ITÄ-SUOMEN YLIOPISTO

University of Eastern Finland

School of Computing

Master Thesis

18.12.2020

Word Cloud on Mopsi

Yunlong Liu

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry,
Joensuu
School of Computing
Computer Science

Yunlong Liu: Word Cloud on Mopsi
Master's Thesis
Supervisors of the Master's Thesis: Pasi Fränti and Mariescu-Istodor
December 2020

Abstract: This thesis focus on develop a tool to generate word cloud on Mopsi web application with the data geotagged photograph contains. The collision detection, shaped word cloud generation and introduction of the tool are introduced in this thesis.

Keywords: word cloud, collision detection, data visualization

Foreword

I want to give thanks to God, he prepared the study experience in University of Eastern Finland and he protect me all the way. I also want to thank God give me wisdom and guidance to help me to complete my Master's degree.

I want to give thanks to my Professor Pasi Fränti, he helped me to make my research objective clear, review my writing and give me useful suggestions. I want to thank my supervisors Mariescu-Istodor, he provided me Mopsi data for testing my application and gave me many useful suggestion on my thesis writing and presentation. I also want to give thanks to Ph.D. student Abu Sayem, who has left us and rest in peace, maybe God accept his soul and bless his family. He helped me to integrate my work on Mopsi application and he helped me to improve my application.

I want to give thanks to my family and my fiancée Dan Gao, they give me strength to go through the hard time when I was under high pressure. They encourage me when I was struggling. I want to thanks my friend Kimmo Kuikanmäki, he helped me proofread my writing and gave me many useful suggestions.

List of abbreviations

UEF	University of Eastern Finland
AABB	Axis-Aligned Bounding Box
TF-IDF	Term Frequency-Inverse Document Frequency
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
T-SNE	T-Stochastic Neighbor Embedding
SEM	Search Engine Marketing
SEO	Search Engine Optimization
JS	Java Script
API	Application Programming Interface
GPS	Global Positioning System
OSM	Open Street Map

Contents

Appendixes

1 Introduction

In the early days, people used the Internet mainly for searching for information. The internet could store static resources on the servers for users to search and query for specific information. As the internet developed, there's nowadays an increasing number of smartphone users with Global Positioning System (GPS) applications. It allow GPS service to identify the location of the phone and where photographs have been taken. It also allows users to take geotagged photographs regardless of time and location. Photos can be stored on the phone or uploaded to the web. There is a large number of photos taken by users every day

A geotagged photograph is a digital photograph that contains information on geographical location. Usually latitude and longitude are assigned to geotagged photographs by GPS service to identify the geographical location. Optionally geotagged photographs may contain other information such as keywords or labels, to identify the content of geotagged photographs. For example, geotagged photographs with keywords 'coffee' or 'hiking' tells the functions or services shown in the geotagged photograph. It may also contain the time record showing when the photograph was taken and a more detailed description than the keywords.

Geotagged photographs are widely used on smartphones and web pages. Nowadays Android phones, iPhones, and Windows phones support the functions of taking geotagged photographs. Millions of users apply and upload geotagged photographs through web service platforms such as Flickr[1], Instagram[2], Facebook, Google Earth every day. That can explain the phenomenon that taking geotagged photos becomes common when using these mobile applications and social network websites. For example, an iPhone application allows users to view photos on Apple Maps. Instagram and Google Earth can display photos on Google Maps.

Thumbnails of geotagged photographs can be displayed nicely on a map. One can open thumbnails in a region and view nice photographs, but it is difficult to know what photographs are about if there are many photographs in a region. For example, if there are 100 geotagged photographs showing on Google Maps in the region of Joensuu, a city in

[1] https://www.flickr.com/
[2] https://www.instagram.com/

Finland, and I want to know what each photograph is about, I have to spend much time clicking every photograph to check what the photograph is about and get a summary of photographs in that region. If I want to know what is the keyword most photographs are referring to, I have to check all photographs and get the result by a simple mathematics operation. I have developed a tool to solve the following problems: (1) What geotagged photographs, that are taken in a region, refer to. (2) Summary of keywords of geotagged photographs that are taken in a region.

Word cloud is a wonderful method to visualize text strings and to give a summary of a document. In my thesis, I will develop a tool to generate a word cloud with keywords extracted from geotagged photographs in the selected regions and display them on Google Maps. Instead of showing geotagged photographs in Google Maps, I show word clouds to show the summary of geotagged photographs in a region. My major task in my thesis is: (1) Generate word clouds with data from geotagged photographs. (2) Discuss the limitations of displaying the word cloud in the shape of the selected region.

1.1 Mopsi

Mopsi[3] is a location-based social network application developed by the Machine Learning Unit, School of Computing at the University of Eastern Finland [1]. Mopsi features include photograph sharing, bus timetable, and service recommendation among other services. Service recommendation is to show the variety, availability of service, and what is around. Mopsi will get user's current location as default and show available services around that location. The available services will be displayed as a list, each service contains service name, description, street address, keywords, and distance to user's current location. The available services also display a snapshot image of the service on Google Maps.

1.2 Word cloud

Word Cloud, also called Tag cloud, is a weighted words list to present visual summary of text data sets. In web technology, the word cloud is typically used to depict keywords metadata on

[3] http://cs.uef.fi/mopsi

websites or to visualize text data. Usually tags are single words and prioritized by font size in two-dimensional word clouds where words are not allowed to overlap. [2].

In the early days, word clouds were used on geographic maps to show the magnitude of different regions in font size, an early printed word cloud example is weighted English words list Douglas Coupland's Microserfs [3]. Word cloud has become popular since the 2000s because of its frequent usage on social media, especially on web pages [4]. With the development of internet technology, millions of users signed up for web blogs and word cloud became a tool of navigation to help users to reach the final web page quickly and to summarize overviews of contents from web blogs [5]. Word cloud can be generated from any document. During the process if a word appears in multiple documents the same words can be placed in one location with the same color and orientation. With this idea, the comparisons can be made easily for similar documents by utilizing similar word clouds [6]. A word cloud can also be used for the assessment of comments and evaluate risks for public safety [7].

In a word cloud a group of words combines into one and the importance of a word is demonstrated by font size with Hypertext Markup Language (HTML) elements applied. In Mopsi I use HTML Canvas elements to present word clouds. Word font size in word cloud indicates different meanings. Based on the definition of importance of words, the word clouds can be classified into three types.

- Frequency-based word cloud
- Significance word cloud
- Categorization word cloud

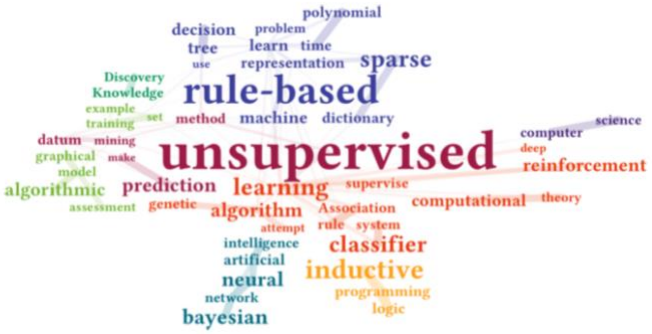
In a common *frequency-based word cloud* the font size of a word represents the frequency of occurrence of the words in a document. Frequency-based word clouds appear prevailing on Social media. For example, we often see word clouds summarizing news and word font size shows how frequently the words are used. **Figure 1 (a)** shows a frequency-based word cloud of a public letter released on the White House official website. The data i used is the text of the letter from the president to the speaker of the house of representatives and the president pro tempore of the senate. The letter title is “Letter from the President -- Report with Respect to Guantanamo”, released by the White House on January 19, 2017. In the word cloud the word “Guantanamo” has the largest font size, which means the word “Guantanamo” was mentioned the most often in this news. The word “States” has a smaller font size than the

word “Guantanamo”, which means the word “States” was mentioned less often than the word “Guantanamo”.

In *significance word cloud*, word font size represents the importance of words in all documents. *Term frequency-inverse document frequency* (TF-IDF) is a technology to reflect how important a word is to a document or corpus [8]. Scoring each word by TF-IDF and showing word score distribution is one kind of significance word cloud. In [9], it shows another kind of significance word cloud. The paper scores co-occurrences of words with CoreNLP [10] and then draws words in two-dimensional space with *t-distributed stochastic neighbor embedding* (T-SNE) [11]. CoreNLP is a *Natural Language Processing* (NLP) toolkit which is developed by Stanford University. T-SNE is a technology to visualize high-dimensional data in low-dimensional space [12]. **Figure 1 (b)** shows a word cloud ”Machine Learning” article from Wikipedia. The word “unsupervised” is the most important word for this article.

A *categorization word cloud* is another kind of word cloud in which words represent the category and word font size shows the number of items in that category. Usually it is used in *search engine marketing* (SEM) and *search engine optimization* (SEO). In SEM, words can present a website category and the font size of the word shows how many websites that category contains. In SEO, a website can be classified into different categories and each category contains web pages. If we consider a website to be a document and categories are words, then the number of the webpages where the word appears is the font size of the word. Categorization word clouds can help users to navigate the content in the information system. The website DigitalMeetsCulture uses the categorization word cloud. **Figure 1 (c)** shows the categorization word cloud in DigitalMeetsCulture website. The word “digital preservation” has larger font size than ”3D”, which means there are more articles on “digital preservation” than articles on “3D”.

(a) Letter from the President



(b) "Machine Learning" article from Wikipedia

(c) Navigation service in website DigitalMeetsCulture

Figure 1: Three different types of the word cloud. (a) Frequency-based word cloud which is generated from the data from the news. (b) Significance word cloud of an article. (c) Categorization word cloud of content category in website DigitalMeetsCulture.

According to the different appearance of the word cloud there are two types of word clouds:

- Non-shaped word cloud
- Shaped word cloud

Non-shaped word clouds have shape, but I call it 'Non-shaped word cloud' because the shape of this kind of word cloud is changing by different text data input. This is a kind of typical word cloud, words usually will be packed into a square region. This kind of word cloud does not have a fixed shape. For example, the web applications WordCloud[4] and TagCrowd[5] allow one to generate a word cloud without shape online while WordClouds[6] allows one to generate shaped word clouds. In **Figure 2**, it shows three word clouds generated by these three web applications with the same text data input. **Figure 2 (a)** and **Figure 2 (b)** are the non-shaped word clouds, these two word clouds do not have fixed shapes. **Figure 2 (c)** is a heart-shaped word cloud, the main idea of the heart-shaped word cloud is to pack all words into a two-dimensional heart-shaped space. After all words are placed, the shape of all words is heart. In the shaped word cloud the shape is usually a two-dimensional geometric space. In **Section 6** I

[4] https://www.jasondavies.com/wordcloud
[5] https://tagcrowd.com
[6] https://www.wordclouds.com

will illustrate the basic algorithm to generate a shaped word cloud with a given weighted words list.

(a) Non-shaped word cloud by WordCloud. (b) Non-shaped word cloud by TagCrowd[7]

(c) Heart-shaped word cloud by WordClouds

Figure 2: Using the same text data input to generate word clouds by three different word cloud generator web application tools. (a) and (b) are non-shaped word clouds while (c) is a heart-shaped word cloud.

1.3 Thesis structure

My thesis consists of seven sections. **Section 1** introduces what are Mopsi and word clouds. **Section 2** explains how basic word clouds are generated. **Sections 3, 4,** and **5** introduce how word clouds are generated by steps. **Section 3** shows linear font size function and logarithmic font size function and also explains how these two font size functions can affect word clouds. **Section 4** shows primitive collision detection and its limitations. Because the function of primitive collision detection is slow, I will introduce AABB and quadtree with faster performance in collision detection. Also, I show how to find a new position when a word collides with another one. In this section, I will introduce Archimedean Spiral that can solve

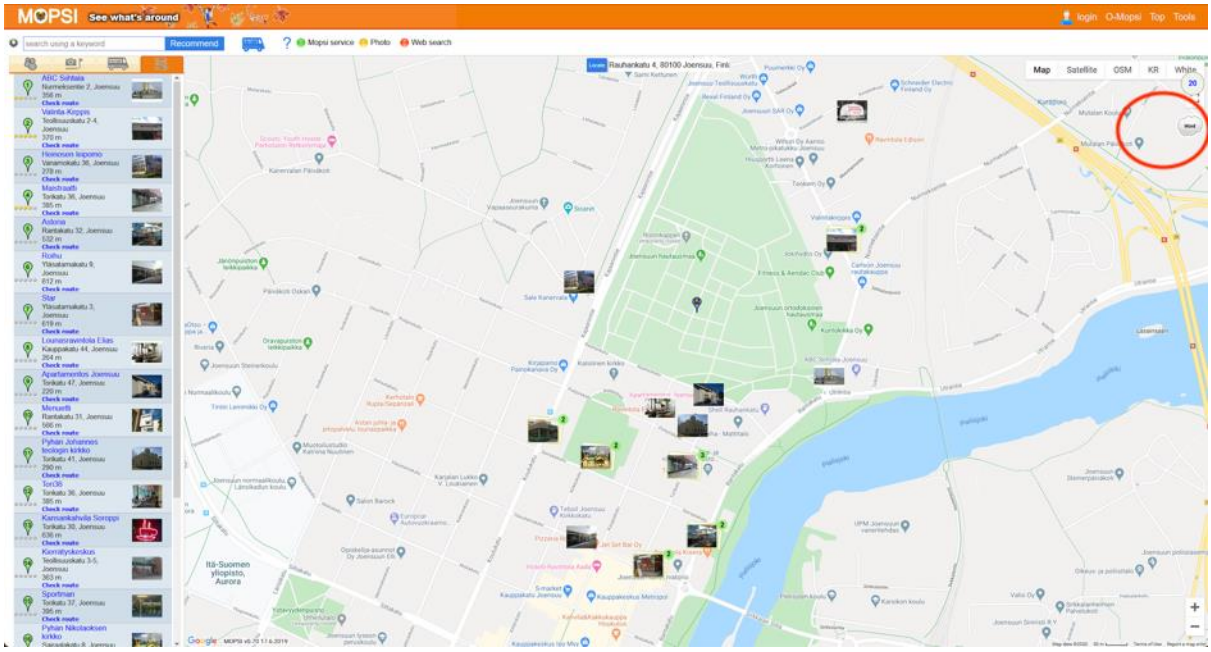
[7] https://tagcrowd.com

such problems. **Section 6** illustrates how to create city shapes and word clouds in shapes. I show how to generate word clouds in heart shape and tree shape while using various word quantities, font size functions, and different word weight distributions to explore how these elements affect word clouds and their visually recognizable shapes. In **Section 7** I will introduce how to apply word clouds on Mopsi. At the end there is the list of all references that are cited in my thesis and the appendixes.

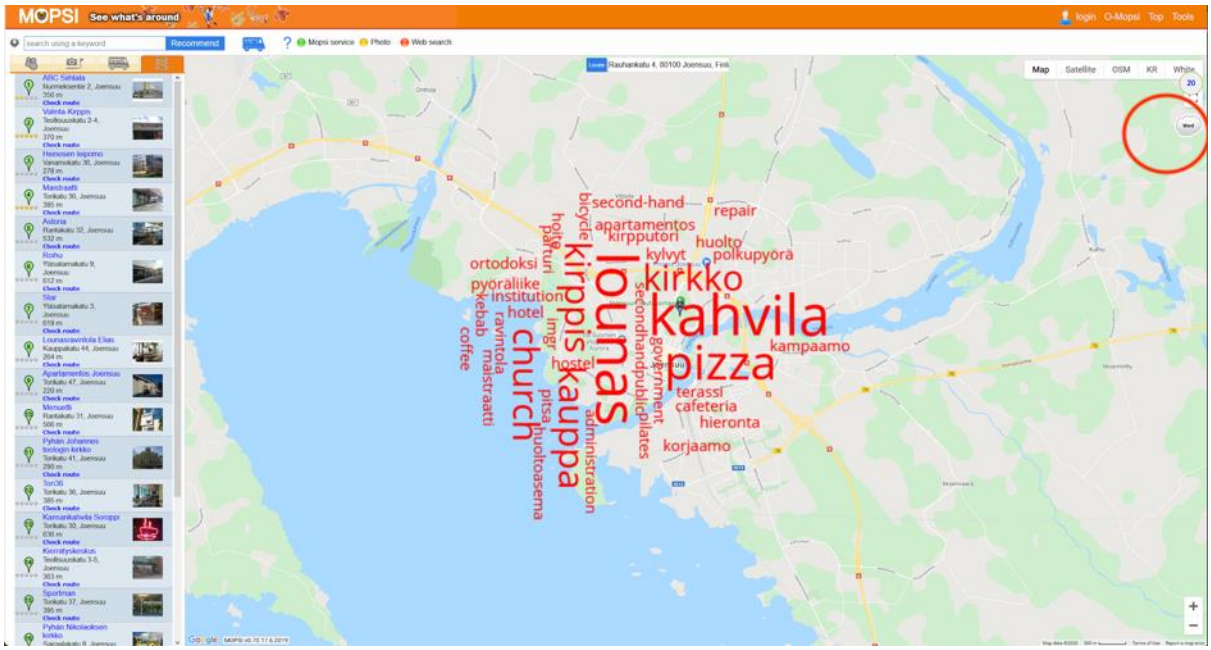
2 Word cloud on Mopsi

In this section, I introduce the whole picture of the word cloud on Mopsi. The main idea of word clouds on Mopsi is to generate word clouds in an image format with given input data and draw it on Google Maps. To implement it, the word cloud on Mopsi contains three main steps: (1) Preparation of input data, (2) Generation of a word cloud as an image within given input data, (3) Drawing word cloud image on Google Maps. Mopsi will prepare data for input for my tool. **Section 2.1** will describe where data comes from and what input data looks like. My main work in this thesis is to generate a word cloud and place word cloud on Google Maps. I will introduce the main algorithm to generate a word cloud and place word cloud on Google Maps in **Section 2.2** and **Section 2.3**.

Figure 3 shows how the word cloud tool works in Mopsi. **Figure 3 (a)** lists many geotagged photographs on the left side and it also shows same geotagged photographs on Google Maps. One cannot see clearly what those photographs refer to. So, I will generate word clouds to represent the photographs. After clicking the word cloud icon which is located at the top right corner, a word cloud will be generated from geotagged photographs that are listed on the left side and the word cloud will be placed on Google Maps. **Figure 3 (b)** shows how it looks. The word cloud gives us a better summary of those photographs. In **Figure 3 (b)**, we can see that many of the photographs refer to “lounas” and “kahvila”.



(a) Before click word cloud icon, the photographs are shown on Google Maps



(b) After click word cloud icon, the word cloud is shown on Google Maps instead

Figure 3: How to generate a word cloud with the tool in Mopsi

2.1 Preparation of input data

In my thesis, input data is extracted from Mopsi. Mopsi will provide each photograph's information and process it for my use. In this section, I am going to introduce what the source data is and what the prepared data is.

Geotagged photographs and trajectories are two types of Mopsi data. There were more than 35,000 geotagged photographs generated by 2,400 registered users in 2017, according to [13]. In Mopsi each geotagged photograph contains keyword property, latitude property, and longitude property. Keyword property of geotagged photograph is a text describing the geotagged photograph. Latitude property and longitude property form the location in the geographic coordinate system, to show where the geotagged photograph was taken. Mopsi will extract keywords, latitude, and longitude from geotagged photographs. **Table 1** shows the data structure of keyword property, latitude property, and longitude property.

Table 1. Source data properties

Column	**Type**	**Description**	**Example**
Keyword	String	Text string of geotagged photograph	kahvila, ravintola
Longitude	Number	Longitude value of geotagged photograph	23.184691 (23° 11' 4.8876")
Latitude	Number	Latitude value of geotagged photograph	62.926880 (62° 55' 36.7674")

Table 2. Properties of input data that Mopsi prepared

Column	**Type**	**Description**	**Example**
Word	String	Unique text string	kahvila
Frequency	Number	How many times the keyword appears in the source date	5
Longitude	Number	Longitude value of geotagged photograph	23.184691 (23° 11' 4.8876")
Latitude	Number	Latitude value of geotagged photograph	62.926880 (62° 55' 36.7674")

The input data that Mopsi provides is an object array. Each object contains word property, frequency property, latitude property, and longitude property. The keywords of source data are a series of text strings. The regular expressions, sequences of characters that define a pattern and describe a certain amount of text, will create a list of unique keywords and present their frequency in the source date[8].

Table 2 shows the input data structure that Mopsi prepared. Word property is the content of what the word cloud will draw. Frequency property will determine word font size in the word cloud. I will use latitude and longitude properties to calculate where the word cloud is placed on Google Maps.

2.2 Generation of word cloud

In this section, I will introduce how to generate the word cloud as an image with given input data. The main idea is to draw all words into a two-dimensional space and save it as an image. In this thesis, I will draw all words on an HTML canvas and save it in image format.

In the thesis, I will use *frequency-based word* clouds [14]. It means if a word has a higher frequency value than other words, the word has largest font size in the word cloud. **Section 3** will introduce how to calculate font size for each word with a given word's frequency.

The amount of geotagged photographs can be different in different regions and the keywords of each photograph can be modified. So, the number of words might be different in different regions. To show all words in the word cloud, I will set a big two-dimensional space and draw all words near the center of the two-dimensional space. After all words are drawn, the program will crop the part with words and save it as an image. In this thesis, two-dimensional space is a square where height and width are the product of the highest length of word among the words multiplied by twice the number of words.

There are two main steps in generating word clouds with given data. (1) Calculate each word weight and font size. (2) Find an optimized position where each word can be drawn in the two-dimensional space. **Generate Word cloud** shows the algorithm to generate a word cloud. Step 1 in **Generate Word cloud** is to calculate font size for each word, it shows how

[8] http://www.regular-expressions.info/print.html

big each word will be drawn. **Section 3** will introduce how to convert word frequency to the word font size. Steps 2 to 22 illustrate how to find positions for drawing words. In step 7, v and ω are the parameters of Archimedean Spiral which I am going to introduce in more detail in Section 5.

Generate Word cloud: generate word cloud
Input: - $\boldsymbol{w} = [w_1 \dots w_i]$: the ordered words object array
Output: word cloud image
Algorithm:

1: Calculate weight and font size for each word w
2: Calculate highest length of word among $\boldsymbol{w}$ **and assign to** $length_W$
3: Calculate number of w_i and assign to $number_w$
4: Assign a square that side length is $length_W * number_w * 2$ to $\boldsymbol{P}$
5: Initialize a square where length of each side equals $length_W$ and mark as $\boldsymbol{square}$
6: Initialize a random position in $\boldsymbol{square}$ for each word w
7: Set v, ω with fixed value
8: Draw first word at central position in $\boldsymbol{P}$, add to $\boldsymbol{W}_{placed}$
9: Remove first word form $\boldsymbol{W}$
10: Assign position of w to $[w_{positionX}, w_{positionY}]$
11: For each w in $\boldsymbol{W}$:
12: Assign 1 to $step$
13: If **Word Overlap Placed Words** $(w, \boldsymbol{W}_{placed}, [w_{positionX}, w_{positionY}])$ is $true$
14: Add w to $\boldsymbol{w}_{placed}$
15: Draw w at $w_{position}$ in $\boldsymbol{P}$
16: Else
17: Increase $step$ by 1
18: Assign $v * step * cos(\omega * step)$ to $w_{positionX}$
19: Assign $v * step * sin(\omega * step)$ to $w_{positionY}$
20: go back to step 13
21: End If
22: End For
23: Return $\boldsymbol{P}$ as image

Word Overlap Placed Words: check whether current word overlaps with any placed words when rotating current word 0 degree and 90 degree

Input: - w: the current word

- $\boldsymbol{W}_{placed}$: the placed word object array
- $[w_{positionX}, w_{positionY}]$: the position of the current word in $\boldsymbol{P}$

Output: $true$ or $false$

Algorithm:

1: Assign $true$ to $notOverlapped$
2: For each word $\boldsymbol{w}_{placed}$ in $\boldsymbol{W}_{placed}$:
3: If $\boldsymbol{w}$ overlap $\boldsymbol{w}_{placed}$
4: Assign $false$ to $notOverlapped$
5: Break
6: Else
7: Assign $true$ to $notOverlapped$
8: End If
9: End For
10: If $notOverlapped$ is $true$
11: Return $true$
12: Else
13: Rotate $\boldsymbol{w}$ 90 degree
14: For each word $\boldsymbol{w}_{placed}$ in $\boldsymbol{W}_{placed}$:
15: If $\boldsymbol{w}$ overlap $\boldsymbol{w}_{placed}$
16: Assign $false$ to $notOverlapped$
17: Break
18: Else
19: Assign $true$ to $notOverlapped$
20: End If
21: End For
22: If $notOverlapped$ is $true$
23: Mark as rotate 90 degree
24: Return $true$
25: Else
26: Return $false$
27: End If
28: End If

2.3 Place word cloud on Google Maps

In this section, I will introduce how to place the word cloud image on Google Maps. Google Maps will provide an Application Programming Interface (API) to place images on Google Maps as an overlay. I will show an algorithm to calculate the coordinate point at the geographic coordinate system on Google Maps to place the word cloud image.

Google Maps is a web mapping service that was developed by Google [15]. Google Maps offers real-time traffic conditions service, route planning service, satellite imagery service, and so on. Google Maps also provides API for the maps for developers to customize maps with their own content. API is a computing interface that defines how software interacts. According to Google Maps documentation, there are at least Maps JavaScript API, Maps Static API and Maps Embed API for developers to use. In this thesis, I am going to use Maps JavaScript API that Google Maps provides to show word cloud images on Google Maps. Maps JavaScript API has four basic map types:

- Roadmap: this is default map type to display default road map.
- Satellite map: this map will display Google Earth satellite images.
- Hybrid map: normal road map and satellite views are displayed on map.
- Terrain map: displays a physical map based on terrain information.

JavaScript API has three custom map types:

- Standard tile sets map: Standard tile sets consist of all tiles which constitute full cartographic maps.
- Image tile overlay map: this type map usually displays images on the top of the existing map.
- Non-image map: this map type allows developers to manipulate the display of map information at the most fundamental level.

Google Maps overlays[9] are the objects on the Google Maps that are tied to longitude and latitude coordinates, it allows you to draw lines, areas, points, etc. There are many kinds of overlays, such as markers, info windows, shapes, images, and so on. TNTgis is advanced software for geospatial analysis. According to the release of TNTgis[10] in 2012, Google Maps consists of many pieces of tiles, which is called tileset structure. Google Maps overlay will be

[9] https://developers.google.com/maps/documentation/javascript/customoverlays
[10] https://www.microimages.com/documentation/TechGuides/78googleMapsStruc.pdf

set on top of existing map as a piece of tile. In this thesis, I will use image tile overlay map type. I will display an image on existing map at a specific location. In this thesis, each word contains geographic coordinate points value. I use **functions (1)** and **(2)** to calculate where the word cloud image will be placed on the map. In **function (1)**, $latitude_i$ is latitude value of i^{th} word while in **function (2)**, $longitude_i$ is longitude value of i^{th} word. In both **function (1)** and **function (2)** n is the number of words. Average $latitude$ value and $longitude$ value are $latitude_{average}$ and $longitude_{average}$. The word cloud image will be placed at $latitude_{average}$ and $longitude_{average}$ on Google Maps.

$$latitude_{average} = \frac{\sum_{i=1}^{n} latitude_i}{n} \quad (1)$$

$$longitude_{average} = \frac{\sum_{i=1}^{n} longitude_i}{n} \quad (2)$$

In this thesis, I use Maps JavaScript API to place a word cloud image on Google Maps. Maps JavaScript API provides an OverlayOverview class for creating my own custom overlays (see **Appendix 1**). The algorithm **Display Word Cloud** gives a procedure on how to place the word cloud image on Google Maps. $LatLng$ is an array containing a geographic coordinate point where each geographic coordinate point contains latitude value and longitude value. The word cloud image is $\boldsymbol{image}$, which is generated with given input data, which I introduced in **Section 2.2**. The Google Maps object in the web application is $\boldsymbol{map}$. The Google Maps object is initialized by importing Maps JavaScript API and gets a map document object model (DOM) element.

Display Word Cloud: display word cloud on Google Maps
Input: -$\boldsymbol{LatLng}$ = $[latlng_1 \ldots latlng_i]$: The geographic coordinate points array
-$\boldsymbol{image}$: word cloud image
- $\boldsymbol{map}$: Google Maps object
Output: map with image as overlay
Algorithm:
1: Assign all latitudes in $\boldsymbol{LatLng}$ to $latitudes$
2: Assign all longitudes in $\boldsymbol{LatLng}$ to $longitudes$
3: Calculate summation of $latitudes$ and assign to $sum_{latitudes}$
4: Calculate summation of $latitudes$ and assign to $sum_{longitudes}$
5: Calculate length of $\boldsymbol{LatLng}$ and assign to $length_{latlng}$
6: Divide $sum_{latitudes}$ by $length_{latlng}$ and assign to $average_{latitudes}$
7: Divide $sum_{longitudes}$ by $length_{latlng}$ and assign to $average_{longitudes}$
8: Initialize $\boldsymbol{map}$
9: Place $\boldsymbol{image}$ at point $[average_{latitudes}, average_{longitudes}]$ on $\boldsymbol{map}$ as overlay

3 Word font size

In this section, I will discuss how to calculate word weight and font size for each word. In **Section 2.2**, I introduced two main steps to generate word clouds with given data. (1) Calculate word weight and word font size for each word, (2) Find the best position where each word can be drawn in a two-dimensional space. In this section, two functions are discussed to calculate font size for given input words.

In this thesis, I choose the frequency-based word cloud method to generate my word cloud. In the frequency-based word cloud, word weight equals the value of the word's frequency. The basic idea of the frequency-based word cloud is that font size will represent how many times the word appears in the source data. In this thesis, I use the Canvas API to draw words in two-dimensional space through an HTML Canvas element. I use pixels as the unit of font size in the HTML Canvas. There is a problem when I show the frequency-based word cloud on the web. When word frequency is very low, the word font size on HTML Canvas will be too small, and the word cannot be recognized by human eye. When word frequency is very high, then the web page might not show all the words properly as some words will be outside the physical device screen (such as mobile phone, laptop, and so on). I want my word cloud: (1) Show all words on the web page. (2) Have all words easily recognizable by human eye. (3) Increase word font size with word frequency.

Based on my considerations, I have two candidate monotonic functions. The first function is a monotonic increasing linear function, which I mark with $lif(w_{weight})$. The Second function is a monotonic increasing logarithm function, which I mark with $lof(w_{weight})$. The parameter w_{weight} in both $lif(w_{weight})$ and $lof(w_{weight})$ is the weight of the word. I want word font size change in a range, so I predefined the maximum font size of the word, $MAXF$, and the minimum word font size,$MINF$. If all the words have the same weight, I will assign $MINF$ to each word as font size. In my word clouds, all the words have the same font size when all the word weights are the same because the relative font size of the word represents different word weight. How large word font size will be assigned when all the word weights are the same depends highly on how one designs his word cloud application. Based on my, I will check if all word weights are the same or not by comparing $Maxw_{weight}$ and $Minw_{weight}$. The parameter $Maxw_{weight}$ is the maximum word weight in a weighted word list while $Minw_{weight}$ is the minimum word weight in the same weighted word list.

Figure 4 shows an example of how the word font size is changing with different words weight. In this example, I use cities in Finland and their populations as input data. The city name is the word while the population is the word weight. **Appendix 2** shows the input data I used in this example. In the word cloud in Mopsi, I want all words to be seen clearly on the web page and all words to be visible on the screen. So, I predefined $MAXF$ and $MINF$. The value of $MAXF$ and $MINF$ is determined by application itself. It depends on how one wants the application to perform, if my application prioritizes just a few of the top biggest words, then $MINF$ can be zero in the case of $Maxw_{weight}$ and $Minw_{weight}$ are not equal. In this thesis, I set $MAXF$ is 55 and $MINF$ is 15 because:

(1) I want all words in the word cloud in Mopsi show on the screen.

(2) I want all words to be visible and to be seen clearly by human eye.

(3) Personal experience.

$Maxw_{weight}$ is 648,650 in this example. It is not predefined but is determined by my input data.

In **Figure 4,** when using $lof(w_{weight})$ word font size increases dramatically with increasing word weight at the beginning. After word weight is 100,000, word font size increases slowly. In my example, word font size does not change much after word weight reaches 400,000. In this example, the minimum word weight is 20,410 and corresponding font size is 45 after rounding. The maximum word weight is 648,650 and corresponding font size is 55. In **Figure 4,** when using $lif(w_{weight})$ word font size increases at a constant rate with increasing word weight. In this example, the minimum word weight is 20,410 and corresponding font size is 16 after rounding. The maximum word weight is 648,650 and corresponding font size is 55.

$$lof(w_{weight}) = \begin{cases} \frac{log_{10} w_{weight}}{log_{10} Maxw_{weight}} \times (MAXF - MINF) + MINF, & w_{weight} \geq 1 \\ & Maxw_{weight} \neq Minw_{weight} \\ MINF, & w_{weight} \geq 1 \\ & Maxw_{weight} = Minw_{weight} \end{cases} \quad (1)$$

$$lif(w_{weight}) = \begin{cases} w_{weight} \times \frac{MAXF - MINF}{Maxw_{weight}} + MINF & w_{weight} \geq 1 \\ & Maxw_{weight} \neq Minw_{weight} \\ MINF, & w_{weight} \geq 1 \\ & Maxw_{weight} = Minw_{weight} \end{cases} \quad (2)$$

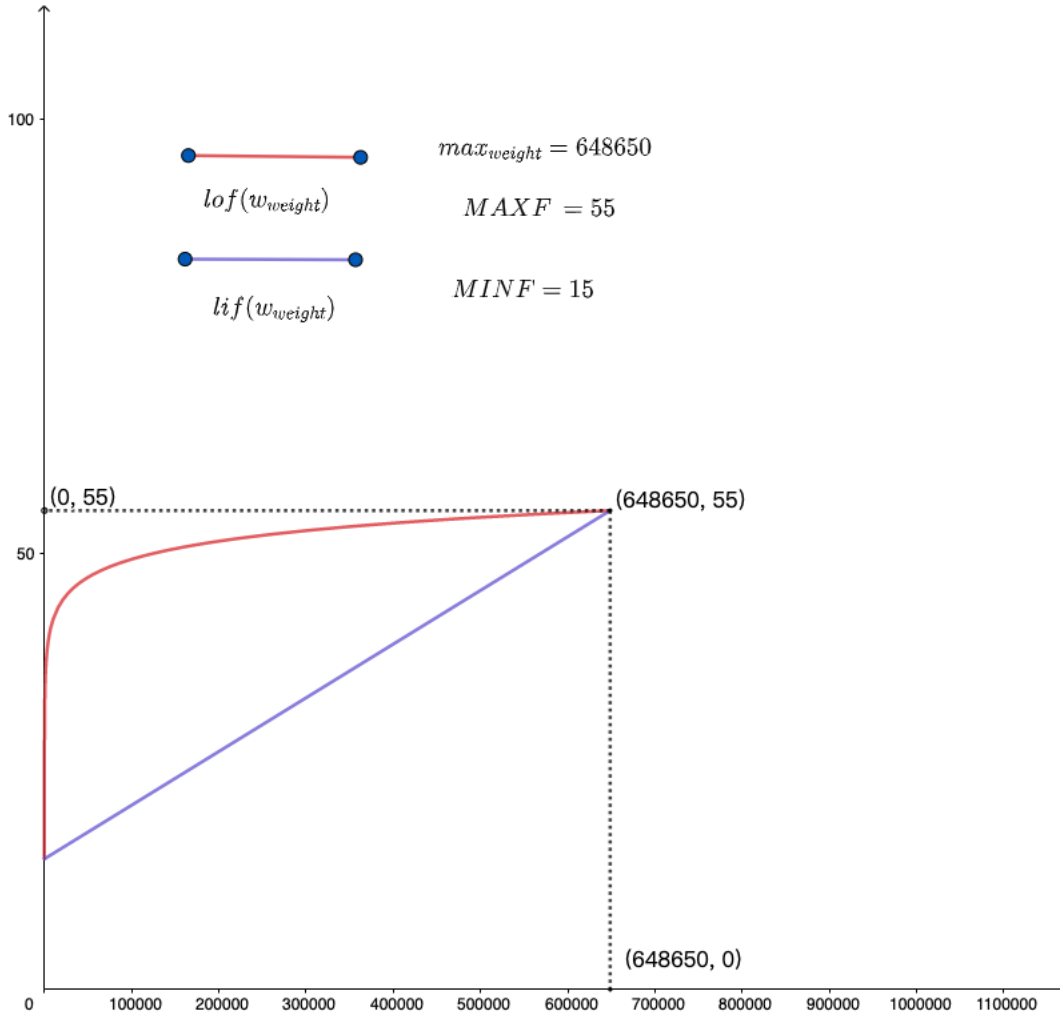


Figure 4: Graph of function $lif(w_{weight})$ and function $lof(w_{weight})$ when $MAXF$ is 55, $MINF$ is 15 and $Maxw_{weight}$ equals 648,650. The red line represents function $lof(w_{weight})$ while the purple line represents $lif(w_{weight})$.

(a) $lif(w_{weight})$ is applied. (b) $lof(w_{weight})$ is applied.

Figure 5: Word clouds of cities in Finland with their populations. The two word clouds are generated with the same input text data and the same word cloud generating algorithm, except the font size function. See input text data in **Appendix 2**.

Figure 5 shows two word clouds that are generated with the same algorithm and the same input text data, except the font size function. The left word cloud in **Figure 5** use $lif(w_{weight})$ to calculate word font size with given word weight while $lof(w_{weight})$ is applied to the right word cloud in **Figure 5**. My input data is the cities of Finland with their populations. The red words in the word cloud are some city names in Finland and the font size represents the cities' population. In the left word cloud in **Figure 5**, we can easily see that Helsinki has the highest population. Espoo, Vantaa, and Tampere have the second highest and roughly similar populations. From the data table that is shown in **Appendix 2**, it can be seen that the population of Helsinki is 648,650 and it is the highest population among the cities in that list. The populations of Espoo, Vantaa, and Tampere are 281,886; 226,160, and 234,441 respectively. The population of these three cities are not the same, but quite similar. This matches my requirements in my thesis. One cannot recognize quickly which city has the highest population from the right word cloud in **Figure 5**. The red line in **Figure 4**

shows that the word font size does not increase very much when word weight is over 20,000. The city that has the smallest population is Hamina, which has population of 20,410 according to **Appendix 2**. From the right word cloud in **Figure 5**, if we check carefully, we can see that the font size of the word Helsinki is a little bit larger than the font size of the word Hamina. But it is not easy for human eye to recognized which one is bigger: the word Helsinki or the word Hamina. Based on my consideration, I will use monotonic increasing linear function $lif(w_{weight})$ in this thesis, and I will predefine $MAXF$ as 55 and $MINF$ as 15.

4 Collision detection

In **Section 2.2** I introduced the algorithm **Generate word cloud** which is to generate word clouds. After that I presented the algorithm to calculate the font size for each word within given wordlist. I will then place all the words in a two-dimensional space to find the best position for each word. This step I call word placement, I will introduce it more thoroughly in **Section 5**. The best position for a word in this thesis means:

(1) The word has a location in a two-dimensional space. I will draw words in HTML Canvas and the location will be represented by a coordinate point. For example, (10, 10). The position of the word has to be inside the HTML Canvas, otherwise I am not able to draw words on the HTML Canvas.

(2) The words do not collide with each other. If the word is placed with its font size at a position in the HTML Canvas, the word cannot collide with any other words which have already found a position in the HTML Canvas.

When two or more objects intersect with each other in a two-dimensional space or three-dimensional space it is called collision. In digital imaging, pixels are the smallest controllable elements of an image presented on screen. All objects on screen are made of group of pixels. In this thesis, if a word shares one or more pixels with another word, then I determine the word collides with another word. In **Figure 6** square represents pixel. Letters H, D, H and I are formed by groups of pixels. Letters H and D share one pixel, so I determine that letter H collides with letter D. Another letter H does not collide with letter I, because they do not share pixels.

Collision detection is a fundamental problem of detecting intersections of two or more objects in the fields of computer graphics, surgical simulations and robotics. Normally collision detection algorithms focus on two-dimensional collision detection and three-dimensional collision detection. The collision detection is divided into broad-phase collision detection and narrow-phase collision detection by Hubbard [16]. The broad-phase collision detection will list all pairs of objects which have possibly collided while the narrow-phase collision detection will determine if objects actually have collided or not and report only those which have actually collided. There are many algorithms to detect whether two or more objects collide. These algorithms are classified as [17]:

- Feature-based algorithm: The featured based algorithm works on geometric primitives of the objects directly. For example, V-Clip [18] and SWIFT [19].
- Simplex-based algorithm: The simplex-based algorithm works on convex hulls. It calculates the Euclidean distance of two convex hull sets to determine if two objects collide. The Gilbert–Johnson–Keerthi distance algorithm [20] is an example of this.
- Image-space based algorithm: The image space-based algorithm is to convert a three-dimensional geometry into two-dimensional image and manipulate image pixels. The Cider [21] is one of well-known examples.
- Volume-based algorithm: The Volume-based algorithm is the same idea as the Image-space based algorithm, but it uses different methods to compute images. The Volume-based algorithm will construct a geometry mesh by its vertices to represent object image. Then compare images' geometry to determine whether objects collide or not. The Gundelman [22] introduced an example of the volume-based algorithm.
- Bounding Volume Hierarchies: The Bounding Volume Hierarchies is the data spatial structure to recursively divide space or object itself. All objects are wrapped in bounding volume as a leaf node in a tree data structure.

In this section, I will discuss how one checks collision detection in word clouds on Mopsi. In **Section 4.1** I will introduce my naïve collision detection method, Axis-Aligned Bounding Box(AABB). AABB is a fast way to check if a word collides with another word or not, but it has low accuracy. In order to improve accuracy, in **Section 4.2** I will introduce quadtree, a hierarchical spatial tree data structure.

(a) **(b)**

Figure 6: (a) Letter "H" and letter "D" collide. (b) Letter "H" and letter "I" do not collide.

4.1 AABB

In this section I will introduce my naïve collision detection method: Axis-Aligned Bounding Box (AABB), one kind of bounding box, and how to do collision tests by comparing two bounding boxes of the two words.

A *Bounding Volume* (BV) is a common method to simplify object representation by using the composition of geometrical shapes that enclose the object [23]. There are four common BV's according to previous research [24,25] which are shown in **Figure 7**: Spheres [26,27], *Axis-Aligned Bounding Box* (AABB) [28,29,30], k-direction Discrete oriented polytopes (k-Dops) [16], Oriented Bounding Box (OBB) [31,32].

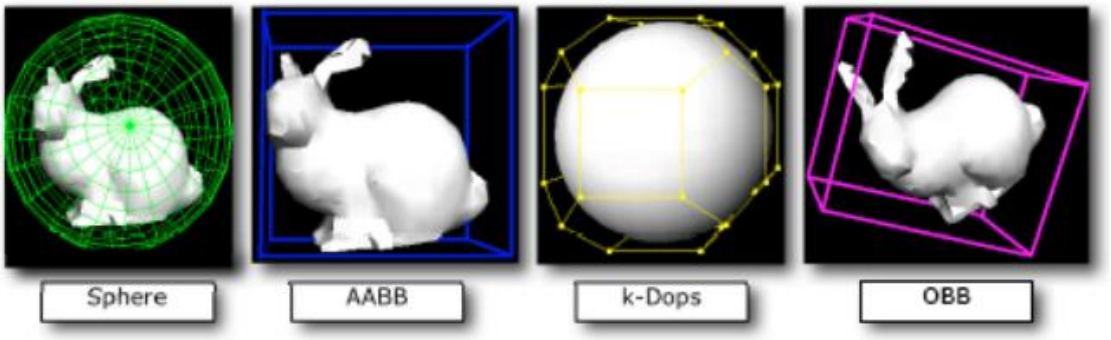
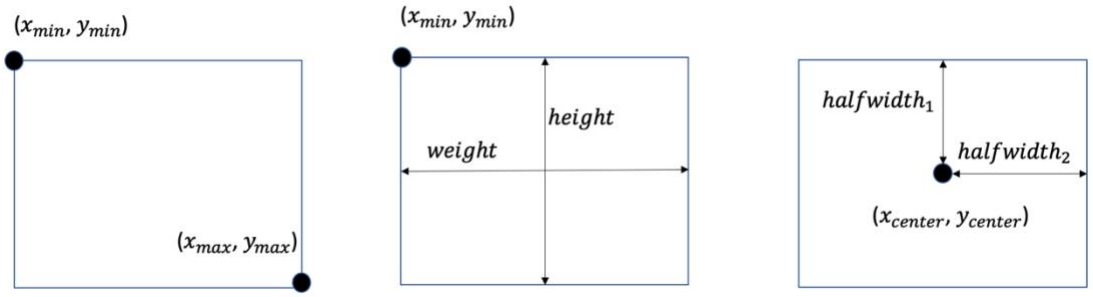


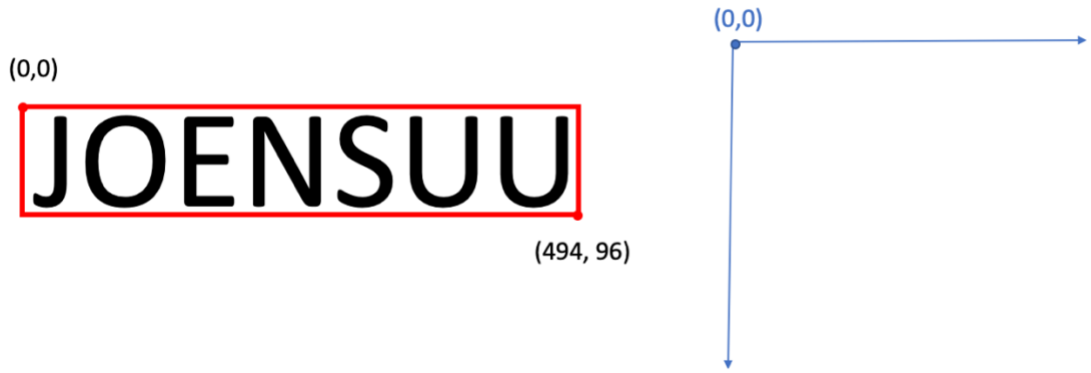
Figure 7: Four common BV's according to previous research [24,25].

AABB is an enclosed axis aligned rectangle that wraps a polyhedron. AABB is a common and efficient method to detect if two or more objects overlap or not. The reasons why I choose AABB to do collision detection tests in this thesis are: (1) AABB is easy to construct. (2) It is straightforward to do collision tests with AABB. When I need to check if one word collides another word or not, I just need to do a collision test between two AABB's of candidate words. **Figure 8** shows three common AABB representations. They are: (a) min-max, (b) min-width-height and (c) center-halfwidths. In this thesis, I will use (a) min-max to present AABB. **Figure 9** shows the AABB of the word JOENSUU in the HTML Canvas coordinate system. In **Figure 9 (a)** the red rectangle is the AABB of word JOENSUU. The values (0, 0) and (494, 96) are the representation of AABB of the word JOENSUU in the coordinate system. **Figure 9 (b)** illustrates the coordinate system in HTML Canvas. If the value is negative, the content will not be shown on HTML Canvas. So, I use only positive values to construct AABB in my thesis.



(a) min-max (b) min-width-height (c) center-halfwidths

Figure 8: The three common ways to represent AABB: (a) Using minimum and maximum coordinate values along each axis to represent AABB. (b) The AABB is represented by the minimum coordinate value, height and width of the object. (c) Using central coordinate value and halfwidths to present AABB.



(a) AABB of word JOENSUU (b) Coordinate system in HTML canvas

Figure 9: The AABB of word JOENSUU and the coordinate system in HTML Canvas: (a) Drawing word JOENSUU at the origin of HTML Canvas coordinate and constructing the AABB for the word JOENSUU. (b) The coordinate system in HTML Canvas. The origin is at the top left; from top to down and left to right are positive directions.

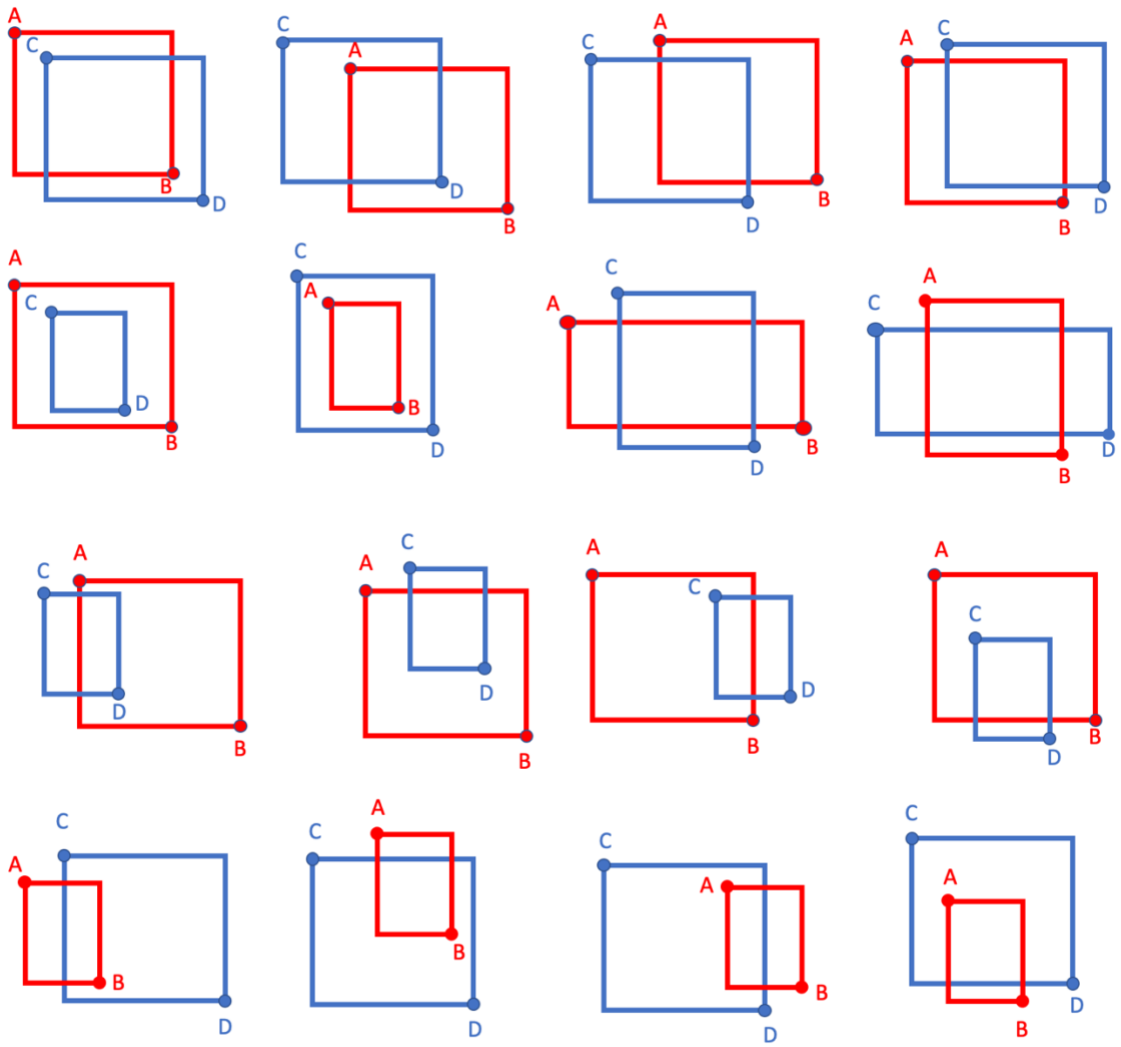


Figure 10: All the situations when two AABB collide. The red rectangle and blue rectangle are AABBs. The point A, B, C and D are the coordinate points.

Figure 10 shows all the situations when two AABB's collide. Based on all these situations, I have the conditions for collision of two AABB's:

- $A_x \leq D_x$
- $B_x \geq C_x$
- $A_y \leq D_y$
- $B_y \geq C_y$

If two AABB's match all the condition above, then the two AABB's collide.

4.2 Quadtree

The AABB is easy to construct and to do collision detection, but why do I still need quardtree? Because when words' AABB's collide, the words themselves might not collide. I cannot list all such cases. **Figure 11** shows three cases where the words themselves do not

collide but words' AABB's collide. In **Figure 11**, the $AABB_{word}$ represents the AABB of the word. The AABB of the word has the same color with the word. In **Figure 11 (a)**, the word JOENSUU collides the word APPLE if compared $AABB_{JOENSUU}$ and $AABB_{APPLE}$, but the red word APPLE does not overlap the black word JOENSUU. The same situation happens in **Figure 11 (b)** and **Figure 11 (c)**. So, I will use *quadtrees* to do further collision tests in my thesis. According to my consideration, my method to do collision detection between two words is to compare AABB's of the two words. If the two AABB's do not collide then the two words do not collide. If the two AABB's collide, I will compare *quadtrees* of the two words to determine whether the two words collide or not. The flowchart of collision detection in my thesis is showing in **Figure 12**.

(a) $AABB_{JOENSUU}$ collides with $AABB_{APPLE}$ (b) $AABB_{ORANGE}$ collides $AABB_{APPLE}$

(c) $AABB_{Check}$ collides $AABB_{Fix}$

Figure 11: The three examples to show AABB's of words collide but the words themselves do not collide. (a) AABB of the word JOENSUU collide AABB of the word APPLE; $AABB_{JOENSUU}$ is the black rectangle while $AABB_{APPLE}$ is the red rectangle. (b) $AABB_{ORANGE}$ is the purple rectangle and $AABB_{APPLE}$ is the blue rectangle. (c) $AABB_{Check}$ is the blue rectangle and $AABB_{Fix}$ is the green rectangle.

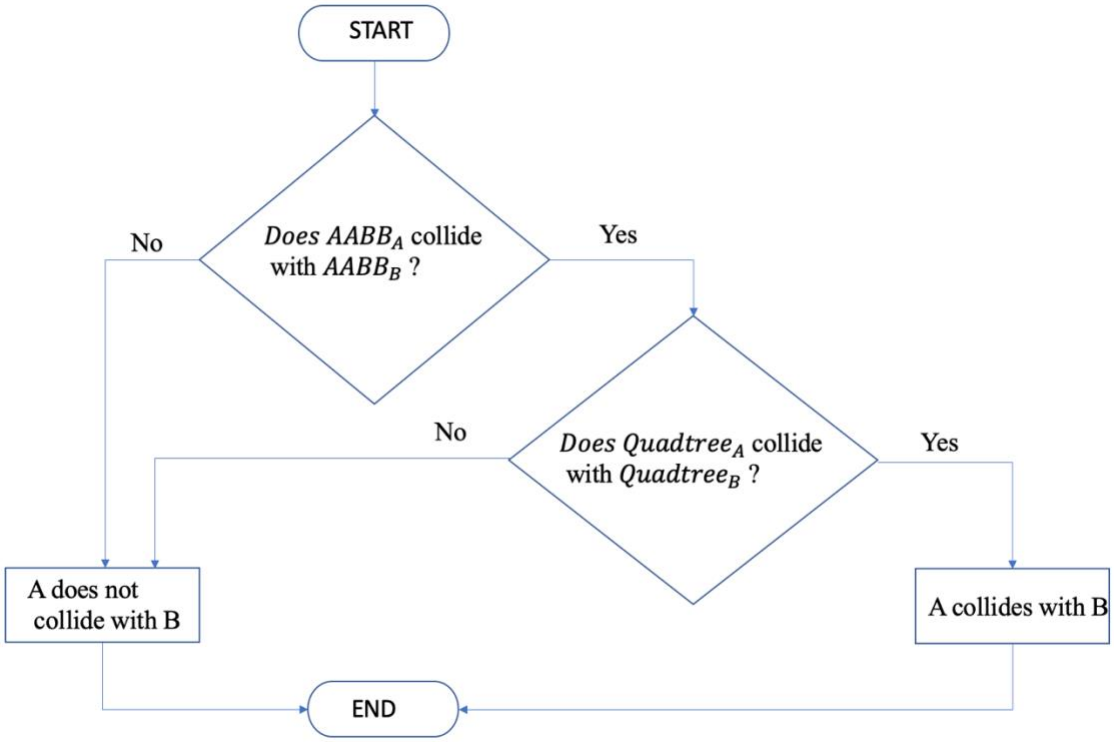


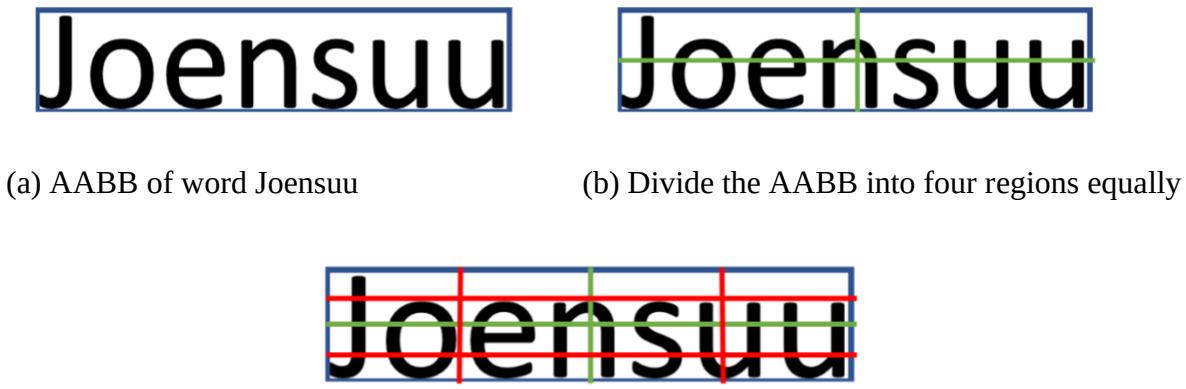
Figure 12: The flowchart of collision detection of two words in my thesis.

Quadtree is a hierarchical spatial tree data structure in which each internal node has four children. The name *quadtree* was given by Raphael and J.L. Bentley in 1974, in their paper titled "Quad tree: A Data Structure for Retrieval on Composite Keys". There are many types of quadtrees and it can be generalized to any dimensional space. But the idea of *quadtree* is always to decompose space recursively [33]. In *quadtree*, each node represents a unit of important spatial information. In my thesis, *quadtree* stores information in two-dimensional space. I will divide AABB of the word recursively into four regions. In my thesis, the important spatial information means pixels of the word. *Quadtree* has already been proven to be a simple and quick data structure for image manipulation [34] [35]. In my thesis, the *quadtree* I will use is called *Region Quadtree*. The *Region Quadtree* represents a partition of a two-dimensional space by decomposing the two-dimensional space into four equal regions. Each node in standard *Region Quadtree* has exactly four children or no children. In the thesis, I will use modified *Region Quadtree* to represent an image or AABB of the word that consists of many pixels, where each pixel value is either 0 or 1. **Figure 13** illustrates how to divide a two-dimensional space during building a *quadtree* and **Figure 14** shows corresponding *quadtree.* All leaf nodes of *quadtree* in this thesis are AABB's. If the AABB does not contain any part of the word, I will mark it as 0, otherwise I mark it as 1, because I am only interested about the AABB's that contains parts of the word. Since decomposition

strategy of quadtree is highly dependent on the application, and based on my consideration above, I have some conditions on building a *quadtree*:

- If all the pixel values in a region are 0, the algorithm does not continue to divide the region and does not add the region to the node as a child.
- If all the pixel values in a region are 1, the algorithm does not continue to divide the region, but adds the region to the node as a child.
- If the size of the region is smaller than $minimumBoxsize$, the algorithm does not continue to divide the region, but adds the region to the node as a child. $minimumBoxsize$ is the predefined threshold in my thesis.

Based on the above conditions to build a *quadtree*, we can see that it is easy to construct a *quadtree*. The **Build Quadtree** illustrates how to construct a *quadtree* for the $AABB$ of the word.



(a) AABB of word Joensuu

(b) Divide the AABB into four regions equally

(c) Divide sub-region into four regions equally

Figure 13: Illustration of dividing two-dimensional space to build a *quadtree.*

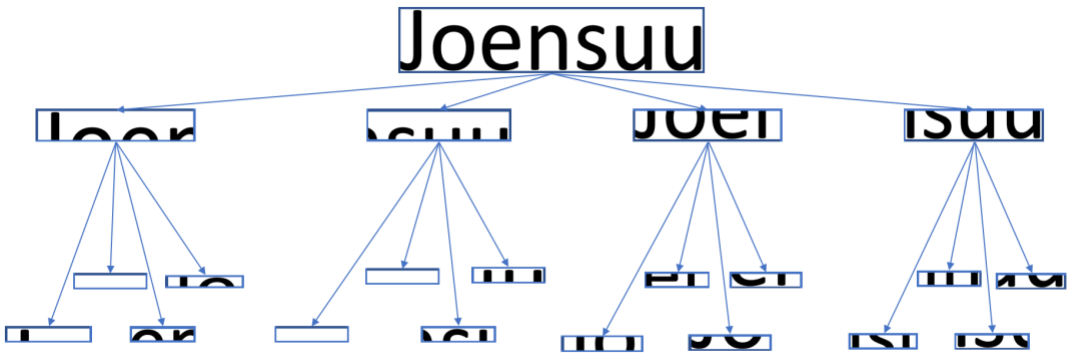


Figure 14: The *quadtree* of the AABB of the word Joensuu in a two-dimensional space.

Build Quadtree: constructing the $Quadtree$ for the $AABB$

Input:

-$AABB$: the $AABB$ of the word

- $minimumBoxsize$: a constant integer

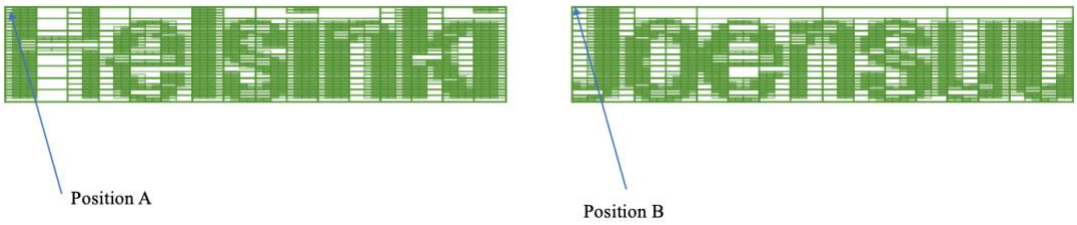
Output: $Quadtree$

Algorithm:

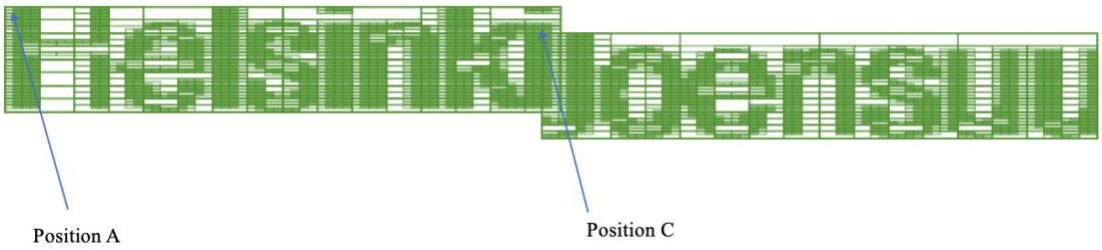
1: Divide current box equally into four *sub-box*es
2: For each *sub-box*:
3: If size of the sub-box is bigger than $minimumBoxsize$
4: If every pixel's value in the *sub-box* is 0
5: Do not store the sub-box and do not continue dividing
6: Else If every pixel's value in the *sub-box* is 1
7: Store the sub-box to the current box as a child
8: Do not continue dividing
9: Else
10: Store the *sub-box* to the current box as a child
11: **Build Quadtree (**sub-box, $minimumBoxsize$**)**
12: End If
13: Else
14: If every pixel's value in the sub-box is 0
15: Do not store the *sub-box* and do not continue dividing
16: Else
17: Store the sub-box to the current box as a child
18: Do not continue dividing
19: End If
20: End If
21: End For

In a two-dimensional space, I will try to place the word at a position in this space. If the word does not collide with any other words already placed there, I will place the word at that position. If the word collides with one or more words already placed there, I will find a new position for it and do the collision test again. **Figure 15** shows different collision situations of the *quadtree* of the word Helsinki and the word Joensuu. In **Figure 15 (a)**, when the word Helsinki is located at Position A and the word Joensuu is located at Position B, the *quadtree* of the word Helsinki does not collide with the *quadtree* of the word Joensuu. In **Figure 15 (b)**, 'i', the last alphabet of the word Helsinki overlaps the AABB of the word Joensuu at Position C, but the two words do not collide. In **Figure 15 (c),** when placing the word Joensuu at Position D, the two *quadtree*s collide. It means the word Helsinki which is located

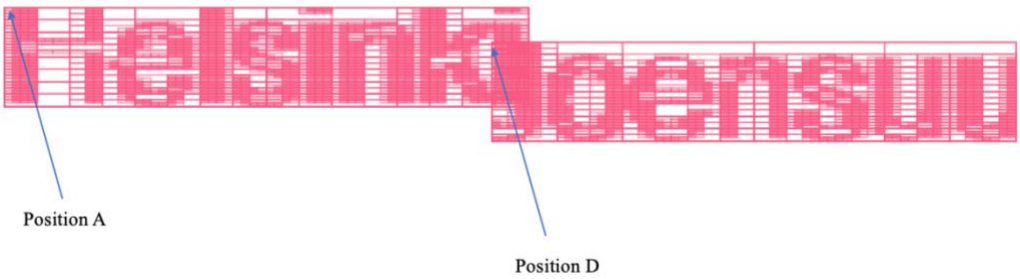
at Position A collides with the word Joensuu which is located at Position D. The algorithm **Quadtrees Collide** shows how to test whether two *quadtree*s collide or not



(a) The word Helsinki which is located at Position A does not collide with the word Joensuu which is located at Position B



(b) The word Helsinki which is located at Position A does not collide with the word Joensuu which is located at Position C



(c) The word Helsinki which is located at Position A does collide with the word Joensuu which is located at Position D

Figure 15: Different quadtree collision situations of the word Helsinki and the word Joensuu at a two-dimensional space. The green color means quadtrees do not collide while the pink color means the quadtrees collide. The Position A, Position B, Position C and Position D mean a position in the current two-dimensional space. (a) The quadtree of the word Helsinki does not collide with the quadtree of the word Joensuu. (b) The quadtree of the word Helsinki does not collide with the quadtree of the word Joensuu. (c) The quadtree of the word Helsinki does collide with the quadtree of the word Joensuu.

Quadtrees Collide: testing whether one quadtree collides with another quadtree
Input:

-$Quadtree_A$: Quadtree of the word A
-$Quadtree_B$: Quadtree of the word B

Output: true or false

Algorithm:

1. If $Quadtree_A$'s root AABB overlaps $Quadtree_B$'s root AABB
2. If $Quadtree_A$ does not have children
3. If $Quadtree_B$ does not have children
4. return true
5. Else for every child in $Quadtree_B$
6. **Quadtrees Collide (**$Quadtree_A$, $Quadtree_B$'s child**)**
7. End If
8. Else for every child in $Quadtree_A$
9. If $Quadtree_B$ does not have children
10. **Quadtrees Collide (**$Quadtree_A$'s child, $Quadtree_B$**)**
11. Else for every child in $Quadtree_B$
12. **Quadtrees Collide (**$Quadtree_A$'s child, $Quadtree_B$'s child**)**
13. End If
14. End If
15. Else
16. Return false
18. End If

4.3 Alternatives

Quadtree can be used to check if a word collides with another word or not. But that is not the only way to do collision detection in my case. My case is special, since I set a fixed size HTML Canvas and work on pixels directly, so the *look up strategy* becomes possible to do collision detection. In this section, I am going to introduce one alternative solution for my case.

4.3.1 Look up strategy

The main idea of the look up strategy is to assign a unique color to the object, then look for the object through its color. For example, I draw words Joensuu, Kuopio and Helsinki on a fixed size HTML Canvas, as shown in **Figure 16**. Each word is drawn in a unique color, represented as RGB (Red, Green, Blue) code. The words Joensuu, Kuopio and Helsinki have

RGB colors (0, 0, 255), (255, 0, 0) and (0, 128, 0) respectively. When I click any location on the canvas, I will read the color at that location. I can determine which word I clicked by referring to the color code.

Figure 16: Words with different colors on HTML Canvas

How do I do collision detection by using look up strategy? First, I create a fixed size HTML canvas with transparent background in RGBA (Red, Green, Blue, Alpha), background color is represented as RGBA, (0, 0, 0, 0). The word color on canvas is black, represented as RGBA, (0, 0, 0, 1). Then I need to get the bounding box and location of each pixel of the word. When I want to know if a word collides with already placed words or not, I just need to check candidate word pixel location on canvas: Is it transparent or not? If the corresponding location on canvas is transparent, then the candidate word does not collide with any placed words, otherwise, it collides with one or more placed words.

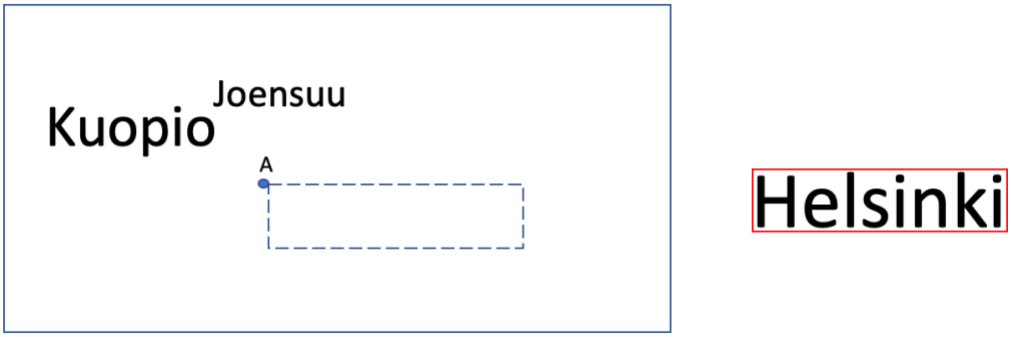


Figure 17: A look up strategy to check whether the word Helsinki collides with the words Kuopio and Joensuu or not.

For example, the words Kuopio and Joensuu are drawn on transparent HTML Canvas, as shown in **Figure 17**. The red rectangular surrounding word Helsinki is a bounding box. The dotted line rectangle is the corresponding location of the word Helsinki bounding box in HTML Canvas. When I check whether the word Helsinki collides with placed words or not, I get the pixel location of word Helsinki bounding box and pixel color of corresponding location in dotted line rectangle. If all pixels of corresponding location in dotted line rectangular are transparent, then word Helsinki does not collide with placed words, otherwise, it collides.

5 Word movement

In the word cloud on Mopsi, there are words with different weights to be placed in two-dimensional space. Which word should be placed first and which word should be second? In my thesis, my solution is simple. I arrange the sequence by word weight, the word with the highest weight will always be placed first. To start with the first word placement, where should I find the place for the first word? After the first word is allocated, where should I place the second word? For example, I have the word Helsinki and the word Joensuu to generate a word cloud. The word Helsinki is already placed, I need to place the word Joensuu at the next step, but which position in this two-dimensional space should be the first position for the word Joensuu to be tried? In this section, I will discuss how to initialize a word's position and how to find a new position for a word that collides with placed words.

In **Figure 18**, the word Helsinki is already placed at $position_a$, when trying to place the word Joensuu at $poaition_b$, the word Joensuu collides with the word Helsinki. How can I find a new position for the word Joensuu to try again? Should I try to place the word Joensuu to a little bit the left, or would the right side has more chance to find a position that does not collide with the word Helsinki. If the word Joensuu collides with the word Helsinki at the new position, how can I find the next position for the word Joensuu? In my thesis, this problem is called *word movement*.

In **Section 5.1**, I will introduce how I initialize the position for each word and explain the reason why I chose this way to initialize the position. In **Section 5.2**, I will introduce *the Archimedean Spiral* which is a method to solve the *word movement* problem in my thesis. Also, I explain why I use *the Archimedean Spiral* to find new possible positions for words instead of just simply moving the word to the left or to the right.

(a) The word Helsinki is placed at $position_a$

(b) The word Joensuu collides with the word Helsinki when the word Joensuu is at $position_b$

Figure 18: Drawing words Helsinki and Joensuu in two-dimensional space. (a) Draw the word Helsinki at $position_a$, (b) Draw the word Joensuu at $position_b$ after the word Helsinki has been placed at $position_a$.

5.1 Word position initialization

The *frequency-based word cloud* has been introduced previously in **Section 1.2** and it is applied to a word cloud on Mopsi. The font size of the word represents the appearing frequency of the word. In my thesis, the word which has the highest appearing frequency is most important in the application. So, the word with the largest font size is the most important word for word cloud on Mopsi. Based on this principle, I make all the words revolve around the word with the largest font size, so the word cloud is tight and easy to

understand. When initializing the words, I always draw the largest font size word first and then place other words around it one by one.

Before I initialize the position of each word in two-dimensional space, I need to initialize a two-dimensional space where all words can be placed in. My two-dimensional space is a square, where height and width are the product of the highest length of word among the words multiplied by twice the number of words, marked as P. Then I initialize a square, where height and width are the highest length of word among the words, marked as p. I place p at center of P. Finally, I initialize a position for each word inside p randomly. After word position is initialized, words may collide with each other and words may be out of p, that is acceptable. Because the main idea of initializing word position in p is keeping all words packed tightly and next section I will introduce a method which can find an ideal position for each word in P. In **Figure 19**, my input words are Helsinki, Espoo, Joensuu, and Kuopio. Corresponding font sizes are 50 pixels, 30 pixels, 14 pixels, and 14 pixels. The highest length of words is 175 pixels and I have four words, so I initialize words positions inside a 175 pixels wide and 175 pixels high square. The red dots are the initial position of the words in p.

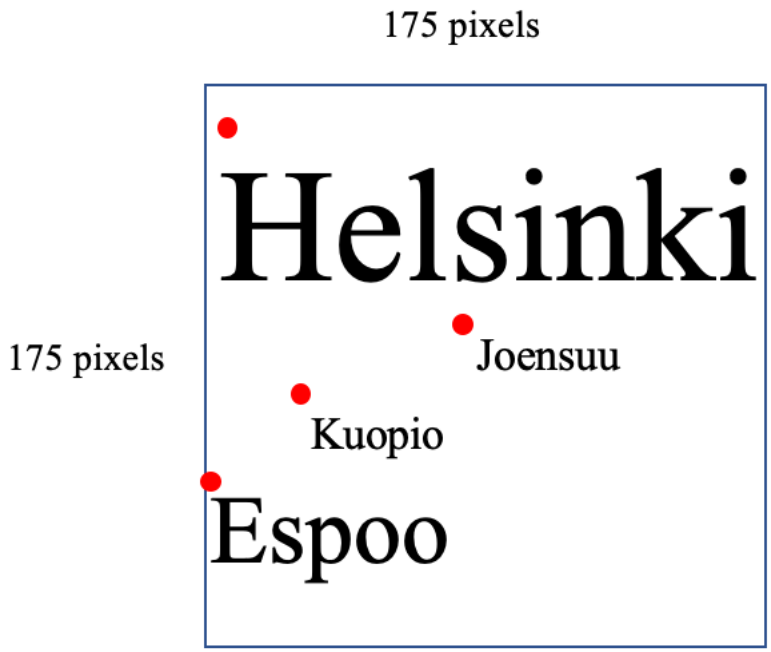


Figure 19: All the words' initial positions will be within this square. The width is the product of the highest length of word among the words. The red dots are the initial position of the word in the square.

The reason for giving each word a random initial position is that it can reduce collision times of words except for when the worst case occurs. I use Archimedean Spiral (I am going to introduce Archimedean Spiral in **Section 5.2**) to be my word movement solution and constant parameters are assigned to it. It means if the words have the same initial position, then the words will have the same movement locus in the same starting location. For example, **Figure 20 (a)** shows the procedure for the word Joensuu when the word Joensuu and the word Helsinki have the same initial position. The black dot is the initial position of the word Helsinki and the word Joensuu. When placing the word Joensuu at the black dot, it collides with the word Helsinki, then I place the word Joensuu at the yellow dot and it still collides with the word Helsinki. However, moving Joensuu to the red dot there is no collision that occurred. Eventually, the word Joensuu does three times collision tests with the word Helsinki. **Figure 20 (b)** shows the word Joensuu looking for a position to be placed when the word Helsinki is already placed, these two words have different initial positions. The black dot is the initial position of the word Helsinki while the red dot is the initial position for Joensuu. The word Joensuu needs to do collision test just once with the word Helsinki. So, a random initial position can reduce collision times in a good case.

(a) The word Helsinki and the word Joensuu share same initial position

(b) The word Helsinki and the word Joensuu have different initial positions

Figure 20: Finding a position for the word Joensuu while the word Helsinki has already been placed. (a) Helsinki and Joensuu share the same initial position; Joensuu needs to do three times collision tests with Helsinki. (b) Helsinki and Joensuu have different initial positions; these two words need only one collision test.

Another reason why assign each word a random initial position within a square is that I want the word cloud to organize tight. The shape does not matter, I can randomly generate initial positions within a rectangular, circle, or other shapes. Choosing the proper square size is also important. Otherwise, the word cloud does not look tight. In **Figure 21**, on the left side is a word cloud where all the words are black. This word cloud looks tight because the initial position of each word is within a good size square. The word cloud where every word is red in **Figure 21** looks not so tight because the initial position of each word is within a too large space.

Figure 21: On the left side is a word cloud where each word has a random initial position within a relatively small fixed size square. On the right side is a word cloud where every word has a random initial position within too large space.

5.2 Archimedean spiral

During collision detection, if a word collides with any of the words which have already been placed, then I need to find a new position for that word, *Archimedean Spiral* can determine where the word will move to in my thesis. A point moves away from the fixed point with constant horizontal speed and rotates with constant angular velocity at the same time, the locus of this point is called an *Archimedean Spiral*. It is widely used in processing digital light [36], bacterial determination in food microbiology [37], and processing medical images [38]. In the polar coordinate system, it is described as equation [38]:

$$r(\theta) = a + b\theta \tag{5}$$

Where:

r: radial distance from origin.

a: distance between starting point and central point of solar coordinate system.

b: constant value, controls distance between successive spiral lines.

θ: polar angle, such as $\frac{\pi}{4}$, $\frac{\pi}{2}$, π *and* 2π.

Figure 22 shows one example of an *Archimedean Spiral* in polar coordinate system. In the example, $a = 0;\ b = 2\ and\ 0 \leq \theta \leq 4\pi$ are given. The coordinate point is represented as (r, θ). The point $(0, 0)$ means the distance between original point and the moving point is 0 when moving point moves 0 radians. The point $(4\pi, 2\pi)$ means the distance between original point and the moving point is 4π when the moving point moves 2π radians around original point. If θ is 2π then r is 4π. The parameter a determines where the starting point is. The parameter b determines the distance between successive spiral lines. θ represents how many radians the point moves around the starting point or the original point.

$$r(\theta) = 2\theta$$

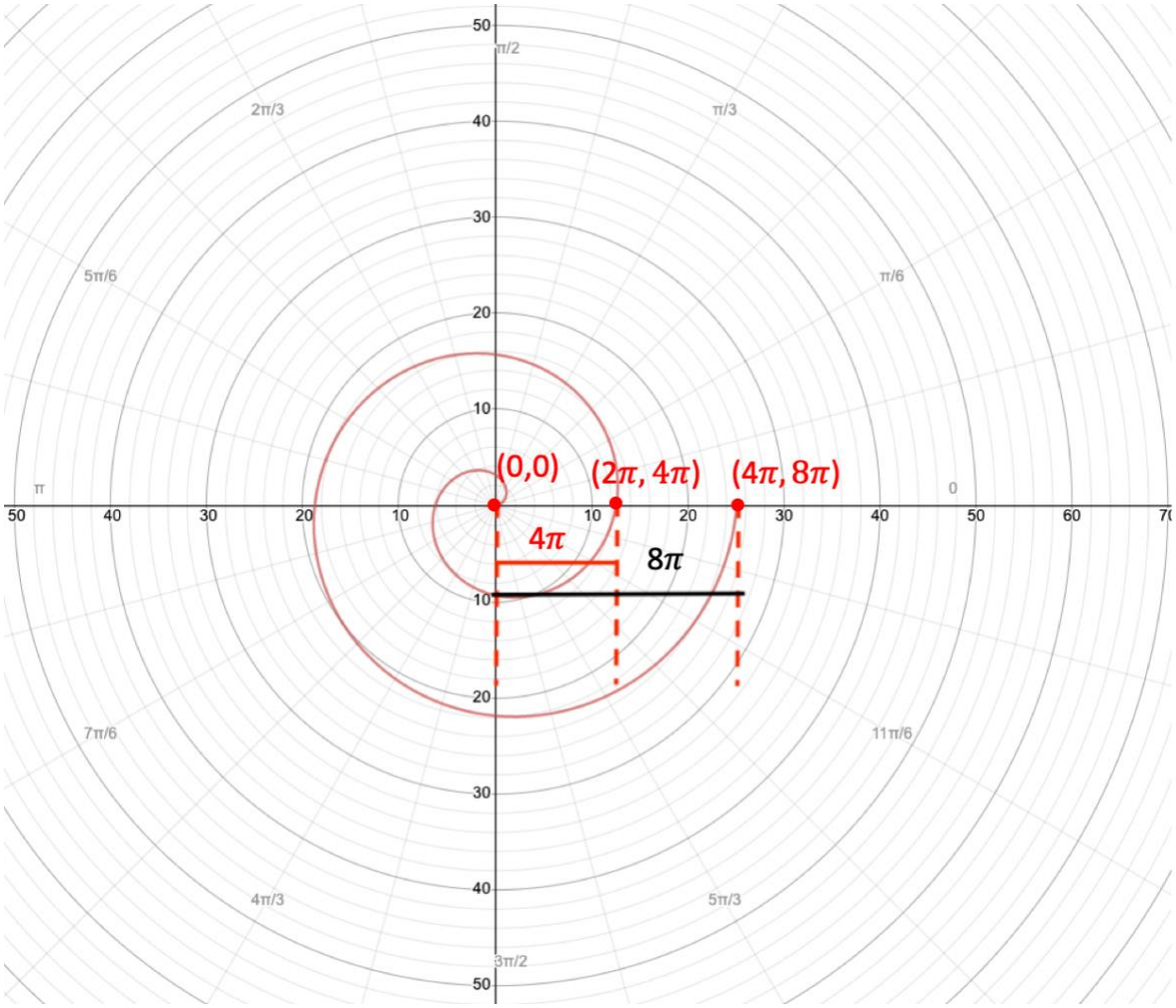


Figure 22: *Archimedean Spiral* geometry in polar coordinate system when $a = 0$; $b = 2$ *and* $0 \leq \theta \leq 4\pi$.

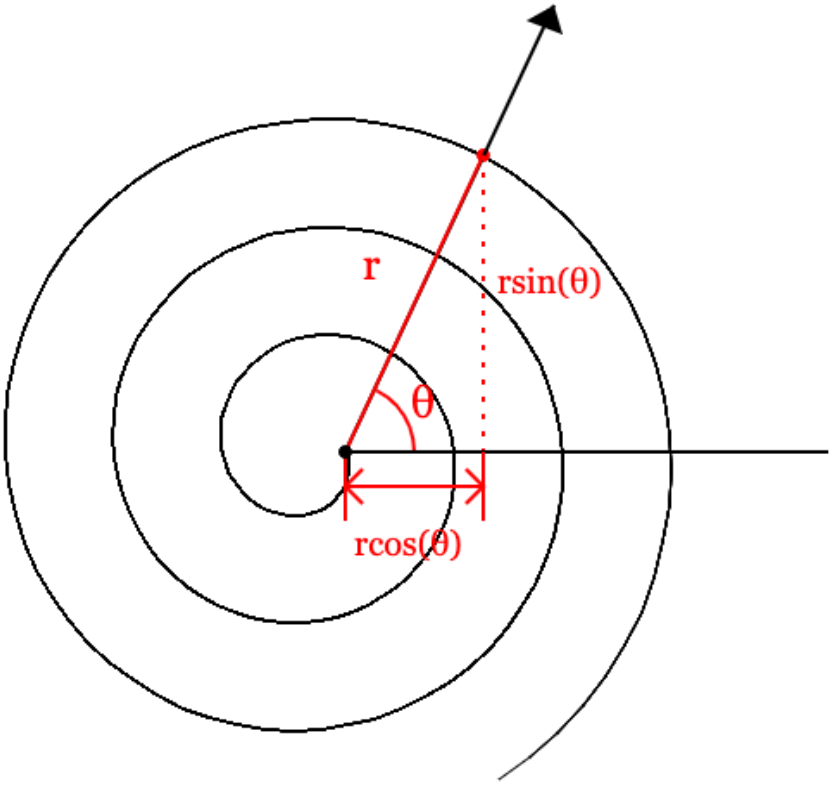


Figure 23: *Archimedean Spiral* can be described in both polar coordinate system and Cartesian coordinate system

Considering that I will work on the Cartesian coordinate system in the thesis, I need to convert the *Archimedean Spiral* equation from polar coordinate system to the Cartesian coordinate system. **Figure 23** shows how to express each equation in parametric form in the Cartesian coordinate system. The x-coordinate and y-coordinate can be expressed as below.

$x = r * cos(\omega t)$

$y = r * sin(\omega t)$

Combined with equation 5, we will have equation 6 and 7:

$x = (a + b\omega t) * cos(\omega t)$ (6)

$y = (a + b\omega t) * sin(\omega t)$ (7)

x: X-axis value in Cartesian coordinate system
y: Y-axis value in Cartesian coordinate system
t: time
ω: constant angular velocity

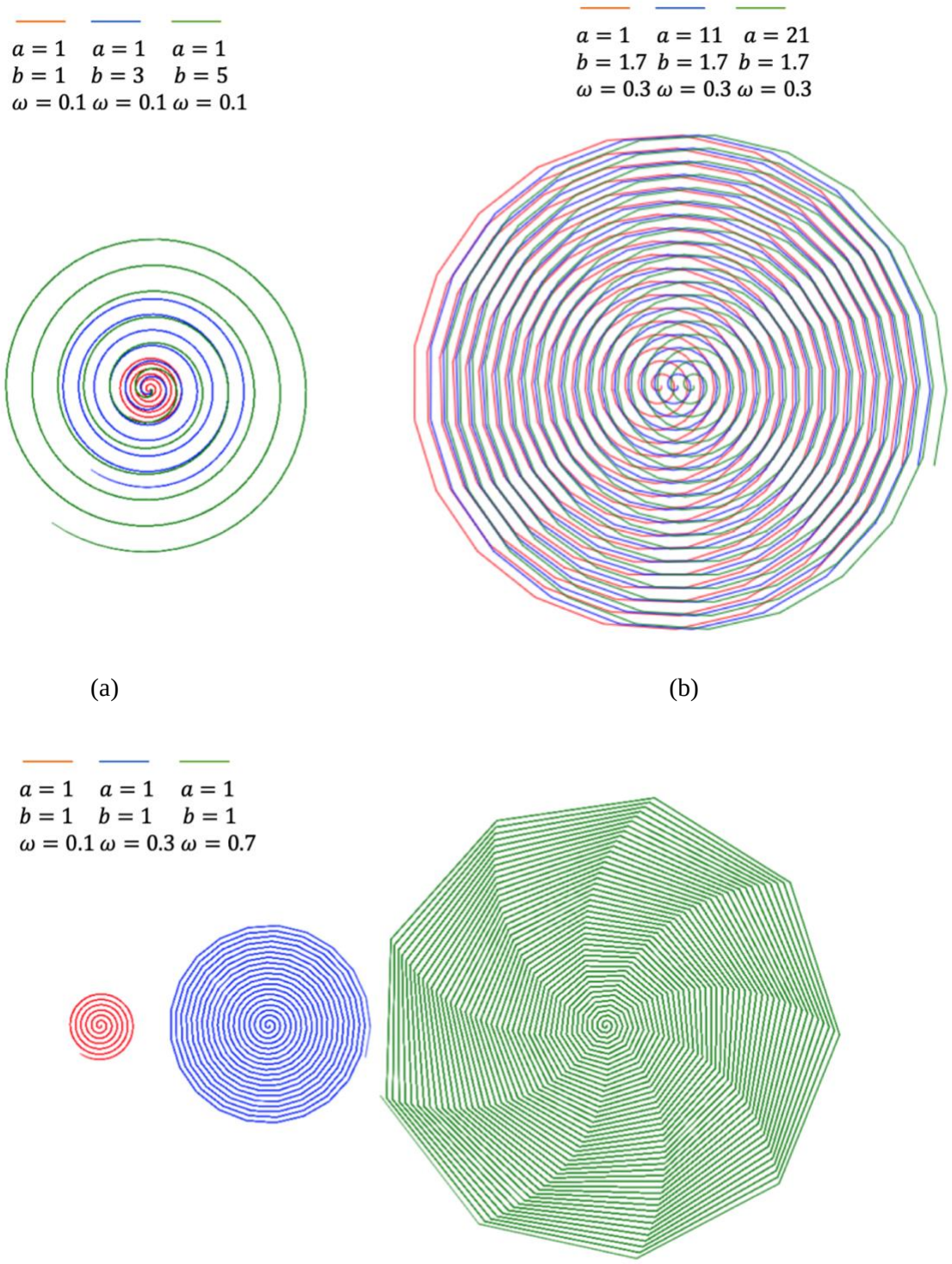


Figure 24. How parameters a, b and ω affect *Archimedean Spiral* when t = 400

Figure 24 shows how the parameters a, b, and ω affect the shape of *Archimedean Spiral*. **Figure 24 (a)** shows how the parameter b affects *Archimedean Spiral* when the parameter a is 1 and the parameter ω is 0.1. From **Figure 24 (a)** one can see that if the value of parameter

b is increased, the radial distance between moving point to the starting point will be increased. **Figure 24 (b)** shows how the parameter a affects *Archimedean Spiral* when the parameter b is 1,7 and the ω is 0.3. From **Figure 24 (b)** one can see that the starting point will move to the right if the value of parameter a is increased. **Figure 24 (c)** shows how the parameter ω affects *Archimedean Spiral.* In **Figure 24 (c)**, the parameters a and b are 1, so the starting point and distance between successive spiral lines are the same. When I increase the value of the parameter ω, the moving point rotates faster around starting point and it makes more revolutions in the same number of steps.

(a) Three words are already placed. (b) Looking for a position for the word Joensuu

Figure 25: How to move in steps when the word Joensuu collides with other words.

Figure 25 shows an example on how to use *Archimedean Spiral* with word clouds. The example shows how the word Joensuu moves when the word Joensuu cannot be drawn at the current position. In this example, my input data are the words Helsinki, Espoo, Vantaa and Joensuu, their corresponding font sizes are 50px, 15px, 30px and 15px. The parameters of the *Archimedean Spiral* I used in **Figure 25** are $a = 12;\ b = 5\ and\ \omega\ = 0.7$. The equations in cartesian coordinate system are represented as:

$x = (12 + 3.5 * t) * cos(0.7 * t)$

$y = (12 + 3.5 * t) * sin(0.7 * t)$

Parameters x and y describe the location of the word Joensuu in a two-dimensional space. The parameter *t* means moving steps of the word Joensuu. If Joensuu hasn't yet moved, the value of *t* is 0. In this example, the words Helsinki, Vantaa, and Espoo have been placed, as shown in **Figure 25 (a)**. Red dots in **Figure 25 (b)** indicate the positions tried for Joensuu.

The initial position of the word Joensuu is at position 1. When Joensuu is placed in position 1, it collides with other words, so I move Joensuu to position 2. It still collides with other words, therefore I move Joensuu to position 3 and see the effect. After 11 times of trial, it is found that the word Joensuu does not collide with any words which have already been placed. So, how many times it has been tried become the value for the parameter *t.* In the beginning, Joensuu has not moved, so *t* is 0. When Joensuu collides with other words at position 1 which is the first time Joensuu needs to move, then *t* is 1 and Joensuu has a new position which is a red dot with the number 2. This is how I use the *Archimedean Spiral* in my thesis. The main function of the *Archimedean Spiral* is to help the word to find the next possible position near starting point.

6 Shaped word cloud

With the popularity of word clouds on social media, there are some tools that can create shaped word clouds, for example, Tagxedo[11] and WordArt[12]. Shaped word clouds are also used in learning web apps, such as ABCya[13] In this section, I will explain how to generate shaped word clouds and discuss their limitations. Shaped word cloud is a type of word cloud that has certain shape boundary for fitting words into it [39]. In **Section 2.2**, I have previously introduced how to fit word clouds into proper two-dimensional space, which is to place the largest font size word as the first step, then put the rest of words around the largest font size word. The algorithm **Generate Word cloud** which is introduced in **Section 2.2** is the main algorithm to generate word clouds. In the algorithm **Generate Word cloud**, it's not necessary to check whether words exceed two-dimensional space or not, as the size of two-dimensional space is large enough. The algorithm to generate shaped words cloud is almost the same as the algorithm Generate Word cloud, however, there is an extra step which is to check if the word exceeds the two-dimensional space.

Preparing shaped two-dimensional space and fitting words are two main steps to generate shaped word clouds. I have introduced how to generate rectangular word clouds earlier. Since generating shaped word clouds will adopt similar method I will pass the introduction for the algorithm. Here I will focus on how to prepare a shaped two-dimensional space. In **Section 6.1** I will present how to get a two-dimensional city shape.

Section 6.2 mainly discusses the limitations of shaped word clouds by a given example. In the example, I classify five groups of words, each group has 5, 10, 20, 40, 80, and 100 words and these words will be assigned with equal weight distributions, linear increasing weight distributions, and exponential increasing weight distributions. The font size functions will be monotonic increasing linear function, which I mark as $lif(w_{weight})$ and monotonic increasing logarithm function, which I mark as $lof(w_{weight})$. The two-dimensional spaces will be presented in heart shape and tree shape. I will demonstrate how the heart-shaped word cloud and tree-shaped word cloud affect visually recognition of the shape by applying different word quantities, weight distributions, and font size functions.

[11] http://www.tagxedo.com/
[12] WordArt.com. WordArt website. http://www.wordart.com/ last visited 11/2019.
[13] ABCya.com.ABCya website. https://www.abcya.com/games/word_clouds last visited 11/2019.

6.1 City shape

In order to generate a shaped word cloud, I need to place words into a two-dimensional space of a certain shape. The shape is determined by a boundary. If I want to obtain city shape, all I should do is to get the city boundary. **Figure 26** shows the city shape of Joensuu on Google Maps. The orange line is the city boundary line. **Figure 27** shows the two-dimensional space that has the city shape of Joensuu. In this section, I will explain how to obtain the two-dimensional space with city shape step by step.

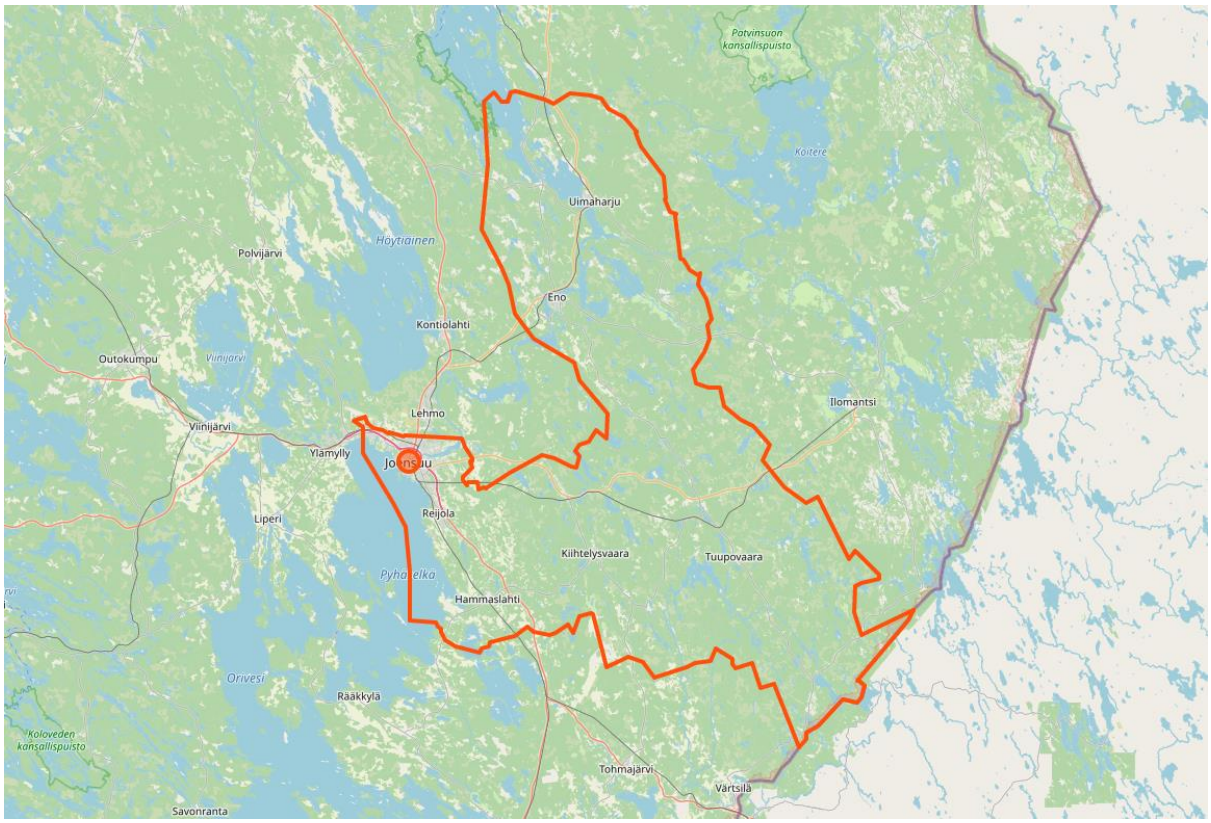


Figure 26: The Joensuu city boundary on Google Maps.

Figure 27: The two-dimensional space that has the shape of Joensuu city.

Nominatim API[14] is a tool for searching *OpenStreetMap* (OSM) data by name and address, and to generate synthetic addresses of OSM points. I will use Nominatim API to get the city boundaries by providing city name. The output is a group of dots in the geographic coordinate system. If I want to get the city boundary of the Joensuu city, the API[15] is to retrieve the Joensuu city boundary. The result is a group of geographic coordinate values which represent latitude and longitude. **Appendix 3** is the Joensuu city boundary in the geographic coordinate system.

After receiving the city boundary in the geographic coordinate system, I need to draw all the dots on the HTML Canvas and line them up to get the city shape. The latitude and longitude I got here are in Spherical Coordinate System, so I have to translate latitude and longitude from sphere to the points on plane, otherwise city shape will be twisted. The function fromLatLngToPoint (see **Appendix 8**) that is provided by Google Maps API is used to get

[14] https://nominatim.openstreetmap.org/search?q=CITYNAME&polygon_geojson=1&format=json
[15] https://nominatim.openstreetmap.org/search?q=joensuu&polygon_geojson=1&format=json

projected points. I marked projected latitude as latitudeProjected while longitudeProjected represents projected longitude.

The unit of HTML Canvas is one pixel. So, the location on HTML Canvas can be represented as $(integer, integer)$. The difference in latitudeProjected and longitudeProjected are quite small, so, if I draw latitudeProjected and longitudeProjected on HTML Canvas directly, the city shape will be small. So I have to zoom in the geographic coordinate values. My idea is to zoom in the difference between the lowest latitudeProjected value and the highest latitudeProjected value and the difference between the lowest longitudeProjected and the highest longitudeProjected value. For example, I want either the city shape width is between 600 pixels and 700 pixels or the city shape height is between 600 pixels and 700 pixels. If both cannot be satisfied at the same time, then the higher value of the height and width will be between 600 pixels and 700 pixels and the other value would be lower. When the projected coordinate point is $(latitudeProjected, longitudeProjected)$, I will have the boundary condition (8) and boundary condition (9). The difference between the lowest latitudeProjected and the highest latitudeProjected is $differ_{latitudeProjected}$ while $differ_{longitudeProjected}$ is the difference between the lowest longitudeProjected and the highest longitudeProjected. The number I am looking for is $multiply_{value}$. When $multiply_{value}$ matches the boundary condition (8) or the boundary condition (9), I will choose the lowest $multiply_{value}$. After I have found $multiply_{value}$ I let all the latitudes and longitudes be multiplied by $multiply_{value}$. For example, $differ_{longitudeProjected}$ and $differ_{latitudeProjected}$ are 1.478642 and 0.7966174 respectively. There are many ways to get $multiply_{value}$ in boundary condition (8) and boundary condition (9). The algorithm in my thesis gives $multiply_{value}$ 437.5. I will get the result shown in **Figure 28** after I draw all the zoomed values on the HTML Canvas.

$$600 \leq differ_{latitudeProjected} * multiply_{value} \leq 700 \quad (8)$$

$$600 \leq differ_{longitudeProjected} * multiply_{value} \leq 700 \quad (9)$$

Figure 28: Zoomed in Joensuu city shape on HTML canvas.

6.2 Shaped word cloud discussion

In this section, I will generate heart-shaped word clouds and tree-shaped word clouds with different word quantities, weight distributions and font size functions. I will also examine how the word quantities, weight distributions and font size functions affect the shaped word cloud. To analyze how word quantities affect the shaped word cloud, I use the same font size function to generate shaped word clouds and I will keep the word weight distribution the same. Next I generate some shaped word clouds with the same word quantity and same word weight distribution, but different font size functions to analyze how font size functions affect the shaped word cloud. When I want to know how word weight distribution affects generating shaped word clouds, I will generate shaped word clouds by using the same words and same font size function. In **Figures 31, 32, 33** and **34**, wN means the quantity of input

words (it is possible that all of them don't fit into the word cloud), wS is the word font size function and wD is word weight distribution. wN=5 means there are 5 input words. fS = lof means the font size function is monotonic increasing logarithmic function while fS = lif means the font size function is monotonic increasing linear function. wD = equal means all word weights are equal. wD = linear means word weight is increased linearly while wD = exp means word weight is increased exponentially.

wN=5, fS=lof, wD=equal

wN=5, fS=lof, wD=linear

wN=5, fS=lof, wD=exp

wN=10, fS=lof, wD=equal

wN=10, fS=lof, wD=linear

wN=10, fS=lof, wD=exp

wN=20, fS=lof, wD=equal

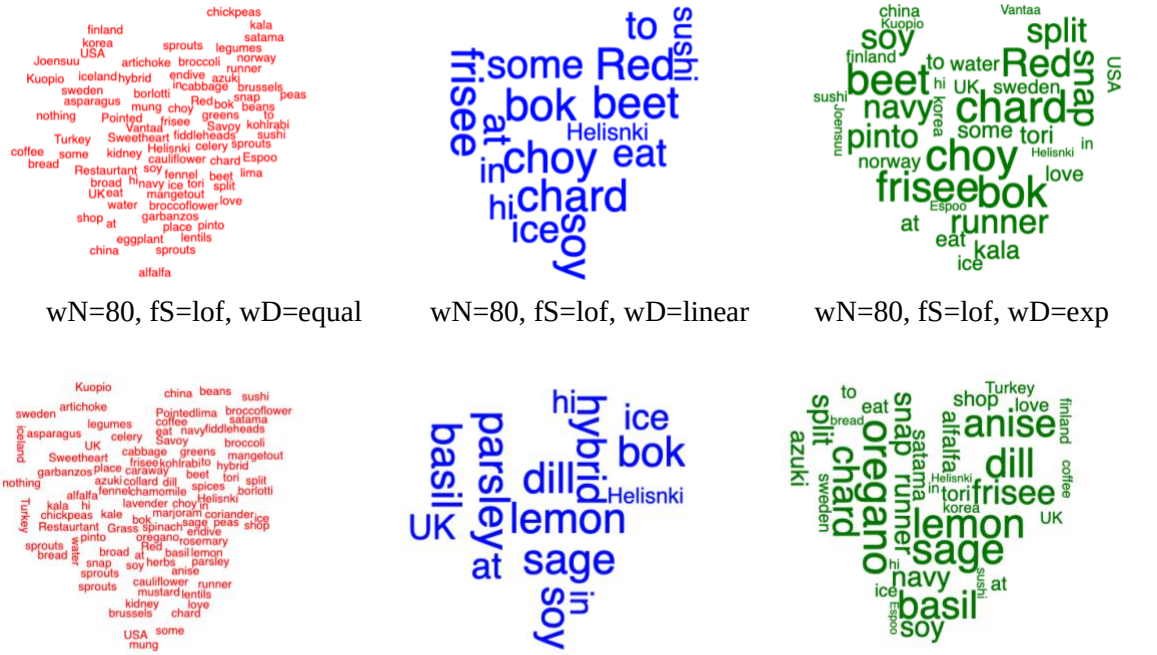
wN=20, fS=lof, wD=linear

wN=20, fS=lof, wD=exp

wN=40, fS=lof, wD=equal

wN=40, fS=lof, wD=linear

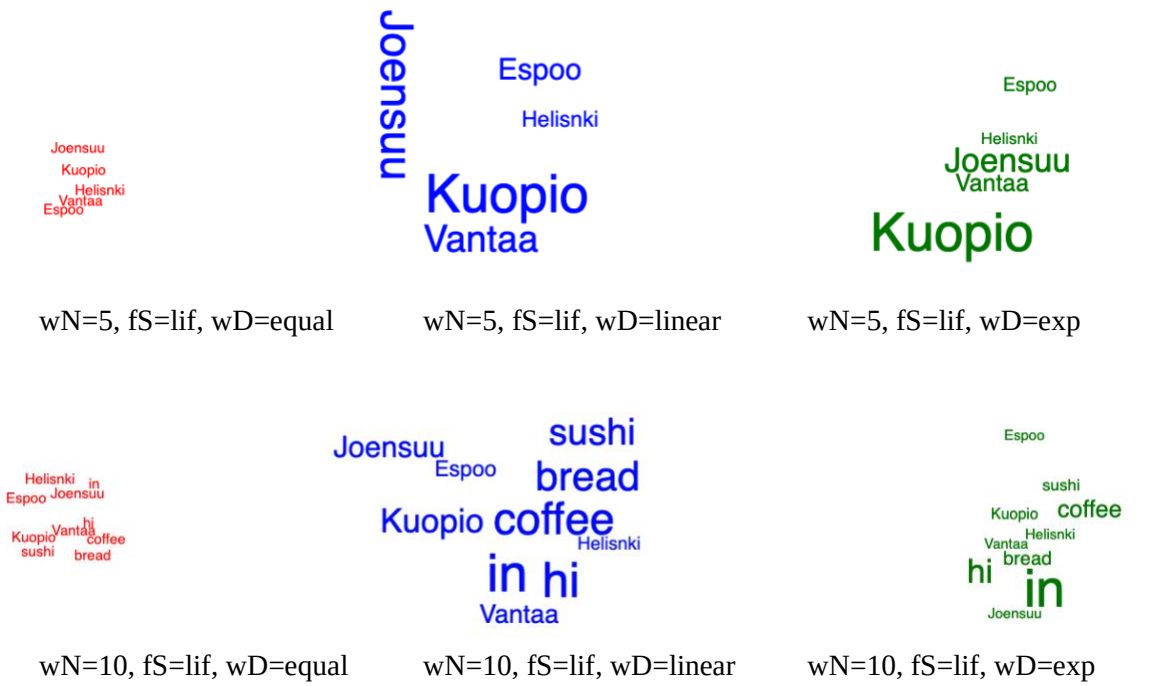
wN=40, fS=lof, wD=exp



wN=80, fS=lof, wD=equal | wN=80, fS=lof, wD=linear | wN=80, fS=lof, wD=exp

wN=100, fS=lof, wD=equal | wN=100, fS=lof, wD=linear | wN=100, fS=lof, wD=exp

Figure 29: Generation of heart-shaped word clouds. The monotonic increasing logarithmic font size function is applied to all the word clouds above. The number of input words is 5, 10, 20, 40, 80 or 100. The word weight distribution is equal, linear, or exponential distribution.



wN=5, fS=lif, wD=equal | wN=5, fS=lif, wD=linear | wN=5, fS=lif, wD=exp

wN=10, fS=lif, wD=equal | wN=10, fS=lif, wD=linear | wN=10, fS=lif, wD=exp

wN=20, fS=lif, wD=equal N=20, fS=lif, wD=linear wN=20, fS=lif, wD=exp

wN=40, fS=lif, wD=equal wN=40, fS=lif, wD=linear wN=40, fS=lif, wD=exp

wN=80, fS=lif, wD=equal wN=80, fS=lif, wD=linear wN=80, fS=lif, wD=exp

wN=100, fS=lif, wD=equal N=100, fS=lif, wD=linear wN=100, fS=lif, wD=exp

Figure 30: Generation of heart-shaped word clouds. The monotonic increasing linear font size function is applied to all the word clouds above. The number of input words is 5, 10, 20, 40, 80, or 100. The word weight distribution is equal, linear, or exponential distribution.

wN=5, fS=lof, wD=equal

wN=5, fS=lof, wD=linear

wN=5, fS=lof, wD=exp

wN=10, fS=lof, wD=equal

wN=10, fS=lof, wD=linear

wN=10, fS=lof, wD=exp

wN=20, fS=lof, wD=equal

wN=20, fS=lof, wD=linear

wN=20, fS=lof, wD=exp

wN=40, fS=lof, wD=equal

wN=40, fS=lof, wD=linear

wN=40, fS=lof, wD=exp

wN=80, fS=lof, wD=equal

wN=80, fS=lof, wD=linear

wN=80, fS=lof, wD=exp

wN=100, fS=lof, wD=equal

wN=100, fS=lof, wD=linear

wN=100, fS=lof, wD=exp

Figure 31: Generation of tree-shaped word clouds. The monotonic increasing logarithmic font size function is applied to all the word clouds above. The number of input words is 5, 10, 20, 40, 80, or 100. The word weight distribution is equal, linear, or exponential distribution.

wN=5, fS=lif, wD=equal

wN=5, fS=lif, wD=linear

wN=5, fS=lif, wD=exp

wN=10, fS=lif, wD=equal

wN=10, fS=lif, wD=linear

wN=10, fS=lif, wD=exp

wN=20, fS=lif, wD=equal

wN=20, fS=lif, wD=linear

wN=20, fS=lif, wD=exp

wN=40, fS=lif, wD=equal

wN=40, fS=lif, wD=linear

wN=40, fS=lif, wD=exp

wN=80, fS=lif, wD=equal

wN=80, fS=lif, wD=linear

wN=80, fS=lif, wD=exp

wN=100, fS=lif, wD=equal

wN=100, fS=lif, wD=linear

wN=100, fS=lif, wD=exp

Figure 32: Generation of tree-shaped word clouds. The monotonic increasing linear font size function is applied to all the word clouds above. The number of input words is 5, 10, 20, 40, 80, or 100. The word weight distribution is equal, linear, or exponential distribution.

From **Figures 29, 30, 31** and **32**, we can see that word quantity is the most important factor to affect generating shaped word clouds. If there are not enough words, it is not easy to recognize the shape of the word cloud. In **Figure 29** and **Figure 30**, when the word quantity is greater than 40, one can recognize heart shape from heart-shaped word cloud, regardless of what the font size function and the word weight distribution is. One can see the same happen to tree-shaped word clouds. **Figure 31** and **Figure 32** show that when the word quantity is greater than 80, one can easily recognize tree shape from a tree-shaped word cloud. So the word quantity is the main factor to determine whether word cloud shape can be recognized or not.

Appendixes 4 and **5** show when using linear font size function and logarithmic font size function, there are numbers of words that cannot find a position to draw on heart-shaped word clouds. In **Appendixes 6** and **7** there are the tables of how many words cannot be drawn on tree-shaped word clouds. I made 20 tests for each shaped word cloud and collected the number of words that cannot be drawn. From **Appendixes 4, 5, 6** and **7**, one can conclude that when word weight is exponentially distributed, one can display more words on the word cloud than linear word weight distribution word cloud. Almost all the words can be shown in the word cloud if the word weight is the same.

From **Figures 29, 30, 31** and **32**, one can see that the linear font size function performs better than logarithmic one in forming shaped word clouds. Linear font size function requires fewer words than logarithmic one to generate shaped word clouds where the shape can be recognized. From **Appendixes 4, 5, 6** and **7**, one can see that the linear font size function is always able to shows more words on the word cloud than the logarithmic font size function.

7 Word cloud implementation

I have already introduced how to generate basic word clouds and shaped word clouds in the previous chapters. In this section, I will introduce how to use word cloud applications. Currently, there are two word cloud applications on Mopsi. The first one is a diagnosis code word cloud and the second one is a recommendation word cloud.

Figure 33 shows two word clouds of diagnosis code. Red marker 'H' indicates the location of a health center where there are patients with certain disease (diagnosis) codes. Assume the diagnosis code 18N means chronic kidney disease, I11 means hypertensive heart disease. I apply diagnosis codes to generate word clouds for the purpose of analyzing geographical distribution of the disease codes for patients. The font size represents the number of patients who got the disease. In **Figure 33 (a)**, the biggest font size diagnosis code is I69, which means patients who suffer sequelae of the cerebrovascular disease are most common. The smallest font size diagnosis code I95 means that the patients who have hypotension are least common among all the patients. **Figure 33 (b)** shows the diagnosis code word cloud in another health center, where a higher number of patients got E11 than any other disease.

(a) (b)

Figure 33: The diagnosis code word cloud

Figure 34: Recommendation word cloud of the city Helsinki on Mopsi

The recommendation word cloud aims to show the services in a region. **Figure 34** is a word cloud generated in the Helsinki region on Mopsi for recommendation services, the word cloud shows what is recommended around Helsinki city center. Red words in **Figure 34** are the keywords to represent the services. For example, the Finnish word ravintola means restaurant and buffet is one kind of meal that a restaurant can offer. From this word cloud, we can easily tell that there are many ravintola and kahvila (Finnish word for coffee shop) in Helsinki city center. There are also ferries and parks but the numbers are small compared with coffee shops and restaurants.

References

[1] R. Mariescu-Istodor and P. Fränti, “Detecting user actions in location-based systems”, Int. Conf. on Location Based Services (LBS), Adjunct proceedings, Zürich, Switzerland, 1-6, January 2018.

[2] M. Halvey and Mark T. Keane, “An Assessment of Tag Presentation Techniques ”, 2017-05-14 at the Wayback Machine, poster presentation at WWW 2007, 2007.

[3] D. Coupland, “Microserfs”, Harper Collins,1995.

[4] J. Stefan and S.Gerik. “TagSpheres: Visualizing Hierarchical Relations in Tag Clouds”, International Conference on Information Visualization Theory and Applications, 15-26, 2016.

[5] M. Gupta, R. Li, Z. Yin, J. Han, “An overview of social tagging and applications”, Social Network Data Analytics, pp. 447–97, 2011.

[6] Q. Castellà, C. Sutton, “Word Storms: Multiples of Word Clouds for Visual Comparison of Documents”, Proceedings of the 23rd international conference on World wide web, 2014.

[7] K. Dressel and Steffen A. Schüle. “Using Word Clouds for Risk Perception in the Field of Public Health – the Case of Vector-Borne Diseases”, European Commission, 7th Framework Programme, 2014.

[8] A. Rajaraman, J. Leskovec, Jeffrey D. Ullman, “Mining of Massive Datasets”, Cambridge University Press, 2011.

[9] E. Schubert, A. Spitz, M. Weiler, J. Geiß, and M. Gertz, “Semantic Word Clouds with Background Corpus Normalization and t-distributed Stochastic Neighbor Embedding”, CoRR, 2017.

[10] Christopher D. Manning, M. Surdeanu, J. Bauer, J. Rose, J. Finkel, Steven J. Bethard, and D. McClosky, “The Stanford CoreNLP Natural Language Processing Toolkit”, Conference: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2014.

[11] G. Hinton and S. Roweis, “Stochastic Neighbor Embedding”, Advances in Neural Information Processing Systems 15, 2003.

[12] Laurens van D. Maaten and G. Hinton, “Visualizing Data using t-SNE” Machine Learning Research 9, 2579-2605, 2008.

[13] R. Mariescu-Istodor. “Efficient management and search of GPS routes”, University of Eastern Finland, 2017.

[14] Y. Jin, “Development of Word Cloud Generator Software Based on Python”, Procedia Engineering, 174: 788-792, 2017.

[15] P. Verma, and J.S. Bhatia, “Design and development of GPS-GSM based tracking system with Google map-based monitoring”, International Journal of Computer Science, Engineering and Applications, 2013.

[16] P. M. Hubbard, “Interactive Collision Detection”, In Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, pp.24-32, 1993.

[17] S. Kockara, T. Halic, K. Iqbal, C. Bayrak and R. Rowe, “Collision detection: A survey”, IEEE International Conference on Systems, Man and Cybernetics, 2007

[18] B. Mirtich, “V-Clip: Fast and Robust Polyhedral Collision Detection”, ACM Transactions on Graphics, pp. 177-208, 1998.
[19] S. A. Ehmann, and M. Lin, “Accelerated Proximity Queries between Convex Polyhedra by Multi-level Voronoi Marching”, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2101-2106, 2000.
[20] E. Gilbert, D. Johnson, and S. Keerthi, “A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space”, IEEE Journal of Robotics and Automation, pp. 193-203, 1988.
[21] D. Knott and D. Pai, “Cinder: Collision and Interference Detection in Real–time Using Graphics Hardware”, In Proc. of Graphics Interface ’03, 2003.
[22] E. Gundelman, R. Bridson, and R. Fedkiw, “Nonconvex Rigid Bodies With Stacking”, ACM Transaction on Graphics, 2003.
[23] S. Dinas, José M. Banon, “A literature review of bounding volumes hierarchy focused on collision detection”, Ingeniería y Competitividad, p. 49-62, 2015.
[24] A. Bade, Norhaida M. Suaib, Abdullah M. Zin, “Oriented convex polyhedra for collision detection in 3D computer animation”, Proceedings of the 4th international conference on Computer graphics and interactive techniques, 2006.
[25] Norhaida M. Sualb, A. Bade and D. Mohamad, “Collision Detection Using Bounding-Volume for avatars in Virtual Environment applications”, The 4th International Conference on Information & Communication Technology and Systems, 2008.
[26] R. Weller and G. Zachmann, “Inner Sphere Trees”, Clausthal University of Technology, 2009.
[27] F. A. Madera, A. M. Day and S. D. Laycock, “A Hybrid Bounding Volume Algorithm to Detect Collisions between Deformable Objects”, Second International Conferences on Advances in Computer-Human Interactions, 2009.
[28] C. Tu and L. Yu, “Research on Collision Detection Algorithm Based on AABB-OBB Bounding Volume”, IEEE: First International Workshop on Education Technology and Computer Science, 2009.
[29] X. Zhang and Young J. Kim, “Interactive Collision Detection for Deformable Models Using Streaming AABBs”, IEEE Transactions on Visualization and Computer Graphics, 2007.
[30] R. Weller, J.Klein and G. Zachmann, “A Model for the Expected Running Time of Collision Detection using AABB Trees”, Conference: Proceedings of the 12th Eurographics Symposium on Virtual Environments, 2006.
[31] J.-W. Chang, W. Wang, and M.-S. Kim, “Efficient collision detection using a dual OBB-sphere bounding volume hierarchy”, Computer-Aided Design, 2009.
[32] S. Gottschalk, M.c. Lin and D. Manocha, “OBBTree: a hierarchical structure for rapid interference detection”, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1997.
[33] D. Angelo, “A Brief Introduction to Quadtrees and Their Applications”, CCCG, 2016.
[34] G. M. Hunter, “Efficient Computation and Data Structures for Graphics”, Princeton University, 1978.
[35] G. M. Hunter and K. Steiglitz, "Operations on images using quad trees". IEEE Transactions on Pattern Analysis and Machine Intelligence. 1979.

[36] Glen M. Ballou,, “Handbook for Sound Engineers”, Focal Press, 2008.
[37] J. E. Gilchrist, J. E. Campbell, C. B. Donnelly, J. T. Peeler and J. M. Delaney, “Spiral plate method for bacterial determination”, Appl Microbiol, 1973.
[38] L. Nanni, A.Lumini, and S. Brahnam, “Local binary patterns variants as texture descriptors for medical image analysis”, Artificial intelligence in medicine, 2010.
[39] Y. Wang, X. Chu, K. Zhang, C. Bao, X. Li and J. Zhang, "ShapeWordle: Tailoring Wordles using Shape-aware Archimedean Spirals”, IEEE Transactions on Visualization and Computer Graphics, pp. 991-1000, 2020.

Appendix 1: WordCloudOverlay class

```
class WordCloudOverlay extends google.maps.OverlayView {
   constructor(bounds, image, map) {
     super(bounds, image, map);
     this.bounds = bounds;
     this.image = image;
     this.map = map;
     this.div = null;
     this.setMap(map);
   }
   onAdd() {
     const div = document.createElement("div");
     div.style.border = "none";
     div.style.borderWidth = "0px";
     div.style.position = "absolute";
     const img = document.createElement("img");
     img.src = this.image;
     img.style.width = "100%";
     img.style.height = "100%";
     div.appendChild(img);
     this.div = div;
     const panes = this.getPanes();
     panes.overlayImage.appendChild(this.div);
   }
   draw() {
     const overlayProjection = this.getProjection();
     const sw = overlayProjection.fromLatLngToDivPixel(
       this.bounds.getSouthWest()
     );
     const ne = overlayProjection.fromLatLngToDivPixel(this.bounds.getNorthEast());
     this.div.style.left = `${sw.x}px`;
     this.div.style.top = `${ne.y}px`;
     this.div.style.width = `${ne.x - sw.x}px`;
     this.div.style.height = `${sw.y - ne.y}px`;
   }
   onRemove() {
     this.div.parentNode.removeChild(this.div);
   }
   hide() { if (this.div) {this.div.style.visibility = "hidden"; }}
   show() {if (this.div) {this.div.style.visibility = "visible"; }
   }
 }
```

Appendix 2: Cities in Finland with their populations

Region name in Finnish	Population
Helsinki	648,650
Espoo	281,866
Vantaa	226,160
Kuopio	118,434
Joensuu	76,228
Tampere	234,441
Turku	190,935
Oulu	202,753
Lahti	119,999
Pori	84,566
Vaasa	67,465
Kokkola	47,723
Rovaniemi	62,667
Hamina	20,410
Rauma	39,410
Lappeenranta	72,801
Lohja	46,490
Hyvinkää	46,622
Hämeenlinna	67,713
Jyväskylä	140,812
Mikkeli	53,983
Nokia	33,403
Porvoo	50,224
Savonlinna	33,866
Seinäjoki	63,072

Appendix 3: Joensuu city boundary in the geographic coordinate system.

[29.6132972,62.650188],[29.6135623,62.6495318],[29.6245285,62.6467503],[29.6262394,62.6469752],[29.6282242,62.6465749],[29.6294816,62.6468596],[29.6306851,62.6465955],[29.6317808,62.6453699],[29.6329853,62.6442937],[29.6332535,62.6441699],[29.6345144,62.643861],[29.6353778,62.6438494],[29.636201,62.6437859],[29.6367737,62.6434358],[29.6370784,62.6428747],[29.63828,62.6421506],[29.6384997,62.6416316],[29.6382534,62.6408827],[29.6378852,62.6402832],[29.6379313,62.6397925],[29.6374139,62.6395139],[29.6375719,62.6392546],[29.6376475,62.6382931],[29.6380118,62.6377942],[29.63828,62.6366106],[29.6377143,62.6354993],[29.6378616,62.6350947],[29.6375378,62.6345946],[29.6361771,62.6344954],[29.6362147,62.6336008],[29.6356514,62.6325538],[29.6357963,62.6322837],[29.635174,62.6316203],[29.635469,62.6312878],[29.6353135,62.6309103],[29.6383444,62.6111662],[29.7179918,62.5507293],[29.7510543,62.5215082],[29.7611283,62.4669893],[29.7629083,62.4589701],[29.7632206,62.4074414],[29.8306858,62.4042442],[29.8357357,62.3995097],[29.8448499,62.3965351],[29.8455962,62.3929524],[29.8460686,62.3918436],[29.851718,62.3875229],[29.8565091,62.3830477],[29.8827616,62.3758707],[29.8830849,62.3745457],[29.945134,62.3676496],[29.9496635,62.3702152],[29.9525145,62.3720043],[29.9696054,62.3751455],[29.9742258,62.3803936],[30.0384838,62.3881646],[30.0711485,62.3998894],[30.0875673,62.3934349],[30.1060947,62.3908399],[30.1309821,62.3895424],[30.1453961,62.3914166],[30.1614347,62.3961418],[30.1781991,62.4015388],[30.192302,62.3918651],[30.2104837,62.4116576],[30.2127305,62.412474],[30.2410745,62.4162675],[30.2443928,62.4151471],[30.2519282,62.4021793],[30.2533454,62.397375],[30.2772304,62.3510059],[30.2797537,62.3501237],[30.3167047,62.3579987],[30.3377553,62.3622161],[30.3768839,62.3561705],[30.4071982,62.3373845],[30.4759378,62.3489387],[30.4931636,62.3500916],[30.5133155,62.3541336],[30.5557279,62.3569082],[30.5699938,62.3725592],[30.6123123,62.3591854],[30.6134184,62.356363],[30.6140751,62.3539733],[30.6147664,62.353043],[30.6204698,62.3488243],[30.6660968,62.354695],[30.686387,62.327963],[30.6887029,62.3241743],[30.7229577,62.3294719],[30.7617752,62.2835122],[30.7619135,62.2813418],[30.7656811,62.2784638],[30.7778483,62.2646325],[30.7827913,62.2586477],[30.7860118,62.2550086],[30.7895411,62.2510199],[30.8086553,62.2610043],[30.809476,62.2696122],[30.8109585,62.2706919],[30.8344904,62.2753378],[30.8404391,62.2786442],[30.8472038,62.282009],[30.8602311,62.2885747],[30.8865689,62.3016362],[30.9043505,62.310261],[30.9207199,62.3088002],[30.9307385,62.3067193],[30.9432722,62.3059688],[30.9551691,62.3122652],[30.96225,62.32725],[30.9619598,62.3275432],[30.9619993,62.3277475],[30.9612132,62.3281863],[30.9599971,62.3282108],[30.9584637,62.3283581],[30.9576177,62.3287265],[30.9568774,62.3290211],[30.9556613,62.3290211],[30.9555027,62.3295368],[30.9550268,62.3297578],[30.9539164,62.3299542],[30.953652,62.3303471],[30.9529647,62.3306417],[30.9514842,62.3307645],[30.9506382,62.33101],[30.9511044,62.3313354],[30.9509554,62.3315993],[30.9499372,62.3317538],[30.9498078,62.3320459],[30.948746,62.332244],[30.9479609,62.3322566],[30.9473409,62.3325663],[30.9471002,62.3324601],[30.9471484,62.3321885],[30.9465139,62.332164],[30.9457208,62.332164],[30.9452449,62.332558],[30.9449429,62.3325187],[30.944683,62.3322991],[30.9440288,62.332385],[30.9436058,62.3326

059],[30.9424425,62.3329988],[31.0714363,62.4051672],[31.0773055,62.4073714],[31.091
9394,62.4198906],[31.0870055,62.4187439],[30.9543576,62.3886611],[30.9353118,62.4166
036],[30.9369018,62.4489968],[30.9427434,62.4498921],[30.9550835,62.452594],[30.9996
389,62.457725],[30.999708,62.4631429],[30.9364524,62.4918461],[30.9197226,62.4824411
],[30.8395297,62.5583464],[30.8090771,62.5509541],[30.7501768,62.5698771],[30.692624
6,62.6019453],[30.7129567,62.6215327],[30.6683939,62.6434632],[30.6346576,62.6519599
],[30.5914848,62.6676128],[30.5922107,62.6698347],[30.5806311,62.6906327],[30.551008
1,62.690252],[30.5168224,62.6951996],[30.5141263,62.7007805],[30.5120869,62.7070577],
[30.5098401,62.7113685],[30.5402927,62.7331668],[30.5529092,62.7494573],[30.5477589,
62.7658971],[30.5580596,62.7738685],[30.5565179,62.7767299],[30.5453393,62.7974693],
[30.5219382,62.814972],[30.5271576,62.8285974],[30.5232744,62.8267685],[30.5051641,6
2.8316286],[30.50113,62.8321773],[30.488234,62.8454401],[30.4847739,62.8449725],[30.4
84448,62.8450398],[30.4843139,62.8452406],[30.4838848,62.8454928],[30.4836139,62.845
8233],[30.4822728,62.8457939],[30.4816183,62.8458723],[30.4813179,62.8461758],[30.47
88288,62.8457547],[30.4775722,62.8461905],[30.4762351,62.8462419],[30.4763692,62.846
3974],[30.4763397,62.8464892],[30.4759562,62.84669],[30.4756517,62.8467328],[30.4746
459,62.8506527],[30.4734475,62.8554259],[30.4630598,62.8974439],[30.4650017,62.90010
82],[30.4542622,62.9003662],[30.4552492,62.901905],[30.4489299,62.9033494],[30.43616
26,62.9175614],[30.4349717,62.9181425],[30.4361841,62.9184609],[30.4363343,62.918872
4],[30.4333302,62.9453162],[30.4288456,62.9483208],[30.4179343,62.9523329],[30.41342
82,62.9550648],[30.4115078,62.9581175],[30.4104134,62.9583395],[30.3878185,62.970604
4],[30.3766283,62.9778198],[30.3772114,62.9785687],[30.3721651,62.9811978],[30.35888
28,62.9977608],[30.3613719,62.9984119],[30.351834,63.0034462],[30.3266094,63.0167505
],[30.3041218,63.0216546],[30.2859208,63.0257208],[30.2660467,63.0393436],[30.243327
9,63.041109],[30.2410705,63.0410973],[30.2394403,63.0424329],[30.2040909,63.0433801],
[30.186589,63.0439839],[30.1849335,63.0444112],[30.1826145,63.0443849],[30.1823607,6
3.044097],[30.1402667,63.0323534],[30.1191971,63.0369348],[30.0421843,63.0474023],[3
0.0338539,63.0476373],[30.027079,63.04676],[30.0256618,63.040508],[30.0204424,63.034
8973],[30.0032977,63.0354459],[29.9833186,63.0457416],[29.9578754,63.0347865],[29.95
814,63.0329212],[29.9585536,63.0300061],[29.959986,63.0195471],[29.9601822,63.018522
9],[29.9684929,62.9598347],[29.9527309,62.8810479],[30.0042686,62.836076],[30.005582
1,62.8308837],[30.0232107,62.8149697],[30.0252156,62.8115268],[30.0385234,62.7798748
],[30.0900958,62.7495974],[30.1970427,62.7213801],[30.2010177,62.7159933],[30.204992
8,62.7096863],[30.2106616,62.6997794],[30.2846672,62.6591818],[30.2805884,62.6293367
],[30.2547677,62.6389495],[30.2330603,62.6250138],[30.2108344,62.6213421],[30.203299
1,62.6174633],[30.1990129,62.6130752],[30.2105925,62.6068894],[30.2004992,62.5935751
],[30.1872605,62.5946729],[30.1394213,62.6052513],[30.1120105,62.6083047],[29.956083
8,62.5701613],[29.9437783,62.5668969],[29.9411513,62.5767846],[29.9109407,62.5730592
],[29.9039238,62.5826743],[29.9256842,62.592931],[29.9222054,62.6005318],[29.9277354,
62.6011546],[29.9254068,62.6046012],[29.9137577,62.6159665],[29.9085398,62.6182077],
[29.9101485,62.6191482],[29.9005427,62.628217],[29.7461305,62.63234],[29.7421771,62.6
327508],[29.6945928,62.6376944],[29.6650735,62.6417294],[29.6496916,62.6474474],[29.
6432278,62.6556254],[29.6373692,62.6546904],[29.6132972,62.650188]

Appendix 4: Number of words that cannot find a position to draw on heart-shaped word cloud with linear font size function.

	Linear Font Size Function																	
	N=5			N=10			N=20			N=40			N=80			N=100		
	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal
1	0	0	0	0	0	0	3	0	0	15	0	0	45	0	0	66	4	3
2	0	0	0	0	0	0	2	0	0	14	0	0	48	1	0	63	10	4
3	0	0	0	0	0	0	2	0	0	16	0	0	46	1	1	66	11	2
4	0	0	0	0	0	0	3	0	0	14	0	0	48	0	0	63	7	2
5	0	0	0	0	0	0	1	0	0	12	0	0	48	1	0	58	8	4
6	0	0	0	0	0	0	3	0	0	12	0	0	45	1	0	68	7	2
7	0	0	0	0	0	0	2	0	0	15	0	0	48	1	0	65	4	4
8	0	0	0	0	0	0	3	0	0	14	0	0	45	1	0	63	10	3
9	0	0	0	0	0	0	2	0	0	14	0	0	47	1	0	63	5	7
10	0	0	0	0	0	0	3	0	0	16	0	0	45	1	0	62	9	6
11	0	0	0	0	0	0	3	0	0	14	0	0	46	0	0	64	10	4
12	0	0	0	0	0	0	1	0	0	15	0	0	46	1	0	63	10	3
13	0	0	0	0	0	0	2	0	0	15	0	0	43	1	0	63	8	4
14	0	0	0	0	0	0	2	0	0	14	0	0	45	1	0	63	11	4
15	0	0	0	0	0	0	1	0	0	14	0	0	46	1	0	64	10	6
16	0	0	0	0	0	0	3	0	0	15	0	0	45	1	0	64	8	1
17	0	0	0	0	0	0	2	0	0	15	0	0	45	1	0	64	7	4
18	0	0	0	0	0	0	2	0	0	15	0	0	43	1	0	63	10	4
19	0	0	0	0	0	0	3	0	0	14	0	0	45	2	0	63	4	3
20	0	0	0	0	0	0	1	0	0	17	0	0	44	0	0	62	8	1

Appendix 5: Number of words that cannot find a position to draw on heart-shaped word cloud with logarithmic font size function.

	Logarithm Font Size Function																	
	N=5			N=10			N=20			N=40			N=80			N=100		
	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal
1	0	0	0	4	0	0	10	4	0	29	12	0	66	49	0	84	65	3
2	2	0	0	3	0	0	10	4	0	27	14	0	63	49	0	86	66	7
3	1	0	0	2	0	0	10	3	0	27	13	0	64	47	1	84	69	6
4	2	0	0	3	0	0	10	1	0	27	13	0	65	47	0	84	63	7
5	1	0	0	2	0	0	10	3	0	28	10	0	65	47	0	86	63	5
6	1	1	0	4	0	0	10	2	0	28	10	0	65	44	0	85	65	4
7	1	0	0	2	0	0	11	2	0	27	13	0	66	45	0	84	61	4
8	1	0	0	3	0	0	10	3	0	27	15	0	66	43	1	84	63	4
9	0	1	0	3	0	0	9	2	0	26	14	0	65	43	0	84	66	3
10	1	1	0	4	0	0	11	2	0	27	13	0	66	44	0	84	64	2
11	1	1	0	3	0	0	10	3	0	26	14	0	65	46	0	84	62	5
12	1	0	0	3	0	0	9	2	0	28	14	0	65	45	0	86	64	2
13	1	1	0	3	0	0	10	3	0	27	12	0	65	45	0	83	62	4
14	1	1	0	3	0	0	10	2	0	27	17	0	65	46	0	85	64	5
15	1	1	0	3	0	0	10	3	0	27	14	0	65	48	0	85	61	3
16	3	0	0	4	0	0	10	3	0	27	15	0	64	48	1	85	65	5
17	1	0	0	3	0	0	10	2	0	27	13	0	64	46	0	87	65	4
18	2	1	0	4	0	0	11	2	0	29	14	0	64	46	0	86	67	2
19	1	1	0	2	0	0	11	2	0	29	15	0	64	42	0	85	63	5
20	1	0	0	2	0	0	10	2	0	26	13	0	65	50	0	84	62	4

Appendix 6: Number of words that cannot find a position to draw on tree-shaped word cloud with linear font size function.

	Linear Font Size Function																	
	N=5			N=10			N=20			N=40			N=80			N=100		
	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal
1	0	0	0	0	0	0	2	0	0	13	0	0	44	0	0	65	4	0
2	0	0	0	0	0	0	2	0	0	14	0	0	44	0	0	65	4	1
3	0	0	0	0	0	0	2	0	0	13	0	0	47	0	0	67	4	0
4	0	0	0	0	0	0	2	0	0	14	0	0	46	0	0	64	4	0
5	0	0	0	0	0	0	2	0	0	14	0	0	47	0	0	64	3	1
6	0	0	0	0	0	0	2	0	0	12	0	0	47	0	0	64	3	0
7	0	0	0	0	0	0	2	0	0	13	0	0	46	0	0	64	4	0
8	0	0	0	0	0	0	1	0	0	14	0	0	44	0	0	68	3	0
9	0	0	0	0	0	0	2	0	0	12	0	0	46	0	0	64	3	0
10	0	0	0	0	0	0	2	0	0	13	0	0	48	0	0	64	2	0
11	0	0	0	0	0	0	2	0	0	13	0	0	48	0	0	64	4	0
12	0	0	0	0	0	0	1	0	0	12	0	0	46	0	0	64	5	0
13	0	0	0	0	0	0	2	0	0	14	0	0	45	0	0	63	8	4
14	0	0	0	0	0	0	2	0	0	15	0	0	47	0	0	64	1	0
15	0	0	0	0	0	0	3	0	0	14	0	0	43	0	0	64	6	0
16	0	0	0	0	0	0	2	0	0	14	0	0	45	0	0	66	5	0
17	0	0	0	0	0	0	2	0	0	12	0	0	45	0	0	65	2	0
18	0	0	0	0	0	0	3	0	0	12	0	0	47	0	0	64	2	0
19	0	0	0	0	0	0	2	0	0	15	0	0	46	0	0	63	4	0
20	0	0	0	0	0	0	1	0	0	12	0	0	44	0	0	64	2	0

Appendix 7: Number of words that cannot find a position to draw on tree-shaped word cloud with logarithmic font size function.

	Logarithm Font Size Function																	
	N=5			N=10			N=20			N=40			N=80			N=100		
	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal	Linear	Exp	Equal
1	1	0	0	2	0	0	10	1	0	27	15	0	64	45	0	86	62	0
2	1	0	0	2	0	0	11	2	0	26	12	0	66	44	0	85	62	0
3	1	0	0	3	0	0	11	0	0	27	14	0	67	45	0	86	64	1
4	1	0	0	3	0	0	11	2	0	25	10	0	65	44	0	86	66	0
5	1	0	0	2	0	0	11	1	0	28	13	0	65	48	0	84	63	0
6	0	0	0	3	0	0	11	2	0	26	12	0	65	44	0	84	62	0
7	1	0	0	3	0	0	10	1	0	27	12	0	64	46	0	86	60	0
8	0	0	0	3	0	0	10	1	0	28	13	0	65	46	0	84	63	0
9	1	0	0	3	0	0	10	2	0	27	11	0	65	45	0	85	64	0
10	1	0	0	3	0	0	10	1	0	26	13	0	65	46	0	86	63	0
11	0	0	0	3	0	0	11	1	0	26	12	0	65	44	0	86	66	0
12	1	0	0	2	0	0	10	1	0	26	13	0	65	46	0	86	66	0
13	0	0	0	1	0	0	11	2	0	28	12	0	63	46	0	84	66	1
14	1	0	0	2	0	0	11	1	0	26	12	0	64	48	0	85	62	0
15	0	0	0	2	0	0	11	1	0	28	11	0	66	44	0	85	64	0
16	1	0	0	2	0	0	10	2	0	27	11	0	65	47	0	86	66	0
17	0	0	0	2	0	0	11	1	0	29	13	0	65	43	0	87	62	0
18	0	0	0	1	0	0	12	0	0	27	13	0	65	45	0	87	64	0
19	1	0	0	2	0	0	9	1	0	28	13	0	65	47	0	85	65	0
20	1	0	0	2	0	0	11	0	0	28	13	0	63	45	0	85	66	0

Appendix 8: The function fromLatLngToPoint

```
fromLatLngToPoint: function (latLng) {
  const latRadians = (latLng.lat() * Math.PI) / 180;
  return new google.maps.Point(
    GALL_PETERS_RANGE_X * (0.5 + latLng.lng() / 360),
    GALL_PETERS_RANGE_Y * (0.5 - 0.5 * Math.sin(latRadians))
  );
}
```