UNIVERSITY OF EXETER

Computer Science Department

B.Sc. Computer Science
Computer Science Department

Automated 3D Visualisation of 2D Geo-spatial Data

Candidate

Jack Purkiss

Student ID 700012494

Supervisor

Dr. Sareh Rowlands

University of Exeter

Co-supervisor

Academic Year
2022/2023

Abstract

(200 WORDS) Contour plots are an effective, widely accessible and cheap method of representing geospatial data. To allow effective communication of data, the method of visualisation used is an important decision to be made. This project looks to automatically visualise the elevation data that can be extracted from a contour map. A focus is made on the extraction of contour lines found from images, due to the more accessible nature of this area. This data can be extracting through image segmentation methods to isolate the lines. The points for the lines can then be detected, and used to determine the elevations from the contour interval. Clustering methods are used in the process of elevation calculation to group the different hills, to then be used in the interpolation of height values for the plot. This is visualised on a graph to be interacted with by a user to gain a further understanding of the topography of the given contour map. The results demonstrate that while it is challenging to produce a fully automated system, the approach used here shows promising results. The system performed well on basic examples, with a suggestion for further testing and refinement for more complex maps and scenarios.

	Yes	No
I certify that all material in this dissertation which is not my own work has been identified.	☑	☐
I give the permission to the Department of Computer Science of the University of Exeter to include this manuscript in the institutional repository, exclusively for academic purposes.	☑	☐

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

DEM Digital Elevation Model

GIS Geographic Information System

1 Introduction

1.1 Introduction and Motivation

Geospatial data is a widely used form of data with many applications. Many collection methods are utilised including RADAR, LIDAR, satellite images and aerial photographs [15], amongst others. This data is important for studies in deforestation and global warming, as well as mapping isolated areas such as wastelands [16]. It is assumed that approximately 60-80% of data available can be interpreted as geodata or spatial data [7], such as the visualisation of gradient descent optimisation algorithms. For this reason, the methodology and representation of this information is important. One of the most widely used methods for the representation of elevation and geospatial information is contour lines. A contour line is a continuous line with a given elevation, with a collection of them representing the change in elevation across a landscape or surface. The contour interval is the gap seen between contour lines, representing the difference in elevation between those two lines. Thus, the smaller gap between the lines indicates a steep slope in the physical topography and wider gaps show a smaller incline. The visualisation of elevation data is a form of Digital Elevation Model (DEM) [8], with contour lines as an example of a 2D collection of data used for this process. The simplicity of contour plots means that it's a popular format to visualise geospatial data and other forms of data, such as multivariate data [9].

Contour line visualisation

To better understand and interpret terrain features, the visualisation of contour lines becomes incredibly useful. With a traditional 2D contour map, only a flat representation of elevation information is provided which can prove challenging to interpret accurately. As the terrains become more complex, this becomes more arduous. Therefore, 3D plots assist greatly by offering an intuitive representation of the terrain which can help in the comprehension of its characteristics [3].

Contour line visualisation has a major advantage: it helps identify trends and patterns in terrain, making it easier to recognise slope, shape, and relief. For example, flat areas are outlined by widely spaced out lines, while closely spaced contours identify steep slopes and cliffs. Although this is determined straight from the contour map, the visualisation clearly presents it as it would be seen in reality.

Image-based contour extraction

To visualise contour lines, the contour information has to be extracted to provide the elevations for a model. There are multiple approaches that could be made here. Geographic Information System (GIS) models may contain contour information with each line assigned a given elevation that can be accessed directly to be visualised. Using this approach is highly accurate as it can include every minor detail in topography found through methods such as

LiDAR or other high-resolution elevation data sources. It also allows highly scalable projects such as for areas over vast regions by using pre-existing elevation data. In this project, we will look at the visualisation of contour lines extracted from images.

Recreational outdoor activities are an example application for the graphical visualisation of contour lines. Activities such as skiing, hiking or mountaineering can use this to better understand the topography to be able to identify suitable paths by avoiding steep slopes and cliffs as well as choosing more suitable routes based on their fitness and experience levels. Image-based extraction allows this to become more accessible than a GIS data-based approach, such as in scenarios where elevation data may not be readily available. This could be in areas that are poorly mapped. There is also greater flexibility in the image source and terrain types that can be used, as there are many sources that can provide contour data, including satellite imagery, aerial photographs, or user-generated photos.

Extracting contour lines from images provides the ability to use specific requirements or preferences to be able to customise the visualisations. This can be used to adapt to different image quality, resolutions and colour maps, which allows for tailoring visualisations to needs such as particular areas of interest, highlighting distinct terrain features or adapting to certain mapping or navigation requirements.

Existing methods for contour line extraction and visualisation may have some limitations. One of the gaps is that many methods can require manual digitisation of the lines, which is time-consuming, labour-intensive and prone to human error [25]. Manual digitisation requires careful tracing of the contour lines by going point by point and for large or complex terrains can take a lot of time and a lot of effort to accurately capture every detail found. Any approach made here is still prone to human error such as mistakes tracing the lines, misinterpreting contour elevations or errors in capturing the spatial relationships between contour lines.

1.2 Project Specification

The project aim is to produce a program that can take an image of a contour plot, extract the contour line data and generate a 3D interactive DEM that graphically represents the elevation data calculated from the contour plot. This will have a front end that allows the user to upload an image, adjust parameters such as the contour interval, and the threshold for image segmentation to produce a binary image that contains only the contour lines. There will be a viewing window that displays the visualisation of the contours for the user to interact with as well. The deliverables for the project can be separated into the following sections:

1.2.1 Contour Extraction

Image segmentation [13] is the first step to extracting the contour lines. This is used in order to produce a binary image that isolates the lines so that there is no other information displayed. Here, we will explore the use of different thresholding techniques to produce a binary image that the contour lines can then be detected from.

Once the contour lines have been isolated, we can extract the data to be stored by the program. We will use a border-following algorithm [26] to trace the segmented lines from the binary image. The points found will be stored in an array and these will be used in the visualisation process.

1.2.2 DEM Visualisation

Contour maps contain a collection of hills showing different peaks. In order to calculate the elevations for each of the contour lines, the hierarchy of contours for each peak can be clustered for individual elevation calculation. Alternative clustering methods will be provided and compared to find what methods are most appropriate for the application.

The contours will be plotted onto an interactive graph so that the user can see the topography and can move around it to fully understand it. To do this, the clustered contour arrays will be used alongside the given contour interval to calculate the elevations for each of the lines. With these elevations provided, a mesh grid can be generated by interpolating the Z values on a graph to then be plotted either through triangulation or with a regular grid method.

A GUI will be created so that the user can fully interact with the program and provide a simple way for them to see their visualised contour lines. This will have different parameters for them to adjust including clustering options, threshold options, and display options to help accurately generate a DEM which will be displayed to them.

1.2.3 Evaluation Methods

With the main project aim being a visualisation tool, many evaluation methods will be qualitative. We will make a visual analysis between thresholding methods under a collection of varying test cases to evaluate which is most appropriate for an automated program. Additionally, a mixture of qualitative and quantitative measures will be made to test the clustering methods. Here we will look at which methods are best at automating the number of clusters using metrics such as the Silhouette Score and Calinksi-Harabasz index (section 3.2), as well as a visual analysis of how the clusters should be grouped into separate hills. Following examples in literature, [27, 8, 18] a comparison is made to existing DEMs that have been generated with different techniques. We will make a qualitative analysis on the comparison between surface plots and their accuracy in their visualisation.

1.2.4 Data source

For this project, the data used will be different examples of raster images of contour lines. A variety of these will be provided, with different colour maps, topographical areas and human-defined ones to test different test scenarios. There are many websites [2, 19, 28] that can provide the contour lines for different areas on a map from around the world, which can provide different test cases for a variety of terrains.

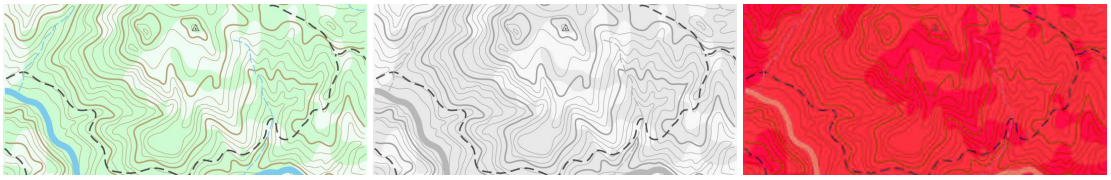
2 Design

2.1 Contour Extraction

The elevation data for the DEM is calculated from the contours extracted from the image. Finding this data involves isolating the contour lines from the image using image segmentation and then detecting the lines from the image using contour detection. In this section, we will discuss the methods used for this process and compare alternatives.

2.1.1 Image Segmentation

The first step in image segmentation is the colour channel used on the image. The options we have here are a grey-scale or an HSV conversion, depending on the image properties. An image containing contour lines undisturbed by other map features can use a grey-scale image to have a threshold applied. However, an HSV conversion may be most appropriate if the image contains features with multiple colours. Often in maps, topographical contour lines can appear in a brown colour. Figure 2.1 shows an original image followed by its grey-scale and HSV channels. In this case, the 'value' channel from an HSV image can be the most appropriate as this will isolate the lines of these values from the image ready for the threshold. Thresholding is a common technique used in image processing to segment an image into a given foreground and background region [22]. This process converts the input image into a binary image based on the threshold given. Figure 2.2 shows a comparison between the three threshold techniques on the same image.



((a)) Original image ((b)) Greyscale image ((c)) HSV image

Figure 2.1: Images in original, greyscale, and HSV formats

Global Thresholding

Global thresholding is a simple technique [12] that involves assigning a binary value of one or zero to each pixel in an image based on whether it falls within a specified threshold range. For a given pixel $g(x, y)$, the threshold value is defined by:

$$g(x,y) = \begin{cases} 1 & if f(x,y) > T \\ 0 & otherwise \end{cases} \tag{2.1}$$

This technique works well for images with consistent lighting and contrast, but in images that contain uneven lighting or varying contrast, it may be less successful.

Adaptive Thresholding

Adaptive thresholding [4] is a technique that uses a threshold that varies across the image. This is based on the local pixel intensity variations. Adaptive thresholding is effective in images that have uneven illumination and varying contrast throughout the image. Many contour maps will not have these features, however, in the case that an image does have some uneven lighting or contrast, then the use of an adaptive threshold could be useful. For example, an image of a map taken on a camera. The formula [5] for this threshold is

$$\text{dst}(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{T}(x, y) \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Otsu's Thresholding

Otsu's threshold [20] is an automatic thresholding technique that calculates an optimal threshold based on the histogram of an image. The idea behind this technique is to minimise the intra-class variance of pixels between the foreground and background classes. A threshold value is chosen where the sum between the intra-class variances is minimised by iterating through the different possible threshold values and calculating the variance for each of them. This can be summarised by the following:

- Compute histogram p_i for the input image
- Compute cumulative sums $\omega_1(t)$ and $\omega_2(t)$, and cumulative means $\mu_1(t)$ and $\mu_2(t)$, where:

$$\omega_1(t) = \sum_{i=1}^{t} p_i, \quad \omega_2(t) = \sum_{i=t+1}^{L} p_i \tag{2.3}$$
$$\mu_1(t) = \frac{\sum_{i=1}^{t} i p_i}{\omega_1(t)}, \quad \mu_2(t) = \frac{\sum_{i=t+1}^{L} i p_i}{\omega_2(t)}$$

- Compute between-class variance $\sigma_w^2(t)$ for all possible thresholds $t = 1, 2, \ldots, L-1$, where:

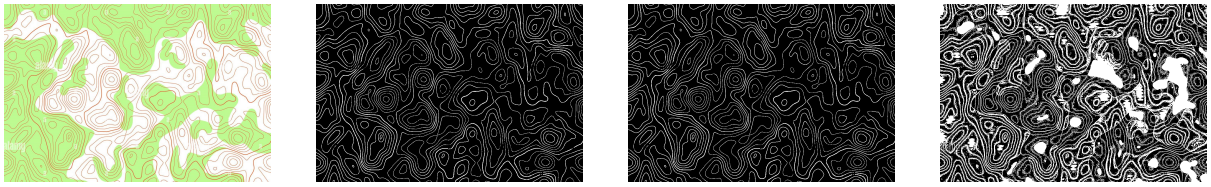
$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

  and

$$\sigma_1^2(t) = \frac{\sum_{i=1}^{t}(i - \mu_1(t))^2 p_i}{\omega_1(t)}, \quad \sigma_2^2(t) = \frac{\sum_{i=t+1}^{L}(i - \mu_2(t))^2 p_i}{\omega_2(t)}$$

- Choose the threshold value that maximizes $\sigma_w^2(t)$

Otsu's thresholding is most effective on images with bimodal histograms, where there is a clear division into two classes. For this reason, the appropriate colour channel used is an important factor, as the initial colour conversion should isolate the lines into a foreground and background. One of the main advantages of Otsu's thresholding is that it's an automatic technique. Therefore, for this project where the aim is to automate the process, this is a valuable attribute as we don't know the given threshold value for every image that might be used.



((a)) Original ((b)) Global ((c)) Otsu's ((d)) Adaptive

Figure 2.2: Contour maps with different thresholding methods.

2.1.2 Contour Detection

To detect the contour lines from the image, a few different approaches could be made. Line thinning [29] is one approach here, which would look to take all of the lines in the image in varying thicknesses and reduce them down to lines all a pixel width. Then we would have a binary image containing all of the line data needed and a line-finding algorithm can be used to isolate the lines by tracing along the neighbouring pixels and storing all of the points in an array. We can use morphological functions to reduce these lines to a desired thickness however with variations in the line thickness throughout different curves and the way they are drawn differently across different maps, using methods such as erosion can make line thinning challenging. The erosion operation is defined as follows:

$$(I \ominus B)(x, y) = \min_{(b,d) \in B} I(x + b, y + d) - b \tag{2.4}$$

where I is the input image, B is the structuring element, $\ominus$ denotes the erosion operation and (x, y) denotes a pixel in the image. The erosion operation removes the pixels around the boundary of the foreground objects in the image, resulting in a contraction of the foreground. Experimental work on this suggested that line thinning was not the most effective technique as it often led to breaks in the lines or some areas still being thicker than the desired width. This led to inaccuracies when extracting the lines for height interpolation, as areas of greater thickness provide more points than what would be accurate for building the DEM, and areas with broken lines mean the interpolation cannot work at all.

OpenCV's contour detection

To detect the contours from the binary image, we will be using the algorithms described in Suzuki and Abe's paper on the "Topological Structural Analysis of Digitised Binary Images by Border Following" [26]. The process of this algorithm begins with scanning the image for the first non-zero pixel to be used as the starting point. Then it begins tracing the boundary of the region by moving along the contour. The direction it moves in is determined by a hierarchy of contours that the algorithm constructs during its tracing. Each white pixel visited is marked so that when it reaches a point it has already visited, it knows that the tracing of the region's boundary has been completed. Throughout this process, information about the contours is stored including their position, size and shape. Hierarchy information that describes each contour's relationships to the other contours in the image is also assigned in this process. This algorithm is implemented by the `findContours` function from OpenCV. By applying this

technique to our binary image with the segmented lines, we are able to isolate the line data to a pixel-thick outline stored in an array for all of the points in the line.

2.2 DEM Visualisation

Having extracted the contour data and stored them in an array, we now need to be able to take this data and use it to form a 3D graphical representation of it. This involves finding the elevation using the lines extracted and interpolating them across a surface to then be displayed to the user. In this section, we will look at the process behind finding the elevations through clustering and the methods for plotting this data onto a graph using a mesh grid.

2.2.1 Contour clustering

Elevations on a single peak

So that we can visualize the contours, the elevation has to be found. The contour interval between the lines is defined, so from there, it becomes an iterative process in assigning an elevation to each line in the array with the formula: $E_i = CI \times i$, where E_i is the elevation of the i-th contour line, CI is the contour interval, and i is the index of the line in the array. This formula is then used to form an array that is an equal length to the array of contours containing the elevations for each contour at their corresponding index. This can be implemented with the following Python code:

```
contour_elevations = [x * contour_interval for x in range(len(contours))]
```

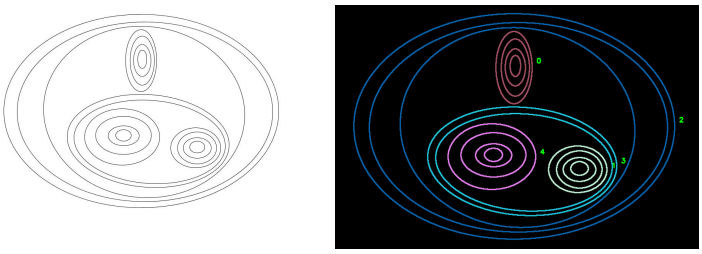
We can then iterate through the contours and create an array of all of the points to be plotted with an array of their complementary elevations so that we can visualise them.

Elevations on multiple peaks

With real-world terrains, there will be many variations in elevations, often with many different peaks. In these scenarios, this iterative process through the lines won't work, as the order of all of the contours in the array will no longer be from largest to smallest and this is no longer the order we need the contours to be in for visualisation.

To be able to plot these, we need to group them into their separate peaks to be able to find each of these groups' elevations independently. This can be achieved through clustering. There are a few different scenarios that determine the number of clusters needed for this to be successful. If there are two peaks stemming from one initial hill, this will require three clusters; one for the starting hill and then two for the new hills coming from it. This scales with the number of hills there are, with each hill containing peaks requiring to be its own cluster and every time a new peak forms, as seen in Figure 2.3.

Once the contours have been clustered, a similar process to the elevation calculation used for the single peaked contours can take place. For each of the clusters, we can check whether its largest contour is contained by another contour from a different cluster. If it is, we can then



((a)) Original Contour Map ((b)) Clustered Contour Map

Figure 2.3: Contour map clustering example

use that contour's elevation as its starting elevation and assign the elevations as before, as seen in code snippet A.1.

K-Means Clustering

To achieve clustering, we used the K-Means algorithm [21], a popular unsupervised technique that groups data points into separate clusters based on their similarity. For this case, each contour is considered a data point, and its features are extracted. These features are normalized and the KMeans algorithm can be applied with a given number of clusters. The algorithm works by iteratively assigning each data point to the nearest cluster centroid and recalculating the centroid of each cluster. This process continues until the centroids no longer move or a maximum number of iterations is reached. The output is an array of clusters, where each cluster contains all the contour arrays that have been grouped together.

The formula for calculating the distance between a data point and a cluster centroid j is:

$$d(x_i, j) = \sqrt{\sum_{k=1}^{n}(x_{ik} - j_k)^2} \tag{2.5}$$

where x_i is the i-th data point, n is the number of features, x_{ik} is the value of the k-th feature of the i-th data point, and j_k is the value of the k-th feature of the centroid of cluster j.

The KMeans algorithm seeks to minimize the within-cluster sum of squares (WCSS) criterion, which is defined as the sum of the squared distances between each data point and the centroid of its assigned cluster:

$$WCSS = \sum_{j=1}^{k} \sum_{x_i \in C_j} d(x_i, j)^2 \tag{2.6}$$

where k is the number of clusters, and C_j is the set of data points assigned to cluster j. The algorithm tries to find the values of the centroids that minimize the WCSS, which can be achieved through the iterative process described above.

One of the issues with using this method for clustering is the requirement for prior knowledge of the number of clusters that need to be used for it to be effective. Too few clusters and the elevations will not be calculated correctly, and too many and it can become more inefficient in calculating them. This becomes an issue when making this an automatic program, as the user

would have to enter a k value for the number of clusters. The Silhouette score [23] can be used to help determine the number of clusters to use with K-Means. The score $s(i)$ for a data point i is calculated by subtracting the mean intra-cluster distance $a(i)$ from the mean nearest-cluster distance $b(i)$. This is then divided by the maximum of $a(i)$ and $b(i)$, as described in the formula:

$$s(i) = \frac{b(i) - a(i)}{\max a(i), b(i)}$$

The silhouette score ranges from -1 to 1, with a higher score suggesting that a data point is well-matched in its cluster. To determine the suggested number of clusters, we can iterate through different cluster numbers and find which one provides the best score.

Hierarchical Clustering

Hierarchical clustering [10] is another form of clustering that we can use to group the contours. One of the main differences between hierarchical and K-Means clustering is that hierarchical can automatically determine the number of clusters it uses by generating a dendrogram and finding a suitable point on there where the clusters split. We begin this with a linkage function, which measures the Euclidian distance between the objects, in this case comparing the different contour lines' features. These are single-linkage (nearest neighbour), complete-linkage (farthest neighbour), average linkage and ward linkage, which is most commonly used. We normalise the features and pass them through this function to form a linkage matrix, which contains information about the hierarchical structure of the clusters. From here, we take the last ten values from the linkage distances, which are the distances between clusters at the final stage of the hierarchy. This allows us to find an elbow point of the resulting curve to determine the optimal number of clusters to use. Some issues still remain here with the automatic generation of clusters, as the program is not guaranteed to be accurate. Figure 2.4 shows the dendrogram generated by using a single linkage method with the clustering found in Figure 2.3 (b).

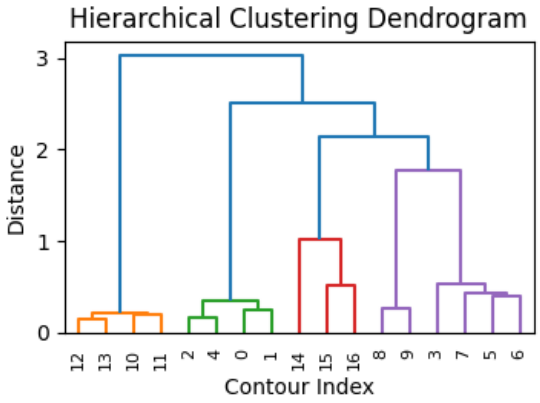


Figure 2.4: Dendrogram from clustered image

2.2.2 Surface Plotting

Once the elevations have been calculated after the clustering, we concatenate all of the contour points and their corresponding elevations into a single-dimensional axis for interpolation. The z values in between all of the points have not yet been found, so they need to be interpolated. Arrays X and Y are generated using a mesh grid the size of the input image size as a base for this surface where `Z[i,j]` corresponds to the elevation at `(X[i,j], Y[i,j])`, which is then interpolated through linear interpolation, due to cubic interpolation causing inaccuracies. We do this by using the `griddata` function from the `scipy.interpolate` module. Figure 3.9 displays a comparison between the two surface plot methods.

Regular grid plotting

One of the methods to plot the surface is through a regular grid method. This creates a surface plot using regular grid data from the X, Y and Z arrays as an input and creates a mesh grid out of them, which is our regular grid. This grid of quadrilaterals is split into two triangles to create a triangulation of the surface, which is what is displayed in the plot. This is using the `plot_surface` function with matplotlib.

Triangulated plotting

We also explore the use of a fully triangulated plot using the `plot_trisurf` function in the matplotlib module. This defines the triangles to be used in the plot more explicitly and uses Delaunay triangulation [1] to generate the triangulated plot. Delaunay triangulation generates a triangulated mesh from a given set of points. It ensures that no point lies within the circumcircle of any triangle of the mesh, making it unique for any given set of points. The output is a more irregular triangulation compared to the regular grid which can provide more flexibility and detail in the plot that is generated. However, it proves to be far more computationally expensive, taking a long time to run and slower with the interaction.

2.3 Graphical User Interface

The interface that the user can interact with is vital for the purpose of this program A.1. Here, they are able to adjust all of the parameters to visualise the contours found in their input images. Whilst the purpose of the project is to make this a fully automated system, issues can be present across different images with varying colour maps or the number of contours for example. In this interface, the user can adjust many settings, including the cluster method, the surface modelling method, the linkage method, and the cluster number as well as see the graphs to help in choosing a suitable number of clusters to use. Importantly, this is where the user also enters the image being read and can see the plot of the visualised contours, so it must be easy to use and flexible to adjust to different images.

3 Results

3.1 Contour Extraction Results

Throughout this section, we will look at the results of the different techniques that have been implemented. One of the issues with a model based on the extraction of existing images is that we don't have any ground truth images to refer to, making it difficult to test how successfully the images are segmented through their thresholds. For this reason, we will aim to use a mix of qualitative and quantitative measures where we can.

To test the thresholding techniques, we will go through a collection of test cases to compare the thresholding method's performances. We will be comparing a global threshold, where a threshold value of 160 was used in the scale [0-255], Otsu's threshold and an adaptive threshold using an adaptive Gaussian algorithm. As the program is based on an automated approach, adjustments in thresholds aren't made to fit suitable images. These tests will be made with a grey-scale and an HSV image, with the majority looking at the value channel, with a few cases that are more suited to the saturation or hue channels. In general within the HSV image, the value channel was the one that most successfully extracted the lines, as this describes the brightness of the image. Therefore, the lines with the biggest difference in pixel intensity will show here. We will then discuss the success of these segmented images in providing a basis for contour detection, and the performance of the contour detection algorithm.

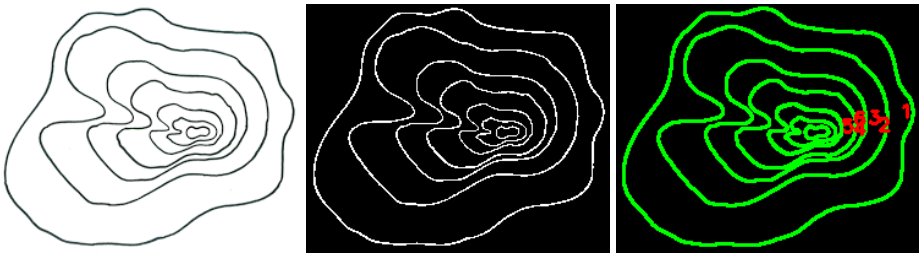
3.1.1 Simple contour lines

These lines are simple maps that are plain and undisturbed with a few contour lines. Here we see a good performance from the global (Figure 3.1 b) and Otsu's thresholding, with them both working perfectly with the grey-scale and value channel images. The adaptive threshold has some issues with areas being included within the binary image that were not wanted, as well as some noise found between lines. This could be due to it looking at the nearby pixel's light intensity to vary its threshold and overcompensating for the area around it. So for this section, Otsu's method and a global threshold are seen to be the most successful.

With the lines successfully segmented from these images, we are able to get good results with the contour detection algorithm. With the lines being thicker and on their own, on these lines it is required to only store every other line, otherwise, there are double the amount of lines detected overall. Figure 3.1 c shows the detected contours from this method with their line numbers displayed next to them.

3.1.2 Coloured maps with additional information

Across these maps, we see a background containing green and white colours. The adaptive threshold is able to extract some of the lines here but does include a lot of white space that

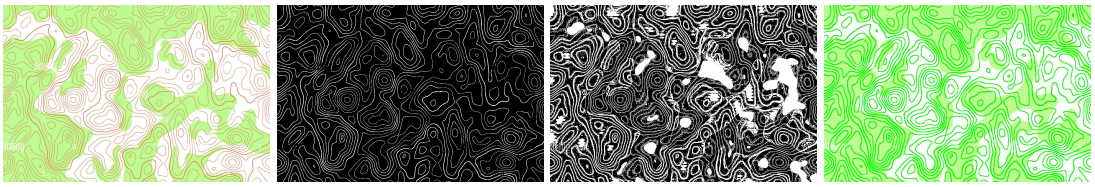


((a)) Original Image ((b)) Global threshold ((c)) Extracted contours

Figure 3.1: Original and segmented images from simple contour example with extracted contours

is picked up also. Other information, such as grid lines, is also extracted here making it an unsuitable method. This means that the threshold it's providing is too low as it is including pixel intensities that we do not want to be extracting. Additionally, it's providing values that are too high in some areas to extract the white space. The global threshold method is an improvement to this, as it picks up more of the lines, however, there are still a few lines that break or thin out too much to be an appropriate technique, meaning the given threshold is not at a value suitable. Otsu's method is the most successful in this area, as it contains almost no line breaks, although it still contains further information from the image, like roads and rivers. This is because in the grey and value channel images, these additional features are appearing at similar values to the contour lines throughout the image, making it harder to differentiate between them.

With Otsu's method providing the best segmentation here with the value channel, we are using this combination to test contour detection. It is somewhat successful, although there are some areas in some images where there are the occasional breaks in the detected contours, or specs that are detected across the image. Because of these problems, the extraction of this type of map wouldn't currently work, as the additional contours that are detected would mean that the feature extraction for clustering wouldn't be able to run properly. Also across these examples, we see the issue with the skipping of lines to avoid double the amount of lines. It means that some lines go 'undetected' and there are more of the extra specs across the image that we don't want.



((a)) Original Image ((b)) Otsu's threshold ((c)) Adaptive threshold ((d)) Extracted contours

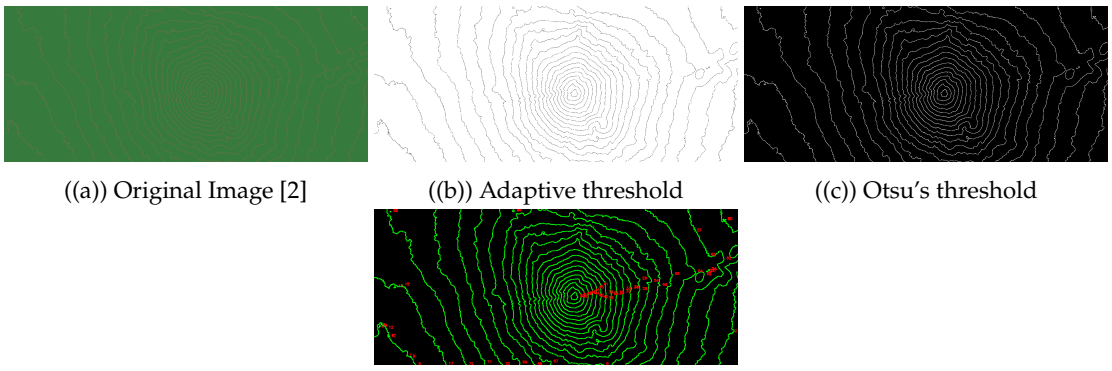
Figure 3.2: Original and segmented images from coloured contour map example with extracted contours

3.1.3 Simple maps with large colour changes

These tests are to see how the threshold methods work with bolder colours that are more difficult to see. In this area, we are also using the hue channel alongside the greyscale and

value channel to test. The hue channel contains colour information, so these images with larger colour changes will show a bigger difference here. The adaptive threshold was quite successful in its grey-scale test, showing an accurate representation of the lines extracted from the image, but was not able to provide a good binary image with the value or hue channels. Due to the fixed threshold of the global technique, it was unable to provide any results for all three of the colour channels, as the pixels across them appeared lower than their given value. Overall here, Otsu's technique had an all-around successful performance, with all three of the colour channels providing a good result. There were some issues in all of the successful results, however, with all of them but Otsu's extraction with the hue channel providing an inverse binary image of the lines. This likely occurs because the different channels may show the line as a lighter shade to the background when typically it is the other way round. This will then inverse the thresholding process, providing an inverse image.

With most of the segmentation results here providing an inverse binary issue, it can cause some issues with the contour extraction. The lines are still extracted appropriately, however, it also then contains the entire border of the image, which would interfere with the methods used for visualisation. It also means that the line-skipping process is disrupted, however, the image we use here does not provide duplicate lines for the border, suggesting that some maps that contain thinner contour lines may not require this additional process.



((a)) Original Image [2] ((b)) Adaptive threshold ((c)) Otsu's threshold

((d)) Extracted contours with text

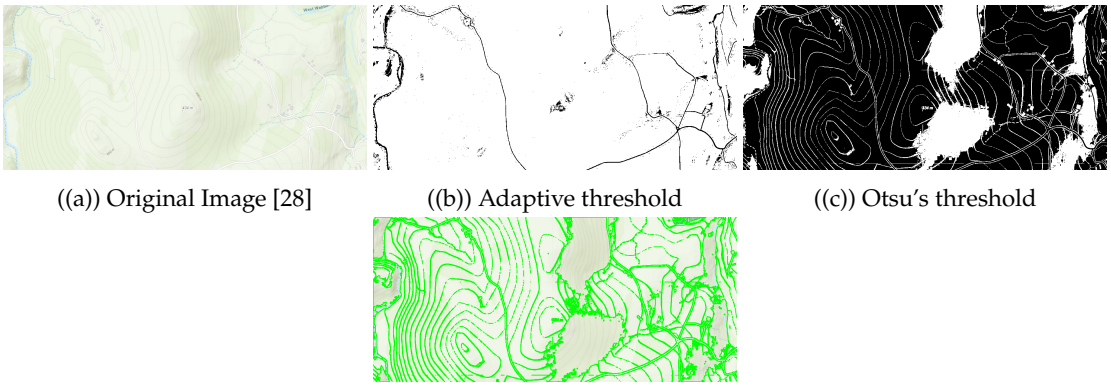
Figure 3.3: Original and segmented images from a dual colour contour map example with extracted contours

3.1.4 Coloured maps with faint contour lines

The maps included in this test are from existing maps containing contour lines. They show maps with fainter colours where the contour lines are less visible across the landscape. A similar issue seen before with the global threshold is found here where nothing appears due to all of the pixels being lower than the fixed threshold provided. The adaptive threshold doesn't show many line details across both colour channels tested here, with a few lines visible but a lot of noise and black or white space where there shouldn't be. Once again, we see Otsu's threshold providing the best results here as some line detail is visible. However, we still see a lot of white space and lines missing. This suggests that the setup for the thresholds as they are

is not suitable for maps with these conditions, and further pre-processing would be required to be able to find the lines in these images. We can conclude that having lines this similar in colour to the background will provide challenges in thresholding as it is difficult to provide threshold values that successfully differentiate between them.

This is apparent in the extraction process on these images, with some lines found but there is a lot of white space that gets in the way of the lines as well as additional information from the image. These images then become unusable in their current state as elevation data cannot be calculated from them.



((a)) Original Image [28] ((b)) Adaptive threshold ((c)) Otsu's threshold

((d)) Extracted contours

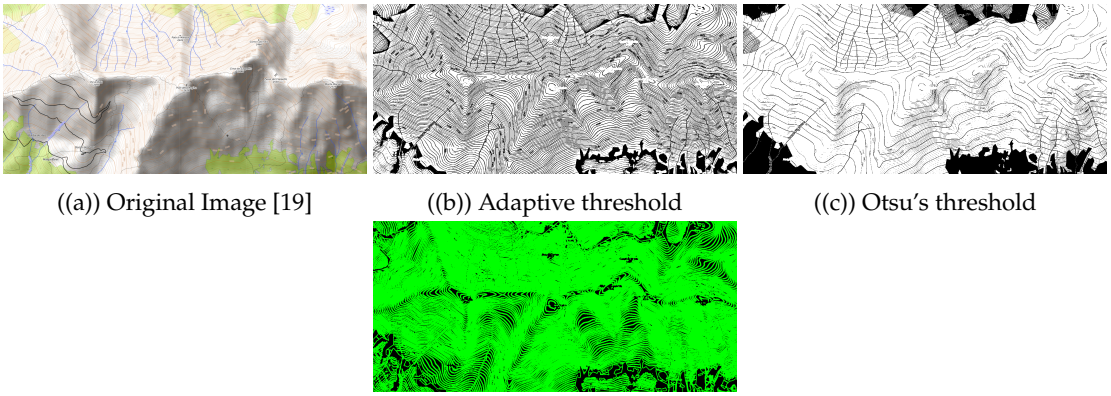
Figure 3.4: Original and segmented images from contour map with faint lines example with extracted contours

3.1.5 Detailed maps with more complex lines

These maps contain a more complex collection of lines, with them being seen closer together and in a greater variety of shapes and thicknesses. For these tests, we are also including the saturation channel as it provided some clearer results. The saturation channel provides the colour intensities, so a map with these bright changes can provide values easier to threshold through here. In comparison to the other methods, the adaptive threshold has some more success here as it is able to extract more lines from the image. Some lines are found with quite a lot of black-and-white space with the grey-scale and value channels, but with the saturation channel, it was able to receive the majority of the lines with a few areas of black space in the inverse image. The global threshold was unable to extract much information, as the lines were seen to be more grey resulting in lots of the image being lower than the threshold provided. Otsu's threshold had some success, and the image without additional information was successful, however, in a more complex situation with shading for the hills, it did not perform as well as the adaptive approach. This suggests that maps containing shading to represent the mountains may be more successful with an adaptive threshold.

Here there was some success in extracting the topographic lines from the segmented image with the adaptive threshold. A lot of line data is collected here, however with the density of the image it proves to be challenging to exclusively extract the data we require. Additionally with

the amount of lines seen, line breaks and overlapping issues are seen throughout the image, further providing problems with the extraction process across this type of image.



((a)) Original Image [19] ((b)) Adaptive threshold ((c)) Otsu's threshold

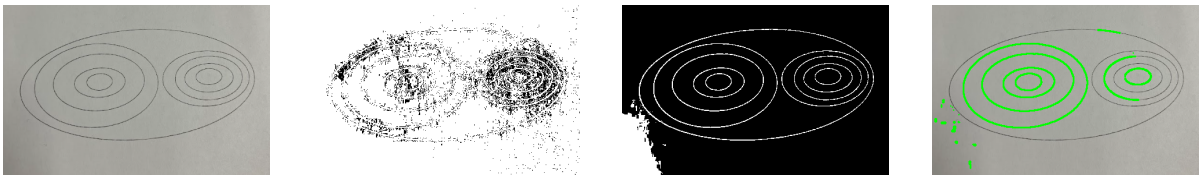
((d)) Extracted contours

Figure 3.5: Original and segmented images from complex contour map example with extracted contours

3.1.6 Photographed contour maps with different lighting conditions

Here we are testing how well the thresholding techniques perform with images of contour lines that have been taken from a camera, which results in different lighting conditions across the lines. It was expected that the adaptive threshold would be the most successful as this is the area an adaptive threshold is typically used, but in these tests, it provided a lot of noise across all cases. Both global and Otsu's thresholds provided similar results, with a degree of success in picking up the lines but included white areas where the shadows were so still weren't quite appropriate for an automated use case. Even so, Otsu's method provided less white space and more lines being connected that are further from the camera.

Aside from the issue with the white space being extracted, we can see that some of Otsu's method provides results that appear to have fully connected lines. As we can see from Figure 3.6 d using the skipping method, some lines that would be full without the skipping still aren't. This means that instead of one continuous line that is found, a collection of lines used to make up the shape are found. This will provide problems in the visualisation process as we will not be able to find features such as the area or centroids that are required for clustering and elevation calculating.



((a)) Original Image [28] ((b)) Adaptive threshold ((c)) Otsu's threshold ((d)) Extracted contours

Figure 3.6: Original and segmented images from photographed example with extracted contours

3.2 Clustering Results

To test the clustering, we can use a combination of qualitative and quantitative measures to analyse the best method and number of clusters for a test example. For more simple contour maps the number of clusters can be determined through a visual analysis of the image. We can use this knowledge to judge the success of the algorithms in their clustering, as well as quantitative metrics such as the Silhouette score [23] and the Calinksi-Harabasz index [6]. The Silhouette coefficient measures the similarity between each data point and its assigned cluster to other clusters. It produces a value in the range from -1 to 1 with 1 indicating a data point very similar to its own cluster and dissimilar to others and -1 providing the opposite. A higher score suggests a good separation between clusters, so the data points within each cluster are more similar than the other data points in other clusters. The Calinksi-Harabasz index looks at how well-separated clusters are from each other. The variance within each cluster is compared to the variance with other clusters, so a higher score indicates more well-defined and separated clusters. In some cases, this is not the best measure to use as it can favour clusters with a large number of points. This means that it is suggested that a higher score here indicates a better clustering group. For this reason, these metrics should be used together to determine the success of the clusters.

Figure 2.3 shows a basic example contour map that, from visual analysis, should be grouped into 5 clusters. To compare algorithms, we have run the automated cluster calculations with the four linkage methods for hierarchical clustering. This is being compared to the suggested K-Means cluster number through silhouette coefficient and comparing this to other values of K. The results are displayed in Table 3.1.

Table 3.1: Cluster evaluation metrics on known cluster number

n clusters	**Method**	**Silhouette**	**Calinski-Harabasz**
4	Ward	0.73	49.12
5	Single	0.79	121.14
3	Complete	0.58	18.54
3	Average	0.58	18.54
5	K-Means (Auto)	0.79	121.14
7	K-Means (k=7)	0.62	189.08
10	K-Means (k=10)	0.44	215.96
15	K-Means (k=15)	0.11	466.83

From the results, we can see that the single linkage method and the automated K-Means clustering methods have provided the best results with the suggested 5 clusters, which is what we predicted. We can see this with the highest silhouette coefficient with a high Calinksi-Harabasz index too when compared to other results. The complete and average linkage methods perform the lowest here, suggesting that they have produced poorly separated clusters. This could be because the average linkage looks at the average distance between all points in two clusters. Complete linkage looks at the distance between the furthest points in two clusters, so also in these cases with contour lines all with different features, this doesn't work properly. We can also see from here that the automatically suggested number of clusters for K-Means

has performed better than the manual tests of 7, 10 and 15 clusters. The increasing Calinski-Harabasz score is due to the number of clusters increasing, as this will lead to a decrease in the within-cluster sum of squares. However, as this increase is happening as the Silhouette coefficient decreases, it can be inferred that the quality of the clustering is lower as the clusters become less well-defined.

In terms of visualisation on the surface plot, having more clusters than necessary will not affect the plot. In this case, it will just reduce the efficiency as more checks are made to see whether the cluster is contained within another. This is due to a greater number of clusters having to be compared in finding the elevation than what we would see with a lower and more accurate number of clusters. However, if the number of clusters is too low we will then have inaccurate elevation data, so it is best for the cluster number to be overestimated for an improved visualisation.

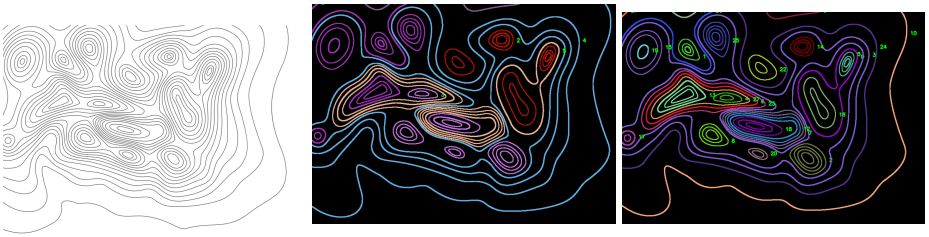
Table 3.2: Cluster evaluation metrics on unknown cluster number

n clusters	**Method**	**Silhouette**	**Calinski-Harabasz**
6	Ward	0.39	34.26
2	Single	0.55	7.92
2	Complete	0.46	9.01
3	Average	0.45	14.21
28	K-Means (Auto)	0.56	127.23
40	K-Means (k=49)	0.45	379.54
55	K-Means (k=55)	0.23	2362.86
63	K-Means (k=63)	0.03	24131.68

Figure 3.7 shows a contour map that contains a more detailed collection of contour lines. This map is less evident on what the correct number of contours should be due to the variation of the lines and hills. Table 3.2 shows the results of the tests, suggesting that the automatic K-means clusters determined by silhouette score are the best number of clusters with 28. This is to be expected as this cluster number is determined by comparing all silhouette scores. By just looking at the silhouette score, it might be suggested that the single linkage method also provides a good estimate for clustering, however, two clusters are particularly inappropriate for this example. The Calinksi-Harabasz index here is quite low with a value of 7.92 which tells us that it's not appropriate. Again, we see an increase in this score as the number of clusters increases, but the decreasing silhouette coefficient suggests that these are inappropriate cluster numbers. Figure 3.8 shows us how too few clusters provide a very inaccurate plot, and as we get past the suggested number of clusters, little difference is made although with this image with some unclosed lines, the bottom of the plot is too flat with the larger number as the elevations are not calculated together, making no starting elevations.

3.2.1 Surface Plot Comparison

Similar to the contour extraction analysis, it is challenging to provide a quantitative analysis of this process as we don't have any original elevation data to compare to when we produce these surface plots. Because of this, we will make a visual analysis comparing the usage of a

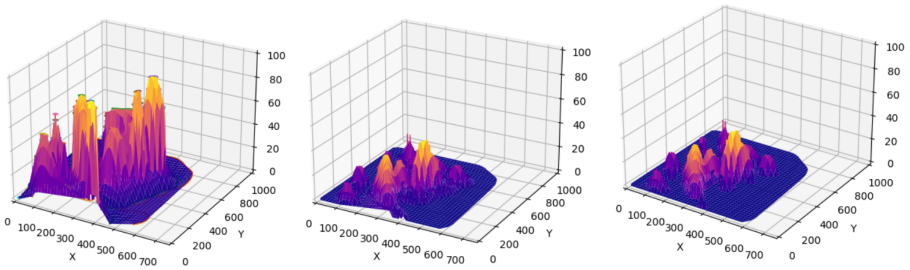


((a)) Original Contour Map

((b)) Too few clusters from ward linkage

((c)) K=28 determined by Silhouette Score

Figure 3.7: Automated contour clusters



((a)) Ward 6 clusters

((b)) k=28

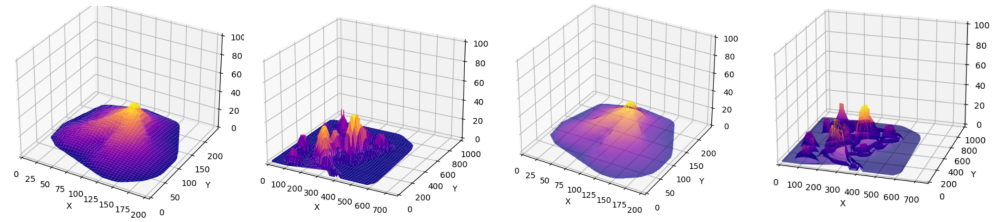
((c)) k=50

Figure 3.8: Plots of different numbers of clusters

regular grid plot to a triangulated plot, and discuss their performance.

Figure 3.9 shows a comparison between plots with a regular grid plot and a triangulated plot. We can see that on the plot made with a regular grid, the quadrilaterals across the surface represent the shape from the contours well, however, they do appear quite jagged in some areas, especially in figure x. This is less apparent with the triangulated plot, which presents a much smoother surface as it contains a much larger number of vertices.

While the triangulated plot provides a more detailed DEM, this program is intended for a user to interact with to visualise a contour map. They should be able to easily plot the graph and interact with it. The problem that comes with the triangulated plot is that so many vertices are generated for the surface, it becomes extremely slow to run. The plot takes longer to generate to be displayed, and then from there is very slow to interact with, especially with more detailed contour maps. For this reason, a compromise has to be made for a less detailed plot that a user can interact with, over a smoother and clearer one that is challenging to run.



((a)) Regular grid surface plot a

((b)) Regular grid surface plot b

((c)) Triangulated surface plot a

((d)) Triangulated surface plot b

Figure 3.9: Suface plot examples

4 Conclusion

4.1 Discussion

In this project, we have explored the use of different threshold techniques for segmentation, the success of border following algorithms, different clustering algorithms and surface plotting. We have found that overall, Otsu's automatic threshold had the best results in our image segmentation, with the most success often being found with the value channel. For it to be fully successful would need further adjustments in parameters to be made.

On simple images, the contour detection algorithm we used was successful and was able to provide the points in the contour lines to be able to calculate elevations to be visualised. This was beneficial over other methods we discussed as we were able to effectively extract the features and iterate over the lines to find the elevations we needed. There were some problems with the algorithm though, largely with it sometimes storing the points for both borders of a thick line. We aimed to fix this by iterating over the array of contours to only store one of these borders, which worked on simple models but did not scale well to more complex examples.

The use of a silhouette score was quite successful in automatic suggestion for cluster numbers, and the use of clustering worked to group hills based on the line features. This provided us with the ability to calculate the elevations for different peaks throughout the image. We didn't see the automatic cluster calculations work as well for hierarchical clustering, largely because they would find a suggested cluster number from the dendrogram but would often cut it too early in the image.

We found that the surface plots were able to provide suitable visualisations of the data that users could easily interact with. The regular grid method provided a less detailed plot, which still showed a good demonstration of the topography that the user could interact with to interpret easily. The triangulated plot produced a smoother and more accurate visualisation, however, it was often too computationally expensive to justify using.

4.2 Future work

The main idea of this project was to provide an automated program to visualise contour lines extracted from an image. In many ways, this system was provided, however on the whole this is not an automatic program. To be able to provide this, a few areas can be improved.

The segmentation was somewhat successful in its ability to extract the lines, but only in simpler images. For the application in a map, where most contour lines would be found, this aim is still not met as it is disrupted by shading in images and other map features. We saw that the thresholding would often either lead to line breaks or additional map features which would then be detected by the contour detection algorithm and disrupt the elevation calculations. Future work on this could involve additional methods to be able to identify and

remove these kinds of features from a thresholded image or identify the contour lines through methods separate from a segmentation approach. This could involve classification approaches [24]. Additionally, it would be important to provide features that allow the reconstruction of broken lines [17, 14], as this is a large issue throughout the contour detection process we have seen in this project. The contour detection algorithm we are using also has the problem seen with providing two lines for each border. We would look to improve this by providing a single unique line for each detected one by comparing similarities in the lines. The two lines for a border will follow the same shape, so we could implement an algorithm to compare these features and determine if they are both from the same line.

Other future areas to improve on would include the elevation calculations based on the contour lines. This would have less of a focus on clustering, as this process groups them together but still largely compares across the image to find if a line is contained by another. Further utilisation of the hierarchy of contours detected could be used here to improve this, or a more efficient algorithm to determine whether lines are contained by others to stack their elevations in a search. Methods such as using a 'steepest slope' [18] approach or Voronoi diagrams [8] could be explored here.

Improvements could be made to the surface plots, as in their current form they either don't provide as much detail or are too detailed to run properly. Other graphical approaches such as using OpenGL [11] to manually automate the creation of a 3D model of the DEM could provide more efficient and accurate visualisations that could still be interacted with. An approach like this could also allow the model to be exported to be used in other applications so users could further interact with it. The system's hardware could be better optimised in this approach for faster responses to it, as well as adjustments to be made for the number of vertices used within a visualisation.

4.3 Conclusion

Overall, we have developed a program to allow users to input an image of a contour map and interact with the DEM produced by it. This has been able to provide users context to different topography that may not be as easily visualised in a 2D form. The approach used to do this work is to produce a segmented image to extract the contours, which can then be clustered to find elevations to plot. The main improvement to be made in all areas is further development towards the direction of automation, and to scale the program to work more effectively in more complex images, such as containing more contour lines or additional details.

References

[1] Franz Aurenhammer, Rolf Klein **and** Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013, **page** 348. ISBN: 978-981-4508-29-8. DOI: 10.1142/8685.

[2] *AxisMap*. https://contours.axismaps.com/. accessed 2023.

[3] Susanne Bleisch **and** Stephan Nebiker. "Connected 2D and 3D visualizations for the interactive exploration of spatial information". **in**2008.

[4] Derek Bradley **and** Gerhard Roth. "Adaptive Thresholding using the Integral Image". **in***Journal of Graphics Tools*: 12.2 (2007), **pages** 13–21. DOI: 10.1080/2151237X.2007.10129236. eprint: https://doi.org/10.1080/2151237X.2007.10129236. URL: https://doi.org/10.1080/2151237X.2007.10129236.

[5] G. Bradski. "The OpenCV Library". **in***Dr. Dobb's Journal of Software Tools*: 25.11 (2000), **pages** 120–126.

[6] Tadeusz Calinski **and** Jerzy Harabasz. "Dendrite method: a new clustering method for large data sets". **in***Advances in data analysis and classification*: 1.1 (1974), **pages** 1–22.

[7] Steve Dübel **andothers**. "2D and 3D presentation of spatial data: A systematic review". **in***2014 IEEE VIS International Workshop on 3DVis (3DVis)*: 2014, **pages** 11–18. DOI: 10.1109/3DVis.2014.7160094.

[8] Qingsong Fan **and** Peng Hu. "DEM generation from contour lines based on Voronoi diagram". **in***Geoinformatics 2007: Geospatial Information Science*: **byeditor**Jingming Chen **and** Yingxia Pu. **volume** 6753. International Society for Optics **and** Photonics. SPIE, 2007, **page** 675320. DOI: 10.1117/12.761876. URL: https://doi.org/10.1117/12.761876.

[9] Gazi **andothers**. "Contour Line Stylization to Visualize Multivariate Information". **in**2021.

[10] Stephen C Johnson. "Hierarchical clustering schemes". **in***Psychometrika*: 32.3 (1967), **pages** 241–254.

[11] Khronos Group. *OpenGL*. Accessed: May 2, 2023. 2021. URL: https://www.khronos.org/opengl/.

[12] Sang Uk Lee, Seok Yoon Chung **and** Rae Hong Park. "A comparative performance study of several global thresholding techniques for segmentation". **in***Computer Vision, Graphics, and Image Processing*: 52.2 (1990), **pages** 171–190. ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(90)90053-X. URL: https://www.sciencedirect.com/science/article/pii/0734189X9090053X.

[13] Stefan Leyk **and** Ruedi Boesch. "Colors of the past: Color image segmentation in historical topographic maps based on homogeneity". **in***GeoInformatica*: 14 (**january** 2010), **pages** 1–21. DOI: 10.1007/s10707-008-0074-z.

[14] Chengming Li **andothers**. "A Reconstruction Method for Broken Contour Lines Based on Similar Contours". English. **in***ISPRS International Journal of Geo-Information*: 8.1 (2019), **page** 8. URL: https://uoelibrary.idm.oclc.org/login?url=https://www.proquest.com/scholarly-journals/reconstruction-method-broken-contour-lines-based/docview/2548556975/se-2.

[15] Ko Ko Lwin, Ronald C. Estoque **and** Yuji Murayama. "Data Collection, Processing, and Applications for Geospatial Analysis". **in***Progress in Geospatial Analysis*: **byeditor**Yuji Murayama. Tokyo: Springer Japan, 2012, **pages** 29–48. ISBN: 978-4-431-54000-7. DOI: 10.1007/978-4-431-54000-7_3. URL: https://doi.org/10.1007/978-4-431-54000-7_3.

[16] Rajesh Kumar Goyal Mahesh Kumar Gaur **and** J.S. Chauhan. "Application of Geospatial Information System for Mapping and Management of Natural Resources – A Review". **in**(2020): **pages** 163–169. DOI: 10.18805/BKAP183. URL: https://arccjournals.com/journal/bhartiya-krishi-anusandhan-patrika/BKAP183.

[17] Shriram Oka, Akash Garg **and** Koshy Varghese. "Vectorization of contour lines from scanned topographic maps". **in***Automation in Construction*: 22 (2012). Planning Future Cities-Selected papers from the 2010 eCAADe Conference, **pages** 192–202. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2011.06.017. URL: https://www.sciencedirect.com/science/article/pii/S0926580511001294.

[18] Prima Oky Dicky Ardiansyah **and** Ryuzo Yokoyama. "DEM generation method from contour lines based on the steepest slope segment chain and a monotone interpolation function". **in***ISPRS Journal of Photogrammetry and Remote Sensing*: 57.1 (2002). Geomatics in Mountainous Areas – The International Year of the Mountains, 2002, **pages** 86–101. ISSN: 0924-2716. DOI: https://doi.org/10.1016/S0924-2716(02)00117-X. URL: https://www.sciencedirect.com/science/article/pii/S092427160200117X.

[19] *OpenTopoMap*. https://opentopomap.org/. accessed 2023.

[20] Nobuyuki Otsu. "A threshold selection method from gray-level histograms". **in***IEEE transactions on systems, man, and cybernetics*: 9.1 (1979), **pages** 62–66.

[21] F. Pedregosa **andothers**. *scikit-learn: Machine Learning in Python*. JMLR 12, pp. 2825-2830. 2011. URL: http://scikit-learn.org.

[22] Wenjing Ran **andothers**. "Raster Map Line Element Extraction Method Based on Improved U-Net Network". **in***ISPRS International Journal of Geo-Information*: 11.8 (2022). ISSN: 2220-9964. DOI: 10.3390/ijgi11080439. URL: https://www.mdpi.com/2220-9964/11/8/439.

[23] Peter J Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis". **in***Journal of computational and applied mathematics*: 20 (1987), **pages** 53–65.

[24] Juan J. Ruiz **andothers**. "Digital map conflation: a review of the process and a proposal for classification". **in***International Journal of Geographical Information Science*: 25.9 (2011), **pages** 1439–1466. DOI: 10.1080/13658816.2010.519707. URL: https://doi.org/10.1080/13658816.2010.519707.

[25] Loh Mun San **andothers**. "Extracting contour lines from scanned topographic maps". **in***Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*: (2004), **pages** 187–192.

[26] Satoshi Suzuki **and** KeiichiA be. "Topological structural analysis of digitized binary images by border following". **in***Computer Vision, Graphics, and Image Processing*: 30.1 (1985), **pages** 32–46. ISSN: 0734-189X. DOI: https://doi.org/10.1016/0734-189X(85)90016-7. URL: https://www.sciencedirect.com/science/article/pii/0734189X85900167.

[27] Hind Taud, Jean-François Parrot **and** Roman Alvarez. "DEM generation by contour line dilation". **in***Computers Geosciences*: 25.7 (1999), **pages** 775–783. ISSN: 0098-3004. DOI: https://doi.org/10.1016/S0098-3004(99)00019-9. URL: https://www.sciencedirect.com/science/article/pii/S0098300499000199.

[28] *Topographic Map*. https://en-gb.topographic-map.com/. accessed 2023.

[29] T. Y. Zhang **and** C. Y. Suen. "A Fast Parallel Algorithm for Thinning Digital Patterns". **in***Commun. ACM*: 27.3 (**march** 1984), **pages** 236–239. ISSN: 0001-0782. DOI: 10.1145/357994.358023. URL: https://doi.org/10.1145/357994.358023.

A

Appendix

```
1
2  def plot_clusters(image, clusters, contour_interval, ax, trisurf=False,
       show_contours=False, show_surface=True):
3
4  """ Previous code """
5
6          # Find the start elevation to be able to calculate contour elevations
7          start_elevation = find_start_elevation(clusters, cluster[len(cluster)-1]) *
       contour_interval
8
9          # Calculate elevations for the contours using the given contour interval
10         contour_elevations = [(x * contour_interval + start_elevation) for x in
       range(len(cluster))]
11
12 """ Continued code """
13
14 def find_start_elevation(clusters, startContour):
15     start_elevation = 0
16
17     # Squeeze contour to 1D array
18     contourPoints = np.squeeze(np.array(startContour)).astype(int)
19
20     # Iterate through other clusters
21     for i, cluster in enumerate(clusters):
22         for j, contour in enumerate(cluster):
23             # Only apply if other contour is larger
24             if len(np.squeeze(np.array(contour))) < len(contourPoints):
25                 continue
26             else:
27                 # Check if the provided contour is contained by the other
28                 if is_inside_contour(contourPoints, contour):
29                     start_elevation += 1
30
31     return start_elevation
32
33
34 def is_inside_contour(innerContour, outerContour):
35
36     # Convert arrays for point polygon test
37     outerPoints = np.array(outerContour)
38     innerPoints = np.squeeze(np.array(innerContour)).astype(int)
39
40     # Find distances between points to find if the contour is contained by the given
        one
```

```
    distances = [cv2.pointPolygonTest(outerPoints, (int(point[0]), int(point[1])),
    False) for point in innerPoints]

    # Check if the contour is contained
    if all(distance > 0 for distance in distances):
        return True
    else:
        return False
```

Code A.1: Clustered elevation calculation

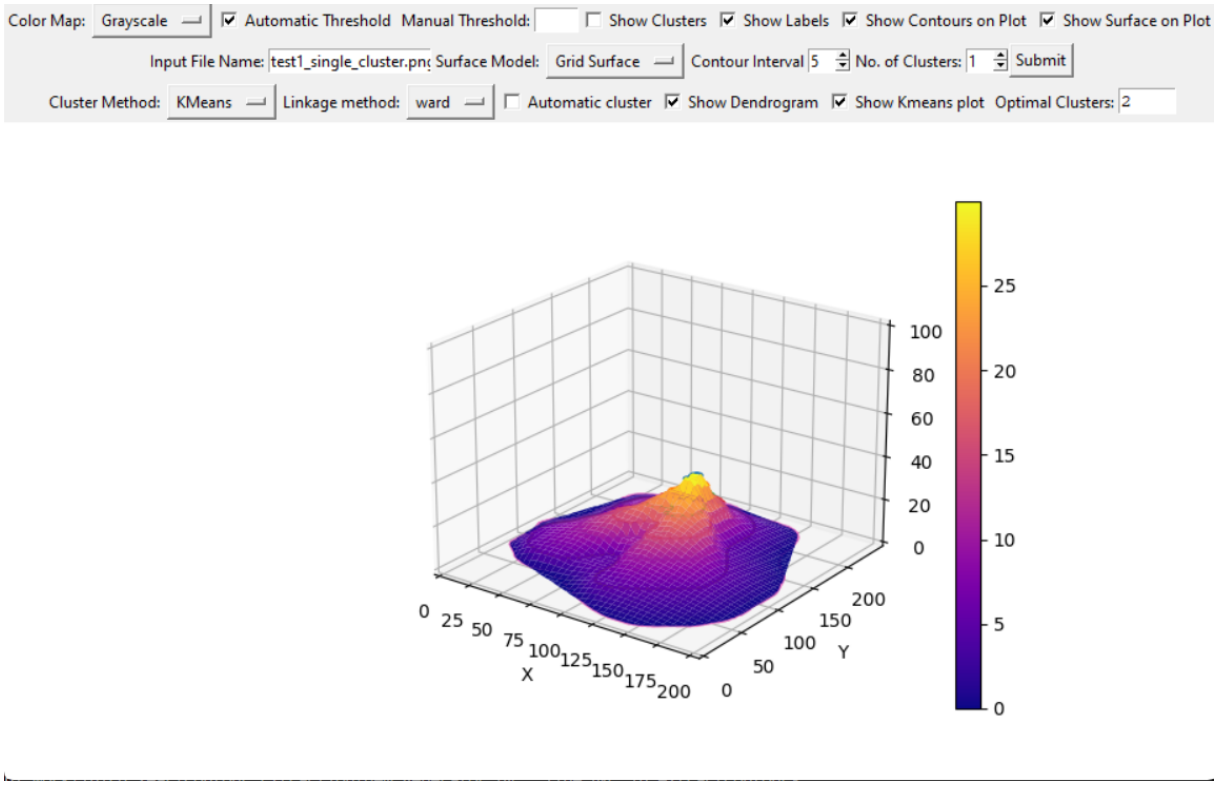


Figure A.1: GUI

Acknowledgments

I extend my deepest appreciation to my supervisor, Dr. Sareh Rowlands, for her incredible support and guidance throughout this project. I am also grateful to my former tutor, Dr. Jacqueline Christmas, for helping me develop my dissertation topic. My heartfelt thanks go to my family and friends for their unwavering support, and to Bryony for covering my shift when I needed it most.