARY 1B CACHED TILES SIMPLE Equal to **FORMAT BINARY 1B CACHED TILES** except the cached tiles used are pre-generalised with the Douglas-Peucker algorithm (see Section 5 and 7.1).

FORMAT BINARY 2B ONDEMAND A custom binary data format where each coordinate consumes 2 bytes (0-65535), and uses on demand tiles (i.e. not cached, but created by the server, after retrieval from the spatial database) that are of size 65536x65536 units (which correspond to the coordinate range). Each tile has a local coordinate system, which needs conversion from the spatial database (where the coordinates are global), and on the client when creating SVG (where the coordinates need to be global).

FORMAT BINARY 2B ONDEMAND SIMPLE Equal to **FORMAT BINARY 2B ONDEMAND** except the cached tiles used are generalised on demand (i.e. by the database during the query) with the Douglas-Peucker algorithm (see Section 5 and 7.1).

FORMAT BINARY 8B ONDEMAND A custom binary data format where each coordinate is a double floating point precision number, that consumes 8 bytes (float numbers does not have a range, as they are differently implemented than integrals). The address-space of this number is great enough to cover an extremely large area, and there is

therefore no need to split the coordinates into tiles.

FORMAT BINARY 8B ONDEMAND SIMPLE Equal to FORMAT BINARY 8B ON-DEMAND except the geometry is generalised on demand (i.e. by the database during the query) with the Douglas-Peucker algorithm (see Section 5 and 7.1).

FORMAT GJ CACHED TILES The popular GeoJSON data format which is a text format (see Section 13.3) and not as compact as the binary formats. The tile model, however, is equal to the FORMAT BINARY 1B CACHED TILES – coordinates with the range 0-255, which needs conversion from the spatial database (where the coordinates are global), and on the client when creating SVG (where the coordinates need to be global).

FORMAT GJ CACHED TILES SIMPLE Equal to FORMAT GJ CACHED TILES except the cached tiles used are pre-generalised with the Douglas-Peucker algorithm (see Section 5 and 7.1).

17.3. Client Architecture

The primary client machine is a Dell Latitude D830, with 4GB RAM, Intel Core Duo T7100 (dual core, 1.80 GHz), however the hardware specifics is more relevant on the server, as the client's task is only to receive, process and render the spatial vector data.

In principle, this could be very reliant on the computers hardware, but since the HTML5 technology is quite new, it is probably a higher chance of the browser and JavaScript performance being the limiting factor. To increase the efficiency of the JavaScript implementation, the implementation tries to adhere to Nokia's best practices for JavaScript performance[185].

The implementation also uses TypedArrays (see Section 10) to store the binary data structure(s) on the client, for faster processing of numbers. Unfortunately, the JavaScript *lossless* compression (see Section 7.2) libraries available did not support both text and binary formats for compression, which was needed to be able to decompress and compare the performance of both text and binary formats, and it was therefore not used in the implementation testing.

17.3.1. SVG

From a developers standpoint, an existing and robust specification and implementation is more attractive than having to "reinvent the wheel", which is more work, and comes with a high chance of introducing bugs in the implementation, which is best avoided if possible.

Lots of HTML5 work has been done with the canvas element, which is a simple specification with a simple canvas, and one has to develop additional functionality or find an existing library (at the time of writing, there are very few).

SVG has the functionality of objects built in, and it was therefore a natural choice for implementing vector maps on the web, compared to the canvas element (see Section 9). Especially

in a scientific context, where the amount of work one has time to complete, is limited, a solution with the canvas element would be even more difficult to achieve.

However, SVG is tedious to work with directly, manipulating and (re)calculating coordinates and paths for spatial features is the primary activity in the client implementation. Therefore, a library was used to avoid manipulating SVG manually.

17.3.2. Choosing a Library

There are currently a handful of libraries capable of manipulating and working with SVG elements available that, in the authors view, are mature enough to use to render vector map data. The most popular are Raphaël[186] and d3[187], and they fulfill both requirements, although differing in their amount of abstraction and interfaces.

Raphaël strives to be as backwards compatible as possible. By supporting both SVG and VML[188] (a former competitor to SVG, now deprecated), the developer can write applications that are compatible with browsers that do not have support for SVG (of the major browsers, only Microsoft Internet Explorer versions 6.0 to and including 8.0 does not have support for SVG[189]), although the cost is an API that supports the "common denominator" of VML and SVG, which partly limits the functionality of the library, as well as introducing an additional abstraction layer.

The d3 library is a generic library for directly manipulating the Document Object Model of a document, which includes the ability to create and edit SVG. Instead of inventing a custom abstraction, which will generally decrease performance, the d3 library tries to work directly with the underlying technologies, making it possible to e.g create SVG objects and define their visual appearance with the established and standardised Cascading Style Sheet (CSS)[84], separate data from style, and – an important feature in a map context – the possibility of vector web maps with dynamic and possibly user-defined style sheets without placing any load on the server (i.e. the re-rendering is only done on the client).

Both libraries make interaction and reuse of existing software straightforward. SVG created in a vector graphics suite such as Inkscape can be reused in both Raphaël and d3 by opening the SVG file in a text editor, and then capture the path string of the object(s). By using a set-based data model, the d3 library is well suited for tile-based maps and vector data, where caching, updating, deleting and replacing data are common operations, and the d3 library was chosen over Raphaël, mainly because of the advantages presented, in the implementation.

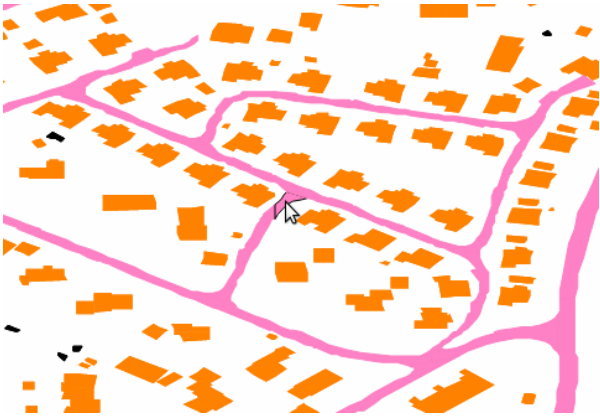
17.3.3. User interaction

The resulting client application (see Figure 18 and 19) have some functions that are common in current web maps, and have the appearance of a regular map, where the controls are familiar, and a user should be able to figure out how to navigate in the map, by using

the controls (zoom, pan)[190]. Note that the edges non-straightness in some cases are due to rounding errors – to be able to store data in as little as a single byte, without any generalisation, and at a relatively small scale, the precision had to be compromised. However, with proper generalisation, and multiple tile layers, this will not happen in a production implementation.



(a)



(b)

Figure 18: The standard implementation map client (a) with a familiar interface for controlling the map, which is composed of vector tiles (b).

The styling and data formats are also very easy to change on the client, by changing the CSS and modifying the client.html, respectively, without touching the server. Overall, the client implementation was re-architected and coded to make it easy to change the data format

used, and to make the comparison fair between data formats.

```
3   border: 2px solid black;
4   margin: 20px;
5 }
6
7 #VEGTABLE.polygon path {
8   /* fill: #ff82c6; */
9   fill : black;
10  stroke: #ff82c6;
11  stroke-width: 1px;
12 }
13
14 #VEGTABLE.polygon path:hover {
15  stroke: #000;
16  stroke: black;
17 }
18
```

(a)



(b)

Figure 19: The styling is completely controlled by the client, which means that a change in the CSS, e.g. road colour fill to black, will be reflected in the map client (d), with no modification on the server.

17.3.4. Native application comparisons

The main goal of this implementation is to use open, standardised technology to achieve efficient vector maps on the web. Although a standard, at the time of writing, the HTML5 spec-

ification is not fully implemented in all browsers yet, and the implementation is balancing at the very edge of new, open and standardised web technology to explore the possibilities of fully functional and platform independent, vector-based web maps.

Native map applications, either outside a web context or web applications using a third-party plugin, would almost certainly perform as well or better than a standards conformant implementation, before HTML5, because the standards were far behind the technology that was needed. This is changing, the standards are catching up, and as the situation with complete support across all major browsers progresses, developing pure web applications using only open web standards will provide a viable alternative for GIS vendors.

17.4. Implementation performance

Testing the implementation is important to get performance results in a realistic environment. Even though the data formats and their resulting data size can be reasoned about with certain accuracy, proof in terms of test results is preferred and reassuring. With latency and processing time, it is much more difficult, or impossible, to predict the results – especially on software with multiple concurrent users, like the map system implementation presented here.

A realistic environment was created for the test, where multiple users collected performance data concurrently while using the map (i.e panning and zooming). The results were collected in JSON data files, and data was differentiated on both data format type and vector layer. The different vector layers have different characteristics (containing buildings and roads). The data files were then collected and processed to a single JSON file in a analysis-friendly format.

Most test data contain a certain amount of spikes or noise, i.e. extreme values that are due to unforeseen events, such as interference by other software, or a fault in the system. To counter this, a median filter[191][192][193] was implemented, as spike values impact mean values directly, and not median values with a large selection window. The filtered data, heavily processed and now ready for use, were rendered and plotted with the flot JavaScript framework[166].

Being able to differentiate between the different entities in the plot is crucial, which is why extra care was taken in picking colours for the different data formats. Generalised variants of the same data formats have dashed line versions and the same colour as the version of the data format that is not generalised (in our legend (see Figure 20). An example is the *SIMPLE* formats, where the data is generalised with the Douglas-Peucker algorithm (see Section 5 and 7.1)). Also, the different data formats have colours – it should be fairly easy to distinguish between them.

- FORMAT_BINARY_1B_CACHED_TILES
- FORMAT_BINARY_1B_CACHED_TILES_SIMPLE
- FORMAT_BINARY_2B_ONDEMAND
- FORMAT_BINARY_2B_ONDEMAND_SIMPLE
- FORMAT_BINARY_8B_ONDEMAND
- FORMAT_BINARY_8B_ONDEMAND_SIMPLE
- FORMAT_GJ_CACHED_TILES
- FORMAT_GJ_CACHED_TILES_SIMPLE

Figure 20: The legend for the test results of the vector implementation

An important premise for research is the ability to confirm and verify other’s claims and results. Extra effort has been made to make the complete source code for the implementation, data processing and the resulting graphs available (see Section 1.1 and Appendix B) for anyone to re-run the tests and verify the end results.

17.4.1. Data size

Most tile-based web maps have multiple layers, often in a quad tree structure, to accommodate for different map scales (see Section 3). By using this technique, the amount of tiles remains fairly constant at all map scales. The implementation presented in this paper, however, due to time constraints, only has a single layer, which means that the amount of tiles will vary between map scales (see Figure 21).

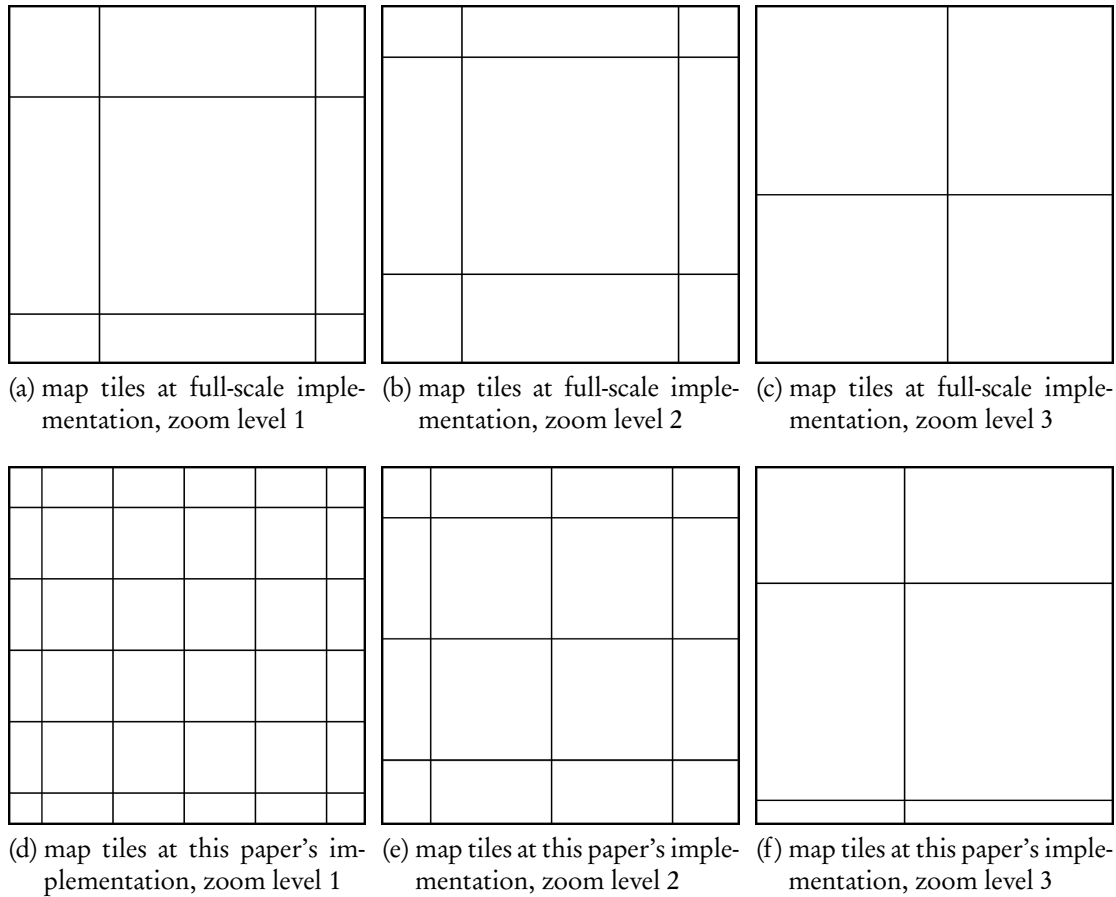


Figure 21: The layers of a full-scale implementation means that the amount of tiles remains constant at different zoom levels (a), (b) and (c). The implementation presented in this paper only has a single layer, which means that the amount of tiles will vary between zoom levels (c), (d) and (e).

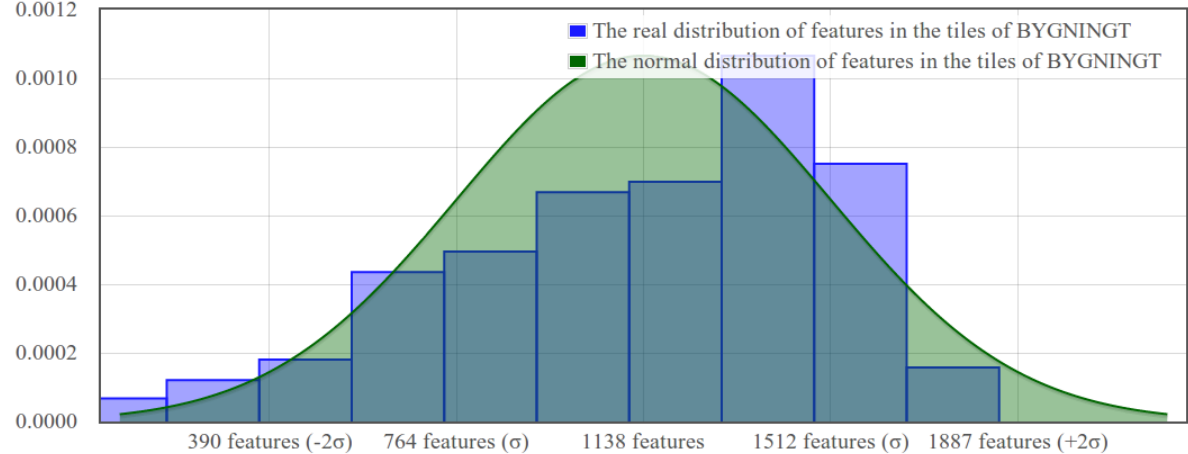
By selecting a map scale where the amount of tiles is equal to what one would find in an full-scale implementation with multiple tile layers (an example is Figure 21f), the total amount of features that are downloaded should be fairly equal to the amount of features that are downloaded at any map scale in a full-scale, multiple tile layer implementation.

By making this simple assumption, we found the number of features for our single layer example at the appropriate zoom level on multiple locations in the map, and used this as a guideline when assessing the results in Section 17.4.2 and 17.4.3. This is very useful for approximating the performance in a full-scale environment, to get an approximate value of the amount of features.

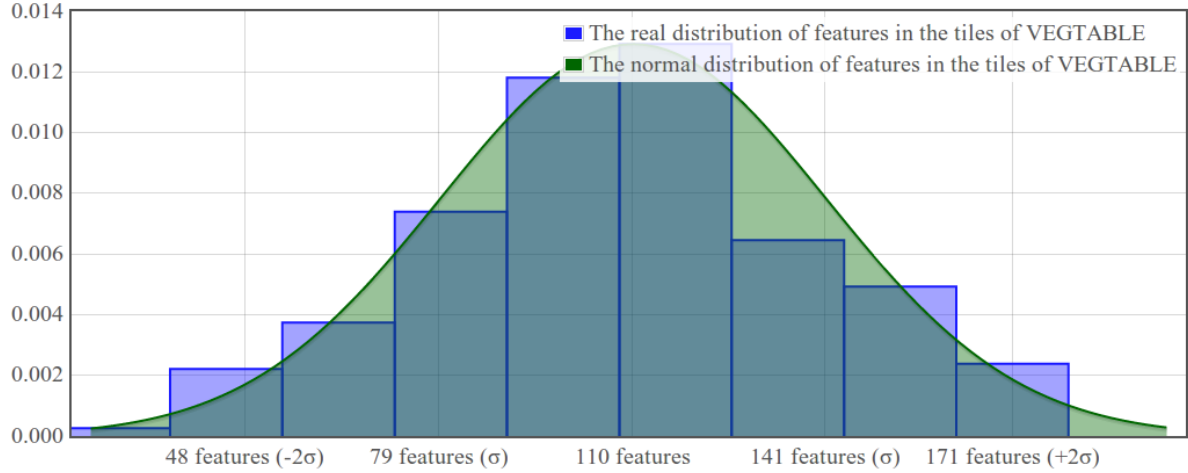
For the collected results, mean value (μ) and standard deviation[194] (σ) were estimated. Then the normal distribution was created with the probability density function (see Equation 1)[195] to visualise the distribution of the observations. There were 618 observations for both the *BYGNING* table and the *VEG* table, which means that n should be large enough

to approximate normal distribution, in both cases[196].) Note that this is more correct for the *VEG* table than the *BYGNING* table.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$



(a)



(b)

Figure 22: The number of features at an appropriate scale, fitted to a normal distribution, in the *BYGNING* table (a) and *VEG* table (b). (See Appendix E for larger versions of the plots.)

To quantify the results, and make it easier to assess the results, an examination of the probabilities of when the number of features is above a certain amount is useful to explore. The natural limits of one and two standard deviations is a good starting point. To be able to use

the values from the table[196], the data need to be converted to standard normal distribution ($N(1,0)$), with the function $Y = \frac{X-\mu}{\sigma}$, where X is our existing variable.

The results for a single and two standard deviations above the mean value for amount of features in the *BYGNING* table :

$$\begin{aligned}
 P(X \leq 1509) &= P\left(\frac{X - \mu}{\sigma} \leq \frac{1509 - \mu}{\sigma}\right) \\
 &= P\left(\frac{X - 1139}{374} \leq \frac{1509 - 1139}{374}\right) \\
 &= P(Y \leq 0.989) \\
 &= \phi(0.989) \\
 &= 0.837
 \end{aligned}$$

$$\begin{aligned}
 P(X \leq 1889) &= P\left(\frac{X - \mu}{\sigma} \leq \frac{1889 - \mu}{\sigma}\right) \\
 &= P\left(\frac{X - 110}{31} \leq \frac{1889 - 1139}{374}\right) \\
 &= P(Y \leq 2.00) \\
 &= \phi(2.00) \\
 &= 0.977
 \end{aligned}$$

The results for a single and two standard deviations above the mean value for amount of features in the *VEG* table :

$$\begin{aligned}
 P(X \leq 141) &= P\left(\frac{X - \mu}{\sigma} \leq \frac{141 - \mu}{\sigma}\right) \\
 &= P\left(\frac{X - 110}{31} \leq \frac{141 - 110}{31}\right) \\
 &= P(Y \leq 1.00) \\
 &= \phi(1.00) \\
 &= 0.864
 \end{aligned}$$

$$\begin{aligned}
P(X \leq 172) &= P\left(\frac{X - \mu}{\sigma} \leq \frac{172 - \mu}{\sigma}\right) \\
&= P\left(\frac{X - 110}{31} \leq \frac{172 - 110}{31}\right) \\
&= P(Y \leq 2.00) \\
&= \phi(2.00) \\
&= 0.977
\end{aligned}$$

For the performance tests, we now have an idea of typical usage, and both the standard deviation and calculations give an estimate on the probable amount of features for the two tables. This means that we should mostly look at less than 1700 features for the *BYGNING* table, and less than 140-170 features for the *VEG* table.

17.4.2. Storage

The importance of generalisation in vector data sets have been stated previously (see Section 5 and 7.1), and the results that show the amounts of coordinates per feature (see Figure 23) confirms that the storage requirements of a vector data format is not able to negate the lack of generalisation.

The data formats with dashed lines have been generalised, while the data formats with darker colours have not. The difference between similar shaded colours notes more efficient packing of data, i.e. less structural overhead. Note that the geometric differences in the *VEG* table and *BYGNING* table, in essence the former consists of roads, with more coordinates per feature, and the latter consists of buildings, with typically less coordinates per feature. Because one does not gain much by generalising a rectangular building, the table with roads gains more from generalisation.

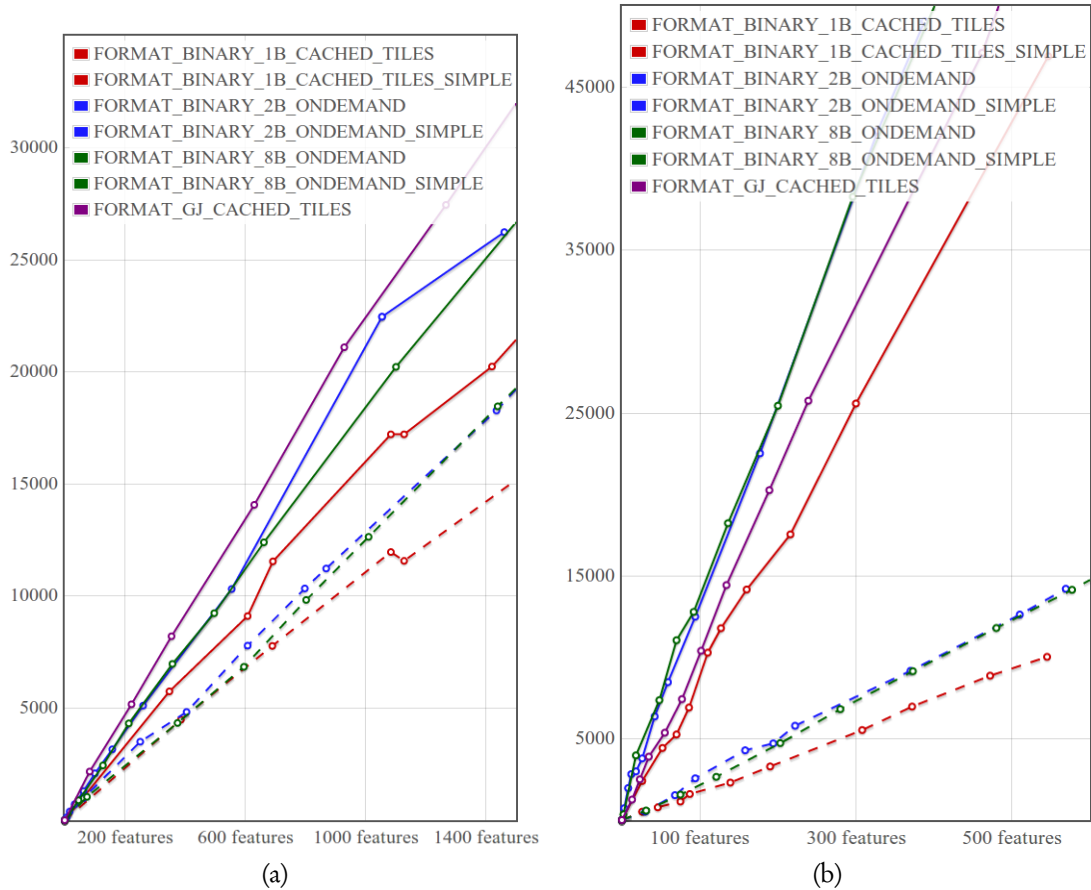


Figure 23: The relationship between the number of features (horizontal axis), and the resulting number of coordinates in the *BYGNING* table (a) and *VEG* table (b). Every generalised data format (dashed lines) consumes a lot less space in terms of number of coordinates, than the data formats that are not generalised. The generalised GeoJSON was removed, because of an error with the median filter. (See Appendix E for larger versions of the plots.)

In other words, even a very naive and simple generalisation – as done in this implementation (see Section 17.1.1) – affects the amount of data in a major way. It is therefore difficult not to draw the conclusion that an efficient vector-based map need to use generalisation to compress the data.

To assess the efficiency of the data formats themselves, one also needs to look at the actual space used to store the data – this is different from the number of coordinates per feature, because the space consumption of a coordinate varies from format to format. The results (see Figure 24a and 24b) favours the binary, single byte, tile-based vector data formats. Although there is a notable difference, the version of it that is not generalised is very efficient as well.

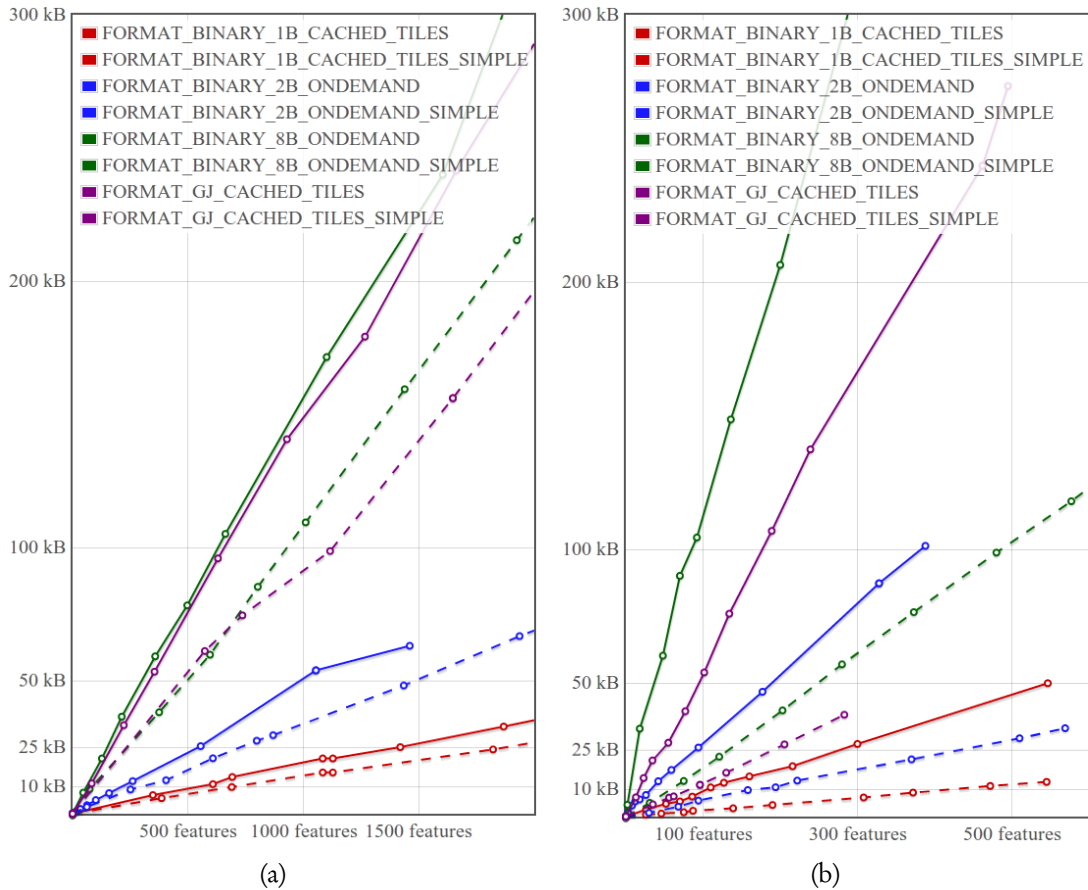


Figure 24: The efficiency of the data formats in the *BYGNING* table (a) and *VEG* table (b). Both the space efficiency of each data format, as well as the impact of generalisation (dashed lines), is visible. The horizontal axis represents the number of features, and the vertical axis is data size in kB. (See Appendix E for larger versions of the plots.)

Because the space consumption of each coordinate is greater with the other binary formats (2 bytes per coordinate, 8 bytes per coordinate), it is not very surprising that the resulting data consumes more space. However, GeoJSON (see Section 13.3) is interesting to observe, as it is the only text format tested in this implementation, and perhaps the only (or one of very few) text-based, relatively efficient, human-readable geospatial data format in wide use. It is apparent from the results that the overhead associated with using this text format does not stack up with the space efficiency of a native, binary data format.

Another observation is how generalisation affects the data size by reducing the number of coordinates (which is what Douglas-Peucker – which is the only generalisation algorithm used in this implementation and test (see Section 17.1.1) – does). Logically, a reduced number of coordinates means a smaller footprint, and a data format where a coordinate is large (for example 8 byte per coordinate) will see a larger reduction than a data format where each

coordinate is tiny (for example 1 byte per coordinate).

The GeoJSON format seems to perform somewhere between the binary formats with 2 bytes per coordinate, and 8 bytes per coordinate, which should be expected with the UTF8 variable bit-length character set used to encode the text (see Section 17.2.2).

17.4.3. Latency

The bandwidth required to handle a data format is only part of the puzzle. It is also interesting to measure the time requirements of database queries, and the latency overhead associated with processing data on the server and on the client – this is especially relevant with vector data, because the performance responsibilities are not constrained to the server in the same manner as raster maps (see Section 6 and 7). The differences between the data formats therefore might predict their suitability for e.g. mobile devices, which are weaker in terms of processing power, and the chosen data format might therefore have more positive or negative impact on the end performance.

All the data formats are built from spatial data stored in a spatial database (see Section 17.1.1), but the differences in retrieval and what structure the data (and query result) is stored in, affects the query performance (see Figure 25a and 25b). The GeoJSON format is the only format where the query's result is text-based, because GeoJSON is supported natively in PostGIS – the generalised version improves on performance, because of the highly reduced amount of data the query engine needs to handle.

The GeoJSON format, together with the 1 byte per coordinate binary format, queries tables with pre-generated tiles, while the 2 and 8 byte per coordinate data formats' queries are generated on demand. The larger difference in latency between generalised and not generalised tile-based and generalised and not generalised on demand data formats is not surprising, because the generalised tile-based data formats query pre-generalised tiles, cached in the database, while the formats that are not tile-based compute the generalised geometries on demand (see Section 13 for details).

The query time for the different data formats presented in the results are not very high – between 25 ms and 100 ms for 1000 features in the *BYGNING* table – but the latencies stack up, and query time is expected to be the least demanding process. In a complete map, one should expect more layers, and an even higher amount of features, and a higher difference in latency between the queries for different data formats.

The latency of transferring data between client and server also comes on top of this. Accurate testing of this latency is very hard if one is not using a naive approach, which would mean accurately synchronising time at microsecond level, ensuring that the time is equal in both server and client during the entire testing session, or continuous synchronisation. It has been proven that Web Sockets is the best technology currently available, and that it is suitable for real-time applications, which suggests that the time spent between server and client is small compared to client and server processing time, and query time (see Section 11

and 17.1.3), which is exactly what is examined here. (The functionality for this measuring is implemented in the software presented here, but it was not accurate enough to provide meaningful results.)

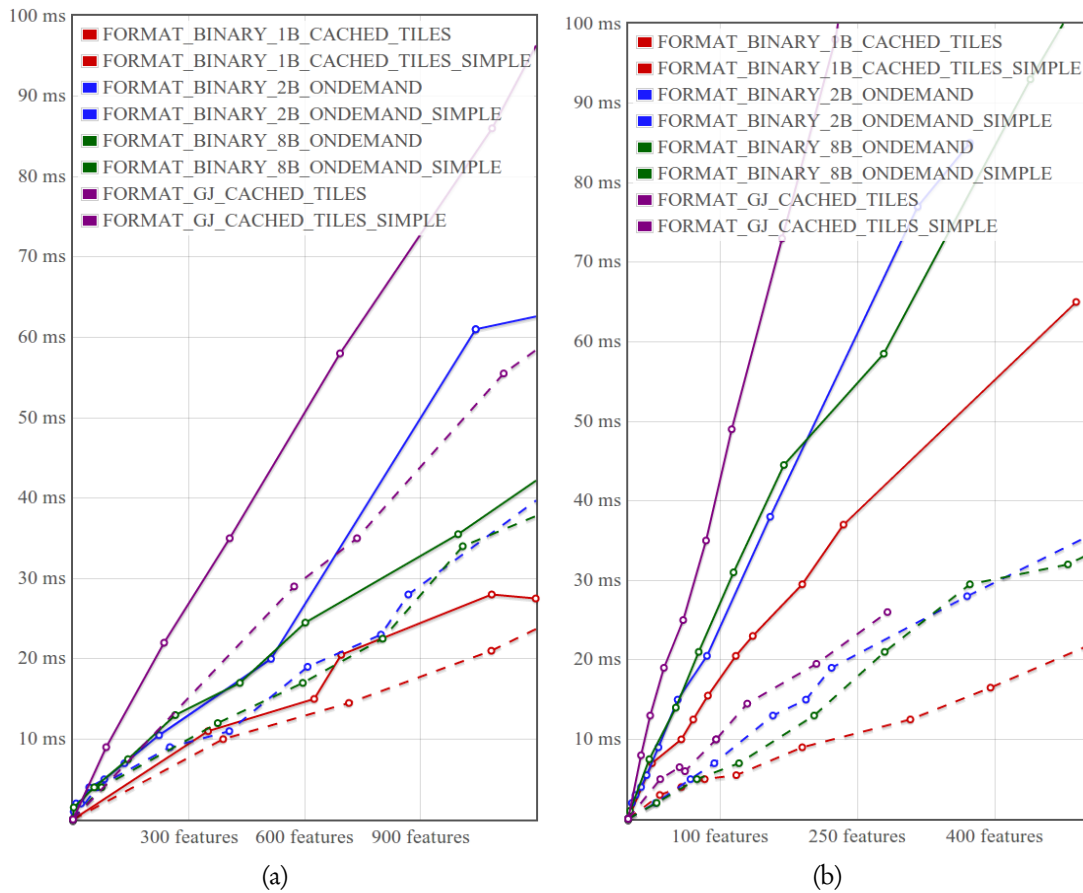


Figure 25: The latency of queries in the *BYGNING* table (a) and *VEG* table (b) for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines. (See Appendix E for larger versions of the plots.)

The server consumes the queries, and generates a data packet that is suitable for transferring to the client with Web Sockets (see Section 11 and 17.1.3). Because the results from the queries are different, and because the data formats sent to the client are different, the latency for this process differs between the data formats. Some require more work on the server, and less work on the client, and some require more work on the client, while less strain is put on the server.

The best performing data format in the results (see Figure 26a and 26b) is the GeoJSON text format. The penalty of the long database query time which resulted in a GeoJSON result pays off, even though every coordinate still needs to be manipulated to the tile's local origo.

The binary data formats require more processing, which is apparent from the higher latency – with a noticeable improvement for generalised geometry in the *VEG* table (see Figure 26b).

The server processing algorithms are written in Python, a language that is highly productive, at the expense of performance (see Section 17.1.2). There is probably lots of room for optimisation in this process, by implementing the algorithms in a more performance-focused programming language (such as C or C++), at the expense of productivity (i.e. it will probably take longer to write). Therefore, the poor latency achieved in this implementation should be considered with these optimisation possibilities in mind.

The most common way to combat high latency in map systems is through caching (see Section 4). In this implementation, caching was limited to pre-generating the tiles in the database, but there is much room for improvement in this area. By creating a tile-cache of binary blobs that are ready to be sent from the server immediately – and do not require pre-processing – the latency on the server could become very small or negligible. The binary blobs could be stored in a database or as a file hierarchy, with a pre-determined structure like the raster maps and their quadtree-structure (see Section 3), eliminating the spatial queries as well.

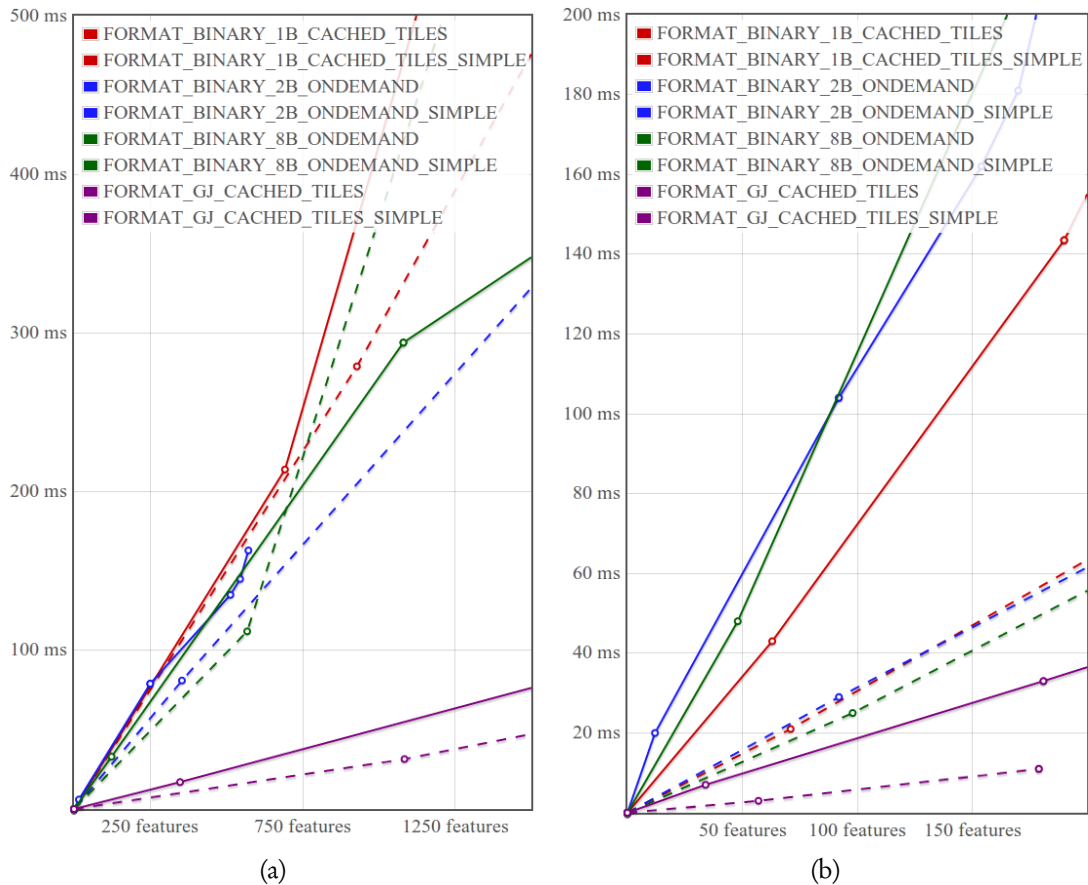


Figure 26: The latency of the server processes in the *BYGNING* table (a) and *VEG* table (b) for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines. (See Appendix E for larger versions of the plots.)

The client plays a much larger part in the resulting performance of a vector-based map, compared to the current raster-based maps (see Section 6 and 7), because it receives a data structure that needs to be parsed, processed and rendered to the screen – with raster maps, the client only needs to render the images it receives.

The largest difference between the server and the client with regards to opportunities for optimisation, is that – to remain a standard web application with wide platform and device support through HTML5 – the client is bound to the browser and JavaScript. There is a good chance that there are optimisation possibilities by improving algorithms, but one is still limited to JavaScript and the client’s JavaScript engine’s performance (see Section 8.3).

With HTML5 and the improvements in technology and especially that JavaScript handles binary data with Typed Arrays, the single byte tile data format is the best performer in the client processing results (see Figure 27a and 27b), even though it does a lot of work to parse and process the data format before converting the geometry to SVG. As the *VEG* table is

more coordinate heavy per feature than the *BYGNING* table, the performance gain is more apparent in Figure 27b.

The GeoJSON text format, which is not processed as heavily, just needs to correct the coordinates from local (using a tile-local origo) to global. On the other hand, more effort is needed to convert the data to SVG, because the numbers need to be parsed from text. When using the larger binary data formats, with 2 bytes per coordinate and 8 bytes per coordinate, the amount of data increases twofold and eightfold, respectively, and the strain on the client becomes proportionally greater as well.

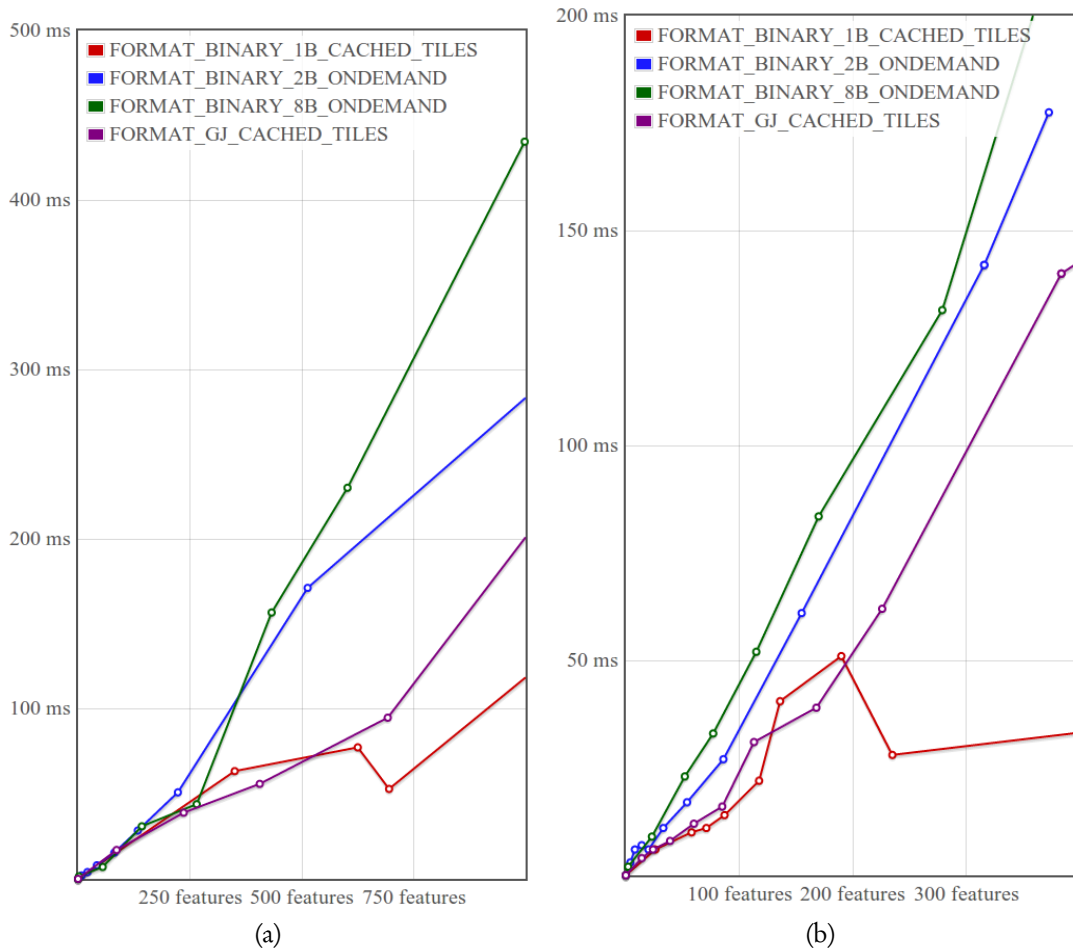


Figure 27: The latency of the client processes in the *BYGNING* table (a) and *VEG* table (b) for the data formats. The generalised versions were removed to avoid clutter, as they did not add value to the graph, in the authors opinion. Note that there are some noise in the data for the single byte binary format in (a) and (b). The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)

By reorganising the data, we can see the total latencies combined. To confirm and the results

(i.e. verify that the median filter algorithm used previously does not invalidate the data) a different smoothing algorithm was applied. It selects a feature data range (i.e. feature 50, with tolerance 10, will select data where the number of features is between 40-60), and applies a filter on the range that only includes data that does not interfere with the mean value in a major way. Since removal of a single value in a large pool of values should not affect the mean much, values that affect the mean in a major way (a custom threshold), is removed. This filter was used to calculate total latencies, and the results are presented below.

The formats affect the server and client processing latencies in different ways, and, although this can be inferred from the independent latency results, it is more apparent when latencies are combined and stacked. When comparing the single byte tile format with the GeoJSON tile format (see Figure 28 and 29), one can see the differences clearly. There are differences in the results between the *VEG* table and *BYGNING* table, because of the differences in coordinate and feature density, explored earlier (see Section 17.4.2), and some characteristics are more visible in one table than the other.

The single byte tile format is, as pointed out earlier, much more heavy on the server. However, with the help of caching (see Section 4), by producing a binary blob that does not require pre-processing, the server latency can be reduced (especially in the *VEG* table, there would be potential to increase the performance with this action). This can also be done for the GeoJSON format, obviously, but since the latency is such a small part of the total latency with this data format, there is not much to gain.

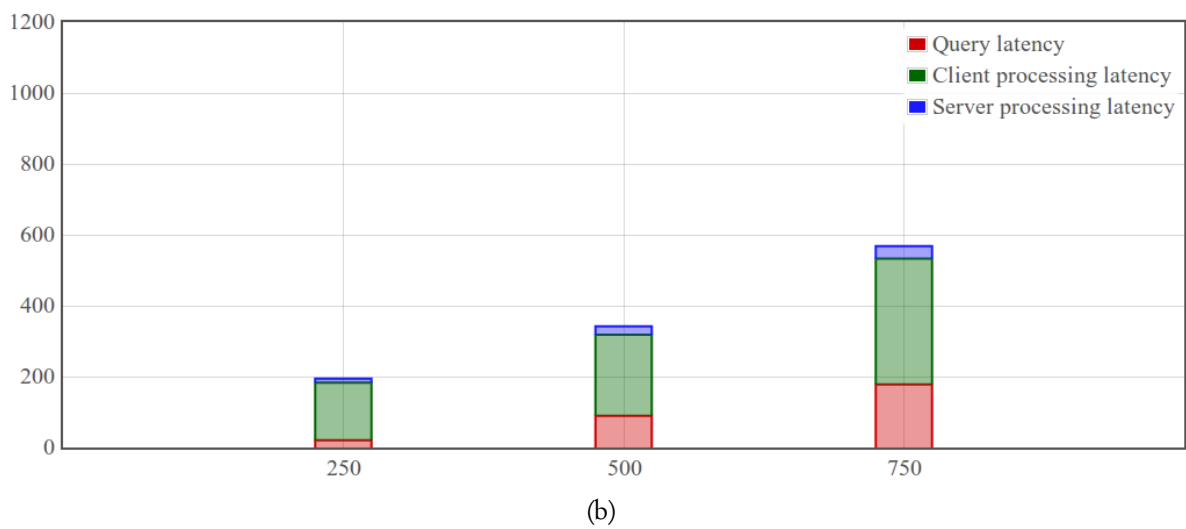
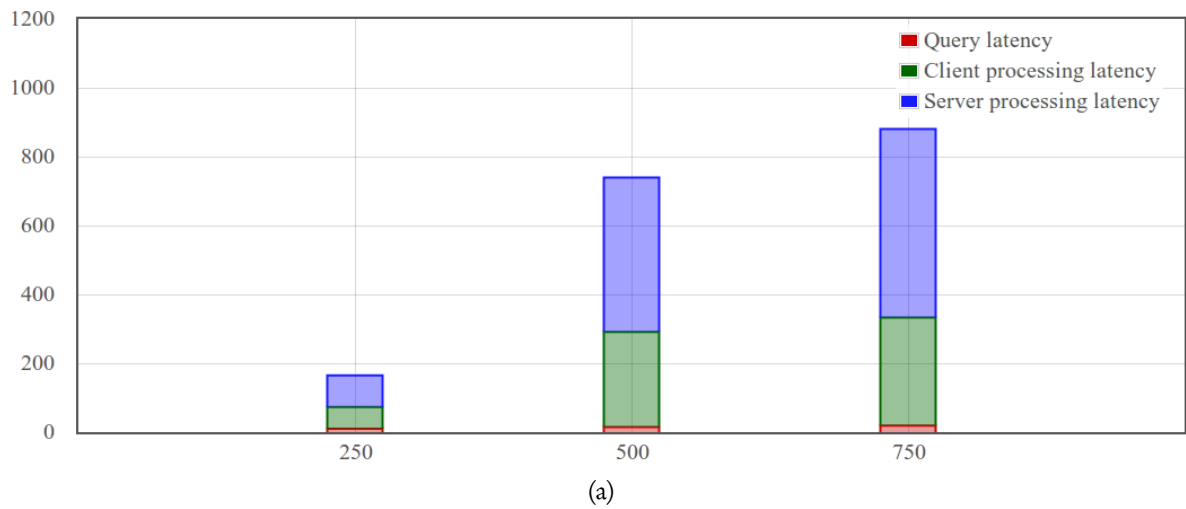


Figure 28: An overview for the latency in the *BYGNING* table for the not generalised versions of the binary single byte tile data format (a) and the GeoJSON tile format (b). The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)

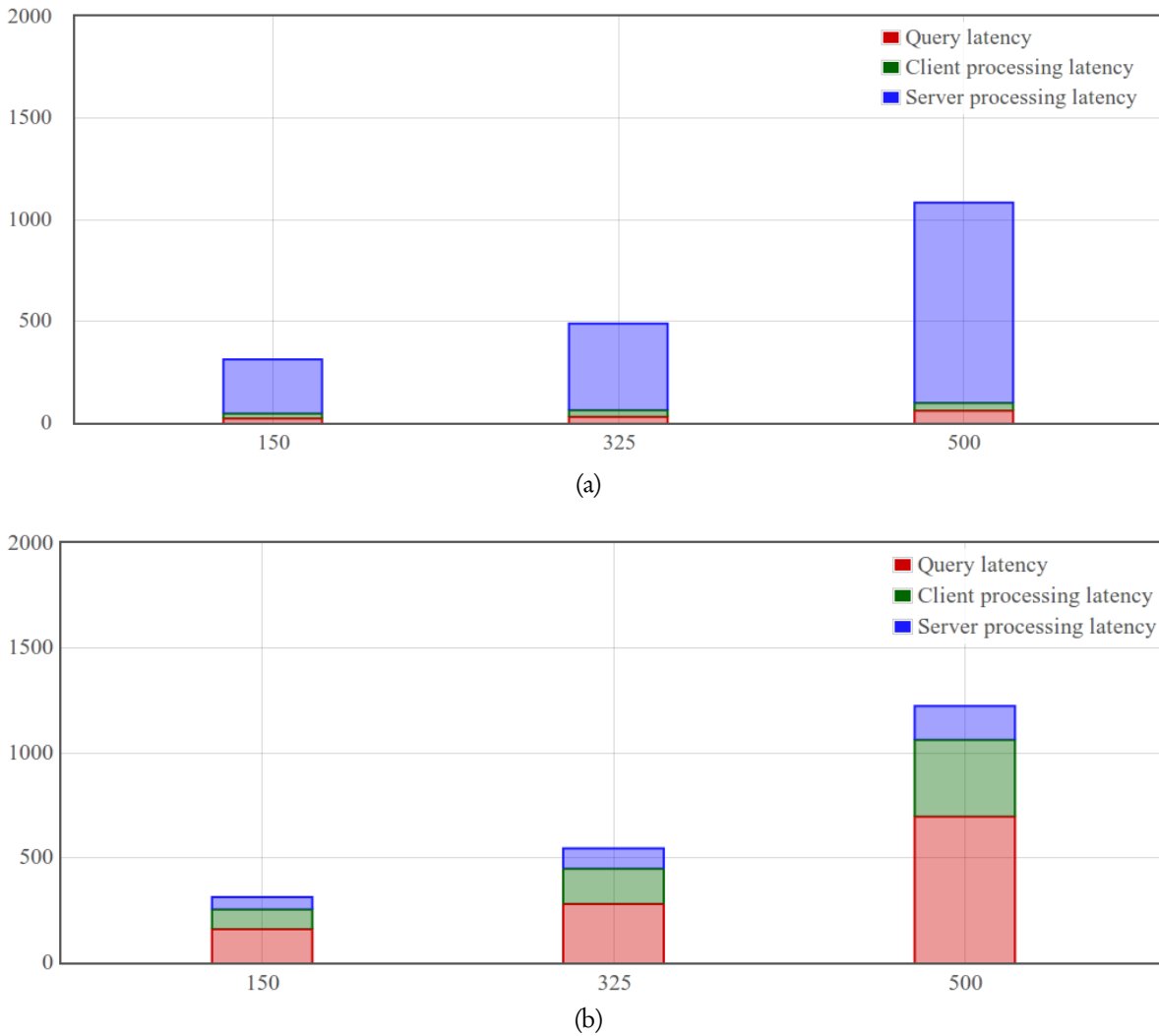


Figure 29: An overview for the latency in the *VEG* table for the not generalised versions of the binary single byte tile data format (a) and the GeoJSON tile format (b). The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)

The load on the client is approximately equal, except in the *BYGNING* table, when the amount of features is small. This might be because of the overhead associated with parsing number values, or because text is more demanding than binary (remember that the Typed Arrays (see Section 10) are used for the binary formats, and they are more optimised than text). There is a large increase in latency when increasing the number of features from 250 to 500 features in Figure 28, possibly because a threshold for storing the data sequentially in memory is passed. The differences in query latency – discussed earlier – also become apparent, and it looks like the latency is constant for the binary data format, while it increases and becomes a fairly large percentage of the GeoJSON latency.

There are not many surprises when comparing the 2 byte and 8 byte binary data formats

(see Figure 30 and 31). They are both less efficient than the binary single byte and GeoJSON data formats, and the difference increases with the amount of features. There are differences between the *VEG* table and *BYGNING* table, because of the differences in coordinate and feature density, explored earlier (see Section 17.4.2).

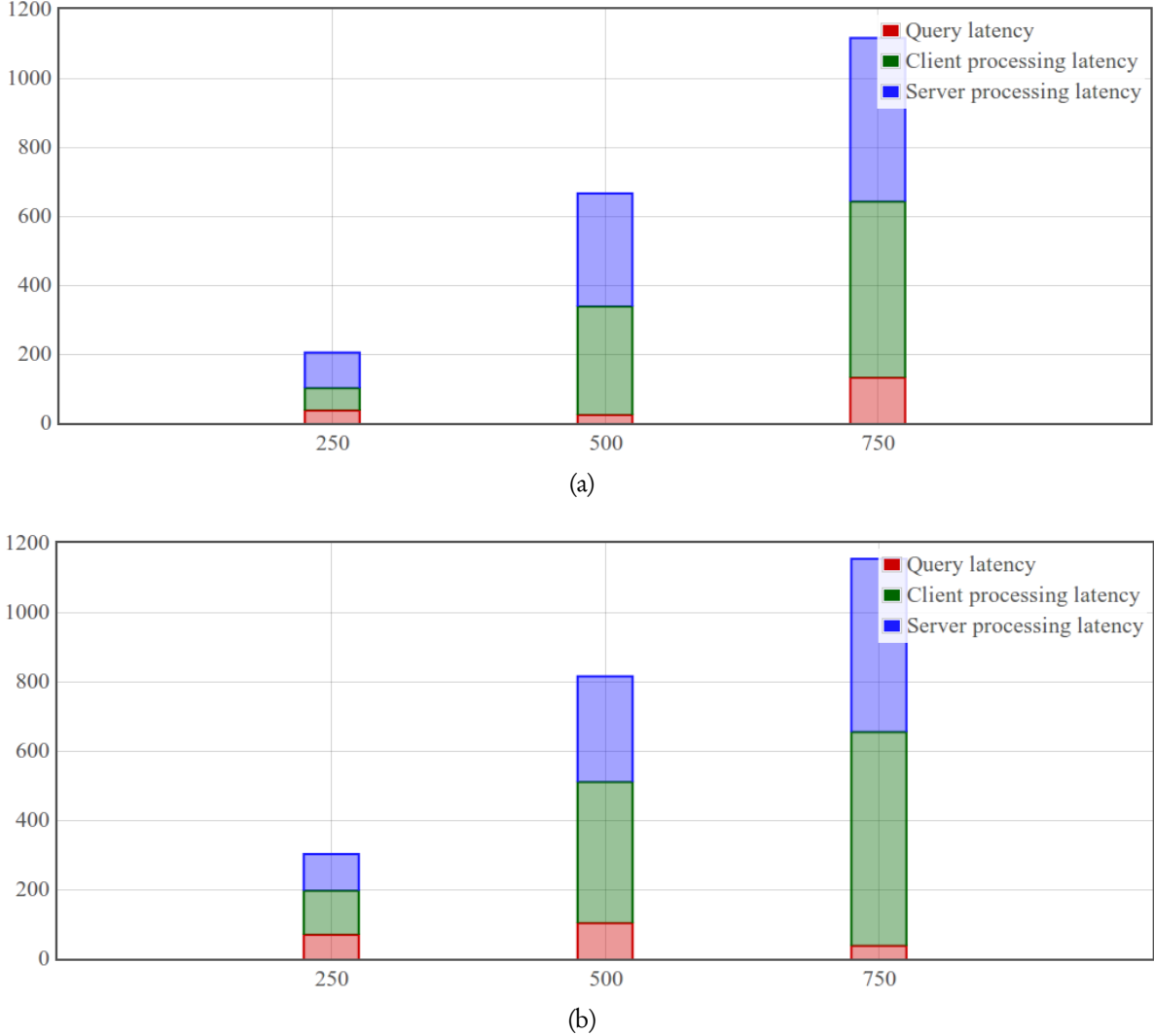


Figure 30: An overview for the latency in the *BYGNING* table for the not generalised versions of the binary 2- and 8-byte formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)

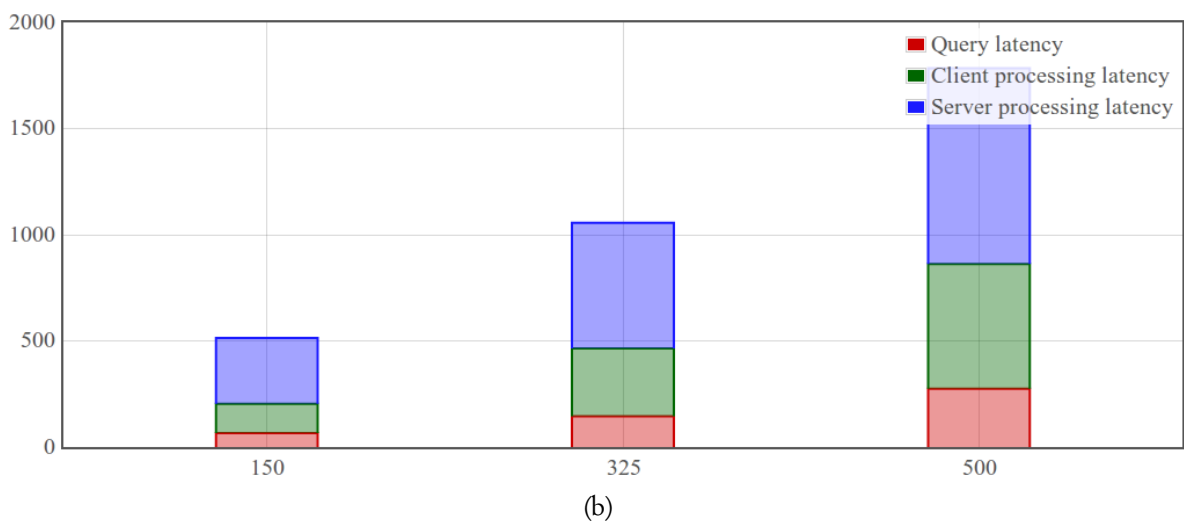
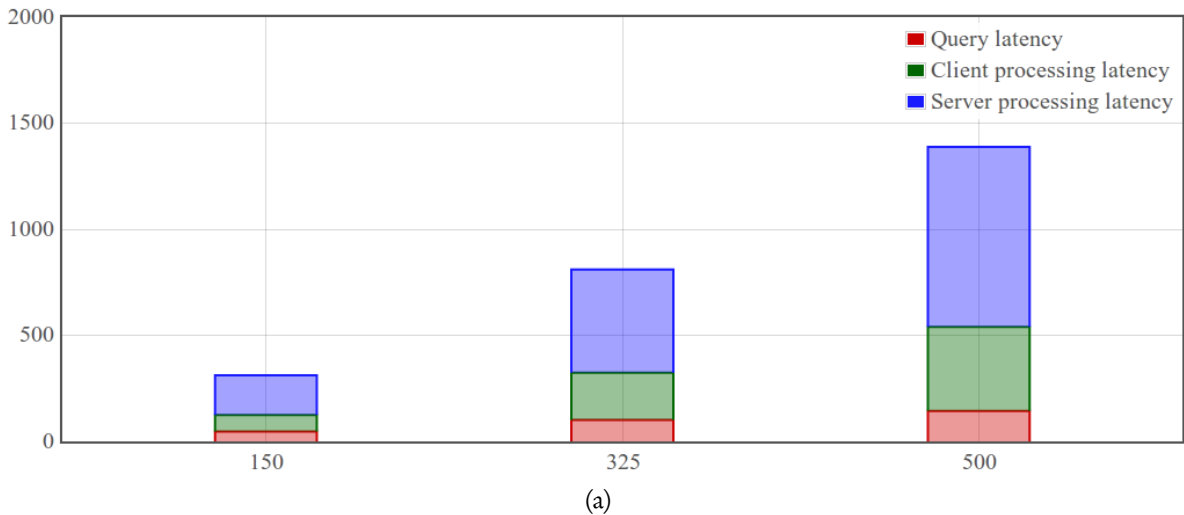


Figure 31: An overview for the latency in the *VEG* table for the not generalised versions of the binary 2- and 8-byte formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)

The implementation tested here gives an overview of the performance possibilities and limits with a vector-based map, implemented with open standards on the client. A lot of work has gone into the implementation and testing, and it has been difficult and time-consuming – especially since the technology is very new, and its use has not been explored in detail yet. The advantages together with further optimisation possibilities makes the single byte binary tile format a clear winner in these tests. As it is also superior in theory (see 13 and 17.2), this is not that surprising, although the confirmation by testing makes the assumptions definite.

The test’s main purpose, is to underline the importance of various aspects of the implementation of web maps, that has not been equally important with raster-based maps. The

increased complexity that involves both client and server, the different data formats and the importance of generalisation are extremely important to get right for a successful vector map implementation, and the tests confirm this. Even though the accumulated latency in the tests are not critically high, a production environment will have added complexity which will impact the latency. It is therefore important to reduce latency where there are major optimisation possibilities.

The responsiveness requirement for users to feel that they are manipulating/using directly in real-time (this is typical for GIS applications where spatial objects are edited and/or queried, perhaps to a lesser degree simple web maps) is 0.1 second[197], and the results shown here does not qualify for real-time using that criteria.

Since users are generally less tolerant of unresponsive desktop applications[197] compared to web pages[198] (i.e. information retrieval), one could perhaps argue that users are used to waiting for web maps, as currently, tile-based raster web maps are not instantaneous (i.e. one can observe that the tiles are rendered). However, to avoid user abandonment, one should strive for real-time operation. The results present several optimisation possibilities, which might close the gap, to achieve real-time, vector-based web maps.

Part IV.

Conclusion

18. Future work

The number of issues that a vector-based web map need to solve, is large. This work has proven that the choice of data format when exchanging vector data has a noticeable effect on the end performance (see Section 17.4). The importance of generalisation of vector data has also been documented by the performance test results, together with the varying processing overhead imposed on the client and server by the choice of data format. However, there is a lot more that needs to be researched at individual and combined levels to be able to conclude definitely on whether a complete, full-featured vector-based web map can compete with its raster-based counterpart.

The test implementation only has one layer of data, because the amount of generalisation work needed to create multiple layers of data would not be achievable in the relatively short timespan of this thesis. Such a model would be an advantage with regards to performance at smaller map scales, where, in the implementation presented here, the amount of tiles increases when the map scale is decreased – ideally this should remain fairly constant like is done with raster-based tiles (see Section 3 and 17.4). The amount of data would obviously become smaller as well, also increasing performance. The results show that some tasks have the possibility for optimisation, e.g. by rewriting the software in a more efficient language,

or more aggressive caching, which needs more exploration.

There are lots of general optimisation possibilities that needs more research. New technologies such as WebCL[199] would be interesting to explore, especially for converting coordinates from the local tile coordinate system, to the global coordinate system, which is a simple operation of addition and subtraction on a large amount of coordinates, and therefore is a prime candidate for parallelisation with GPGPU-techniques[200] (General-Purpose computation on Graphics Processing Units). To be able to render tiles in the background, and not blocking the main user interface, the use of HTML5 Web Workers (see Section 12) on the client for computational-heavy tasks should be explored. It would also be interesting to look at lossless compression algorithms, and the impact they have on data size and latency for both text and binary data formats – however, as there are not currently suitable decompression algorithms for the client, in JavaScript, it will need to be implemented, which is a sizeable task (see Section 7.2).

A large part of the web application when providing vector maps on the web, is the client. The functionality and performance needs more research for exploring the possibilities, advantages and disadvantages with vector maps. Currently there is parallel work on the topology of vector maps on the web, researching e.g. the stitching of features across tiles[201]. Clipped features is a major problem when the map is used interactively, which should be combined with this work for a more complete solution.

Conclusion

The emerge of HTML5 is helping to accommodate the increasingly heterogeneous environment we find ourselves in, with the adoption of hand-held and mobile devices alongside the traditional desktop. The open technology that comes with the HTML5 standard – for example Web Sockets and inline SVG – closes the gap between the desktop application and the web application, and enables us to mimic a desktop application more closely in a purely standard HTML browser environment. The JavaScript language have also received language extensions such as Typed Array, and the JavaScript engines in browsers have been greatly optimised to increase the performance and minimise the difference between the web application and the desktop application.

Efficient and usable vector maps in the web browser using open standards have long suffered from a lack of technology that make the process efficient enough, making the efficiency and usability near impossible to achieve. Because the architecture that uses vector data instead of raster data puts a higher load on the client, the browser needs to have a certain performance, and since the web originally was not intended for fully featured applications, this was simply non-existent.

Therefore, while current maps are mostly using raster images and technologies built on the technology of raster image maps, vector maps are becoming an increasingly relevant competitor. Raster maps have, among other reasons, gained popularity because they do not

require third-party applications or plugins in the browser to be able to work, and they are reasonably fast. The demand for more functionality while retaining the current performance is very hard to do with raster maps, where the server does all the processing, and the pressure and demand for vector maps is therefore increasing, where some tasks are offloaded to the client, and where more interaction with the map is possible.

An implementation of a vector map server and client have been created, with multiple implementations of vector data structures, using the newest – and, in the author’s opinion, best suited – web technologies available at the time of writing. The impact of generalisation and data formats have been discussed thoroughly, and is found to impact the efficiency of vector maps in a major way. The implementation have been thoroughly tested to determine both the best data structure for most efficiency, and also tried to assess the expected performance in a real environment. The results are positive, and suggest that it is possible to achieve performance that either matches or exceeds the current popular raster-based maps.

Appendices

A. Attachment 1: Master Thesis Assignment

MASTER DEGREE THESIS

Spring 2012

for

Student: Mats Taraldsvik

The future of web-based maps: can vector tiles and HTML5 solve the need for high-performance delivery of maps on the web?

BACKGROUND

Current map providers employ raster tiles to deliver maps on the web, but the size requirements and static nature of images makes raster tiles an inflexible solution.

Using vector tiles for web maps is gaining popularity, because the vector data size is smaller, and the data format is not restricted to images.

New technology like HTML5 also gives opportunity to develop platform independent solutions for rendering vector data, which was not previously possible without platform-dependent plugins or extensions.

Since vector-based GIS systems dominate the desktop market, it would be interesting study of how vector-based maps and how the data format between client and server can be tweaked to achieve better performance and minimize data size.

TASK DESCRIPTION

- Discuss the motivation for improving the established method of delivering web-based maps through raster data formats with vector data formats, and its performance implications.
- Explore different approaches of delivering high-performing vector maps to clients only depending on HTML5 technology and established web standards.
- Develop example implementations, and measure how these fare against current raster delivery methods.

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

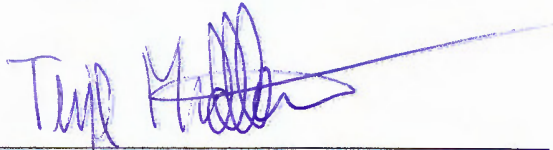
The work on the Master Thesis starts on January 16, 2012

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

Other supervisors: Dr. Ing. Rune Aasgaard, Norkart Geoservice AS

Trondheim, January 16, 2012. (revised: 30.05.2012)



Professor in charge (sign)

B. The implementation and source code accompanying this paper

A *major* part of the thesis work consisted of software development, and extra care was taken to make this source code available to anyone interested in confirming the results, or looking at the source code (see Section 1.1).

dataformats-example

This folder contains examples used in Section 13, and the actual spatial data was taken directly from the data set used to test the vector map implementation. This extra work was done to create realistic examples, and better understand the relationship between the different data formats (whereas if completely random examples were used, this would not be as clear). See Appendix D for more details.

geoserver-wms-client

This folder contains the client implementation of the current map standard – the server is a GeoServer instance, that had to be set up manually (GeoServer does not have proper automation tools), which took a week to complete. Because of this, the source code for the server instance, is not that interesting.

The folder contains the map client, which was used to test the implementation, and also the performance test (the performance-test folder) and also the result data and generated graphs.

tilestache-polymaps-client

The TileStache vector tile map server and generator was tested with PostGIS as a backend, but was too slow and relied on old technology (HTTP), which is why the testing was not included in this paper. Nevertheless, it is included for completeness.

websockets

The folder contains implementations of client and server (the websockets subfolder) for the different data formats, and tile generation. It was the early implementation while the author was familiarising himself with the technology and data structures.

websockets-new

The individual implementations in the websockets folder were combined and re-architected into a single client and server implementation. This re-architecture of the implementation was time-consuming and hard work, but the resulting code is much simpler to inspect, and also ensure that the different data formats are tested on as equal terms as possible.

The performance-tests folder contains both the data acquired when testing, the processing of the data (median filter etc.), as well as the generated graphs and data visualisations.

To start the server, run `runserver.py`. The server will listen on port 9090. The client will work on a local machine, without a web server, as it uses only the HTML5 open standard and JavaScript. It should be able to connect to the server if the ip-address in `client.html` is correct. The data format is chosen with a configuration parameter in `client.html`.

C. HTML5 Support in Web Browsers

HTML5 is a "new" standard. The support status changes at such a rapid phase, that instead of providing a static (and quickly outdated) status report in this paper, one is encouraged to use online resources instead. Sites such as caniuse.com[202] provides an up to date overview of browsers with HTML5 support.

In the wake of lacking support and implementation across browsers, a handful of tools have appeared to check if specific features are supported in the browser – the most popular is probably Modernizr[203]. Libraries that backport several features for use in ageing browsers are also available, a great win for companies on the fence because of lacking support. Use of such tools is encouraged if supporting legacy browsers is a priority.

D. Data Formats Example

Extra measures were taken, and extra work was done, to give a realistic, and comparable, example of the various data formats. Some features used in the vector map implementation (see Section 17) were extracted from the dataset, and processed in PostGIS (on the tables "veg" and "VEG", which contains polygons and linestrings, respectively) (see Code Example 7).

Code Example 7: SQL Query

```
SELECT
  ST_AsGeoJSON(ST_Simplify (geom, 10) , 2)
FROM
  "veg"
WHERE
  geom &&
  ST_SetSRID
  (
    box2d(
      ST_GeomFromText (
        'LINESTRING (590000.0_6645600.0 ,_590010.0_6645610.0) '
      )
    ),
    25832
  )
```

The resulting GeoJSON was then formatted into a single GeometryCollection, and converted to WKT, WKB, GML and SVG. Since the ESRI Shapefile does not handle GeometryCollection, the GeoJSON was split into two separate files, MultiPolygon and MultiLineString, and converted to shapefile. Complete sources are available in the source code (see Section 1.1 and Appendix B).

E. Large versions of the visualisations of the performance test results

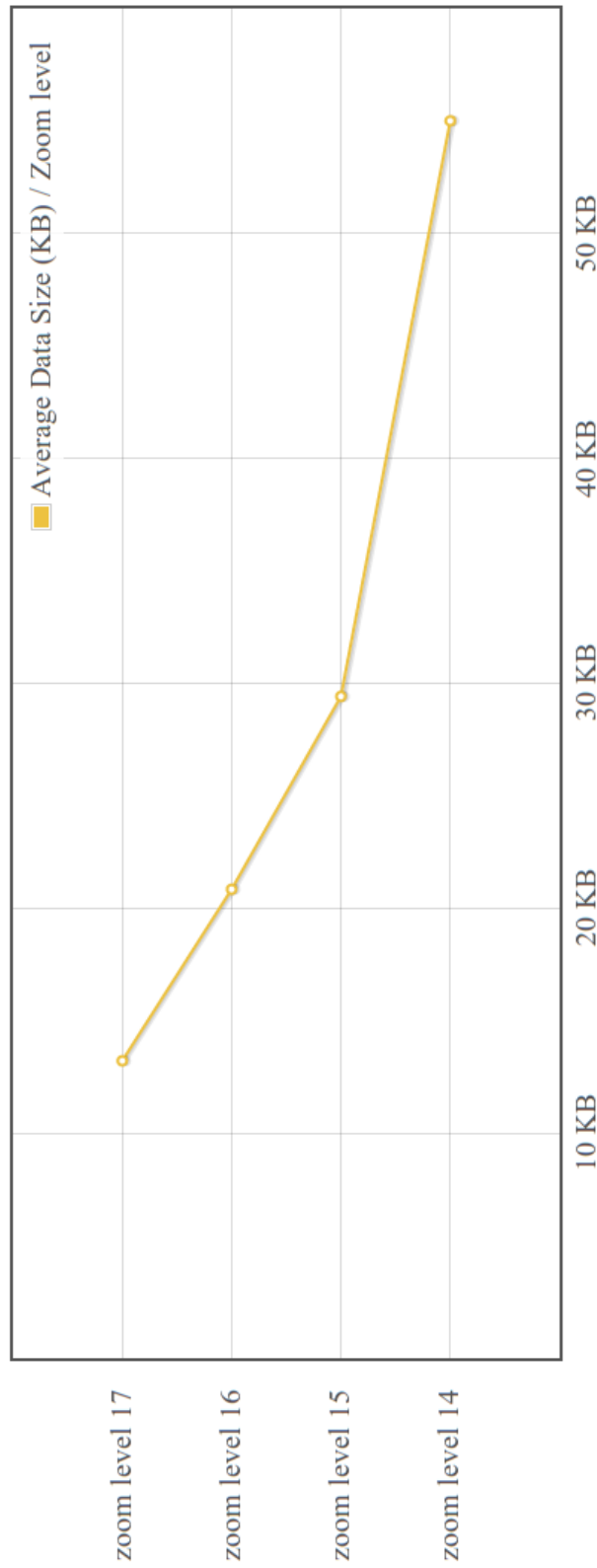


Figure 32: The average tile data size for different zoom levels in the WMS implementation.

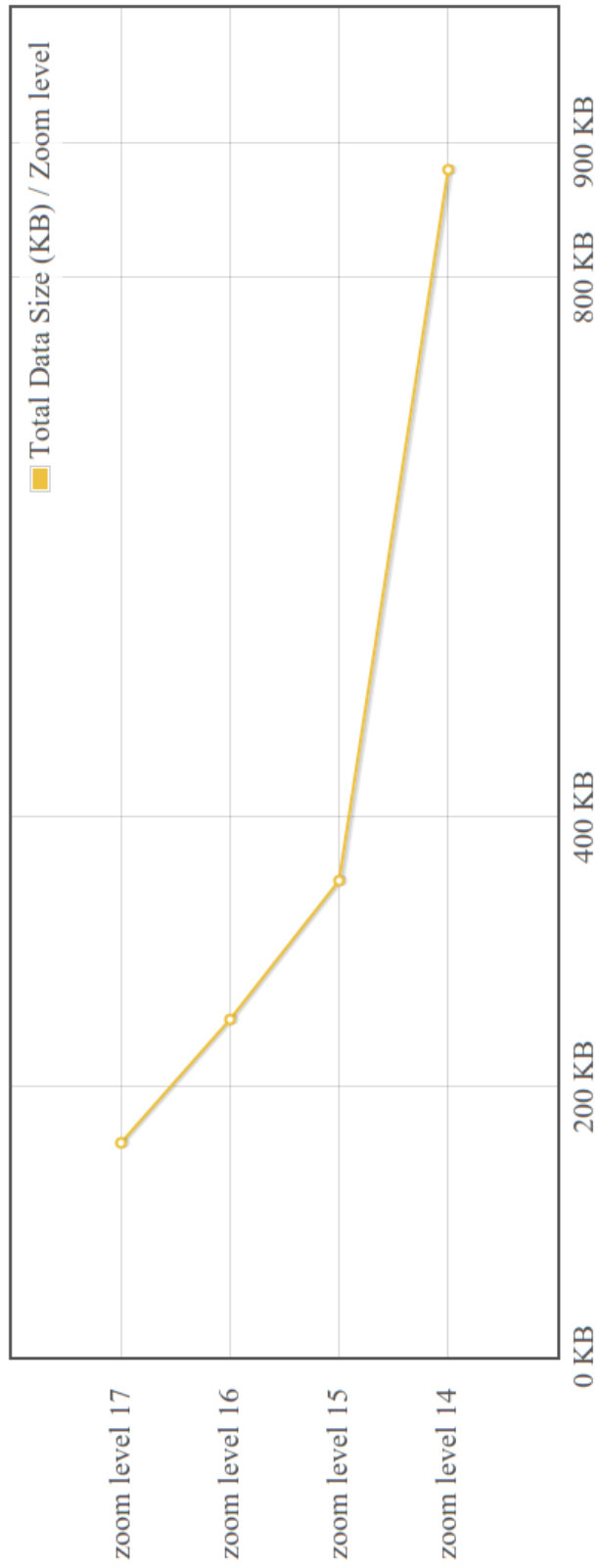


Figure 33: The total tile data size for different zoom levels in the WMS implementation.

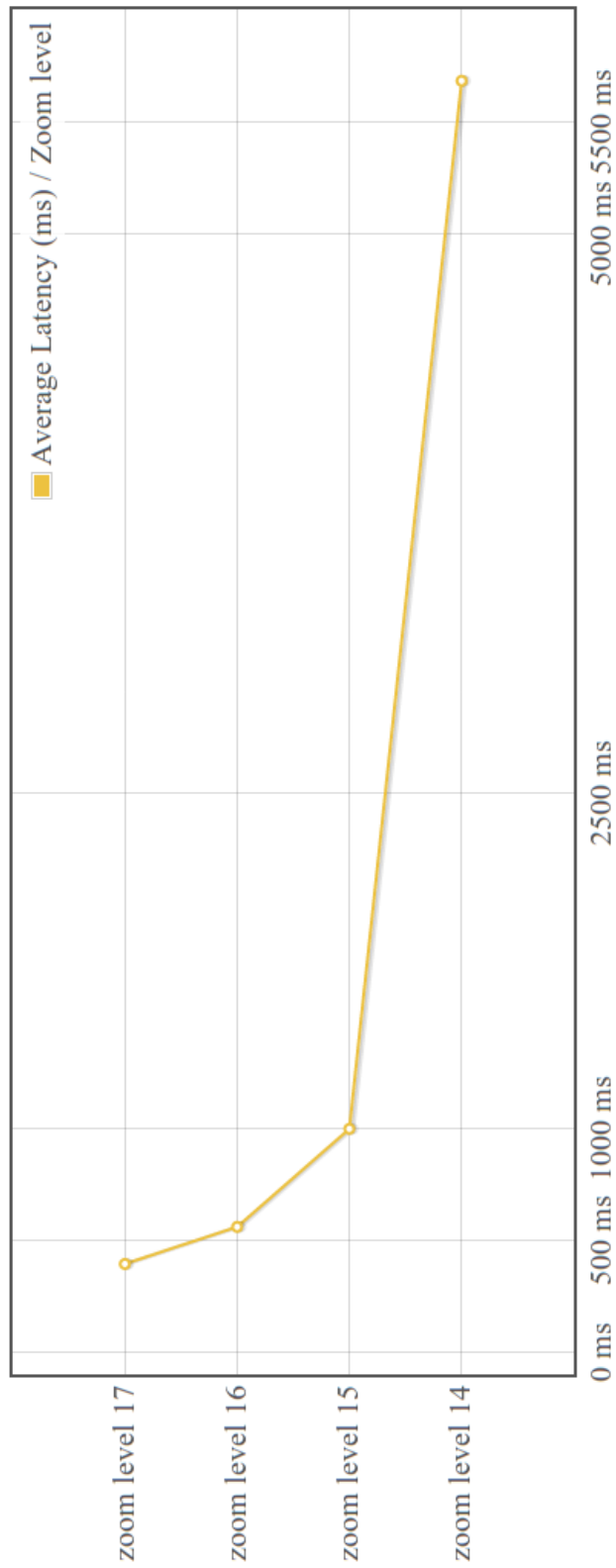


Figure 34: The average latency for different zoom levels in the WMS implementation.

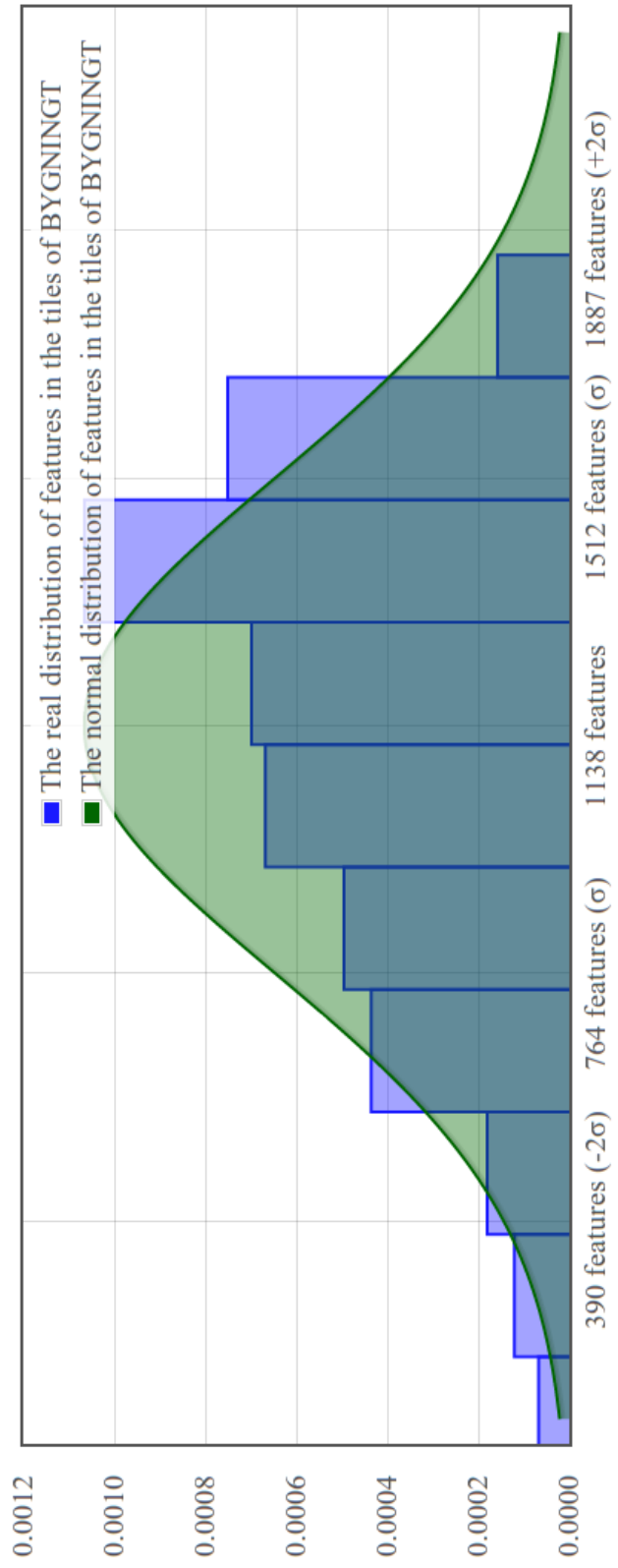


Figure 35: The number of features at an appropriate scale, fitted to a normal distribution, in the *BYGNGT* table.

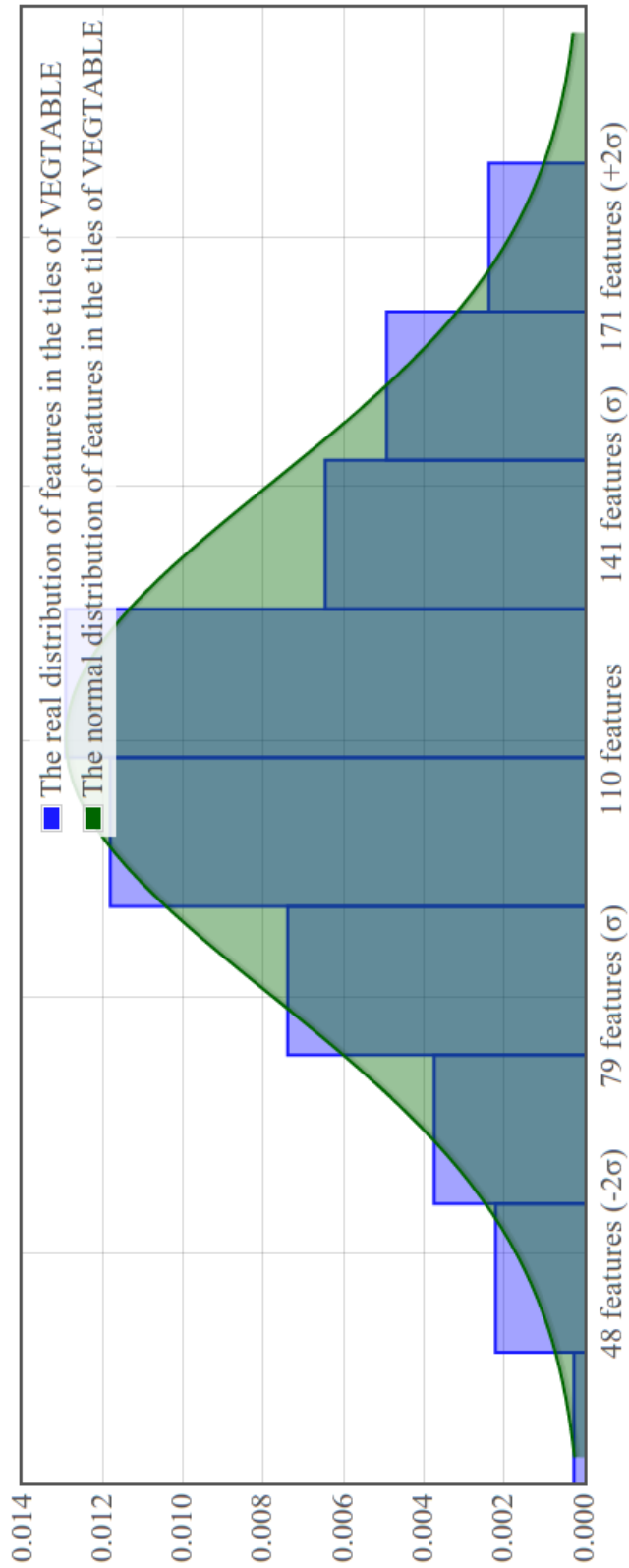


Figure 36: The number of features at an appropriate scale, fitted to a normal distribution, in the VEG table.

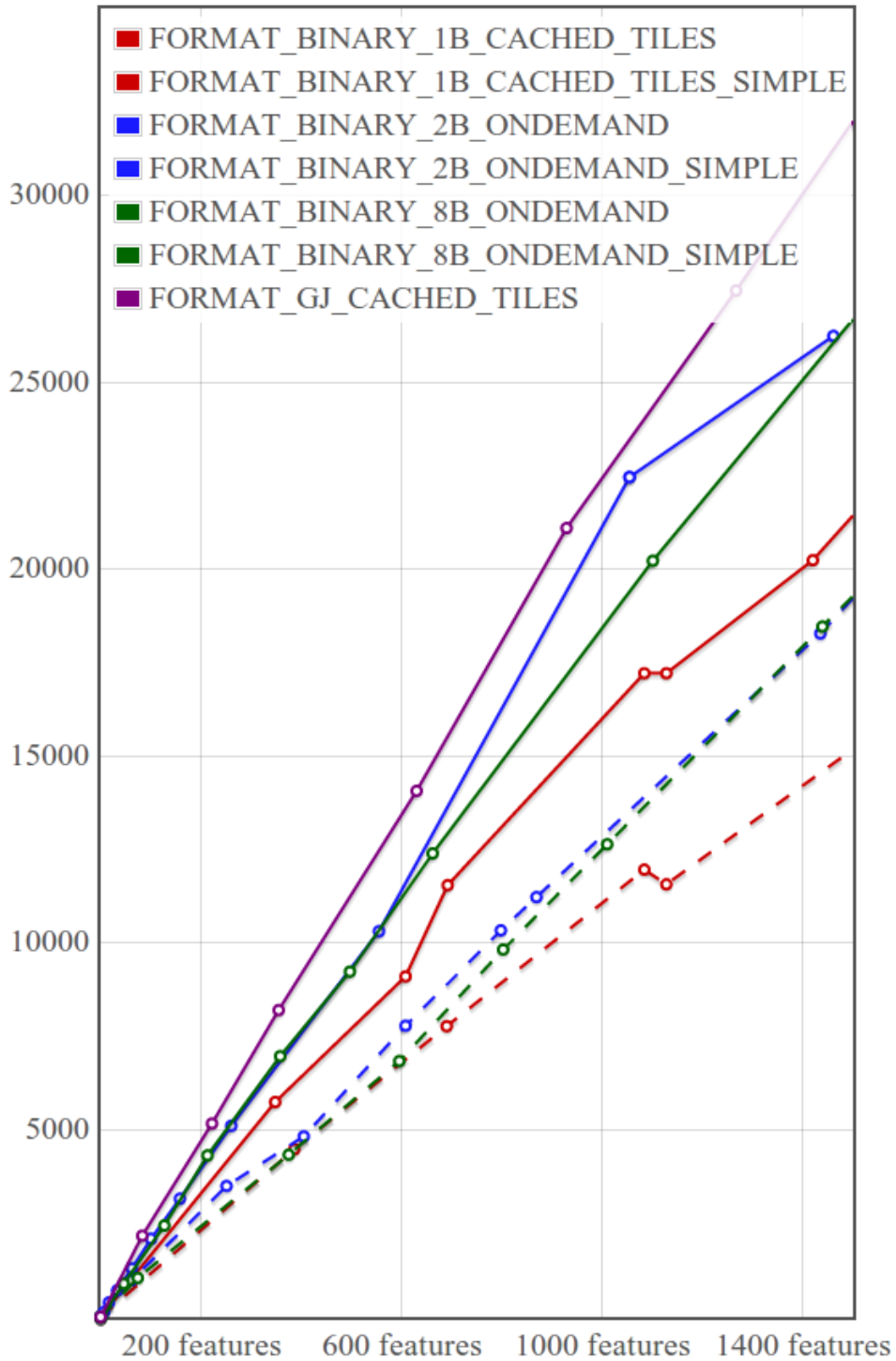


Figure 37: The relationship between the number of features (horizontal axis), and the resulting number of coordinates in the *BYGNING* table. Every generalised data format (dashed lines) consumes a lot less space in terms of number of coordinates, than the data formats that are not generalised. The generalised GeoJSON was removed, because of an error with the median filter.

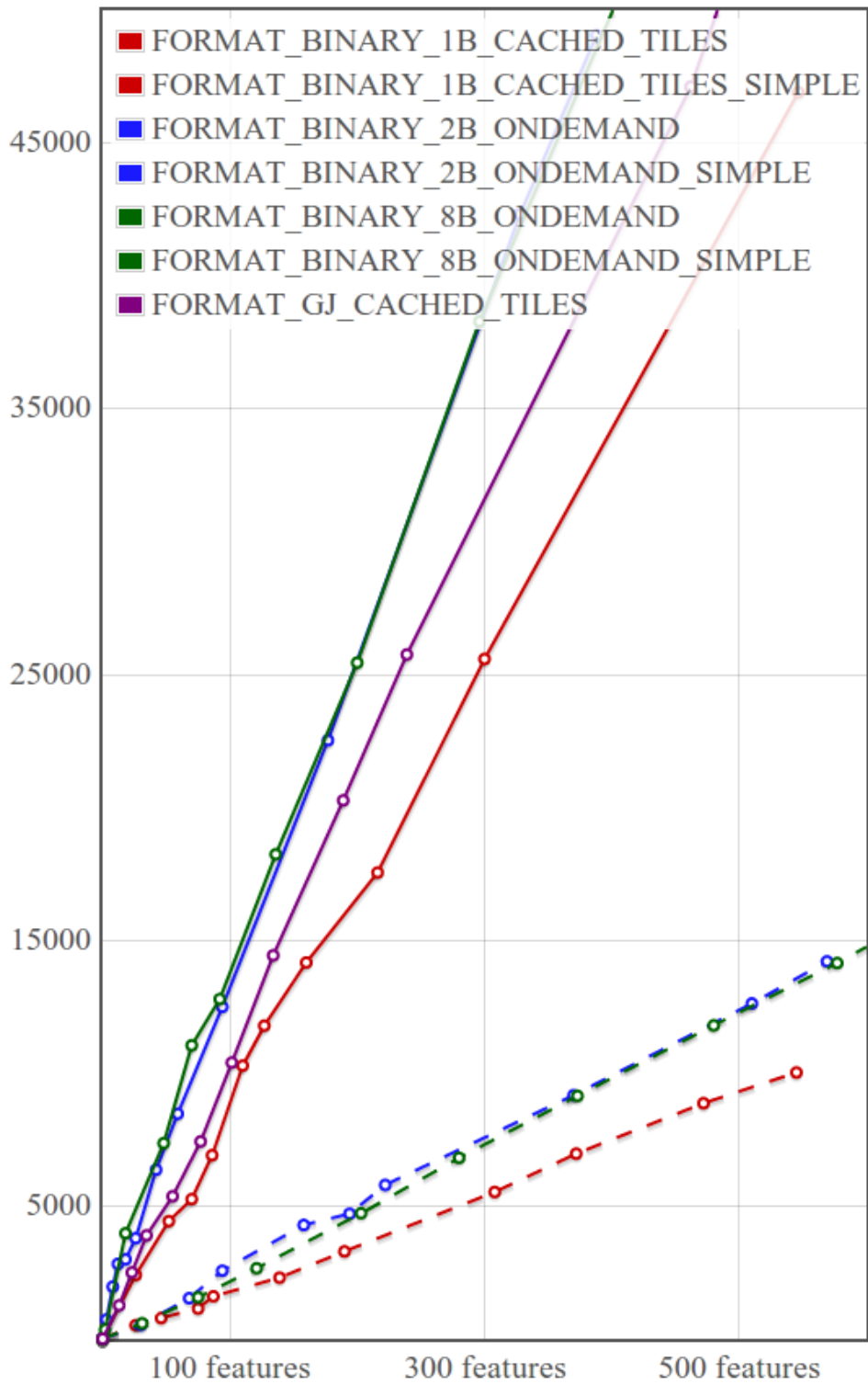


Figure 38: The relationship between the number of features (vertical axis), and the resulting number of coordinates in the *VEG* table. Every generalised data format (dashed lines) consumes a lot less space in terms of number of coordinates, than the data formats that are not generalised. The generalised GeoJSON was removed, because of an error with the median filter.

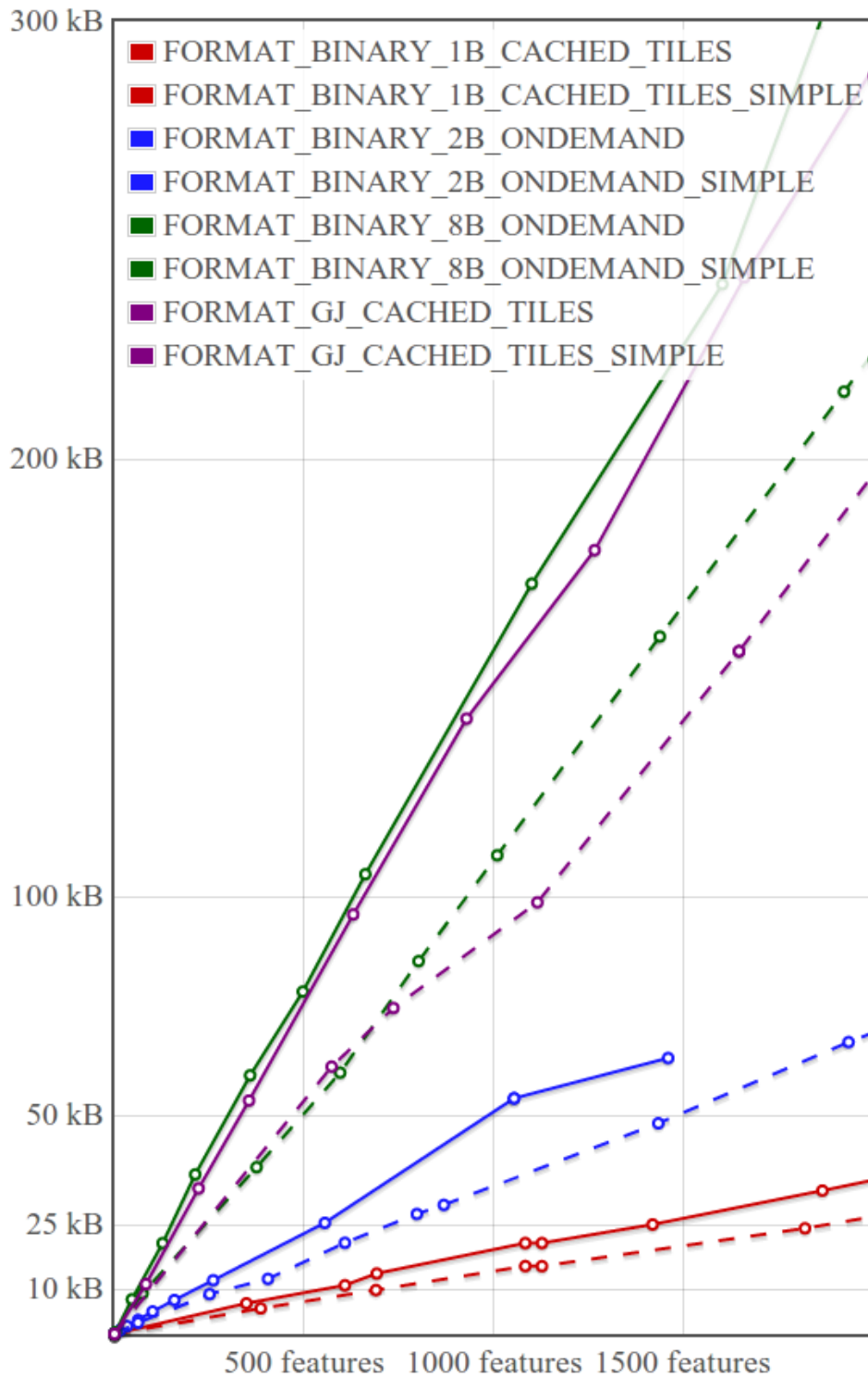


Figure 39: The efficiency of the data formats in the *BYGNING* table. Both the space efficiency of each data format, as well as the impact of generalisation, is visible. The generalised versions of the data formats have dashed lines. The horizontal axis represents the number of features, and the horizontal axis is data size in kB.

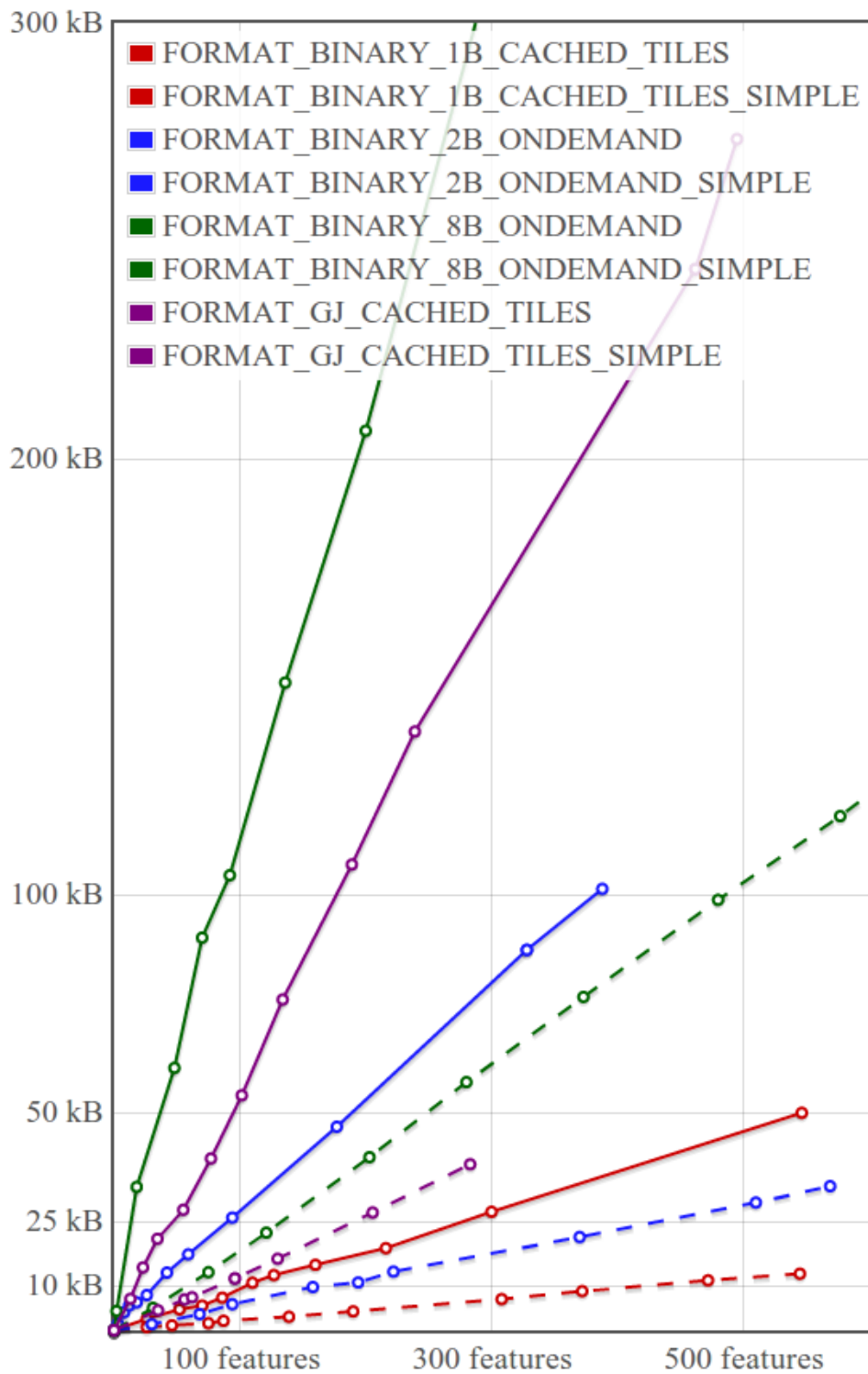


Figure 40: The efficiency of the data formats in the *VEG* table. Both the space efficiency of each data format, as well as the impact of generalisation, is visible. The generalised versions of the data formats have dashed lines. The horizontal axis represents the number of features, and the horizontal axis is data size in kB.

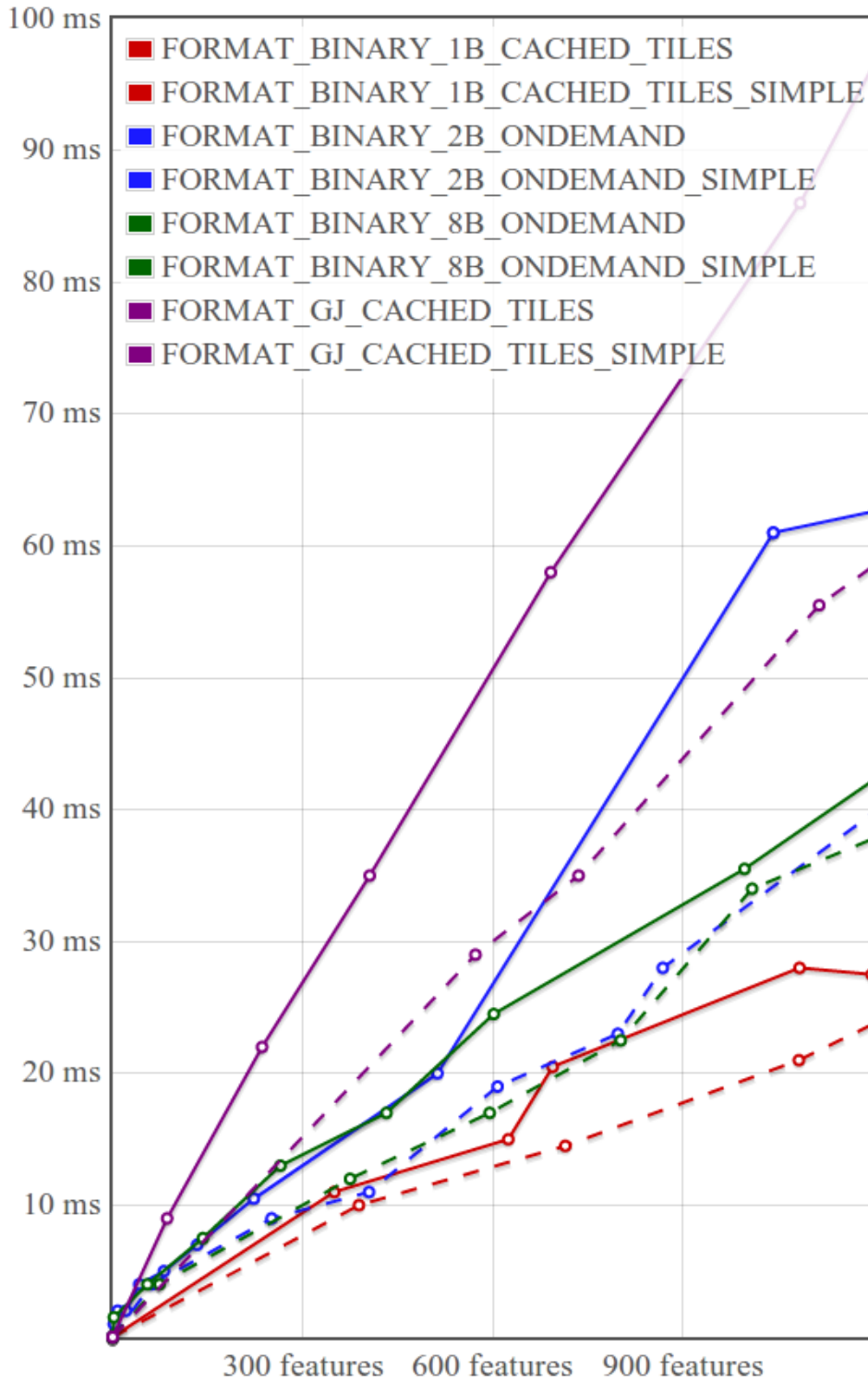


Figure 41: The latency of queries in the *BYGNING* table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.

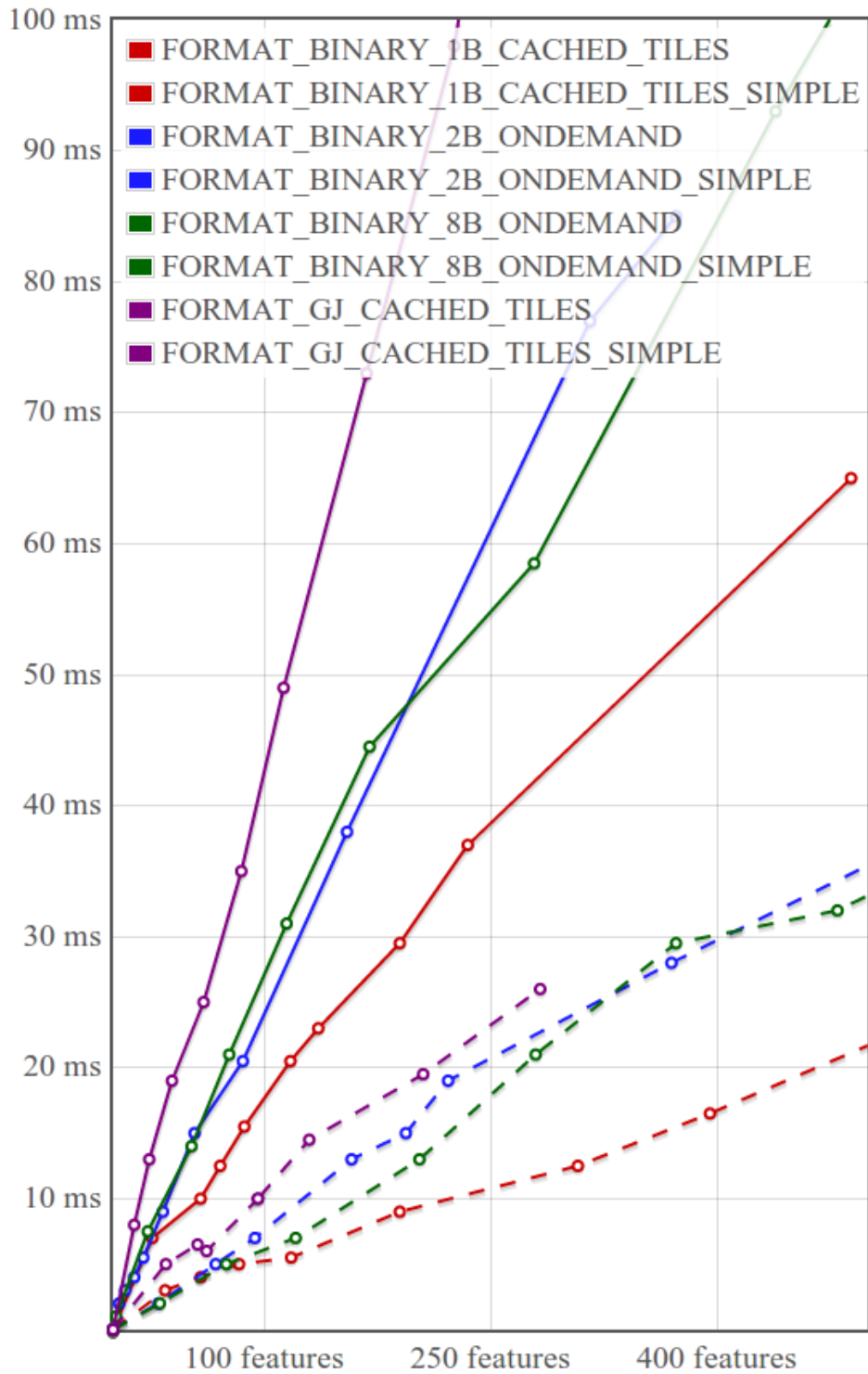


Figure 42: The latency of queries in the *VEG* table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.

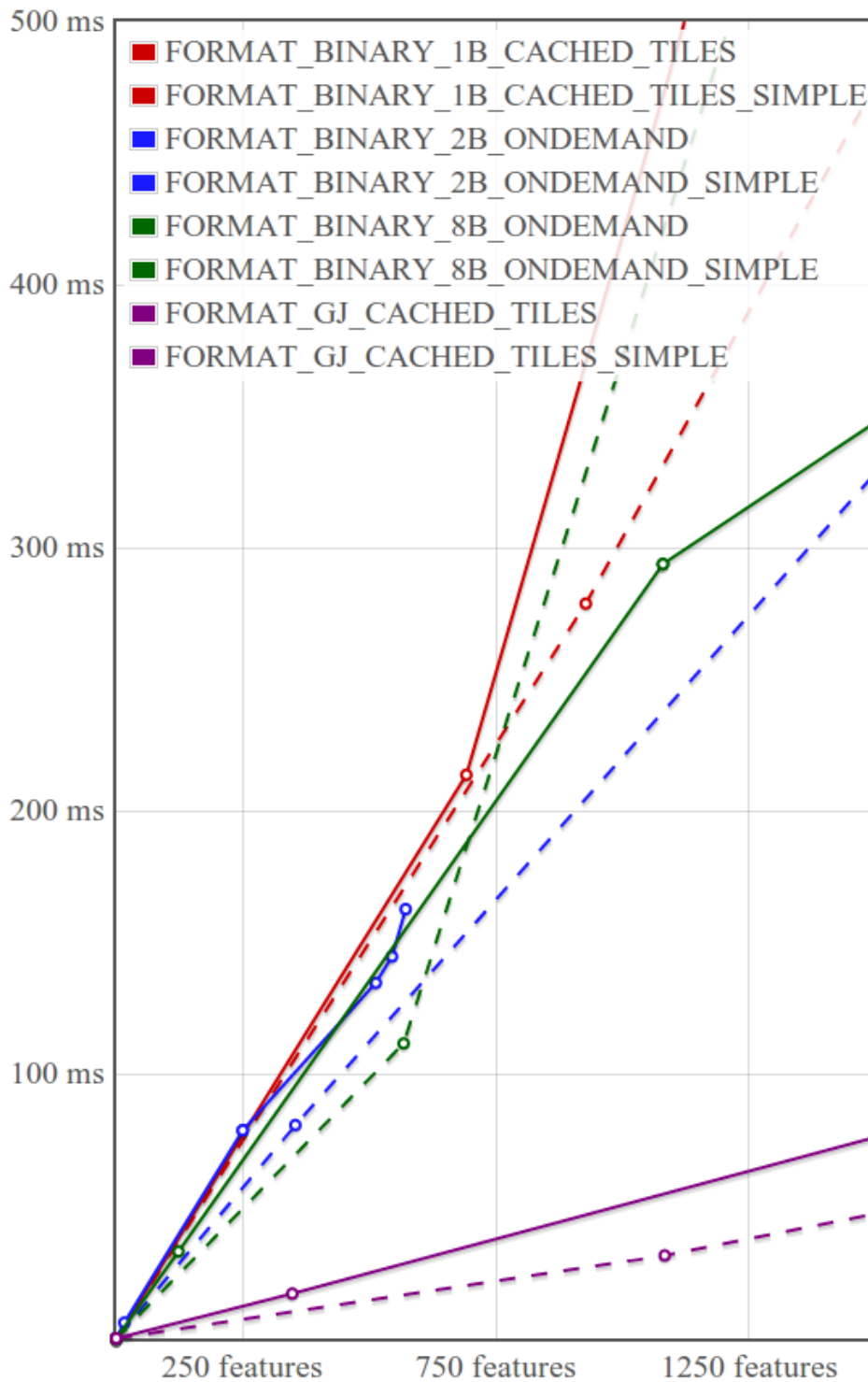


Figure 43: The latency of the server processes in the *BYGNING* table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.

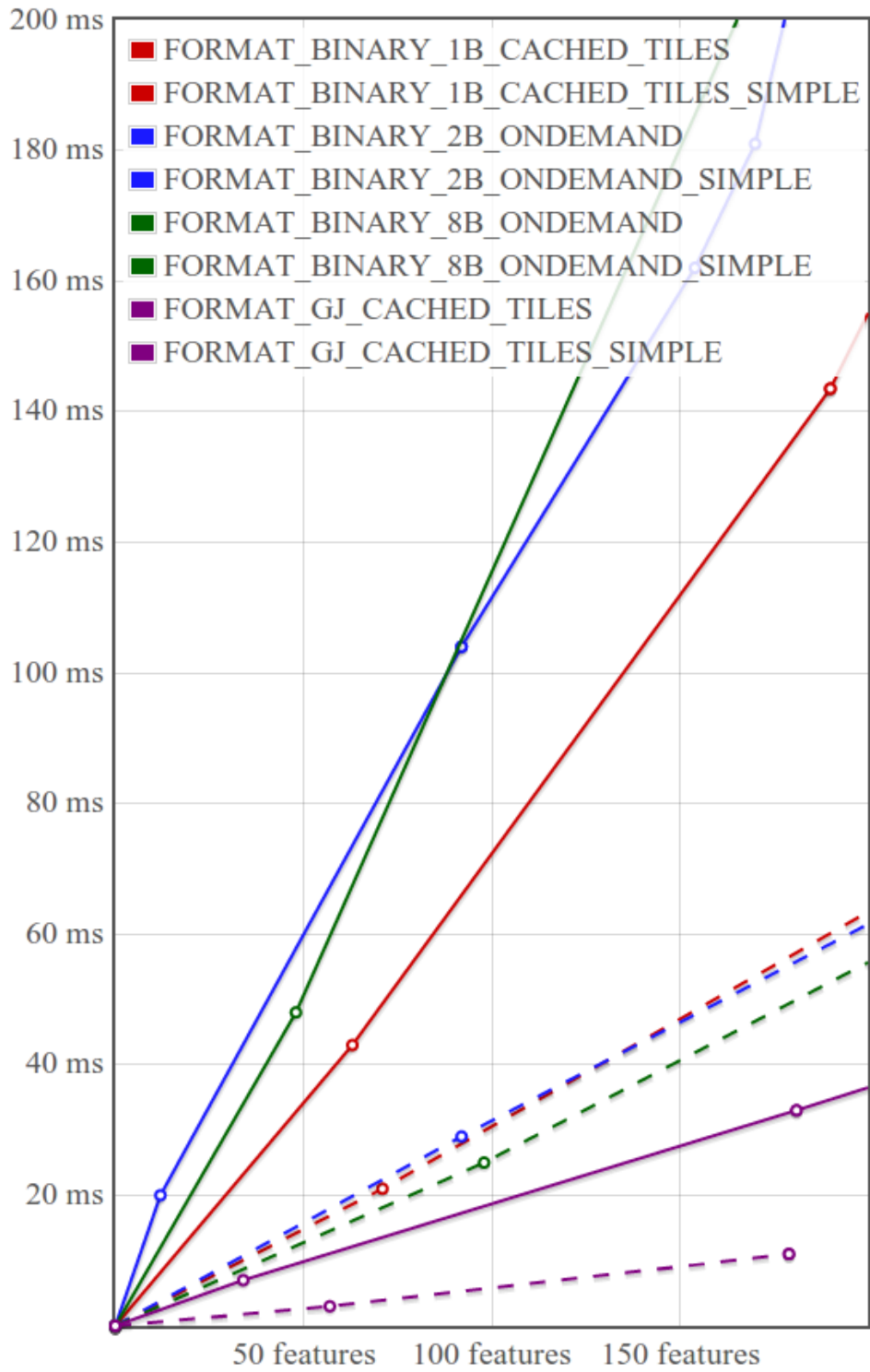


Figure 44: The latency of the server processes in the *VEG* table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.

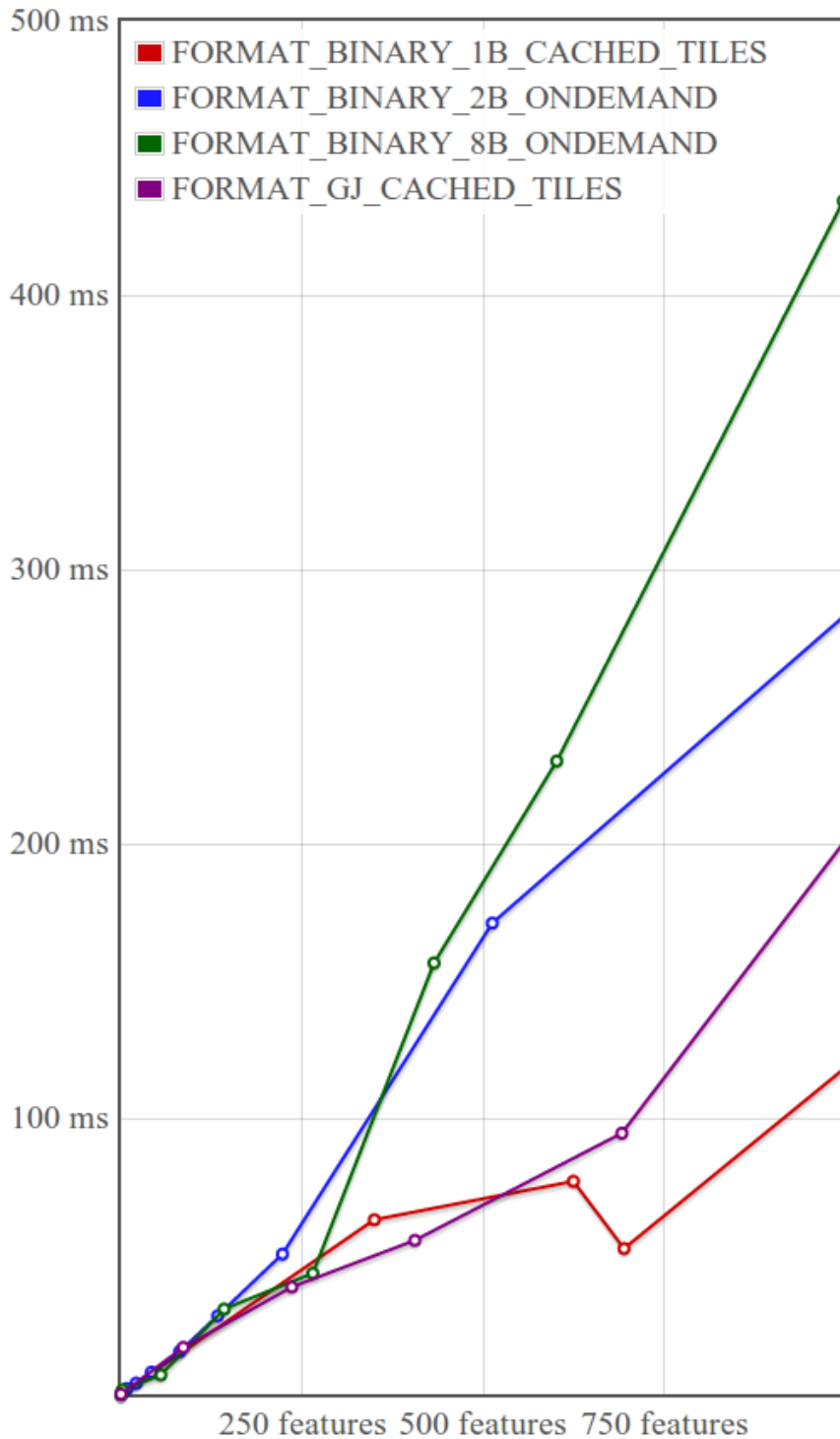


Figure 45: The latency of the client processes in the *BYGNING* table for the data formats. The generalised versions were removed to avoid clutter, as they did not add value to the graph, in the authors opinion. Note that there are some noise in the data for the single byte binary format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

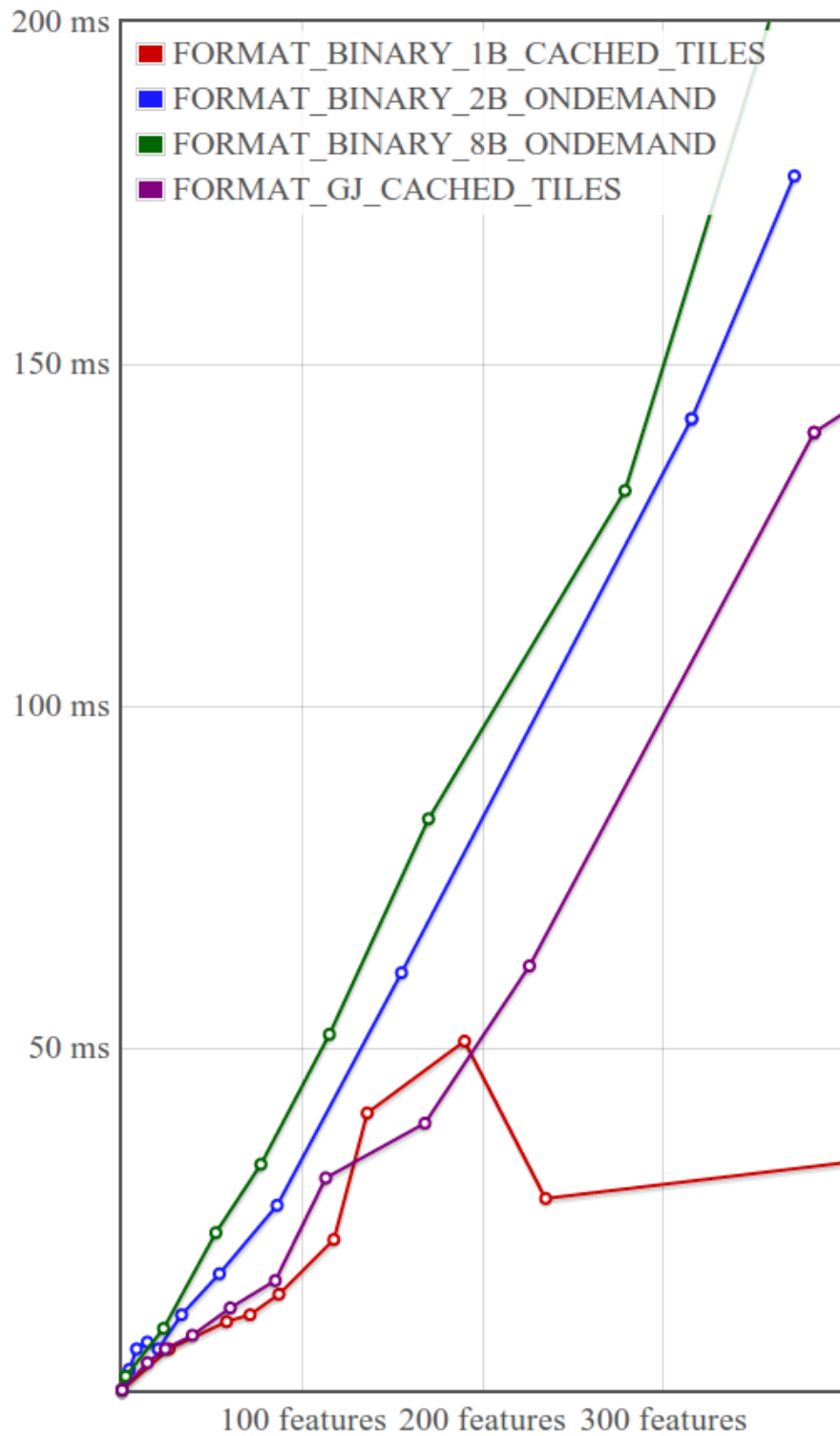


Figure 46: The latency of the client processes in the *VEG* table for the data formats. The generalised versions were removed to avoid clutter, as they did not add value to the graph, in the authors opinion. Note that there are some noise in the data for the single byte binary format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

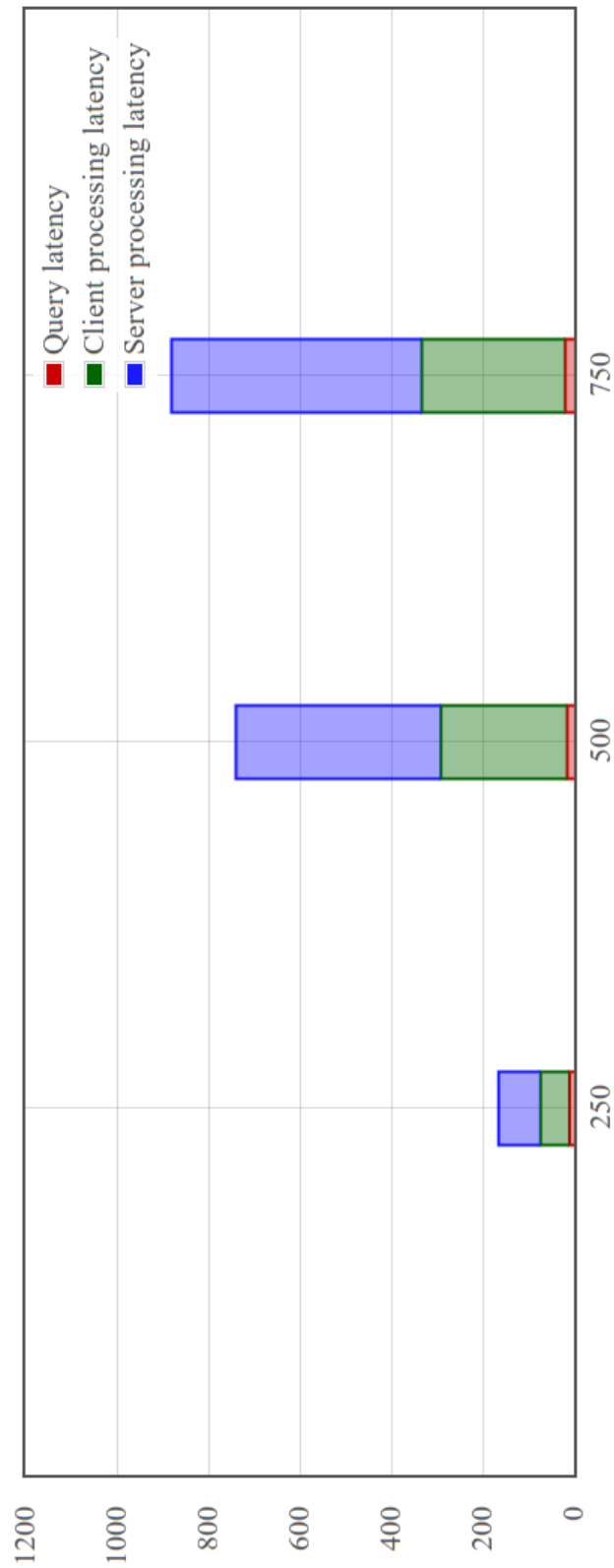


Figure 47: An overview for the latency in the *BYGNING* table for the not generalised versions of the binary single byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

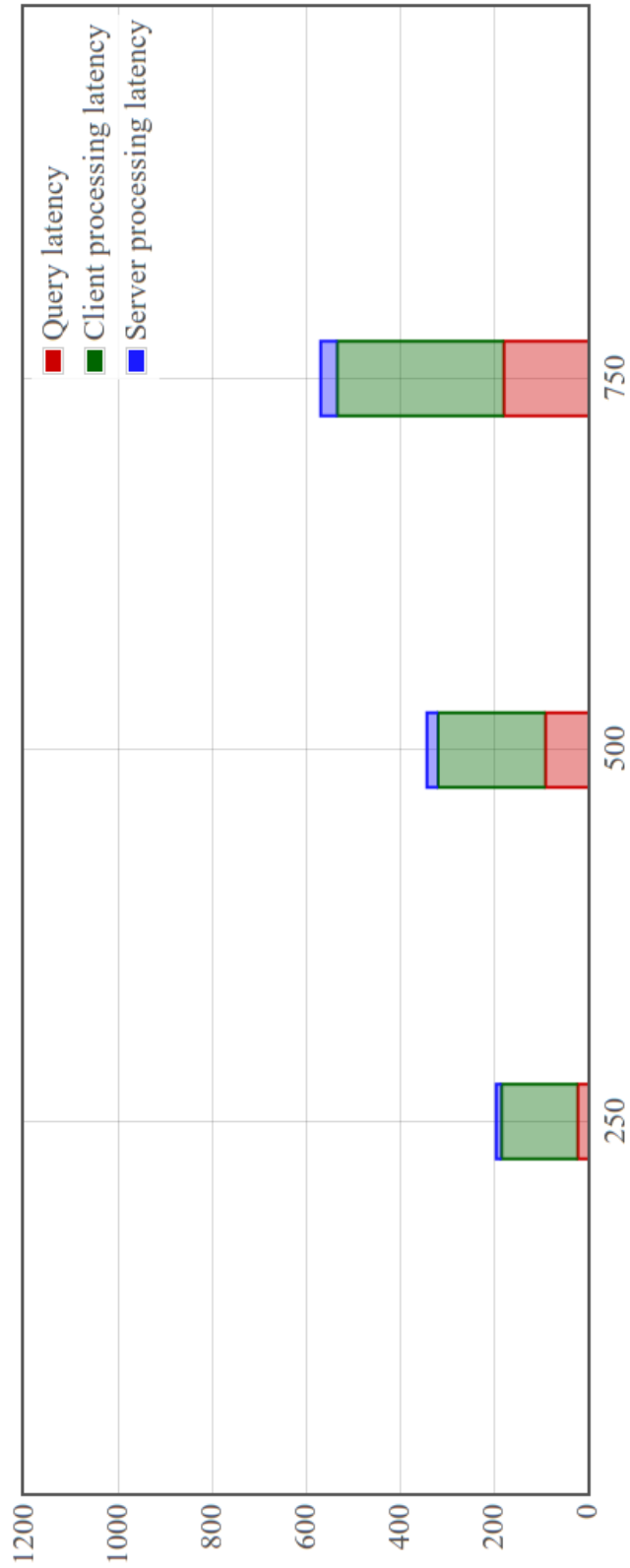


Figure 48: An overview for the latency in the *BYGNING* table for the not generalised versions of the GeoJSON tile format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

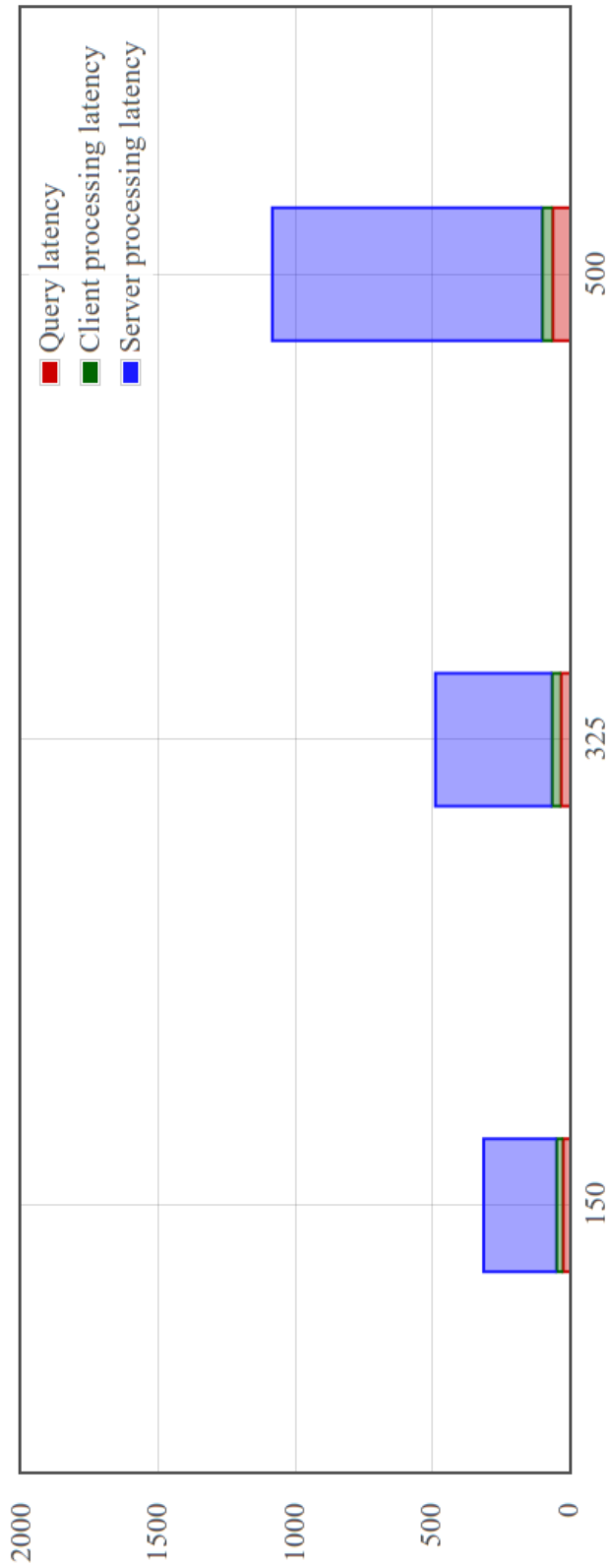


Figure 49: An overview for the latency in the *VEG* table for the not generalised versions of the binary single byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

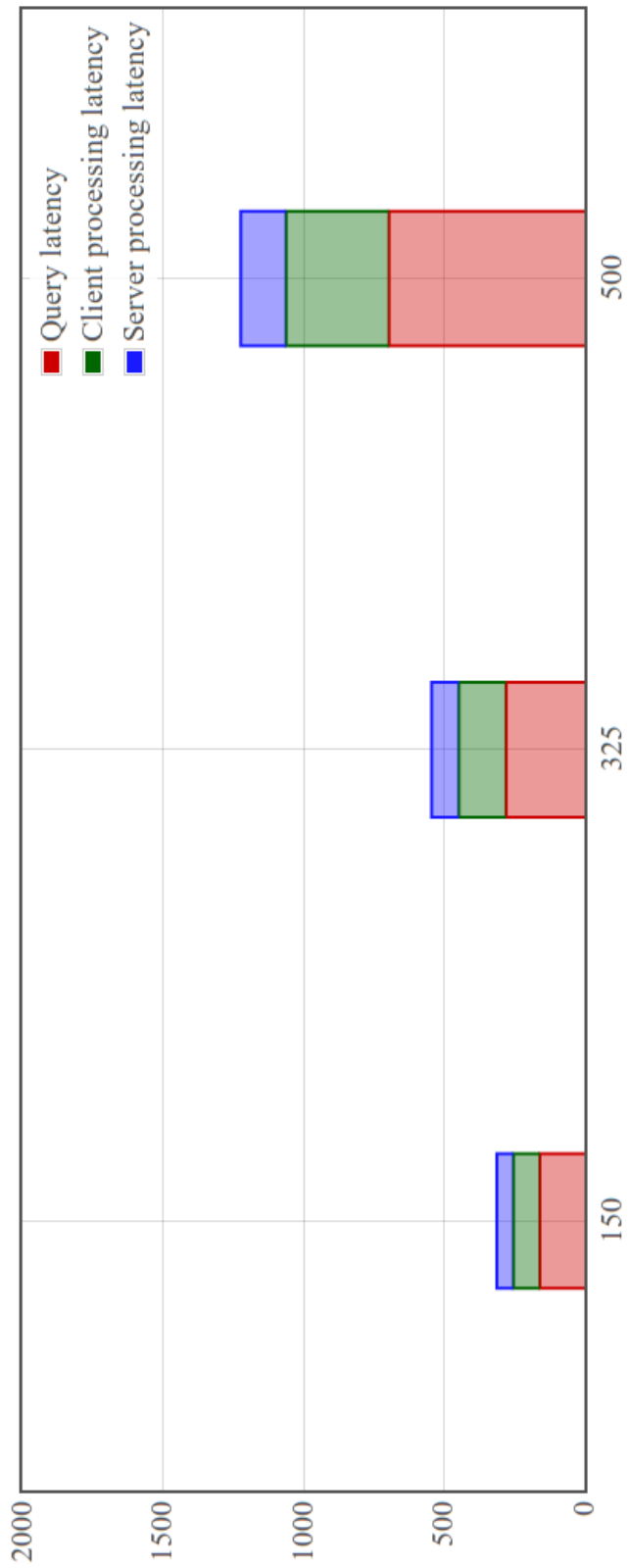


Figure 50: An overview for the latency in the *VEG* table for the not generalised versions of the GeoJSON tile format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

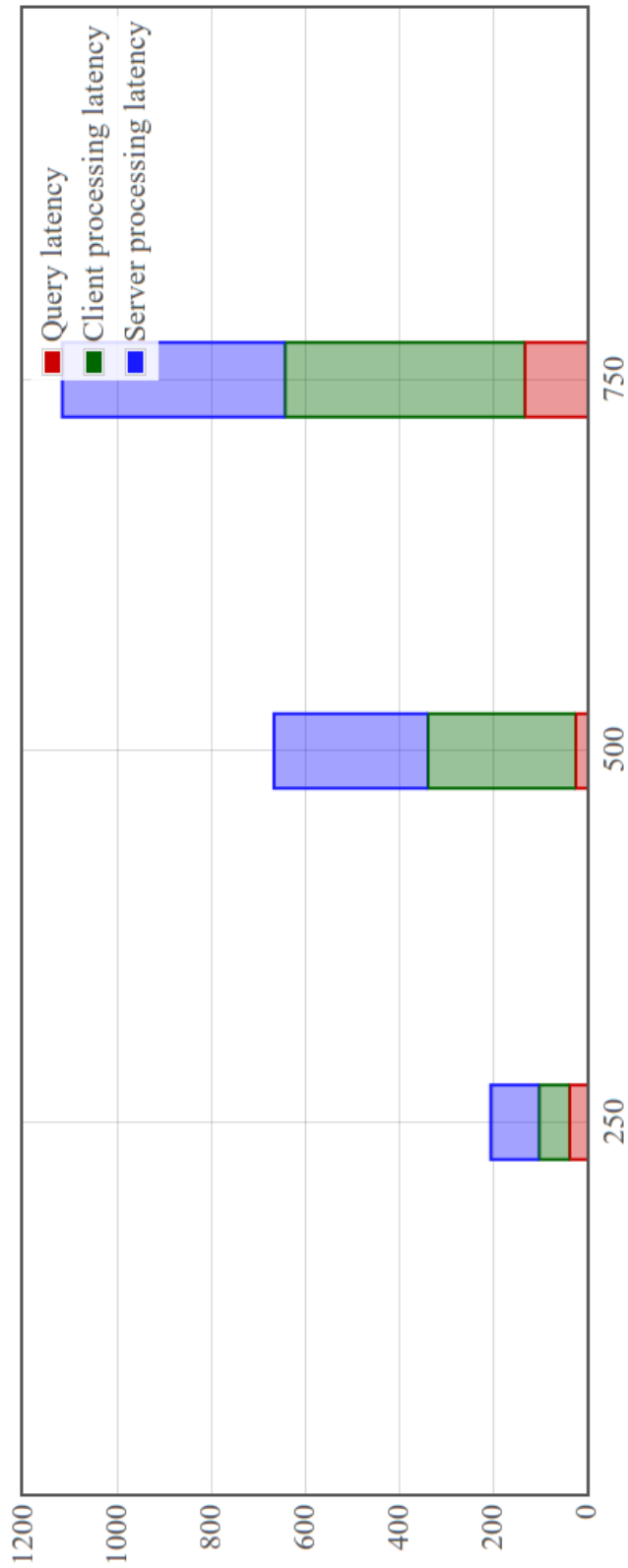


Figure 51: An overview for the latency in the *BYGNING* table for the not generalised versions of the binary 2 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

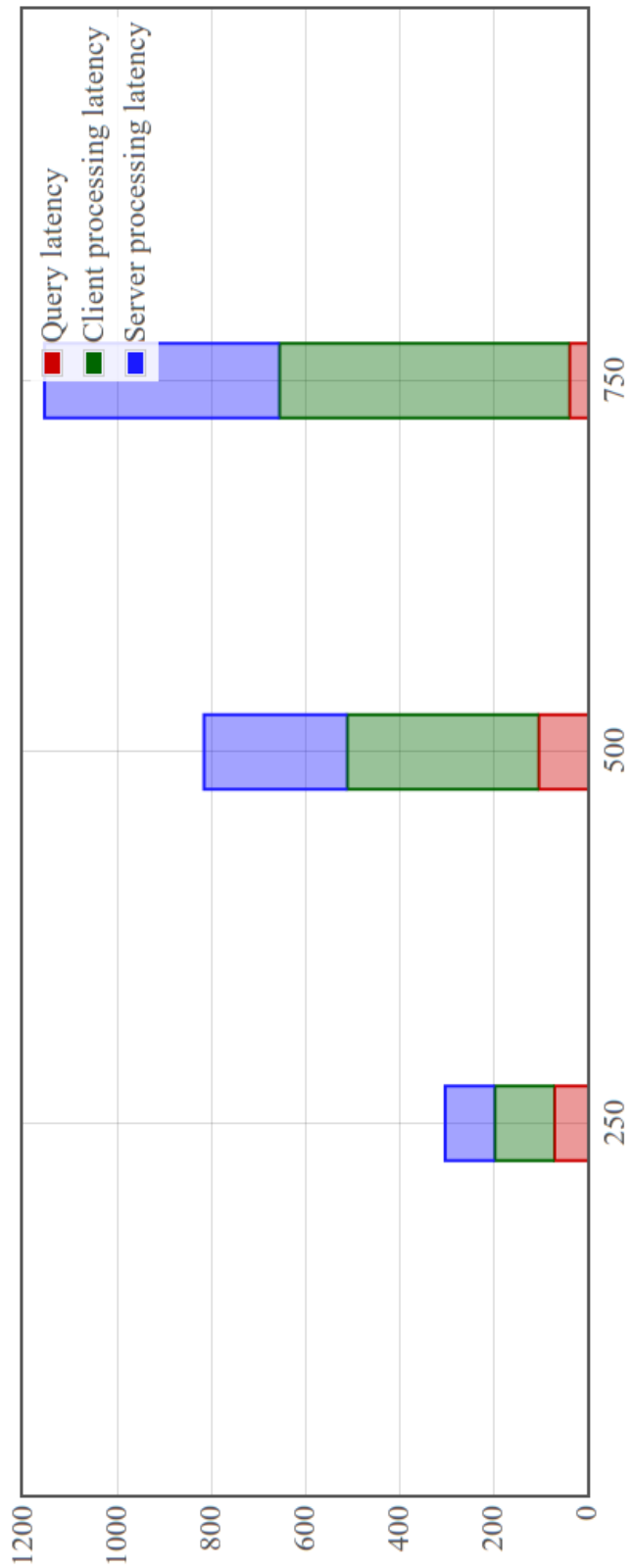


Figure 52: An overview for the latency in the *BYGNING* table for the not generalised versions of the binary 8 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

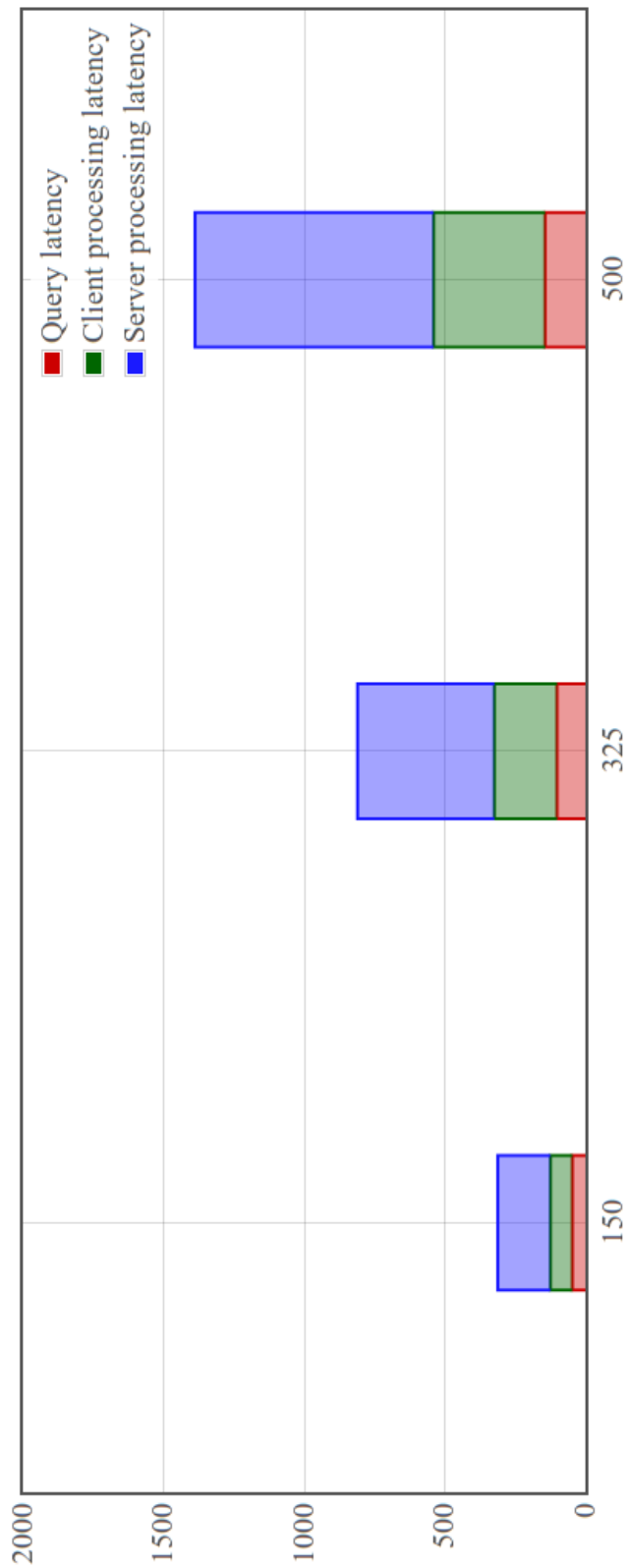


Figure 53: An overview for the latency in the *VEG* table for the not generalised versions of the binary 2 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

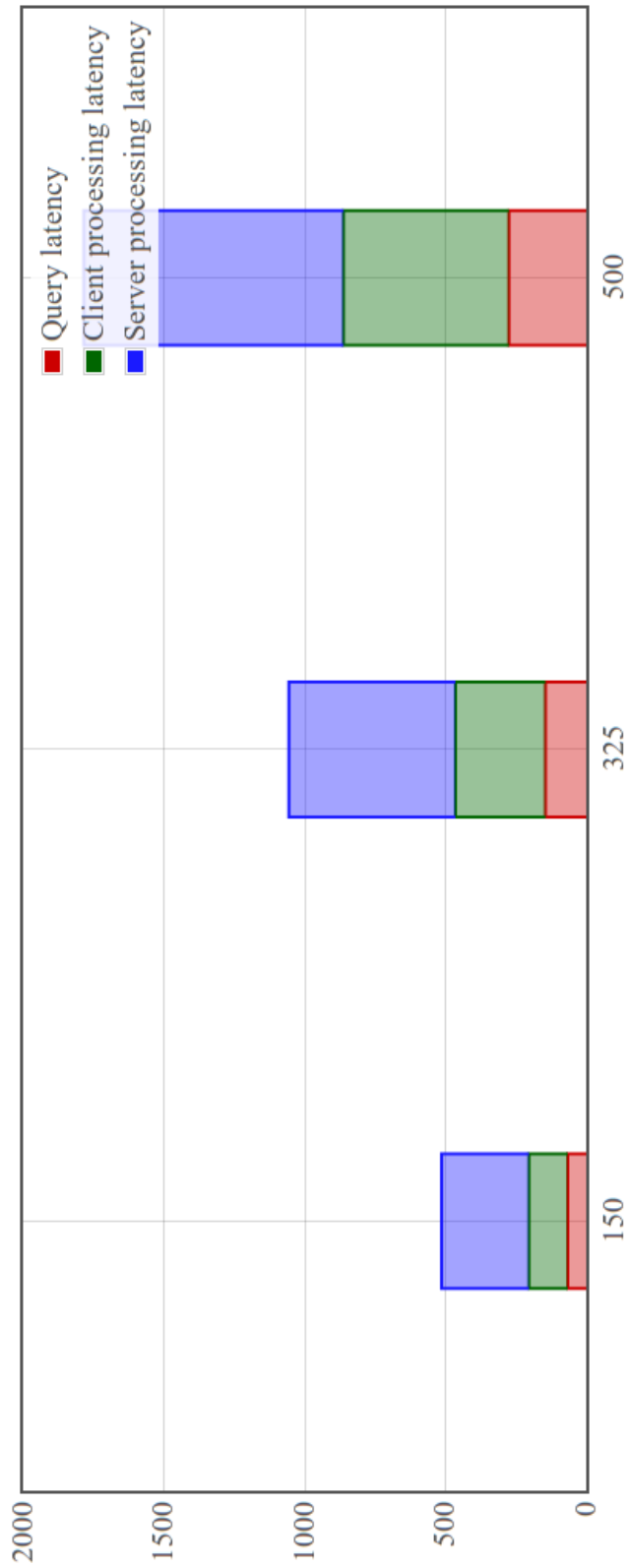


Figure 54: An overview for the latency in the *VEG* table for the not generalised versions of the binary 8 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.

List of Figures

1.	The Xerox PARC Map Viewer with a map of Switzerland	7
2.	Tiling works by dividing the map (a) is into tiles (b) that are loaded independently on demand. For maps that support zooming, the tiling process is done for every map scale (c).	9
3.	Identical tiles are stored once on disk, and referenced when displayed, to avoid redundant data and to save space. For water tiles, which there are many of in some datasets, this technique can save a lot of disk space.	11
4.	Usage of Delaunay triangulation to merge two areas, with a specified triangle size tolerance (b) – and without this constraint (a). [54]	13
5.	Progressive transmission with raster images works by starting with a coarse image (a), and gradually transfer images with higher resolution which are replaced with the coarser image (b), finally resulting in a high-resolution, full-detail image (c).	16
6.	On the server, the original object(s) of full detail is generalised into multiple incremental models, from coarse to full detail ((a), (b) and (c)). The initial object(s), which are coarse and provide limited detail (a), are sent to the client (d). Either automatically, or as the user increase the map scale, the object(s) are upgraded in a number of steps (b), transferring only additional points that increase detail (e), until the original detailed object (c) is transferred to the client (f), providing equal amount of detail on both server and client. . .	17
7.	Example of a line (a) and the line converted to a Bézier curve (b). The points in (a) are compressed to a smooth curve (b), which consists of start, middle (P1) and end points, with three control points (P2, P3 and P4) (the red lines are not visible, only for illustrating the control points' influence on the curve). Depending on the data structure, the Bézier curve could take less space than the original line.	19
8.	Examples of file compression with a dictionary coder and entropy encoding.	20
9.	The tile in (b) is twice as large as the tile in (a). The data type in (b) therefore also has to be twice as large as the data type in (a), to be able to store coordinates for the entire tile. Note that Px in (b) is twice as large as Px in (a) since the size depends on the data type – not the number stored in it.	23
10.	By using SVG, the HTML elements can integrate with individual objects in the SVG element (a), and an application can be created by combining the technologies. This is not possible with proprietary plugins, like Adobe Flash, where the HTML elements can not "see" inside the external object (b).	26
11.	The flow of data in a vector map implementation is moving data from the server's spatial database (A) into a data format that is tiny and fast enough to pass through the limited communication channel (B). The data is received by the client, and, if the data structure does not map to the target representation, it is processed to the desired data structure (C), and rendered (D). . . .	30

12.	The data in (a) and (b) has a type header, and lots of coordinates consuming 8 bytes each. In (a), the data is misaligned, while in (b), parts of the first block is kept empty to align the coordinates to single blocks.	32
13.	The example shapefile used in Code Example 3 and 4 shown in Quantum GIS. Since it is a binary format, the file contents can not be read directly. . .	37
14.	An example where the wait time can be perceived as slower when removing existing data immediately after the request, resulting in a blank map until the new data is completely processed and rendered (a), instead of waiting to replace the existing data until the new data is completely processed and ready to be rendered (b).	40
15.	Example of an interactive WFS layer, where one is able to select and query individual features. Unfortunately, each interaction event, which triggers a request to the server, is very noticeable, and might not be acceptable enough in terms of performance for normal use.[161]	42
16.	The tile sizes and map loading time were tested at different zoom levels, where the tiles have varying amounts of spatial data.	43
17.	The test results for different zoom levels: (a) the average tile data size, (b) the total data size for the map and (c) the average map latency. (See Appendix E for larger versions of the plots.)	44
18.	The standard implementation map client (a) with a familiar interface for controlling the map, which is composed of vector tiles (b).	53
19.	The styling is completely controlled by the client, which means that a change in the CSS, e.g. road colour fill to black, will be reflected in the map client (d), with no modification on the server.	54
20.	The legend for the test results of the vector implementation	56
21.	The layers of a full-scale implementation means that the amount of tiles remains constant at different zoom levels (a), (b) and (c). The implementation presented in this paper only has a single layer, which means that the amount of tiles will vary between zoom levels (c), (d) and (e).	57
22.	The number of features at an appropriate scale, fitted to a normal distribution, in the <i>BYGNING</i> table (a) and <i>VEG</i> table (b). (See Appendix E for larger versions of the plots.)	58
23.	The relationship between the number of features (horizontal axis), and the resulting number of coordinates in the <i>BYGNING</i> table (a) and <i>VEG</i> table (b). Every generalised data format (dashed lines) consumes a lot less space in terms of number of coordinates, than the data formats that are not generalised. The generalised GeoJSON was removed, because of an error with the median filter. (See Appendix E for larger versions of the plots.)	61
24.	The efficiency of the data formats in the <i>BYGNING</i> table (a) and <i>VEG</i> table (b). Both the space efficiency of each data format, as well as the impact of generalisation (dashed lines), is visible. The horizontal axis represents the number of features, and the vertical axis is data size in kB. (See Appendix E for larger versions of the plots.)	62

25.	The latency of queries in the <i>BYGNING</i> table (a) and <i>VEG</i> table (b) for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines. (See Appendix E for larger versions of the plots.)	64
26.	The latency of the server processes in the <i>BYGNING</i> table (a) and <i>VEG</i> table (b) for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines. (See Appendix E for larger versions of the plots.)	66
27.	The latency of the client processes in the <i>BYGNING</i> table (a) and <i>VEG</i> table (b) for the data formats. The generalised versions were removed to avoid clutter, as they did not add value to the graph, in the authors opinion. Note that there are some noise in the data for the single byte binary format in (a) and (b). The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)	67
28.	An overview for the latency in the <i>BYGNING</i> table for the not generalised versions of the binary single byte tile data format (a) and the GeoJSON tile format (b). The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)	69
29.	An overview for the latency in the <i>VEG</i> table for the not generalised versions of the binary single byte tile data format (a) and the GeoJSON tile format (b). The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)	70
30.	An overview for the latency in the <i>BYGNING</i> table for the not generalised versions of the binary 2- and 8-byte formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)	71
31.	An overview for the latency in the <i>VEG</i> table for the not generalised versions of the binary 2- and 8-byte formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. (See Appendix E for larger versions of the plots.)	72
32.	The average tile data size for different zoom levels in the WMS implementation.	86
33.	The total tile data size for different zoom levels in the WMS implementation.	87
34.	The average latency for different zoom levels in the WMS implementation. .	88
35.	The number of features at an appropriate scale, fitted to a normal distribution, in the <i>BYGNING</i> table.	89
36.	The number of features at an appropriate scale, fitted to a normal distribution, in the <i>VEG</i> table.	90
37.	The relationship between the number of features (horizontal axis), and the resulting number of coordinates in the <i>BYGNING</i> table. Every generalised data format (dashed lines) consumes a lot less space in terms of number of coordinates, than the data formats that are not generalised. The generalised GeoJSON was removed, because of an error with the median filter.	91

38.	The relationship between the number of features (vertical axis), and the resulting number of coordinates in the <i>VEG</i> table. Every generalised data format (dashed lines) consumes a lot less space in terms of number of coordinates, than the data formats that are not generalised. The generalised GeoJSON was removed, because of an error with the median filter.	92
39.	The efficiency of the data formats in the <i>BYGNING</i> table. Both the space efficiency of each data format, as well as the impact of generalisation, is visible. The generalised versions of the data formats have dashed lines. The horizontal axis represents the number of features, and the horizontal axis is data size in kB.	93
40.	The efficiency of the data formats in the <i>VEG</i> table. Both the space efficiency of each data format, as well as the impact of generalisation, is visible. The generalised versions of the data formats have dashed lines. The horizontal axis represents the number of features, and the horizontal axis is data size in kB.	94
41.	The latency of queries in the <i>BYGNING</i> table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.	95
42.	The latency of queries in the <i>VEG</i> table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.	96
43.	The latency of the server processes in the <i>BYGNING</i> table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.	97
44.	The latency of the server processes in the <i>VEG</i> table for the data formats. The horizontal axis represents the number of features, and the vertical axis is latency in ms. The generalised versions of the data formats have dashed lines.	98
45.	The latency of the client processes in the <i>BYGNING</i> table for the data formats. The generalised versions were removed to avoid clutter, as they did not add value to the graph, in the authors opinion. Note that there are some noise in the data for the single byte binary format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	99
46.	The latency of the client processes in the <i>VEG</i> table for the data formats. The generalised versions were removed to avoid clutter, as they did not add value to the graph, in the authors opinion. Note that there are some noise in the data for the single byte binary format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	100
47.	An overview for the latency in the <i>BYGNING</i> table for the not generalised versions of the binary single byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	101
48.	An overview for the latency in the <i>BYGNING</i> table for the not generalised versions of the GeoJSON tile format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	102

49.	An overview for the latency in the <i>VEG</i> table for the not generalised versions of the binary single byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	103
50.	An overview for the latency in the <i>VEG</i> table for the not generalised versions of the GeoJSON tile format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	104
51.	An overview for the latency in the <i>BYGNING</i> table for the not generalised versions of the binary 2 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	105
52.	An overview for the latency in the <i>BYGNING</i> table for the not generalised versions of the binary 8 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	106
53.	An overview for the latency in the <i>VEG</i> table for the not generalised versions of the binary 2 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	107
54.	An overview for the latency in the <i>VEG</i> table for the not generalised versions of the binary 8 byte tile data format. The horizontal axis represents the number of features, and the vertical axis is latency in ms.	108

List of Tables

- 1. The resulting code table for the entropy coding in Figure 8, using 4 bit codes to represent the symbols. 20
- 2. The relationship and layout of the PostGIS database tables in the implementation. 46
- 3. The relationship between binary and octal values for a single byte signed integer. Note that the high bit marks a negative number, and that -1 has the highest binary value. 49

List of Code Examples

1.	SVG (see Appendix D for complete file contents)	27
2.	TypedArray Example	28
3.	GML (see Appendix D for complete file contents)	33
4.	GeoJSON (see Appendix D for complete file contents)	34
5.	Well-known Text (see Appendix D for complete file contents)	35
6.	Well-known Binary (see Appendix D for complete file contents)	35
7.	SQL Query	84

References

- [1] Creative Commons Attribution-ShareAlike 3.0 Unported License. URL <http://creativecommons.org/licenses/by-sa/3.0/>.
- [2] GitHub, . URL <http://www.github.com>.
- [3] World Wide Web Consortium. A History of HTML, 2011. URL <http://www.w3.org/People/Raggett/book4/ch02.html>.
- [4] An Interactive Map Viewer, . URL <http://www2.parc.com/istl/projects/www94/mapviewer.html>. Accessed 25.01.2012.
- [5] The World-Wide Earhquake Locator, . URL <http://tsunami.geo.ed.ac.uk/local-bin/quakes/mapsript/home.pl>. Accessed 25.01.2012.
- [6] The Atlas of Canada, . URL <http://atlas.nrcan.gc.ca/site/english/aboutus/index.html>. Accessed 25.01.2012.
- [7] Gazetteer for Scotland: Scottish Towns, Villages, Places, People, Families, . URL <http://www.scottish-places.info/background.html>. Accessed 25.01.2012.
- [8] History of Web Mapping, . URL http://geospatial.referata.com/wiki/History_of_Web_Mapping. Accessed 25.01.2012.
- [9] Open Geospatial Consortium, . URL <http://www.ogc.org>. Accessed 11.04.2012.
- [10] OGC History. URL <http://www.opengeospatial.org/ogc/historylong>. Accessed 05.06.2012.
- [11] Allan Doyle. Www mapping framework. *Open GIS Consortium*, 1997.
- [12] MapQuest Maps, . URL <http://www.mapquest.com>. Accessed 25.01.2012.
- [13] Map quest - definition by the Free Online Dictionary, Thesaurus and Encyclopedia, . URL <http://www.thefreedictionary.com/Map+quest>. Accessed 25.01.2012.
- [14] Esri Info - Our History, . URL <http://www.esri.com/about-esri/about/history.html>. Accessed 25.01.2012.
- [15] MapInfo is now Pitney Bowes Software, . URL <http://www.pbinsight.com/welcome/mapinfo.html>. Accessed 25.01.2012.
- [16] Lesson 0: Introduction to web Mapping, . URL <https://www.e-education.psu.edu/geog863/book/export/html/1904>. Accessed 25.01.2012.
- [17] Slippy Map, . URL http://wiki.openstreetmap.org/wiki/Slippy_Map. Accessed 25.01.2012.

- [18] Andrew J. Turner. What is neogeography? In *Introduction to Neogeography*. O'Reilly Media, Inc., 2006.
- [19] Opengis web map service (wms) implementation specification, . URL <http://www.opengeospatial.org/standards/wms>. Accessed 27.01.2012.
- [20] Google maps, . URL <http://maps.google.com>. Accessed 27.01.2012.
- [21] OpenStreetMap, . URL <http://www.openstreetmap.org>. Accessed 27.01.2012.
- [22] Bing maps, . URL <http://www.bing.com/maps>. Accessed 27.01.2012.
- [23] Yahoo! maps, . URL <http://maps.yahoo.com>. Accessed 27.01.2012.
- [24] Walmart.com - Store Locator, . URL http://www.walmart.com/cservice/ca_storefinder.gsp. Accessed 25.01.2012.
- [25] FixMyStreet, . URL <http://www.fixmystreet.com/>. Accessed 25.01.2012.
- [26] Eric Miltsch. Location-based Services: The Hottest Segment in Social Media, October 2010. URL <http://socialmediatoday.com/ericmiltsch/198296/location-based-services-hottest-segment-social-media>. Accessed 25.01.2012.
- [27] Cris Cameron. Market for Location-Based Services is Heating Up for Startups, May 2010. URL <http://www.readwriteweb.com/start/2010/05/market-for-location-based-services-heating-up-for-startups.php>. Accessed 25.01.2012.
- [28] Helen Leggatt. Location-based services "valuable" to 99% of U.S. users, February 2011. URL <http://socialmediatoday.com/ericmiltsch/198296/location-based-services-hottest-segment-social-media>. Accessed 25.01.2012.
- [29] tiling - gis dictionary, . URL <http://support.esri.com/en/knowledgebase/GISDictionary/term/tiling>. Accessed 24.01.2012.
- [30] Raster tiles, . URL <http://edndoc.esri.com/arcsde/9.2/concepts/rasters/entities/rastertiles.htm>. Accessed 24.01.2012.
- [31] Li Haiting, Peng Qingshan, and Li Yanhong. Data Security Analysis of WebGIS Based on Tile-Map Technique. In *Proceedings of the 2009 International Symposium on Web Information Systems and Applications (WISA'09)*, 2009.
- [32] Cubeserv[®] web map tiling server (wmts), . URL <http://www.cubewerx.com/products/wmts>. Accessed 27.01.2012.
- [33] OpenGIS Web Map Tile Service Implementation Standard, . URL <http://www.opengeospatial.org/standards/wmts>. Accessed 27.01.2012.

- [34] Review: Opengis web map tiling service (wmts) interface specification, . URL <http://xml.coverpages.org/OGC-WMTS-Candidate.html>. Accessed 27.01.2012.
- [35] Zao Liu, Marlon E. Pierce, Geoffrey C. Fox, and Neil Devadasan. Fox implementing a caching and tiling map server: a web 2.0 case study. In *Proceedings of The 2007 International Symposium on Collaborative Technologies and Systems (CTS 2007)* <http://grids.ucs.indiana.edu/ptliupages/publications/CachingTilingMapServer.pdf>.
- [36] Google VP lays down mobile stats, boasts 150 million Maps users, . URL <http://www.engadget.com/2011/03/14/google-vp-lays-down-mobile-stats-boasts-150-million-maps-users/>. Accessed 26.01.2012.
- [37] Big Birthday... Google Maps API Turns 5!, . URL <http://googlegeodevelopers.blogspot.com/2010/06/big-birthday-google-maps-api-turns-5.html>. Accessed 26.01.2012.
- [38] J. D. Blower. Gis in the cloud: implementing a web map service on google app engine. In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application, COM.Geo '10*, pages 34:1–34:4, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0031-5.
- [39] Sterling Quinn and Mark Gahegan. A predictive model for frequently viewed tiles in a web map. *Transactions in GIS*, 14(2):193–216, 2010.
- [40] Yong-Kyoon Kang, Ki-Chang Kim, and Yoo-Sung Kim. Probability-based tile prefetching and cache replacement algorithms for web geographical information systems. In Albertas Caplinskas and Johann Eder, editors, *Advances in Databases and Information Systems*, volume 2151 of *Lecture Notes in Computer Science*, pages 127–140. Springer Berlin / Heidelberg, 2001.
- [41] MBTiles, . URL <http://mapbox.com/mbtiles-spec/>. Accessed 27.01.2012.
- [42] Client caching. URL http://wiki.novell.com/index.php/Client_caching. Accessed 30.05.2012.
- [43] Cache Eviction Algorithms, . URL <http://ehcache.org/documentation/apis/cache-eviction-algorithms>. Accessed 05.06.2012.
- [44] LRU cache implementation in C++, . URL <http://timday.bitbucket.org/lru.html>. Accessed 05.06.2012.
- [45] Cache Eviction, . URL <http://docs.codehaus.org/display/COCONUT/Cache+Eviction>. Accessed 05.06.2012.
- [46] John Krygier and Denis Wood. *Making Maps: A Visual Guide to Map Design for GIS*. Guilford Publications, 2 edition, 2011.

- [47] K. Shea and R. McMaster. Cartographic generalization in a digital environment: When and how to generalize. In *AutoCarto 9*, pages 56–67, Baltimore, Etats-Unis, 1989.
- [48] Dianne E. Richardson and William A. Mackaness. Computational processes for map generalization. *Cartography and Geographic Information Science*, 26, 1999.
- [49] Ferjan J. Ormeling. *Technical Geography: Core Concepts in the Mapping Sciences*. EOLSS.
- [50] William A. Mackaness. Preface. In *Generalisation of geographic information: cartographic modelling and applications*. Published on behalf of the International Cartographic Association by Elsevier.
- [51] R. B. McMaster and K. S. Shea. *Generalization in Digital Cartography*. Association of American Geographers, 1992.
- [52] F. Christ. A program for the fully automated displacement of point and line features in cartographic generalization. *Information Relative to Cartography and Geodasy Translations*, 1978.
- [53] W. A. Mackaness and R. S. Purves. Automatid displacement for large numbers of discrete map objects. *Algorithmica*, (30):302–311, 2001.
- [54] Christopher B. Jones, Geranit Ll. Bundy, and J. Mark Ware. Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems*, 22:317–331, 1995.
- [55] William A. Mackaness. An algorithm for conflict identification and feature displacement in automated map generalization. *Cartography and Geographic Information Systems*, 21:219–232, 1994.
- [56] GENNADY L. ANDRIENKO and NATALIA V. ANDRIENKO. Interactive maps for visual data exploration. *International Journal of Geographical Information Science*, 13(4):355–374, 1999.
- [57] Sharon Oviatt. Multimodal interactive maps: designing for human performance. *Hum.-Comput. Interact.*, 12(1):93–129, March 1997.
- [58] J. Dash and G. Lawton. Gis hits the road. *Software Magazine*, 16, 1996.
- [59] Zhong-Ren Peng and Chuanrong Zhang. The roles of geography markup language (gml), scalable vector graphics (svg), and web feature service (wfs) specifications in the development of internet geographic information systems (gis). *Journal of Geographical Systems*, 6:95–116, 2004.
- [60] Stephan Winter and Andrew U. Frank. Topology in raster and vector representation. *GeoInformatica*, 4:35–65, 2000. 10.1023/A:1009828425380.

- [61] What is the abandonment rate of users who need to install a browser plugin in order to use major features of a web site?, . URL <http://www.quora.com/What-is-the-abandonment-rate-of-users-who-need-to-install-a-browser-plugin-in-> Accessed 13.02.2012.
- [62] What is the average retention rate for a browser extension?, . URL <http://www.quora.com/What-is-the-average-retention-rate-for-a-browser-extension>. Accessed 13.02.2012.
- [63] Bisheng Yang. A multi-resolution model of vector map data for rapid transmission over the internet. *Computers and Geosciences*, 31, 2005.
- [64] Uwe Rauschenbach and Heidrun Schumann. Demand-driven image transmission with levels of detail and regions of interest. *Computers and Graphics*, 23, 1999.
- [65] Philip F. Kern and James D. Carswell. An investigation into the use of jpeg image compression for digital photogrammetry: Does the compression of images affect measurement accuracy? *EGIS Conference Proceedings: European Conference on Geographical Information Systems*, 1994.
- [66] Leila De Floriani and Enrico Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics*, 14, 1995.
- [67] Donggyu Park, Hwangue Cho, and Yangsoo Kim. A tin compression method using delaunay triangulation. *International Journal of Geographical Information Science*, 15, 2001.
- [68] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. *Proceedings of IEEE Visualization Conference*, 1998.
- [69] Bisheng Yang, Ross Purves, and Robert Weibel. Efficient transmission of vector data over the internet. *International Journal of Geographical Information Science*, 21, 2007.
- [70] Bisheng Yang, Ross Purves, and Robert Weibel. Implementation of progressive transmission algorithms for vector map data in web-based visualization. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34, 2004.
- [71] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10, 1973.
- [72] Zeshen Wang and Jean-Claude Muller. Line generalization based on analysis of shape characteristics. *Cartography and Geographic Information Science*, 25(1):3–15, 1998. URL <http://www.ingentaconnect.com/content/cagis/cagis/1998/00000025/00000001/art00001>.

- [73] Mahes Visvalingam and Simon Herbert. A computer science perspective on the bendsimplification algorithm. *Cartography and Geographic Information Science*, 26(4): 253–270, 1999. URL <http://www.ingentaconnect.com/content/cagis/cagis/1999/00000026/00000004/art00002>.
- [74] *Introduction to Data Compression*. Carnegie Mellon University, 2010.
- [75] Bisheng Yang, Ross S. Purves, and Robert Weibel. Variable-resolution compression of vector data. *Geoinformatica*, 12:357–376, September 2008.
- [76] Bezier curve, . URL <http://mathworld.wolfram.com/BezierCurve.html>. Accessed 31.05.2012.
- [77] B-spline, . URL <http://mathworld.wolfram.com/B-Spline.html>. Accessed 31.05.2012.
- [78] Data compression basics, . URL http://dvd-hq.info/data_compression_1.php. Accessed 05.03.2012.
- [79] David J.C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2005.
- [80] Huffman Coding: A CS2 Assignment, . URL <http://www.cs.duke.edu/csed/poop/huff/info/>. Accessed 05.03.2012.
- [81] RFC 1951 - DEFLATE Compressed Data Format Specification version 1.3, . URL <http://tools.ietf.org/html/rfc1951>. Accessed 05.03.2012.
- [82] Gzip - GNU Project - Free Software Foundation, . URL <http://www.gnu.org/software/gzip/>. Accessed 05.03.2012.
- [83] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1, . URL <http://tools.ietf.org/html/rfc2616>. Accessed 05.03.2012.
- [84] Cascading Style Sheets, . URL <http://www.w3.org/Style/CSS/Overview.en.html>. Accessed 04.05.2012.
- [85] Reihaneh Safavi-Naini Qiong Liu and Nicholas Paul Sheppard. Digital rights management for content distribution. *Conferences in Research and Practice in Information Technology*, 21, 2003.
- [86] Digital Rights Management: A failure in the developed world, a danger to the developing world, . URL <https://www.eff.org/wp/digital-rights-management-failure-developed-world-danger-developing-world>. Accessed 06.03.2012.
- [87] World Wide Web Consortium. A History of HTML, 2011. URL <http://www.w3.org/People/Raggett/book4/ch02.html>.

- [88] Web Hypertext Application Technology Group, . URL <http://www.whatwg.org>.
- [89] Html5 – edition for web developers, . URL <http://developers.whatwg.org/>. Accessed 19.04.2012.
- [90] Matthew B. Hoy. HTML5: A New Standard for the Web. *Medical Reference Services Quarterly*, 30:1:50–55, 2011.
- [91] Adobe. Adobe - Flash Player, 2011. URL <http://www.adobe.com/software/flash/about/>. Accessed 19.04.2012.
- [92] Microsoft. Introduction to ActiveX Controls, 2008. URL [http://msdn2.microsoft.com/en-us/library/aa751972\(VS.85\).aspx](http://msdn2.microsoft.com/en-us/library/aa751972(VS.85).aspx). Accessed 19.04.2012.
- [93] Microsoft. About Microsoft Silverlight, . URL <http://www.microsoft.com/silverlight/what-is-silverlight/>. Accessed 19.04.2012.
- [94] Matthew Marshall. Gartner Press Release (on mobile device growth), 2011. URL <http://www.gartner.com/it/page.jsp?id=1543014>. Accessed 20.04.2012.
- [95] Microsoft. Microsoft Windows, . URL <http://windows.microsoft.com/>. Accessed 19.04.2012.
- [96] Apple. Apple OS X, . URL <http://www.apple.com/macosx/>. Accessed 19.04.2012.
- [97] Canonical. Ubuntu Linux, . URL <http://www.ubuntu.com/>. Accessed 19.04.2012.
- [98] Katherine Noyes. Linux and Mac OS are fastest-growing operating systems, 2011. URL http://www.pcworld.com/businesscenter/article/226564/linux_and_mac_os_are_fastestgrowing_operating_systems.html. Accessed 20.04.2012.
- [99] Apple. Apple iOS, . URL <http://www.apple.com/ios/>. Accessed 19.04.2012.
- [100] Google. Android. URL <http://www.android.com/>. Accessed 19.04.2012.
- [101] Canonical. Ubuntu Linux, . URL <http://en.wikipedia.org/wiki/X86>. Accessed 19.04.2012.
- [102] ARM. ARM. URL <http://www.arm.com/>. Accessed 19.04.2012.
- [103] Ecma International. ECMAScript Language Specification, 2011. URL <http://www.ecma-international.org/publications/standards/Ecma-262.htm>. Accessed 20.04.2012.
- [104] C, . URL <http://www.open-std.org/JTC1/SC22/WG14/>. Accessed 20.04.2012.
- [105] C++, . URL <http://www.open-std.org/jtc1/sc22/wg21/>. Accessed 20.04.2012.
- [106] Java, . URL <http://www.java.com>. Accessed 20.04.2012.

- [107] C#, . URL <http://www.ecma-international.org/publications/standards/Ecma-334.htm>. Accessed 20.04.2012.
- [108] Python, . URL <http://www.python.org>. Accessed 20.04.2012.
- [109] Peter Wayner. From PHP to Perl: What's hot, what's not in scripting languages, 2011. URL <http://www.infoworld.com/d/application-development/php-perl-whats-hot-whats-not-in-scripting-languages-175867?page=0,1>.
- [110] Computer Language Benchmarks Game, 2011. URL <http://shootout.alioth.debian.org/u32/benchmark.php?test=all&lang=all>. Accessed 09.12.2011.
- [111] Tail Call Optimization, . URL <http://paulbarry.com/articles/2009/08/30/tail-call-optimization>. Accessed 04.06.2012.
- [112] Javascript memory optimization and texture loading, . URL <http://blog.tojicode.com/2012/03/javascript-memory-optimization-and.html>. Accessed 04.06.2012.
- [113] The Future of JavaScript Engines: replace them with javascript compilers, . URL <http://shorestreet.com/node/43>. Accessed 04.06.2012.
- [114] The JavaScript Revolution, . URL <http://fail-forward.blogspot.no/2012/03/javascript-revolution.html>. Accessed 04.06.2012.
- [115] a quick note on JavaScript engine components, . URL <http://hacks.mozilla.org/2010/03/a-quick-note-on-javascript-engine-components/>. Accessed 04.06.2012.
- [116] Scalable vector graphics (svg) 1.1 (second edition), . URL <http://www.w3.org/TR/SVG11/>. Accessed 30.03.2012.
- [117] 4.8.16 svg – html5, . URL <http://dev.w3.org/html5/spec/svg-0.html#svg-0>. Accessed 30.03.2012.
- [118] Working with binary data using typed arrays, . URL <http://blogs.msdn.com/b/ie/archive/2011/12/01/working-with-binary-data-using-typed-arrays.aspx>. Accessed 26.03.2012.
- [119] Typed array specification, . URL <https://www.khronos.org/registry/typedarray/specs/latest/>. Accessed 26.03.2012.
- [120] Performance of Javascript (Binary) Byte Arrays in Modern Browsers, . URL <http://blog.n01se.net/blog-n01se-net-p-248.html>. Accessed 28.05.2012.
- [121] About websocket, . URL <http://websocket.org/aboutwebsocket.html>. Accessed 16.03.2012.

- [122] Mats Taraldsvik. Exploring the future: is html5 the solution for gis applications on the world wide web? 2011. URL <http://www.github.com/meastp/html5andgis/>.
- [123] Is Microsoft Challenging Google on HTTP 2.0 with WebSocket?, . URL <http://www.readwriteweb.com/enterprise/2012/03/microsoft-sees-googles-hand-fo.php>. Accessed 04.06.2012.
- [124] Hypertext Transfer Protocol Bis (httpbis), . URL <https://datatracker.ietf.org/wg/httpbis/charter/>. Accessed 04.06.2012.
- [125] What's Next for HTTP, . URL http://www.mnot.net/blog/2012/03/31/whats_next_for_http. Accessed 04.06.2012.
- [126] SPDY: An experimental protocol for a faster web, . URL <http://dev.chromium.org/spdy/spdy-whitepaper>. Accessed 04.06.2012.
- [127] SPDY Brings Responsive and Scalable Transport to Firefox 11, . URL <http://hacks.mozilla.org/2012/02/spdy-brings-responsive-and-scalable-transport-to-firefox-11/>. Accessed 04.06.2012.
- [128] The websocket api, . URL <http://dev.w3.org/html5/websockets/>. Accessed 30.04.2012.
- [129] Carl A. Gutwin, Michael Lippold, and T. C. Nicholas Graham. Real-time groupware in the browser: testing the performance of web-based networking. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, CSCW '11, pages 167–176, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0556-3. doi: 10.1145/1958824.1958850. URL <http://doi.acm.org/10.1145/1958824.1958850>.
- [130] About websocket, . URL <http://websocket.org/aboutwebsocket.html>. Accessed 15.05.2012.
- [131] Binary or text (an essay on w3c's design principles), . URL <http://www.w3.org/People/Bos/DesignGuide/binary-or-text>. Accessed 15.05.2012.
- [132] Workers love ArrayBuffer, . URL <http://updates.html5rocks.com/2011/09/Workers-ArrayBuffer>. Accessed 28.05.2012.
- [133] Transferable Objects: Lightning fast, . URL <http://updates.html5rocks.com/2011/12/Transferable-Objects-Lightning-Fast>. Accessed 28.05.2012.
- [134] HTML5 2.8.4 Transferable objects, . URL <http://dev.w3.org/html5/spec/common-dom-interfaces.html#transferable-objects>. Accessed 28.05.2012.
- [135] RFC 3629 – UTF-8, a transformation format of ISO 10646, . URL <http://tools.ietf.org/html/rfc3629>. Accessed 16.04.2012.

- [136] RFC 20 – ASCII format for network interchange, . URL <http://tools.ietf.org/html/rfc20>. Accessed 16.04.2012.
- [137] What is new in PostGIS 2.0.0. URL <http://t.co/fEE1TP5g>. Accessed 24.05.2012.
- [138] Understanding Big and Little Endian Byte Order, . URL <http://betterexplained.com/articles/understanding-big-and-little-endian-byte-order/>. Accessed 05.06.2012.
- [139] An Essay on Endian Order, . URL <http://people.cs.umass.edu/~verts/cs32/indian.html>. Accessed 05.06.2012.
- [140] Open Geospatial Consortium. Geography Markup Language, 2007. URL <http://www.opengeospatial.org/standards/gml>. Accessed 11.04.2012.
- [141] JSON, . URL <http://www.json.org/>. Accessed 13.04.2012.
- [142] GeoJSON, . URL <http://www.geojson.org/>. Accessed 13.04.2012.
- [143] JSON vs XML – Part 1: Data Size, . URL <http://xphone.me/devnotes/2011/02/json-vs-xml-part-1-data-size/>. Accessed 13.04.2012.
- [144] Compress JSON with automatic type extraction, . URL <http://stevehanov.ca/blog/index.php?id=104>. Accessed 13.04.2012.
- [145] json.hpack, . URL <https://github.com/WebReflection/json.hpack/wiki>. Accessed 13.04.2012.
- [146] JSON Compression Algorithms, . URL <http://web-resource-optimization.blogspot.com/2011/06/json-compression-algorithms.html>. Accessed 13.04.2012.
- [147] BSON, . URL <http://bsonspec.org/>. Accessed 15.05.2012.
- [148] MongoDB, . URL <http://www.mongodb.org/>. Accessed 15.05.2012.
- [149] BSON specification, . URL <http://bsonspec.org/#/implementation>. Accessed 15.05.2012.
- [150] Open Geospatial Consortium. Simple Feature Access - Part 1: Common Architecture, 2011. URL <http://www.opengeospatial.org/standards/sfa>. Accessed 12.04.2012.
- [151] PostGIS, . URL <http://www.postgis.org/>. Accessed 13.04.2012.
- [152] PostgreSQL, . URL <http://www.postgresql.org/>. Accessed 13.04.2012.
- [153] 4.1.2. PostGIS EWKB, EWKT and Canonical Forms, . URL http://www.postgis.org/documentation/manual-svn/using_postgis_dbmanagement.html#EWKB_EWKT. Accessed 13.04.2012.

- [154] *ESRI Shapefile Technical Description*. Environmental Systems Research Institute, Inc., 1998.
- [155] Fred Wilson's 10 Golden Principles of Successful Web Apps - Future of Web Apps (Conference), 2011. URL <http://thinkvitamin.com/web-apps/fred-wilsons-10-golden-principles-of-successful-web-apps/>. Accessed 20.04.2012.
- [156] Len Bass, Paul Clements, and Rick Kazmann. Understanding Quality Attributes. In *Software Architecture in Practice*, chapter 4. Pearson Education, second edition, 2009.
- [157] Len Bass, Paul Clements, and Rick Kazmann. Performance Tactics. In *Software Architecture in Practice*, chapter 5.4. Pearson Education, second edition, 2009.
- [158] Opeingis web feature service (wfs) implementation specification, . URL <http://www.opengeospatial.org/standards/wfs>. Accessed 08.05.2012.
- [159] Wenjue Jia, Yumin Chen, Jianya Gong, and Aixia Li. Web service based web feature service. State Key Laboratory for Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University.
- [160] Chuanrong Zhang and Weidong Li. The roles of web feature and web map services in real-time geospatial data sharing for time-critical applications. *Cartography and Geographic Information Science*, 32(4), 2005.
- [161] Wfs: Getfeature example (geoserver), . URL <http://openlayers.org/dev/examples/getfeature-wfs.html>. Accessed 08.05.2012.
- [162] Open source gis: Open scales and wfs, . URL <http://www.webmapsolutions.com/open-source-gis-openscales-wfs>. Accessed 08.05.2012.
- [163] Geoserver, . URL <http://www.geoserver.org/>. Accessed 08.05.2012.
- [164] Lefalet - a modern, lightweight javascript library for interactive maps, . URL <http://leaflet.cloudmade.com/>. Accessed 08.05.2012.
- [165] Chromium, . URL <http://www.chromium.org/Home>. Accessed 21.05.2012.
- [166] flot - Attractive javascript plotting for jQuery, . URL <http://code.google.com/p/flot/>. Accessed 21.05.2012.
- [167] 2.2 Cached Tiles, . URL <http://workshops.opengeo.org/openlayers-intro/layers/cached.html>. Accessed 28.05.2012.
- [168] Advanced Research Lab for Geospatial Information Science and Indian Institute of Technology Engineering. Comparison between postgis and oracle spatial. URL <http://bit.ly/xP1bDj>.

- [169] Microsoft SQL Server, . URL <http://www.microsoft.com/sql/>. Accessed 30.04.2012.
- [170] Oracle Spatial, . URL www.oracle.com/technetwork/database/options/spatial/index.html. Accessed 30.04.2012.
- [171] 8.6. Geometry Outputs, . URL <http://postgis.org/docs/reference.html>. Accessed 30.04.2012.
- [172] 8.3. Geometry Constructors, . URL <http://postgis.org/docs/reference.html>. Accessed 30.04.2012.
- [173] ST_Simplify, . URL http://postgis.refractor.net/documentation/manual-2.0/ST_Simplify.html. Accessed 30.04.2012.
- [174] ST_SnapToGrid, . URL http://postgis.refractor.net/documentation/manual-2.0/ST_SnapToGrid.html. Accessed 30.04.2012.
- [175] JTS Topology Suite, . URL <http://www.vividsolutions.com/jts/>. Accessed 30.04.2012.
- [176] GEOS, . URL <http://www.geos.osgeo.org/>. Accessed 30.04.2012.
- [177] Shapely, . URL <https://github.com/sgillies/shapely>. Accessed 30.04.2012.
- [178] Autobahn.oss, . URL <http://autobahn.ws/developers>. Accessed 30.04.2012.
- [179] AutobahnTestSuite, . URL <http://autobahn.ws/testsuite>. Accessed 30.04.2012.
- [180] AutobahnPython Reference, . URL <http://autobahn.ws/developers/reference/python/index.html>. Accessed 30.04.2012.
- [181] WebSocket++, . URL <https://github.com/zaphoyd/websocketpp>. Accessed 30.04.2012.
- [182] Boost.Asio, . URL http://www.boost.org/doc/libs/1_49_0/doc/html/boost_asio.html. Accessed 30.04.2012.
- [183] wsperf, . URL <http://www.zaphoyd.com/wsperf>. Accessed 30.04.2012.
- [184] UtfEight, . URL <http://code.google.com/p/webgl-loader/wiki/UtfEight>. Accessed 15.05.2012.
- [185] JavaScript Performance Best Practices, . URL https://www.developer.nokia.com/Community/Wiki/JavaScript_Performance_Best_Practices. Accessed 28.05.2012.
- [186] Raphaël – JavaScript Library, . URL <http://raphaeljs.com/>. Accessed 30.04.2012.
- [187] d3.js, . URL <http://mbostock.github.com/d3/>. Accessed 30.04.2012.

- [188] VML - the Vector Markup Language, . URL <http://www.w3.org/TR/NOTE-VML>. Accessed 30.04.2012.
- [189] Creator of Web spots a flaw in IE, . URL <http://www.msnbc.msn.com/id/26646919#.T56H6VTWTRY>. Accessed 30.04.2012.
- [190] Understanding Map Projections, . URL <http://macwright.org/2012/01/27/projections-understanding.html>. Accessed 28.05.2012.
- [191] Algorithm Alley (Median filter), . URL <http://www.drdoobbs.com/parallel/184411079>. Accessed 21.05.2012.
- [192] Dr. Colin Mercer. CLEANING UP DATA - Using a median filter to remove spikes from data. Accessed 21.05.2012.
- [193] Median filter, . URL http://fourier.eng.hmc.edu/e161/lectures/smooth_sharpen/node3.html. Accessed 21.05.2012.
- [194] Wikipedia. Standard deviation. http://en.wikipedia.org/wiki/Standard_deviation, 2011. Accessed 03.12.2011.
- [195] Wikipedia. Normal distribution. http://en.wikipedia.org/wiki/Normal_distribution, 2011. Accessed 03.12.2011.
- [196] Institutt for matematiske fag, NTNU. *Tabeller og formler i statistikk*. 2000.
- [197] Jakob Nielsen. Response Times: The 3 Important Limits. In *Usability Engineering*, chapter 5. 1993.
- [198] Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 2004.
- [199] WebCL, . URL <http://www.khronos.org/webcl/>. Accessed 23.05.2012.
- [200] General-Purpose computation on Graphics Processing Units, . URL <http://gpgpu.org/>. Accessed 23.05.2012.
- [201] Robert Nordan. An investigation of potential methods for topology preservation in interactive vector tile map applications. 2012.
- [202] When can I use... Support tables for HTML5, CSS3, etc, . URL <http://www.caniuse.com>. Accessed 28.05.2012.
- [203] Modernizr, . URL <http://www.modernizr.com>. Accessed 28.05.2012.