UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Dmitri Timašjov

Fast map interface with large number of vector objects

Bachelor's thesis

Supervisor: Vambola Leping
External supervisor: Toivo Vajakas

Author: ..... "...." May 2013
Supervisor: ..... "...." May 2013
Supervisor: ..... "...." May 2013
Professor: ..... "...." May 2013

TARTU 2013

Table of Contents

Chapter 1. Introduction **3**
1.1 Motivation 3
1.2 Problem description 3
1.3 Scope 3
1.4 Purpose and research question 4
1.5 Structure of thesis 4
Chapter 2. Geographic information system **6**
2.1 Representation of geographical data 6
2.2 GIS Development 9
2.3 Map components 10
Chapter 3. Overview of the experiment **14**
3.1 General information about experiment 14
3.2 Technologies used in experiment 15
Chapter 4. Methods used to speed-up the process of rendering **16**
4.1 Background 16
4.2 Cluster strategy 17
4.3 Decimation principle 18
4.4 Data encoding method 19
Chapter 5. Experiment **24**
5.1 Analysis and implementation 24
5.2 Test environment 27
5.3 Performance evaluation 28
5.4 Conclusions 32
5.5 Possible improvements 32
Chapter 6. Discussion and conclusion **34**
6.1 Work overview 34
6.2 Potential improvements and future work 35
Resümee **36**
References **37**
Appendix **39**

Chapter 1. Introduction

1.1 Motivation

The increasing popularity of web-based geographical information systems (GIS) and services for delivering maps for practical and scientific purposes can be regarded as one of the most important developments in the advancement of cartography. The vast majority of developed systems and applications appear to be all kinds of state registers that deal with geographical information. It frequently happens that the amount of spatial data the applications need simultaneously to show to the users is so huge that the browsers may have troubles with data visualization and generalization. So the actual challenge remains to create top-quality and lithesome geographical information systems with respect to velocity and data presentation of the system being described.

1.2 Problem description

The problem that simultaneous display of large amount of geographical data may lead the browsers into the state when they cannot respond to the users requests is typically recognized only after GIS web application has been developed. Only after the detection of such problems the attempts to optimize the web application take place.

The main goal of the work is to study how current geovisualization methods and tools support representation and encoding of very large geographical datasets (more than 100000 objects). This thesis evaluates, which features could provide benefits for fast representation and generalization of spatial data. It will be studied how to combine benefits and create a web application that can handle large amounts of geographical data. Current work tries to take advantage of recent technological advances in data formats. More attention is directed to the developments in open standards for encoding both data and graphics.

1.3 Scope

Like every web application, web mapping systems have a client-server architecture that allows them to exchange information through the Internet. Therefore different

representation and optimization methods can be applied on both the server- and client side of the web-applications. Thesis whereas focuses on the client-side of the GIS web applications. Server-side will also be touched, but to a lesser extent. Spatial databases and database geographic queries are not considered in this paper.

1.4 Purpose and research question

This work tries to analyze and solve the following problems:

- The first problem is that standard graphical user interfaces neither support complex queries to the databases linked to the visual display, nor they enable flexible manipulation of display parameters. Current work makes an overview of the most commonly used ways and technologies to speed-up the process of rendering objects on the map. Various rendering and representation methods are considered which cope with very large datasets of high dimensionality, the scale at which phenomena exist and the level of details used to digitally represent objects on different zoom levels.
- The second problem is geovisualization usability. Thesis analyzes semiotics and meaning of large amount of spatial data, namely, how visual depictions relate to underlying meaning, to desired uses and human-computer interaction.

GIS web application will be built in order to analyze how browsers cope with an excessive amount of geospatial data. Different optimization methods (clustering, decimation principle) and rendering techniques (SVG, canvas) will be implemented in the application in order to order to give the answer, whether these techniques are helpful and can reduce features rendering time, or vice versa, they just slow down the work of the application.

1.5 Structure of thesis

Current work consists of 3 logical parts, divided into 6 chapters. Chapter 1 is introductive, it makes a short overview of a thesis, scope of the work and the problem domain. Chapter 2 is the background of a thesis; it consists of a description of geographic information system, its standards and components. Chapter 3 provides general information about the practical part of the thesis. Most

common technologies used to speed-up the process of rendering are thoroughly overviewed in Chapter 4. Chapter 5 describes the practical part of the thesis and explains how different optimization techniques are implemented in the prototype. This chapter also contains the analysis of the obtained results and some proposals for additional improvements. Analysis is followed by the conclusion in Chapter 6.

Chapter 2. Geographic information system

GIS is defined as a computer-based technology and methodology for collecting, managing, analyzing, modeling and presenting geographically referenced data for a wide range of applications [4]. According to this definition, GIS primary role is to deal with geographic data from input to output. Following chapter will give an overview of the main geospatial data types and their representations on a map.

2.1 Representation of geographical data

2.1.1 Vector and raster data

Geographical data represent objects and phenomena from the real world, such as forests, roads, floods, etc. In terms of graphics, geographical objects are essentially divided into two great classes: raster and vector. Both structures have different methods of storing and displaying spatial data [2]. Raster data is cell-based, where each cell, also called pixel, is used to encode geographic data and contains associated attributes indicating color value of the cell. In comparison, vector data represents mathematically associated objects that are considered as geometrical shapes. This thesis concentrates on vector data model and raster data model will no longer be discussed.

2.1.2 Vector data type

In GIS, vector data is classified as three main types: point, line, and polygon forming an area.

- Point: A point feature is a spot that has no physical or actual spatial dimensions, but does have specific location. Point is shown as a convenient visual symbol to locate the feature it represents and is defined by a latitude longitude pair. However, the point does not indicate the actual length or width of the feature.
- Line: A line is a one-dimensional feature that consists of at least of two points, the endpoints, and optionally points in between.
- Polygon: A polygon or shape is a list of points where the last point is the same as the first, so that the connecting the points encloses a geographic area [3].

A geographic feature is an application object that represents a physical entity. Each feature may or may not have additional associated attributes and be rendered in a different way - this way is determined by a geometry associated with the feature [12]. But representation of the feature mainly depends not only on the feature itself, but on also on the map scale and the way how it is visualized. For instance, point can represent a city with name and population, even though in reality city has area. Lines are typically used for visualizing roads, streams or administrative boundaries. Objects such as agricultural fields, political districts or forest areas can be visualized as polygons.

2.1.3 Spatial data visualization

Human insight is very important to extract high-level information from a dataset. Therefore it is very important to provide end-user with a comprehensive user-centered and accessible design. Multimedia cartography and GIS can play a very important role in the process of visualization of spatial data and making the discovered knowledge understandable and interpretable by human beings.

Geovisualization, short for geographic visualization, or visualization of geospatial data represents the interface between scientific visualization, cartography and image analysis. Due to the capability of combining geospatial information with "human vision and domain expertise", geovisualization sets the basis for effective support of data exploration and decision-making processes and can be applied to all the stages of problem-solving in geographical analysis [10].

Frequently it is necessary to visualize the increasingly large and complex volumes of geospatial data that are becoming available. Main problem here is that given a large dataset, the display may become cluttered to the point where it is impossible to distinguish information [7]. This requires both advances in geovisualization methods to deal with very large data volumes and advances in integration of these methods with geocomputational ones [1].

2.1.4 Spatial data generalization

As is the case with any data relating to geographic information, scale is a critical issue in GIS, because it defines the limits to human observations of the Earth. Different information is required at varying scales, and appropriate detail should be presented at each scale. As scale of display increases, precision also decreases and shapes cannot be precisely represented. As a result, all spatial datasets should be generalized. Spatial data generalization is the process responsible for generating visualizations or geographic datasets at coarser levels-of-detail than the original dataset, while retaining essential characteristics of the underlying geographic information (see Illustration 1) [5].

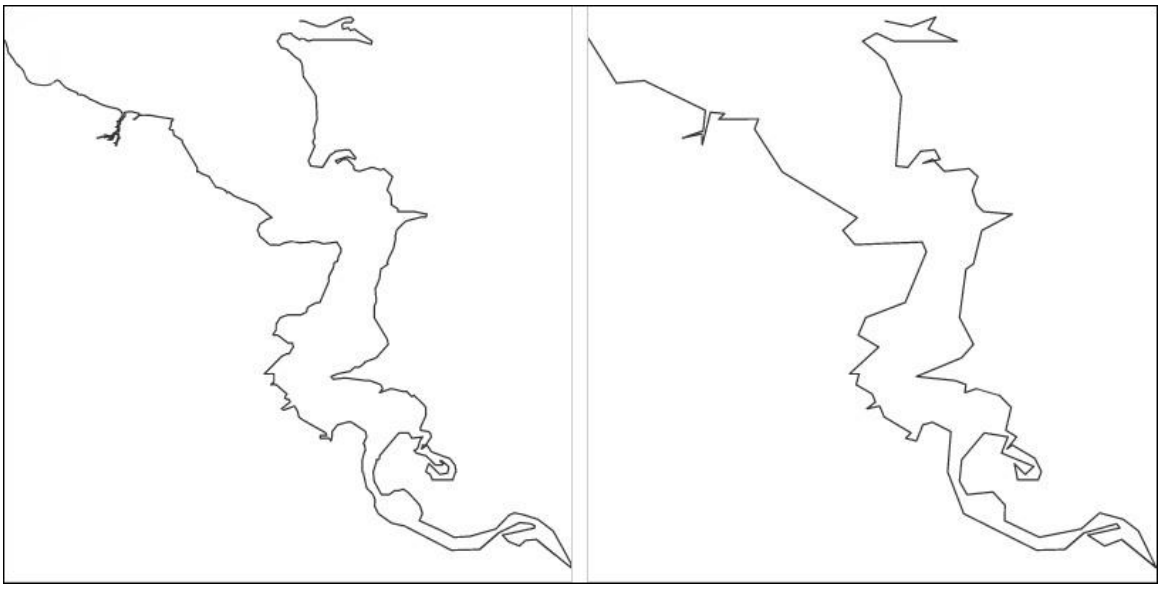

Illustration 1. Generalization of geographic objects

Map generalization is a core stage of map design [9]. While generalizing geographical objects it is necessary to remember that the human factors aspect is very important. This challenge raises many specific issues including: emphasis of the most important map elements, representation of the world in a faithful and recognizable way on different zoom levels and preservation of the object's distinguishing characteristics. Generalization must be done very precisely, because wrong classification, for instance, may hide the characteristic pattern of a statistical surface [13].

A great deal of attempts has been made to automate the process of generalization. Despite the fact that automatic generalization has been a hot research topic for decades, there still does not exist a set of universal rules or algorithms that explicitly define how generalization should be performed [15]. Therefore, the objective of automatic geometric generalization is to automatically preserve the important parts of the data and eliminate or simplify the less important ones in order to create a map that will have good visual communication characteristics [17].

2.2 GIS Development

2.2.1 GIS architecture

Rapid progress of Internet, application of data warehouse technology and the combination of GIS and Internet have paved the way for web distribution of GIS spatial data [19]. Users can browse the spatial data in GIS web application by arbitrary node on Internet, make thematic maps and make all kinds of spatial checkups and spatial analysis, thus, GIS can enter numerous households and become a kind of public instrument [19].

Typically, GIS web application is divided into three tier client/server mode where three major types of service components are: presentation, business and data. Server contains all back-end logics and is responsible for the presentation by displaying the map images. Server receives the request with needed parameters and information and does required geoquery to the database. When the geographical data is fetched from the database and returned to the server then, if necessary, it is processed or edited and transmitted back to the client. Client holds map container which contains all mapping elements and logic engine, which renders features. Most commonly geographic data, vector data in our case, is exchanged through special GIS formats, the most popular of which are GeoJSON (JSON object containing object's geometry), GML (Geography Markup Language) and WKT (Well-Known Text). Such formats are efficient for spatial data exchange, as they compress geodata which makes files smaller, and consequently decreases file transfer time.

2.2.2 Web GIS libraries

In order to visualize a map and features on it, GIS web application should use special libraries that can embed interactive map into a website. These are JavaScript based libraries, such as OpenLayers, Leaflet, Google Maps, and Bing Maps. All this libraries provide methods and tools for creating dynamic, rich and interactive maps: enabling and disabling layers, changing base layer, zooming in and out, selecting element on the map, printing, etc. Key advantage of using such libraries is that they support the delivering of up to date information to the browser, therefore there is no need to refresh the webpage. All these libraries can be customized for satisfying needs of concrete GIS product. An example of such interactive map that is using Google Maps library can be viewed on Illustration 2.

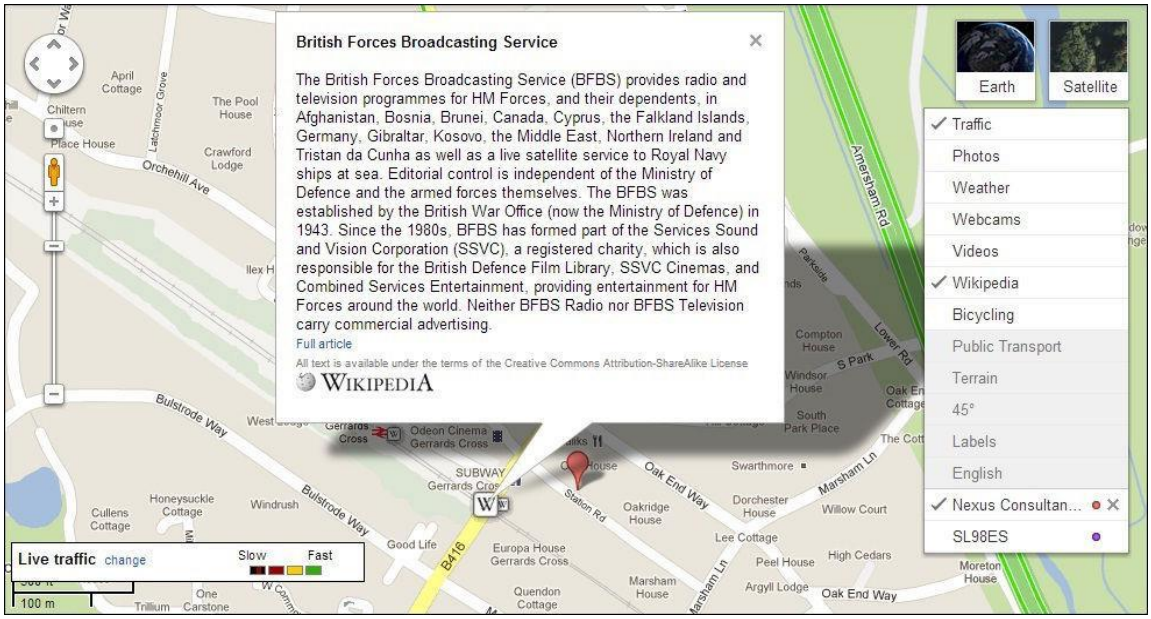

Illustration 2. Web application with map component

2.3 Map components

2.3.1 Layers

One of the key features of GIS is to display data from different sources with varying coordinate systems and projections in a map. A map consists of one or more layers which are thematic representations of geographic information [11]. A layer is a special container for holding features of one type and can have different data sources. Each layer can have its own coordinate system. Due to the fact that map uses exactly one coordinate system, while choosing another layer to show, this

layer coordinate system will be on the fly reprojected on the map's coordinate system.

There exist two kinds of layers: base layer and non-base layer. Only one base layer can be enabled at a given time. That layer specifies map properties, such as projection, units, number of zoom levels available on the map. Non base layer do not control map properties and multiple non base layers can be enabled at a time. Every layer can be used as a base layer - previous base layer becomes non base layer and selected layer becomes a base layer. A base layer can be overlaid with multiple non base layers which must share the same coordinate system as the base layer [11].

2.3.2 Web services for publishing vector data

Usually GIS web applications use different web services to publish geographic data on a map. Nowadays there exists a great amount of different web services, but the most commonly used are Web Map Service (WMS) and Web Feature Service (WFS). Web service is an independent software component that can be accessed through the Internet in the application. Each web service has different approaches for publishing vector data: WMS renders vector features as images which end-user cannot edit or spatially analyze, while WFS does requests for geographical features across the web using platform-independent calls and supports feature manipulation (updating feature, deleting feature, etc.).

For example, layer holding a world map served by WMS can be used as a base layer and two vector layers as non-base layers overlaying base layer and rendering set of geographical objects. As a result, end-user sees an unchangeable image of a world map and changeable set of vector features rendered on a map.

2.3.3 Map tiling scheme

While rendering maps and spatial data on them, it is essentially important to define fixed zoom levels in which the map can be seen. Then for every zoom level the map is rendered with a predefined configuration and cut into tiles, which are usually 256 x 256 pixel images (see Illustration 3). A client can directly access

these tiles which improve the performance: it allows ensuring only the image within the user's view are requested and loaded by the client [14]. Alternatively, single-tile approach can be used which loads all map tiles at once and results in one large image for each layer. However, this can lead to disastrous consequences: if the loading of one tile is delayed because of network related errors, then the whole layer will not be drawn until the request of that one tile returns, successful or not [11].

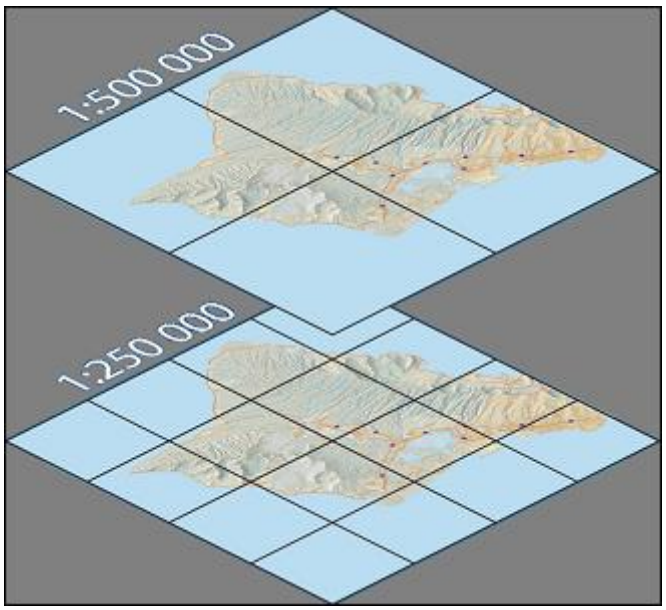

Illustration 3. Map tiling scheme [21]

Additionally, it is important to define a bounding box (BBOX) parameter, which determines the extent of the map. Bounding box is a rectangle oriented to the x and y axes, which bounds a geographic dataset within which all geographic features lie. By default, if BBOX parameter is not specified, all geographic datasets will be shown. For instance, if user has one layer containing features of New York and the other features of Moscow, one will see most of the World. The bounding box also determines the aspect ratio of the map. If only one of width or height is specified, the other will be determined based on the aspect ratio of the bounding box [18].

2.3.4 Rendering technologies

In order to render spatial data, browsers have the support for different technologies: Canvas, Flash, SVG, etc. So to make the process of drawing geometries in the map independent from the vector layer using concrete technology, the concept of renderer is used [12]. In addition, vector layer can have a style associated with it that is used by the renderer to draw the geometry that represents a feature [12]. For example, a layer can visualize a street network and each street is rendered individually using a specific style according to the street's classification (highway, main road, secondary road, etc.) [12].

A vector layer can have one or more strategies attached to it [12]. Strategy is a class that makes processing on the vector layer without layer knowing what is happening [12]. The most popular strategies are:

- Strategy which loads the content of a data source attached to the vector layer only once. This is usually used to load a data file in a vector layer, because it is only required to be loaded once.
- Strategy which loads content from a data source each time the bounding box of the viewport changed [12]. This strategy is useful for vector layers that dynamically loads content from a WFS server and needs to update its contents every time the map bounding box parameter changes [12].

Each library used for rendering maps can also have library specific implementations of strategies - some minor modifications can be implemented in order to improve the performance of concrete strategy.

Chapter 3. Overview of the experiment

3.1 General information about experiment

As for practical part of the thesis, a small GIS web application will be built in order to see how browsers cope with an excessive amount of geospatial data. Some popular optimization and rendering techniques will be implemented in the application: mainly clustering strategy, canvas renderer, and decimation principle. Detailed description of above mentioned methods can be found in the subsequent chapters. All results will be benchmarked and analyzed in order to give the answer, whether these techniques are helpful, or vice versa, they just slow down the work of the application. Moreover, analysis will try to give an answer what additional aspects should be taken into account while developing GIS web applications that simultaneously need to display large amount of geographical data.

Because of the limited time frame allocated for writing this paper and creating web application, this thesis restricts the focus only on one type of the vector data, namely point features. Point features were chosen mainly because their format is not so extensive, so it is easier to deal with their geometries and implement different optimization techniques. Since at the time of writing this paper there was no open-source solutions concentrating on client side of the web application and implementing current optimization techniques considering other vector data types, this work assumes that for other vector data types the result will be the same as a result achieved with point features. In order to reduce complexity of the prototype and focus more on the client-side of the GIS web application and implementation of optimization methods, geographic data used in experiment will be randomly generated and not be queried from the database.

User interface of the web application consists of two main components: input fields, where user can chooses how many points one wants to visualize and what optimization technique to use, and a map, where user can view all the geographical data. Number of displayed data and its structure depends on user choice and is perceived from the input fields. All input field validation takes place in the server-side and if there occurs an error then appropriate message is shown to the user.

3.2 Technologies used in experiment

Created web application consists of two parts: user interface, i.e. what end-user sees in browser, and service code that runs on the server side. Grails web framework was chosen as a main framework for holding both client- and server-side code. As for front-end development, an OpenLayers library (version 2.11) was chosen to embed a map in browser as it is one of the most popular open-source libraries for visualizing maps. Besides this, HTML, CSS, JavaScript, Twitter Bootstrap (version 2.2.2) and jQuery (version 1.8.0) were used for creating and showing elements in the browser. As for back-end development, Groovy language was chosen to implement service code, business logics and some optimization methods. CloudBees service[1] is used as a web hosting provider in order to access the application from the Internet. The application itself can be accessed via the following URL: http://mappoints.timasjov.cloudbees.net/.

[1] Homepage of the service: http://www.cloudbees.com/

Chapter 4. Methods used to speed-up the process of rendering

4.1 Background

There can appear a serious problem when vector layer contains lots of features (more than 10000 objects) and an ugly effect can appear in a map. Depending on the zoom level and layer style, features can be rendered overlapping each other's (see Illustration 4).

Illustration 4. Interactive map containing great amount of overlying features

This can lead to very serious problems: user-interface can become terrible and visual depictions may not properly relate to desired uses - it will be practically impossible to understand where the actual object is located and difficult to select necessary feature. In this case self-overlaps can be deformed by simply replacing punctual symbols with other aggregated symbols.

Another obstacle to the development of vector web mapping is performance. Web maps must be fast maps, and existing web maps based on vector data usually do not meet the minimal requirements in terms of display speed [14]. However, taking

into account that client device memory, processing and connection capacities are always improving, web mapping with vector data is acceptable approach [14]. Compared to raster web mapping, vector data fast representations methods and techniques are not well analyzed and established yet, however, approaches exist to improve considered processes [14]. Such methods and tools are, for example, different clustering strategies, pre-rendering of used vector data, spatial indexing, vector tiling, data generalization, etc. Paragraphs 4.2, 4.3, and 4.4 contain detailed description of some of the most popular vector mapping performance optimization methods.

4.2 Cluster strategy

One of the most common and yet simple ways that allows user successfully to deal with a large number of geographic features is clustering strategy. Cluster is a homogeneous group of objects with similar characteristics. Given the set of features within a layer, cluster strategy on the fly computes the distance among the features and if the distance between them is relatively small, it clusters them. Distance directly depends on the zoom level and most commonly is calculated in pixels, because standard metrics (for instance, kilometers or miles) have different meaning in different zoom levels. So if the distance between features is smaller than predefined number of pixels, a new cluster is created and objects are added to the cluster. The clusters are nothing more than point geometry features with a new count attribute holding references to the vector layer features it contains [12]. When zoom level changes, following steps take place:

- New cluster stores a reference to the previously computed clusters (cluster related to the previous zoom level he came from)
- New cluster is computed and added to the vector layer
- Start a tween to animate from the previous to the new cluster positions

If user zooms in the view, it means one go to a level with more clusters than the previous one, that is, a cluster at level N becomes M clusters at level N+1 [12]. These means that animated cluster makes use of an extra array of clustered features (the previous one). The vice-versa occurs for zoom out actions when M clusters at level N becomes one cluster at level N-1 [12]. At the end of the zooming process positions of the clusters should be refreshed, which implies redrawing the

vector layer. As a result, it takes additional time to perform the necessary calculations. This drawback does not play significant role in the process of visualization of a relatively small number of features, but it can matter when user needs simultaneously to render tons spatial objects. Also, OpenLayers applies special animation that indicates the user about zoom level change and new cluster division or merge.

As a result of applying clustering strategy, there will be no feature overlaps any more, map will have more user-friendly design, appearance and look will be improved. Also, map becomes more consistent and intuitive which directly helps to improve human-computer interaction as it is easier to understand where concrete feature is located (see Illustration 5).

Illustration 5. Geographical data clustering

4.3 Decimation principle

Another strategy that will be discussed is decimation principle, known as subsampling. Decimation principle used in GIS is very similar to the decimation principle used in Roman army or signal processing, namely it is a method used as a low-pass filter. In GIS, the use of this principle helps to reduce the number of features to be rendered on a map.

Subsampling can drastically increase not only the overall performance, i.e. the process of embedding maps to the end-user, but also incredibly improve the geovisualization usability: as the number of rendered features decreases, it becomes easier to read and understand the semiotics and meaning of the map, provided that the map style stays the same. The main advantage of subsampling is that due to the smaller dataset, the data processing and transfer are faster, as it is needed to transfer less objects to the view. As a direct consequence of this, browsers needs less memory and machine resources to render required dataset.

Decimation principle takes place before the data is rendered on the vector layer. After the content of the data source is loaded, following steps take place:

- Special function loops through the dataset and leaves only each n-th feature. The number of features to be removed from original dataset should be manually defined
- Processed dataset is added to the vector layer and drawn on the map

This means that this strategy does not use any additional data structures nor performs any additional calculations. No special animation or style changes take place.

Primarily decimation principle helps not only to improve web vector mapping performance, but also to solve scalability problems as well as enhance map readability and accessibility. In addition to this, user will be able to more accurately track and manage the content of the map.

4.4 Data encoding method

The speed of data transmitting and visualization greatly depends on the architecture that web application is using. Technology trend within information technology has made it possible to move towards distributed computing and Service-Oriented Architectures (SOA) and to no longer use the familiar computing paradigms such as static server-client approaches as their centralized architecture results in harder management solutions and practices for data representation. Modern large scale systems require more flexible asynchronous communication models to cope with the high number of participants and transfer of larger datasets

between them [16]. Precisely for this reason more and more geographic web applications started to implement service-oriented architecture as it helps to create loosely coupled and component oriented architecture and provides greater interoperability to cope with very large datasets of high dimensionality.

SOA can be implemented using different technologies, including Simple Object Access Protocol (SOAP), Representational State Transfer (REST), Common Object Request Broker Architecture (COBRA), etc. All implementations provide their specific set of protocols and interfaces for accessing different data sources and information exchange services.

In the web application client usually downloads vector data and displays it on top of raster images, which are usually published through different web services [14]. SOA helps to solve certain problems associated with interoperability between architecture components, but the well-known limit of this approach is the long time usually necessary to transfer and decode vector data. Different data formats can be used for interfacing with SOA service, but XML format is the common choice for transmitting structured data. However, usually special GIS data formats are used for encoding, compressing and transmitting geographic data structures as processed files are smaller than XML files. For example, a GeoJSON document containing dataset of two features with attributes (see Illustration 6).

```
{
   "data": {
       "type": "FeatureCollection",
       "features": [
           {
               "type": "Feature",
               "geometry": {
                   "type": "Point",
                   "coordinates": [59.4339, 24.7549]
               },
               "properties": {
```

```
                    "name": "Tallinn"
                }
            },
            {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [58.3706, 26.7157]
                },
                "properties": {
                    "name": "Tartu"
                }
            }
        ]
    }
}
```


Illustration 6. Dataset containing two features in GeoJSON format

Provided that 1 byte is needed to represent 1 character if UTF-8 encoding is used, it is simple enough to calculate how many bytes is required to transfer each data file. GeoJSON file contains 270 characters, which respectively gives 270 bytes, where each feature takes 111 characters (or bytes). For instance, user views a 256 x 256 pixel area and is at a zoom level so high that each feature is rendered as one pixel. Assuming that the area is full of objects and that GeoJSON file format is used to transfer features, it takes 111 x 65536 = 7277493 bytes or about 6.94 megabytes storage cost. It requires a lot time and machine resources to perform the transfer and decoding of such dataset. As can be seen from the example, the use of service-oriented architecture and special GIS data formats cannot completely solve the problem of representation and transportation of spatial objects.

Another possibility to transfer data over Internet is to use a binary scheme format, where each byte means specific information about an object. User can on one's pat decide what information one wants to hold in a single band image (i.e. black and white). There are 256 different options that can be used for encoding to store information in 1 byte. For example, it can be defined that 1 means that the object color is green, 2 - yellow, and 3 - purple and so forth. Thus, the raw storage cost of such binary scheme is 256 x 256 x 1 (one band only) = 65536 bytes or 0.0625 megabytes. Furthermore, different image compression techniques can take place in order to store or transmit data in a more efficient form. Described actions take place in server-side map engine which does all the work to generalize the data and create a meaningful tile for every zoom level that has the data encoded.

Before the advent of HTML5 it was not easy to present binary data in JavaScript. All binary data had to be escaped so that it could be placed into a string element in JSON and passed forward to special GIS JavaScript libraries. As a result, a lot of extra time was needed to process the data. However, solution to the problem was found when new features of HTML5 came out, particularly canvas. Canvas element is a part of HTML5, which allows drawing graphics on the fly. Its main advantage is that it can render bitmap images and manipulate the pixels directly. The appearance of canvas element has significantly affected the whole process of spatial data rendering: now server can send the data over the wire encoded on what appears to be an image, then client can add that image into an HTML5 canvas, which will render it on a map, which represents canvas drawing surface. Many GIS JavaScript libraries (Leaflet, OpenLayers) already support canvas, which significantly simplifies the process of rendering. Furthermore, HTML5 canvas does not even need to be used to draw, it can be only used as a binary decoding mechanism that will generate JSON objects. GIS client-side map libraries will process decoded data, create real vector objects and add them to the map. Also, different event handlers can be added to the vector features.

Canvas element made it possible to decode a huge amount of meaningful geo-referenced data in a highly compressed format and manipulate them in JavaScript.

This technique guarantees that the user will see a dynamic map containing tons of vector features that can be quickly refreshed.

Chapter 5. Experiment

This chapter contains background information about the experiment, explains the realization of implemented rendering methods and compares them by running performance tests. Section gives a first evaluation of each implementation and summarizes the results.

As a result of each implementation, user will see dynamic map containing required number of geographical objects. All implementations use default tiling strategy: the tiles are drawn on the map once they are ready. In some cases it could produce a flickering effect when the map is panned several times in short intervals.

5.1 Analysis and implementation

5.1.1 Default renderer

The first scenario considered in the experiment is rendering geographical objects using default OpenLayers renderer and without applying any optimization techniques. OpenLayers uses SVG (Scalable Vector Graphics) as default renderer. All major browsers have a support for SVG and can render vector objects directly which makes the use of *<svg>* HTML tag. All actions take place in the thick client which means that once the data is generated and initialized, it is in the browser and there is no need to request the same data again.

To begin with, vector layer is created using *OpenLayers.Layer.Vector* class and added to the map which, in turn, is created using *OpenLayers.Map* class. Vector layer will hold vector objects that will be subsequently created. Once the layer is initialized, OpenLayers creates SVG element and inserts it into the layer container. SVG renderer is responsible for drawing geometries and other elements (labels, borders, texts, etc.). Once renderer is initialized, necessary number of features is generated using *OpenLayers.Feature* class and added to the vector layer. Afterwards vector layer is added to the map which means that SVG renderer starts drawing features on the map. Also, special styling (afterwards default styling) is applied using *OpenLayers.Style* framework: features are red with black

stroke. This framework is used to control the styling of each object attached to vector layer allowing the use of custom styling properties and rules.

5.1.2 Clustering

Cluster implementation developed for this thesis is the first considered technique that tries to improve web vector mapping performance. Cluster implementation developed for this thesis uses two different approaches regarding creation of clusters: client-side clustering and server-side clustering.

Speaking about the client-side clustering, one of the first things to note is that all processes are performed on the client side. As in the default renderer technique, map and vector layer are created in the same way. SVG renderer is also used to draw geometries. After initialization of the map and vector layer, required number features are generated as standard *OpenLayers.Feature* features and clustered using *OpenLayers.Strategy.AnimatedCluster* class which is a subclass of a default *OpenLayers.Strategy.Cluster* class, so it inherits all the properties from the superclass and extends some extra features. Afterwards created objects (i.e. features that represent clusters) are added to the vector layer which is, in turn, added to the map. Animated cluster strategy holds original features in cache and loads clusters dynamically depending on the user needs. When user changes zoom level or pans the map, all objects in vector layer are destroyed and new clusters are generated from cached features. To sum up, one can say that this implementation tries to open new opportunities in visualizing vector data on the client side.

Server-side clustering implementation developed for this thesis uses a slightly different approach. The main computations are performed on the server-side and thin client is used only for rendering. The workflow is as follows: client sends a request to the server with number features user wants to render as parameter. Server generates required number of objects providing each object with random geometry. Thereafter server initializes map by dividing it into squares (or cells). Each cell is relative to fixed decimal degrees of latitude and longitude values and its size does not depend on the map zoom level. When map grid is ready, special

method is invoked, which loops through all features and for each feature by its coordinates determines in which cell it is located. In order to improve performance, *java.util.Map* interface is used to hold cells, which maps cell coordinates to the remaining cell properties. Features inside a cell are then grouped into cluster. If there are no features in the cell, then cluster will not be created. Cluster coordinates are found by calculating the average latitude/longitude for the properties of the corresponding cell. After the process of clustering is finished, all created objects (i.e. clusters) are returned back to the client who transforms them into standard *OpenLayers.Feature* features, adds them to the vector layer and renders on a map. If user changes zoom level, then all clusters have to be recalculated and rendered again - all features in vector layer are destroyed and the original workflow is repeated again. The grid-based approach has a fast processing time performance and directly depends on the size of the grid and the number of features to render.

Special styling is applied for both clustering techniques: in order to distinguish objects three colors are used to style the clusters depending on the number of features they contain:

- Red, if cluster contains more than 50 features
- Yellow, if cluster contains from 15 to 50 features
- Green, if cluster contains less than 15 features

5.1.3 Decimation principle

Another strategy to be implemented in the experiment is decimation principle. The essence of this technique is to reduce the initial dataset and simultaneously show fewer objects on the map. These actions may greatly improve performance and scalability issues. As well as in client-side clustering, all actions are performed in the thick client. OpenLayers default renderer is used to draw objects. Map and vector layer are also created and initialized in a standard way as in the clustering technique. After necessary number of features are generated as *OpenLayers.Feature* features and identified, special function loops through the created dataset and leaves only each 10-th feature, i.e. features nr 1, 11, 21, etc.

Just as in the previous strategies, features are added to the vector layer and rendered on a map. Considered strategy uses default styling.

5.1.4 Canvas rendered

One more technique that focuses on improving web vector mapping performance is the use of the canvas renderer. OpenLayers has built in support for canvas technologies already available in browsers which makes use of the *<canvas>* HTML tag. OpenLayers canvas renderer is based on 2D canvas drawing element and is responsible for drawing objects on the map. It is not used as a binary decoding mechanism (see data encoding method in Chapter 4).

Map and vector layer are created in the same way as in the default renderer technique. Once the vector layer is initialized, OpenLayers creates HTML canvas element and inserts it into the layer container. Canvas renderer draws objects using *redraw()* method, which draws features using the *lineTo()* and *stroke()* methods. Once the renderer is initialized, required number of features are generated as *OpenLayers.Feature* features and added to the vector layer. Just as the vector layer is added to the map, canvas renderer will automatically start drawing features on the map.

It should be noted that canvas renderer has several disadvantages, for example, it redraws the whole layer when even a single feature needs to be redrawn. In addition, like the SVG renderer, canvas renderer does not refresh the view during the dragging of the map due to performance reasons [11].

5.2 Test environment

All test cases were run with a 1026 x 500 pixel map in browser Chrome 25.0.1364.97 on Windows 7 with a quad core CPU (3.3 GHz) and 16 GB memory RAM. Browser Chrome was chosen mainly because recent studies indicate this is the fastest in performance tests [20].

5.3 Performance evaluation

The prototype was tested on following datasets: 1000, 10000, 100000, 500000, and 1000000 features. Following test cases were executed for each dataset:

- Render all features - the initial drawing of the vector layer containing specified number of features
- Change zoom level - the process of zooming in to a specific to zoom level
- Select feature - hovering the mouse over the specified feature. This test case was carried out on a random feature.

In order to evaluate performance all test cases were executed 10 times and then average result was found. Such a large number of tests carried out will exclude element of randomness and produce reliable results. Before each test browser cache was cleared in order to avoid temporary storage of static resources. Time-taking was measured in milliseconds and was started after the server response to the input validation request. Average results for each test case can be found in Table 1, 2, 3. Further information about each test case can be found in Appendix A.

Test case 1: Render all features

The table below (see Table 1) shows an average time needed to draw a vector layer containing specific number of vector objects.

Table 1. Average rendering time (milliseconds) for test case "Render all features"

	Number of objects				
Technique	**1000**	**10000**	**100000**	**500000**	**1000000**
None	42.2	367.8	4454.3	26620.2	54434.4
Clustering (client)	37.3	201.3	1901	9827.3	19835.4
Clustering (server)	203.4	376.5	1870.9	5429.7	10916.3
Canvas	67.4	439.5	4657	26640.7	55258.7
Decimation	17.9	143.6	1277.7	6128.9	23108.1

To start with, results in Table 1 indicate that OpenLayers default SVG renderer is slightly faster than canvas renderer. The latter one is approximately 16% slower when showing less than 10000 objects and 1% slower when there are more than 100000 vector features on the map. Moreover, SVG renderer additionally creates

two special SVG group elements in the DOM which hold basic feature attributes, such as ID, type, geometry, style. This allows accessing every feature from DOM by ID. OpenLayers canvas renderer does not hold any feature attributes and just draws geometries using 2D canvas on the drawing surface. However, both renderers do not cope well with their duties when it is needed to show more than 100000 vector objects on the map as user will have to wait almost 5 seconds. Furthermore, features density goes too high and map becomes unreadable. In this case some optimization techniques may be applied.

According to the received data, both optimization techniques are extremely effective and can reduce the time needed to draw features. For instance, rendering of 100000 features using client-side clustering and decimation principle is respectively 57% and 71% faster when comparing with the time needed to show all vector objects using SVG renderer. Furthermore, server-side clustering allows rendering features even faster than above mentioned methods. However, this algorithm is not so efficient when there is less that 10000 vector objects on the map as it takes additional time to transfer features from the server. But in general, one has to keep in mind that clustering and decimation principle are two completely different techniques. Faster execution time using clustering is explained by the fact that objects are grouped according to their geometry and only clusters representing features are rendered. Though, speaking about decimation principle, it should be kept in mind that only N/10 objects are shown on the map. It also worth noting that both clustering methods are with linear time complexity and therefore are more suitable techniques for large datasets as the amount of time is directly proportional to the number of vector objects.

This test case demonstrates that canvas and SVG renderers have practically the same capabilities in visualizing vector data on the client side. They are not so good at visualizing large datasets and therefore special optimization techniques should be applied in order to improve performance.

Test case 2: Change zoom level

Before running this test case, 500000 features were randomly generated and rendered on the map. Table below (see Table 2) shows how long it takes to zoom in to a specific zoom level (level 5) and how much objects are visible on the screen within the current map view. Zoom level 5 was chosen because after zooming it becomes possible to distinguish where exactly vector object is located.

Table 2. Average response time for test case "Change zoom level"

Technique	**Time** (ms)	**Visible Objects** (number)	**Time to render one object** (ms)
None	9641.7	11442.4	0.84
Clustering (client)	5296.1	10332.1	0.51
Clustering (server)	5281.7	13865	0.38
Canvas	4578.4	11417.9	0.4
Decimation	2115	1134.5	1.86

The first thing to note is that in all cases except decimation principle, at the end of the zooming process there is approximately the same number visible objects on the map within the user's view. Once zooming process ends, OpenLayers checks prospective bounding box parameter and automatically recalculates which objects should be shown. These calculations heavily depend on the renderer. For example, SVG renderer automatically recalculates coordinates of all geographical shapes, checks if feature style has not changed and renders them on the map, while canvas renderer does not perform any calculation and just redraws features depending on the BBOX parameter. Due to this restrictions canvas renderer is nearly twice as fast as SVG renderer when it comes to zooming. This test case clearly demonstrates that zoom time directly depends on the renderer.

The second thing to note is that number of visible objects on the screen using decimation principle is about 10 times smaller comparing with the same parameters using other techniques. Despite the fact there are rendered only N/10 objects, it takes three times longer to render one feature. This suggests that zoom time is not dependent on the number of vector objects.

Results in Table 2 indicate that both clustering techniques can improve vector web mapping performance. This is proved by the fact that after zooming process it takes 39% less time to render one vector object when using client-side clustering and 68% less time when creating clusters on the server.

Test case 3: Select feature

Before running this test case, 500000 features were randomly generated and rendered on the map. The table below (see Table 3) shows an average time needed select a randomly chosen vector feature. In this test case, time measurement starts when the mouse pointer enters vector feature and finishes when tooltip appears beside the selected element.

Table 3. Average response time for test case "Select feature"

Technique	**Time** (ms)
None	2.9
Clustering (client)	27.3
Clustering (server)	12.0
Canvas	2.9
Decimation	2.8

Results listed in Table 3 indicate that default OpenLayers renderer perfectly copes with its work when it comes to events and event handlers. The same result is also obtained with canvas renderer. In accordance with the achieved data, it is needed less than 3 milliseconds to select the vector object. Nevertheless, both renderers have different methods for determining the feature to select. In SVG, specific event can be attached to concrete feature, while in canvas event in attached to the whole drawing surface. Canvas renderer determines the position in pixels where user has clicked, finds all possible features located in that area and executes the event.

The result obtained when applying decimation principle practically does not differ from previous results, because default OpenLayers renderer is used. However, the use clustering technique is not very effective: it is 89% (client) and 76% (server) slower than the above mentioned method. This stems from the fact that in the former case one feature represents exactly one feature with its geometry and

additional attributes, while in the latter case one feature rendered as cluster represents N vector objects. Cluster holds information about all N objects, therefore more time is needed to fetch necessary data, insert it to the DOM and show a tooltip. Moreover, in some cases cluster tooltip can be of a very large size and may do not fit on the screen. This factor may lead to the fact that map usability will suffer and the end-will have difficulties with navigation on the map.

OpenLayers has a great support for events and event listeners. It allows to almost instantly get information about a certain object. This test case demonstrates that if the purpose of the web application is to quickly find and display information, then optimization strategies will not improve the overall performance.

5.4 Conclusions

By comparing received results achieved during performance evaluation, it can be concluded that each technique has its benefits and drawbacks. For instance, clustering (either server- or client-side) technique may be applied when it is needed simultaneously to show tons of geographical objects or perform zooming operations, while decimation principle when handling object events. Tests showed that OpenLayers default SVG renderer is a little bit slower than HTML5 canvas renderer, but the difference is not so tangible. To sum up, concrete renderer and optimization method should be chosen depending on the type and the purpose of the web application.

5.5 Possible improvements

Several possibilities exist to improve rendering and usability issues. One such possibility is to apply progressive object loading. The principle of progressive transmission and streaming methods is to load necessary dataset progressively and display loaded data continuously, until the full transmission is complete. These methods have been developed for many kinds of data including raster and vector data. Each vector object that has been downloaded is displayed starting with a simplified view progressively enriched with additional details [14]. Also, progressive loading of the data may also be obtained using asynchronous queries to the server

for each vector object [14]. This allows user almost immediately to start operating with visible features, instead of waiting all dataset to be downloaded. However, progressive loading do not contribute to solve the performance problems or improve overall rendering time: they, on the contrary, could drastically improve the user experience and usability issues connected with manipulation of display parameters. Nowadays such methods have become very popular and are already used in many different GIS web applications.

Also worth noting that vector web mapping performance issues can be improved by efficiently querying spatial data and serving only the relevant data to the user's client. Given technique is called vector tiling. The underlying idea is that offline preprocessor prepares the dataset by decomposing the vector data into different parts, summarizes, and chops it into same geographic regions. In other words, vector data on the server is cut into pieces and divided up into exactly the tiles that the raster image is divided into. Every time the map is updated (i.e. moved, panned, etc.), screen bounding box is calculated and if it has changed, all tile geometries will be removed and new tiles for the new bounding box will be loaded. After the features are added to the map, they are also cached and stored in the user's browser cache. Vector tiles are used on a session-by-session basis and are retrieved from the browser cache when the same request (URL) is issued more than once [6]. Vector tiles are only cached on the client-side and no additional server actions is required. Vector tiling can be especially useful at a small scale as it prevents client from multiple data requests and ensures that only features within the current map extent will be retrieved. Despite the fact that considered technique can greatly improve performance and user experience issues, it has several limitations, namely, startup time and additional browser memory required to download and preprocess a large dataset.

Chapter 6. Conclusion

This chapter concludes the research done in this thesis. After summarizing the results, potential improvements as well as the future work will be presented.

6.1 Work overview

The main purpose of the current paper was to study which features could provide benefits for fast representation and generalization of large volumes of spatial data. Work concentrates on vector data type and provides the overview of the most common technologies used to speed-up the process of rendering objects on the map. The main emphasis was on the practical implementation of several optimization methods, namely, clustering, canvas renderer, and decimation principle. Map readability and geovisualization usability issues were also discussed.

Prototype created as a practical part of the thesis was used as the input for measuring the effectiveness of a particular technology, where the time needed to render vector objects as well as map readability were considered as main criteria. The results were computed for various cases, using different number of objects. The data obtained during the experiment indicate that none of the examined approaches allows to solve alone the performance problem - each optimization method has its advantages and disadvantages. For example, while comparing client-side clustering with rendering objects without applying any strategies, then it allows to render features in double-quick time, but it takes up to 10 times longer to select the feature.

If the goal of the GIS web application is to provide an efficient vector web mapping in satisfying time and allow manipulation with the data, then several optimization and rendering approaches should be used together. If the client faces performance issues, it means the vector data have not been simplified or generalized enough [14]. Depending on the specific context, the user needs and the nature of the data one wants to display, suitable cartographic visualization techniques should be developed [14].

With the simultaneous display of a large number of vector data on a map, it can become very dense and it will be difficult to distinguish objects. Maps readability could certainly be improved by clustering objects, replacing them with other aggregated symbols or removing overlapping objects. However, these actions need additional memory and processing capabilities, which can result in the inability to display required dataset in satisfying time. A good balance between map readability and the time needed to render objects on the map has to be found.

6.2 Potential improvements and future work

As this is mostly academic work so far, there remains need for improving server-side clustering algorithm and user interface of the created GIS web application. For example, possibility of selecting cell size depending on zoom level can be added to the algorithm.

This work can be used as an input for future works in investigation and implementation of different optimization techniques. One of the promising future works related to this research could be the introduction of the spatial indexing. Spatial indexing makes possible the development of new innovative cartographic visualization techniques, especially dynamic visualizations with moving and changing objects [14].

Kiire kaardiliides mahukate vektorandmetega

Resümee

Dmitri Timašjov

Tänapäeva maailmas esineb sageli vajadus kujundada ja arendada igasuguseid veebipõhiseid geograafilisi infosüsteeme (GIS), praktilistel ja teaduslikel eesmärkidel. Tihti juhtub, et geograafiliste andmete maht, mida infosüsteem peab samaaegselt kasutajale näitama ja edastama on nii suur, et veebilehitsejatel võivad tekkida probleemid andmete visualiseerimisega ja kiire esitamisega.

Peamine töö eesmärk on uurida kuidas olemasolevad geoandmete visualiseerimise vahendid toetavad suuri geograafilisi andmekogumeid (rohkem kui 100000 objekte), nende kodeeringut ja kujutamist kaardil. Tähelepanu pööratakse ülevaatele, kus kirjeldatakse kõige sagedamini kasutatavad meetodit ja tehnoloogiad, mis töötavad väga suuremahuliste ja mitmedimensionaalsete andmetega ja võimaldavad määrata kuvatavate andmete hulka erinevatel suumiastmetel. Töös võrreldakse visualiseerijaid (SVG, canvas), serveri- ja kliendipoolseid klasterdamise tehnikaid ning detsimeerimise printsiipi. Lisaks uuritakse geoandmete visualiseerimise kasutusmugavust. Väga suuremahuliste andmete kuvamisel ei ole kasutajale tihtipeale arusaadav, mida on kaardile visualiseeritud.

Lõpptulemusena luuakse väike GIS veebirakendus, kus rakendatakse mõningaid optimeerimis- ja visualiseerimistehnikad, et näha kuidas veebilehitsejad saavad hakkama väga suuremahuliste andmemahtudega. Rakenduse testimisel saadud tulemused korrastati ja analüüsiti, et hinnata valitud meetodite töökindlust ja jõudlust. Tulemused näitasid, et igal optimeerimise meetodil olid omad plussid ja miinused ning konkreetne tehnika tuleb valida sõltuvalt veebirakenduse tüübist.

References

[1] A. M. MacEachren, M-J. Kraak (2000): Research Challenges in Geovisualization.

[2] B. E. Davis (2001): GIS: A Visual Approach.

[3] Google Inc., Google Fusion Tables, Geographic Data Types, [Online]. Available: http://support.google.com/fusiontables/answer/174680 [Accessed 29 January 2012].

[4] K. Eldrandaly (2007): Expert Systems, GIS, and Spatial Decision Making: Current Practices and New Trends.

[5] S. Mustiere, M. Sester, F. van Harmelen, P. van Oosterom (2009): Generalization of Spatial Information.

[6] Esri Headquarters, Out of the Box Vector Tiling Using Feature Layers, [Online]. Available: http://blogs.esri.com/esri/arcgis/2011/06/06/out-of-the-box-vector-tiling-using-feature-layers/ [Accessed 20 February 2013].

[7] P. B. McNeally (2008): Holistic Geographic Visualization of Spatial Data with Applications in Avalanche Forecasting.

[8] D. R. Green, K. Bojar (2010): "YthanView" – Visualizing and Estuary and Virtual Fieldwork at the Ythan Estuary, Scotland, UK.

[9] J. Gaffuri (2011): Improving Web Mapping with Generalisation.

[10] D. De Chiara (2011): From GeoVisualization to Visual-Analytics: Methodologies and Techniques for Human-Information Discourse.

[11] T. Sauerwein (2010): Evaluation of HTML5 for its Use in the Web Mapping Client OpenLayers.

[12] A. Santiago, Animated Marker Cluster Strategy for OpenLayers, [Online]. Available: http://acuriousanimal.com/blog/2012/08/19/animated-marker-cluster-strategy-for-openlayers/ [Accessed 25 January 2013].

[13] J-C. Muller (1991): Generalization of Spatial Databases.

[14] J. Gaffuri (2012): Toward Web Mapping with Vector Data.

[15] P. Wang, T. Doihara (2002): Automatic Generalization of Roads and Buildings.

[16] G. Aydin (2007): Service Oriented Architecture for Geographic Information Systems Supporting Real Time Data Grids.
[17] S. Wiesmann, B. Stern, L. Hurni, M. Werner (2013): Generalisation of Map Data.
[18] GeoServer, WMS Animator, [Online]. Available:
http://docs.geoserver.org/2.1.3/user/tutorials/animreflector.html [Accessed 20 March 2013].
[19] L. Luqun, L. Jian, T. Yu (2002): The Study of Web GIS Architecture Based on JNLP.
[20] M. Muchmore, Performance, [Online]. Available:
http://www.pcmag.com/article2/0,2817,2349496,00.asp [Accessed 19 March 2013].
[20] M. Muchmore, Performance, [Online]. Available:
http://www.pcmag.com/article2/0,2817,2349496,00.asp [Accessed 19 March 2013].
[21] Esri Headquarters, Cached Map Service [Online]. Available:
http://webhelp.esri.com/arcgisserver/9.2/dotnet/manager/publishing/static_map_svcs.htm [Accessed 19 March 2013].

Appendices

Appendix A. Vector data performance tests

Table 4. Rendering time (milliseconds) for the test case "Render all features" without applying any strategies

	Number of objects				
No	**1000**	**10000**	**100000**	**500000**	**1000000**
1.	89	356	5350	26058	48225
2.	40	390	4265	24691	51202
3.	38	382	4267	25911	53516
4.	37	369	4272	26251	51284
5.	38	360	4506	27193	57325
6.	37	378	4595	28881	57439
7.	37	356	4431	28093	53770
8.	34	379	4292	27333	57817
9.	35	338	4318	26261	55128
10.	35	377	4247	25530	58638

Table 5. Rendering time (milliseconds) for the test case "Render all features" with applying client-side clustering strategy

	Number of objects				
No	**1000**	**10000**	**100000**	**500000**	**1000000**
1.	63	189	1648	9376	19312
2.	36	210	1935	9211	19280
3.	34	176	2031	10613	18820
4.	35	209	1629	10394	18944
5.	26	214	1866	9827	19616
6.	27	175	1983	9259	22003
7.	54	169	2077	9755	22583
8.	36	192	1763	8492	19399
9.	31	247	2043	9708	9356
10.	31	232	2035	11638	19041

Table 6. Rendering time (milliseconds) for the test case "Render all features" with applying decimation strategy

	Number of objects				
No	**1000**	**10000**	**100000**	**500000**	**1000000**
1.	26	163	1363	6330	21849
2.	19	129	1310	5809	22771

3.	15	164	1303	6856	21834
4.	17	127	1290	5974	25677
5.	15	110	1243	5770	23358
6.	15	140	1350	6398	22551
7.	14	143	1236	5965	22858
8.	16	148	1269	5867	23031
9.	23	136	1260	5712	24069
10.	19	176	1153	6608	23083

Table 7. Rendering time (milliseconds) for the test case "Render all features" with canvas renderer

	Number of objects				
No	**1000**	**10000**	**100000**	**500000**	**1000000**
1.	79	476	5133	25545	52657
2.	80	479	4103	24753	53319
3.	49	429	4748	25229	52711
4.	57	417	4619	28867	53359
5.	60	438	4710	29451	54016
6.	77	442	4562	29530	55993
7.	76	414	4642	25979	58171
8.	92	417	4613	25866	58573
9.	51	443	4766	25654	56872
10.	53	440	4674	25533	56916

Table 8. Rendering time (milliseconds) for the test case "Render all features" with applying server-side clustering strategy

	Number of objects				
No	**1000**	**10000**	**100000**	**500000**	**1000000**
1.	202	385	2995	5575	10496
2.	189	366	2002	5404	11945
3.	199	369	1694	5402	10507
4.	206	374	1602	5419	10977
5.	185	373	1989	5851	10856
6.	322	418	1547	4906	10945
7.	191	370	1611	5760	11045
8.	171	375	1624	5375	10187
9.	187	364	2093	4797	11989
10.	182	371	1552	5808	10216

Table 9. Results for the test case "Change zoom level". For each result, number shows how much time is needed to zoom in to specific zoom level (level 5), and number in parentheses shows how many objects is visible on the screen within the current map view.

Data	None	Clustering (client)	Canvas	Decimation	Clustering (server)
1.	8810 (11419)	5885 (11271)	4110 (11310)	2127 (1211)	5576 (13639)
2.	10609 (11492)	6211 (11163)	5078 (11406)	2062 (1172)	6312 (13665)
3.	11147 (11393)	3929 (11286)	5751 (11492)	2126 (1168)	5445 (13841)
4.	8936 (11526)	3391 (11334)	4585 (11363)	2098 (1111)	5846 (13803)
5.	9021 (11396)	7269 (11376)	4228 (11486)	2169 (1138)	4724 (13943)
6.	8896 (11372)	5216 (11461)	4681 (11355)	2082 (1060)	5109 (13845)
7.	10285 (11428)	6159 (11523)	4210 (11289)	2120 (1148)	5009 (13902)
8.	9249 (11467)	3934 (11423)	4094 (11527)	2333 (1097)	4883 (14125)
9.	9147 (11495)	8014 (11316)	4962 (11374)	2051 (1128)	5055 (13822)
10.	10317 (11436)	3553 (11168)	4085 (11577)	1982 (1112)	4858 (14071)

Table 10. Response time (milliseconds) for the test case "Select feature"

Data	None	Clustering (client)	Canvas	Decimation	Clustering (server)
1.	4	9	2	2	14
2.	2	41	3	3	8
3.	3	30	2	4	13
4.	2	10	2	2	14
5.	3	18	4	2	14
6.	3	6	3	3	13
7.	2	34	3	3	14
8.	3	59	5	3	13
9.	3	43	3	4	9
10.	4	23	2	2	8

Non-exclusive licence to reproduce thesis and make thesis public

I, **Dmitri Timašjov** (01.11.1991),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Fast map interface with large number of vector objects,

supervised by Vambola Leping and Toivo Vajakas,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **10.05.2013**