Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Contour map representation through augmented reality

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Borja Ruiz Torres

DIRECTOR: Josep M. Yúfera Gómez

DATE: June 21 th 2012

Title: Contour map representation through augmented reality

Author: Borja Ruiz Torres

Director: Josep M. Yúfera Gómez

Date: June, 21th 2012

Overview

This document proposes a set of processes necessary to convert a 2D topographic map into 3D information. First of all, the image information has to be processed in order to be translated into a digital format. This processing is formed by some sub-steps: image processing, acquisition of the image information (contour lines), contour line processing and number recognition (elevation numbers). The output of these processes could be displayed using either 2D or 3D data. The first can be represented through the same contour map but showing the elevation values by means of a colour range. Otherwise, the relief will be illustrated using a 3D mesh, which can be directly displayed or employing marker-based augmented reality.

Some of the main topics included in this document are: image processing, optical character recognition and augmented reality.

INDEX

INTRODUCTION

Time ago the only way to store terrain information was through the use of topographic maps. As a consequence, nowadays there exists a huge quantity of them in the public domain. With technology evolution these maps have been pushed right into the background, because new systems permit to store terrain information more accurately. Furthermore, this information is also easier to manage and represent.

The main objective of this project is to convert the terrain information included in a contour map in a currently used terrain information format and, finally, to display through Augmented Reality the 3D surface of the terrain. Either the software application presented in this project and the ones in charge of acquire topographic information permit port terrain data to digital formats. This way, paper maps are gradually less used and it involves a less waste of paper to represent topographic data, which is beneficial for the environment.

This document describes the steps to achieve our purpose. It is organized into seven chapters.

Chapter 1 briefly introduces some theory about contour maps and the information storage in geographic information systems (GIS), the current technology that has substituted the first ones.

From chapters 2 to 5 there are contained the processes in charge of achieving the main objective of this project. Each of them presents the same structure: a description of the issue in question and the solution to solve it. The first step has always been to search related work to the issue in literature and check if these solutions fitted our particular scenario. In case none of them was useful an own solution has been proposed.

Chapter 2 explains how the contour map image is processed in order to acquire the terrain information (elevation numbers and contour lines).

Chapter 3 illustrates the processes taking place to get the elevation value for each pixel in the image.

Chapter 4 determines the way to adapt number information to be able to recognize the number values from the previously detected number areas.

Chapter 5 shows different ways implemented to display the terrain information. It describes the different application of augmented reality and how it is used in this project.

Chapter 6 analyses the computing time and success rate in some contour maps to determine the actual functionality and efficiency of the software.

Chapter 7 is a guideline to know how the designed software application works.

CHAPTER 1. TOPOGRAPHIC INFORMATION

1.1. Contour Maps

A contour map is a representation of the Earth surface, which shows the relief variations of a terrain (Figure 1.1). These variations are represented through contour lines. Apart from contour lines, other geographical factors are included, as vegetation, hydrography, towns… all of them with their correspondent colour and symbol. Contour maps have a scale equal or inferior than 1:10.000 (1 cm on the map is 100 m in the real space).

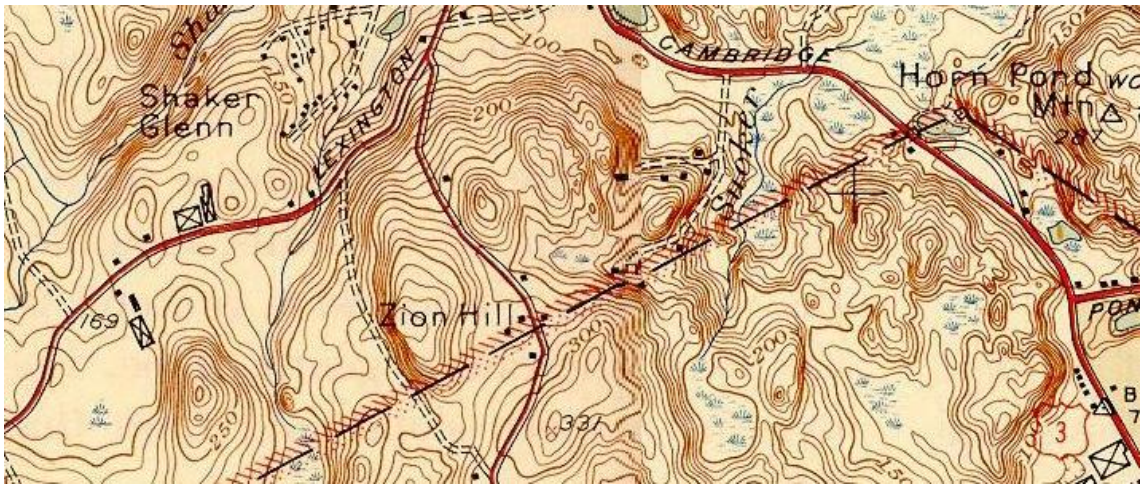


Figure 1.1 Scanned topographic map

A contour line is a drawn line, which joins points with the same high. Depending on what kind of terrain is represented, the contour line can be drawn in a different colour: brown or orange tones for ground and blue for glaciers, oceanic and lacustrine depths. In topographic maps, the surfaces between two contour lines are usually filled with determinate conventional inks (hypsometric ink). Hypsometric inks are used to represent in a symbolical way the relief of a terrain by applying different colours to the different heights. For example, dark green for depressions under sea level, light green for medium heights, browns for high altitudes and red or violet for the highest peaks on Earth.

There are some rules that contour lines obey:

- All contour lines are closed. Although contour lines that run into the edge of the map do not seem to close on themselves, if you got the adjacent map you would see that this contour line would close on itself.

- Every point on a contour line represents the same height.

- Contour lines can never cross one another, because each line represents one height and two different elevations are not possible in the

same point. There are two exceptions, when there is an overhanging cliff or a cave.

- The space between contour lines depends on slope characteristics. The closer the contour lines are between them, the steeper is the slope. So, if contour lines are close together it represents a steep slope. If contour lines are more separated it represents a gentle slope.

- A series of closed contours one inside the other represents a hill. Depressions are marked with short, straight lines inside the contour line that point down slope to the depression.

- When contour lines are closed to a valley or a stream, they form a V shape. In case a stream or river flows through the valley, the V points upstream.

1.2. Geographic Information Systems

A Geographic Information Systems (GIS) is a computer system able to capture, store, handle, analyse and display geographically information. It is composed by hardware, software and geographic data.

This project, in relation to GIS, will only be focused on the different methods to store topographic information, which enable an easier management of the data. That is why, today, they have substituted paper contour maps. They are principally divided in two groups: raster and vector data.

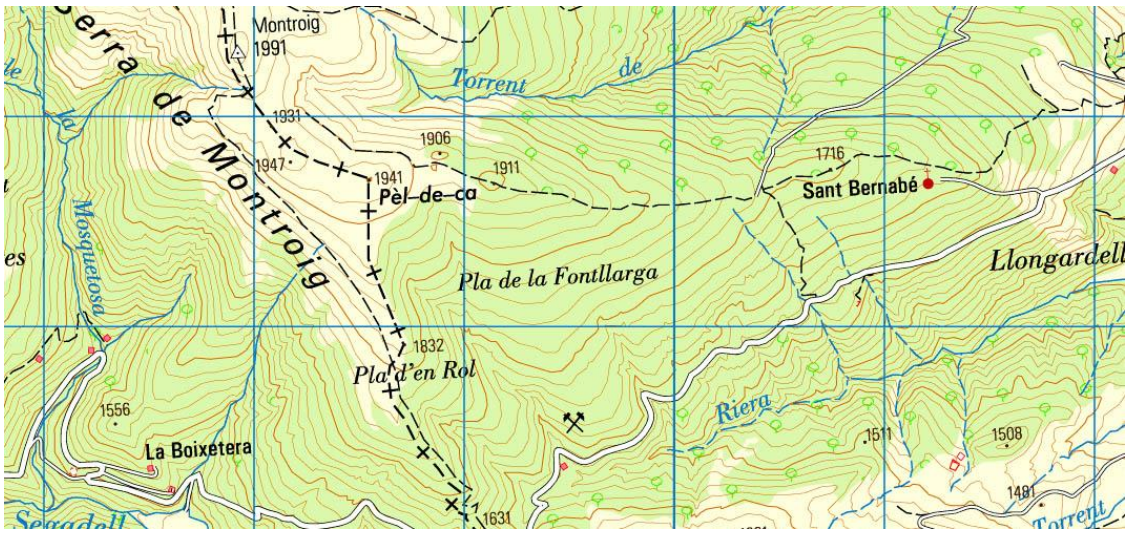


Figure 1.2 Digitalized topographic map

Normally, raster data (Figure 1.2) allows us to store the same information included in contour maps but in digital formats. In these cases, the common ways to acquire the data are such as scanned paper maps and aerial

photography. Preserve the terrain properties as whether a land is being farmed or if an area is urban or not, is the main objective in this raster information class. On the other hand, it obtains a lower accuracy in terms of location and height. Typically, the file formats used for saving this specific raster information are: JPEG, TIFF, etc.

DTM (Digital Terrain Model) files are other kind of raster data. Unlike the previous ones, its main goal consists of saving height measurements with high accuracy; otherwise, the land properties are not considered. For that, a height grid is stored in an ASCII file (Figure 1.3). Through these files and the appropriate software, all type of terrain characteristics can be deducted: slopes, hillsides, hollows, views and hidden areas, etc. Furthermore, synthetic contour lines can be generated by means of data processing.

```
NCOLS 5841
NROWS 4081
XLLCENTER 901600
YLLCENTER 4586800
CELLSIZE 5
NODATA_VALUE -999
251.593 251.702 251.896 252.339 252.838 253.382 253.96 254.544 255.126 255.707
256.29 256.813 256.997 257.231 257.688 258.164 258.733 259.371 260.177 261.013
261.697 261.316 260.546 260.094 259.739 259.375 258.988 258.546 257.992 257.615
```

Figure 1.3 DTM file

These files are made up of a header and elevation data. The header contains variables that determine the area extension and the data position respect from the geographic coordinates. In particular, these variables are:

- NCOLS: number of columns in the grid.
- NROWS: number of rows in the grid.
- XLLCENTER and YLLCENTER: specify the geographic coordinates at the midpoint of the grid.
- CELLSIZE: refers to the resolution of the grid.
- NODATA_VALUE: represents missing data.

On the other hand, vector data is defined using coordinate pairs relative to a cartographic system. Through a coordinate pair and the elevation value a point can be managed. A point is the basic entity used on this model. The main idea consists of grouping points to obtain lines and grouping lines to obtain polygons.

Vector data permits us to handle and manage easily the ground surface topography; hereby many GIS applications are based on this technology. The

strengths of this model lie in the accuracy of the data. They are able to manage a wide range of information as for example, ground characteristics and water properties. Some format files using topographic data in vector technology are: Drawing eXchange File (DXF), Autocad Drawing Files (DWG), etc.

CHAPTER 2. CONTOUR LINE ACQUISITION SYSTEM

This chapter describes the different processes carried out to adapt the image and finally acquire its information. First, a filter is used to obtain a binary image. When the filtering process has finished, the pixel density is computed to detect number areas. Then, the thinning process is carried out over the output image from the filtering process; as a result, the terrain information is one-pixel thickness. On this image, number areas previously detected are removed. As a consequence, some gaps are generated. Reconstruction process is used in order to complete contour lines. Finally, an acquisition step is needed to obtain the pixels that make up the contour lines.

Next diagram schematically illustrates the workflow of this chapter. The symbol ⊖ means that the outputs of the previous steps (Number detection and Thinning) are subtracted. In other words, detected number areas are removed from the thinned image.

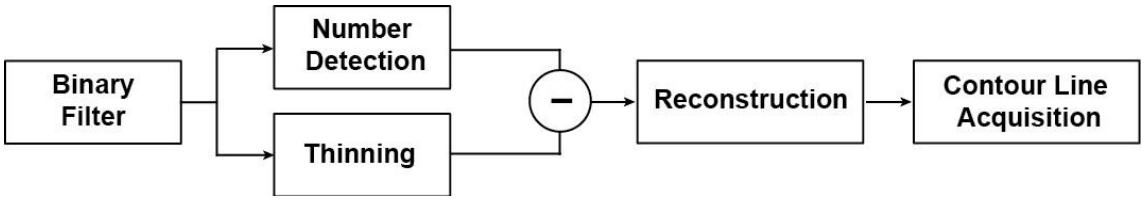


Figure 2.1 Contour line acquisition system diagram

2.1. Binary filter

2.1.1. Description

As section 1.1 explains, most of contours maps use a wide range of colours to represent the different terrain characteristics, so it is necessary to isolate contour lines and elevation levels, which is the function of the filtering process. As a result of this filter a binary image will be obtained.

2.1.2. Solution

Several ways to resolve this issue have been studied, although these can be largely grouped into two sets: those that change RGB[1] to HSV[2] model and

[1] Colour model based on adding the three primary colours: red, green, and blue. Changing the amount of the primary colours all the visible colours can be created. It is widely used with computer displays.

[2] Colour model (Hue, Saturation, Value), often called HSB (Hue, Saturation, Brightness), is a non-lineal transformation of points in a RGB colour model to cylindrical-coordinate representation.

those that work with colour clustering. As the best option was unknown to me, both methods were implemented trying to test which one was better to solve the trouble.

In my opinion, Ghircoias and Brad proposal [1], which uses the colour clustering method, is a smart and reliable way to solve the problem. Some of the most important steps are: colour quantization, in order to reduce the set of colours used in the image, and colour clustering so as to group colours by means of Euclidean distance in the chrominance plane.

Other methods [2, 3] consist of transforming from RGB to HSV space. A cube represents RGB model, so the segmentation colour is difficult because the colour space is not perceptually uniform. Otherwise, HSV is closer to an artist point of view because the colours are represented in terms of hue, saturation and value. This model makes possible an easier colour range division. So, the extraction of colours is achieved through a specific threshold. Using this procedure on Figure 1.2 displayed above, colour segmentation has been obtained. For that, HSV space has been ordered according to the hue. Here, as elsewhere, contour lines are painted using brown and orange tones. As Figure 2.2 shows, brown and orange range is between hue values 0.05 and 0.19.

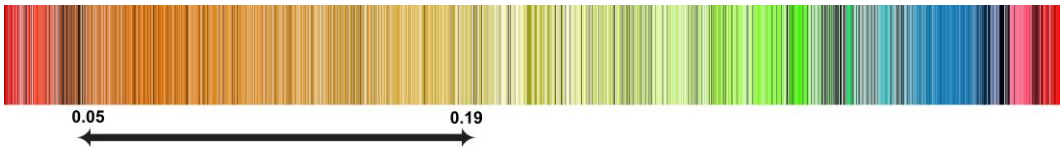


Figure 2.2 HSV space in order of hue

To isolate the contour lines of a map, it is used this threshold (0.05-0.19) and adds two new ones (saturation > 0.2 and value < 0.6) to discard colours close to black and white. Figure 2.3 illustrates the result of the previously explained.

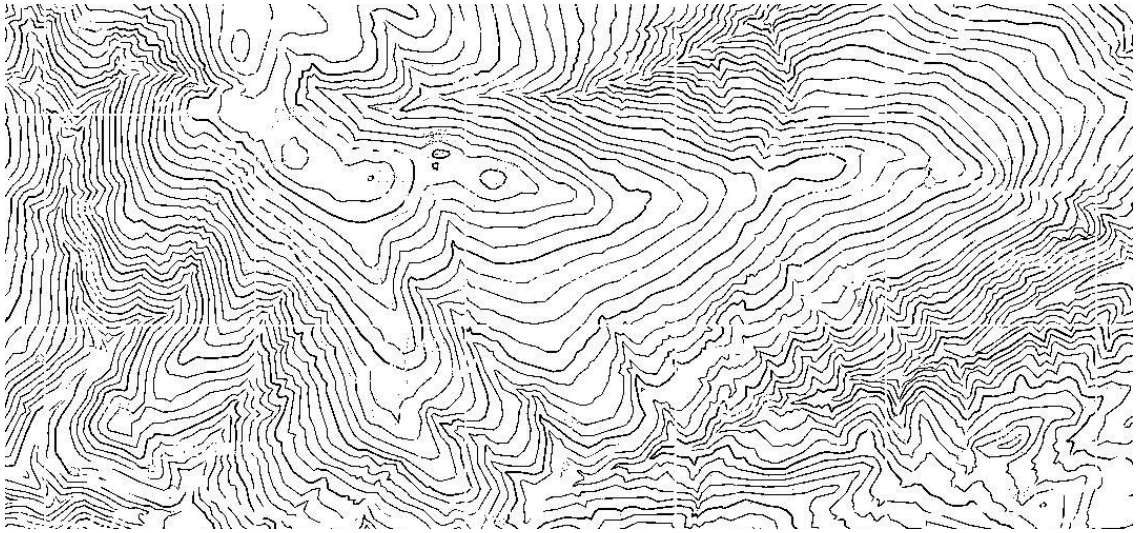


Figure 2.3 Output of filtering process

Last technique is easier and faster because it involves a lower number of processes and, therefore, a lower computational load. Its main drawback is the lower tolerance when the colour used to fill the contour lines has changed; the threshold has to be modified. However, colour clustering dynamically detects changes in colours, which makes it a robust filter but involves the fact that the software needs the intervention of a person, which has been tried to avoid.

Contour lines and elevation information have to be separated from other data to meet our objectives. This is a problem because contour lines, motorways and some specific roads share the same range of colour. It happens, in the same way, with height information and other characters on the map (highway or motorway designation, city names, hill names etc.). This makes it impossible to distinguish between them.

Trying not to divert my goals, the choice has been to work with black and white images. Despite this, black and white images are actually a greyscale, so it is necessary to use a filter to obtain a binary image. Both methods are still useful for my purpose, although in this case the needs are lower so I opted for the second one, RGB to HSV model, resulting in better software performance.

Figure 2.4 , which is a greyscale image, has been the chosen to illustrate in this document the different processes of the project.

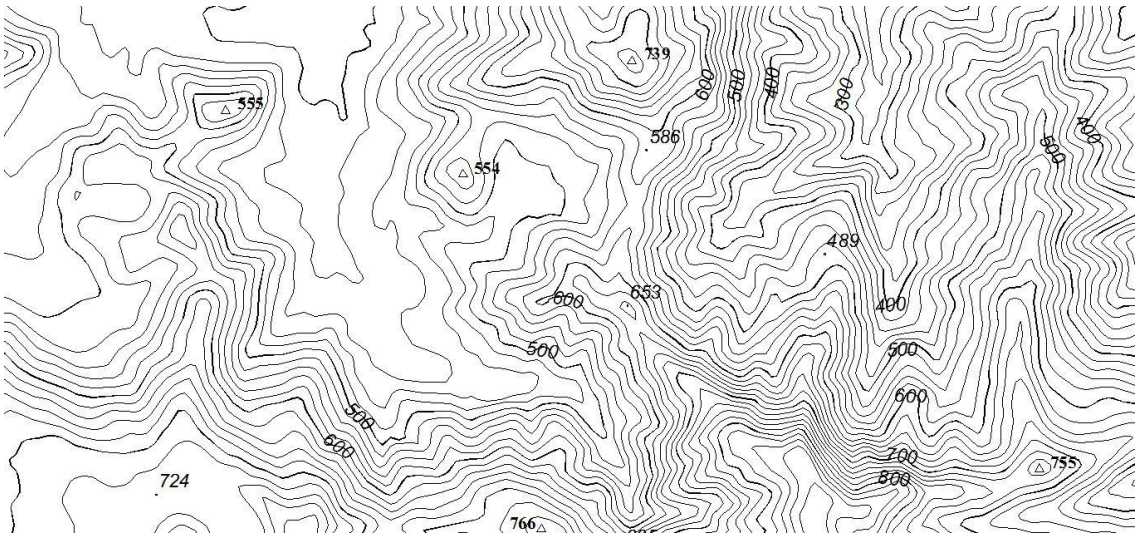


Figure 2.4 Sample contour map image used to illustrate all the processes

2.2. Number detection

Once the filtering is done, a binary image is obtained. This new image shows contour lines with elevation numbers.

2.2.1. Description

The greatest handicap that involves working with binary images is the numbers extraction from the lines, because both share the same colour. To solve this problem, different examples of binary contour maps images have been analysed. In conclusion, generally, in places where there were numbers on the image a highest concentration of black pixels was found. Therefore, a mechanism to detect black pixel density has to be used.

2.2.2. Solution

The easiest approach is creating a sliding window with a specific size such as 10 x 10 pixels size. This will traverse the image calculating black pixels within it and analysing whether this value exceeds a defined threshold. In this method the processing time depends on the window and the image size.

To prevent this dependence on the window size, the summed-area table algorithm [4] has been implemented. In this algorithm the image is though as an integral, where each field's content is the sum of the whole area from the upper left quadrant to this field, including it. This process requires a temporary image I, where the summed values are stored. It can be computed efficiently in a single pass over the image through the following formula:

$$I(x,y) = image(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1) \qquad \textbf{(2.1)}$$

When the summed area table has been computed, the density evaluation can be accomplished in constant time with just four array references. The value of those references points depends on the window size chosen; in this case it is equal to 4.

$$Density(x,y) = I(x,y) + I(x+4,y) - I(x+4,y+4) - I(x,y+4) \qquad \textbf{(2.2)}$$

After studying many images, a threshold has been estimated equal to 10 to detect numbers. It assumes that can exist a number when there are ten or more black pixels within the 4 x 4 window.

Whereby, it is possible to detect the focus of density (Figure 2.5 b), although it is not enough to extract the number information from the image, because the detected area in most cases was smaller than the number area. So, an expansion procedure has been done in order to adapt it to our scenario. As its name suggests, this process inflates the identified area allowing merging several nearby detected areas (normally the digits forming a number) or simply increasing this area to cover the number area (Figure 2.5 c).
Before keeping the calculated number area, an additional step has been included to reduce the quantity of detections, because many of these are wrong

(in that area there is no number). This merely consists of averaging the detected area sizes and removing those areas that are lower than the average (Figure 2.5 d). Despite this, it is always better to capture more information than is needed. This extra information will be returned to the image in the reconstruction stage (section 2.4).

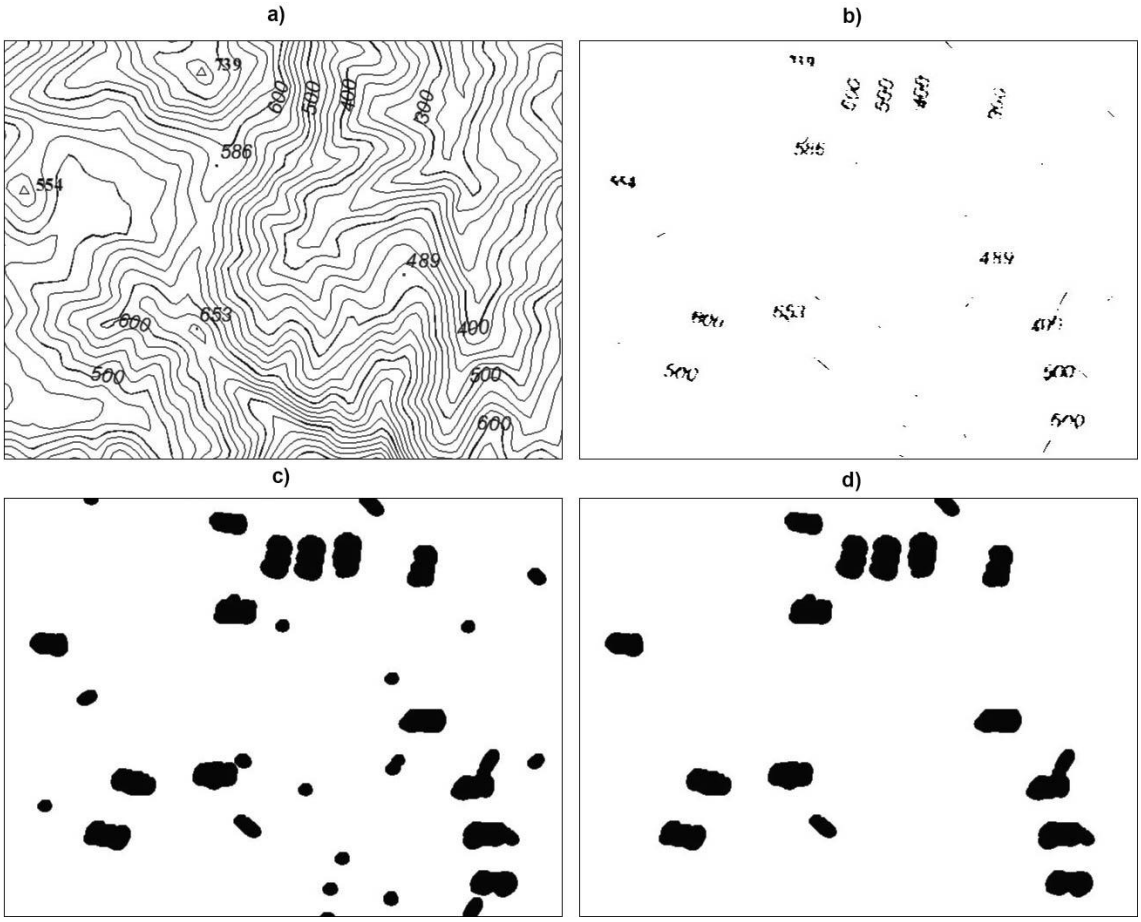


Figure 2.5: a) Original map fragment; b) Output of summed-area table algorithm with 4 x 4 window size and threshold equal to 10; c) Output of inflator procedure; d) Output of removing detected numbers with size lower than the average

Although, this step could be done before or after the thinning process, the choice of using it before is not random. If this process is previously carried out, the number thickness is taken into account, which enables a highest reliability in number detection (the numbers width is generally thicker than the lines).

It is important to know that the output of this stage does not involve any changes in the main image; only the area locations with higher density will be saved.

2.3. Thinning

This process starts with the same image used as input in the previous step. The image has contour lines with elevation numbers. The lines and numbers thickness are irregular due to the standards of the maps design (those lines indicating elevations are thicker) and the filtering process (which can involve a border reduction).

The main goal of the acquisition step is digitalizing the data to make its processing easier. Without reducing line thickness, some necessary steps as end-points detection and calculation of the line direction are more complex, so it becomes necessary to apply a thinning process.

2.3.1. Description

There exist a large amount of algorithms known as thinning algorithms. Thinning is a morphological operation used to remove selected foreground pixels in images. It is usually used on binary images and the output is another binary image.

After reading some papers [5] [6] I concluded that some basic properties have to be considered on thinning algorithms: one-pixel thickness, connectivity and maintenance of the original shape.

2.3.2. Solution

Most of the papers describing these algorithms are not public so I have just worked with the ones I have found. Those that seemed better have been implemented taking into account the results reported and later they have been tested. The proposal of Zhang and Sue [6] has been compared with Huang, Wan and Liu [5] and the main features of thinning algorithms are summarized in Table 2.1.

Table 2.1. Comparison between thinning algorithms

Thinning algorithm	Thickness	Shape	Connectivity
Huang, Wan and Liu	Large reduction of the foreground pixels.	Preserves the topology of the image	Perfect
Zhang and Suen	Less reduction of the foreground pixels.	Many spurious branches appear	Some disconnections are produced

As can be seen, Huang et al. algorithm is more suitable to contour maps shapes, which has conducted me to better results, so it has been the chosen

one. It specifies a set of steps or rules to perform the thinning. These rules are applied to each pixel simultaneously.

In some specific cases the thinning process does not perform properly. If the connectivity does not want to be lost, the objective of this process, one-pixel-width lines, is not achieved. Sometimes, the algorithm tries to keep the skeleton and the thickness is forgotten, so in these cases two-pixel thickness is obtained. That is why a new rule, implemented for our specific scenario, has to be added to the thinning process. Once Huang et al. algorithm has finished, our rule starts. It analyses each pixel checking its number of neighbours. In case the value is higher than two, the unnecessary pixels to maintain the connectivity are removed, imposing one-pixel thickness.

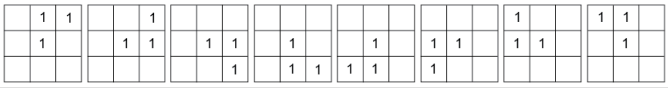
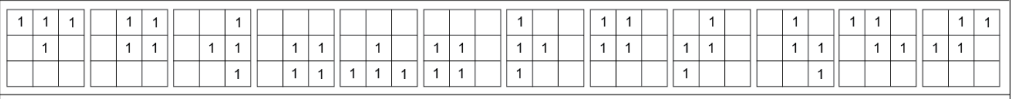
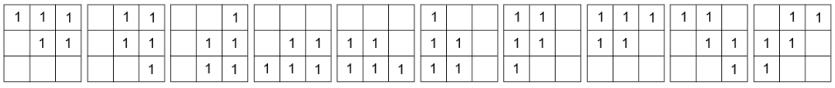
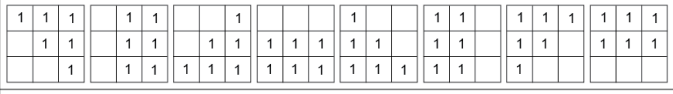
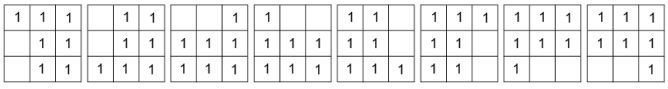
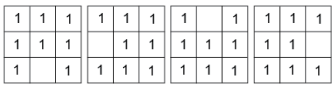
Both steps are going to be essential for a good performance of the software. As a consequence, it permits to detect cuts in contour lines, makes easier to calculate line direction and the number of pixels the program has to handle with is reduced.

Elimination rules described in Huang et al. are exposed in the following lines:

1. The index number (number of black pixels in 8-neighbours) is calculated for each pixel.

2. Use the index number and Table 2.2 to decide if the pixel is eliminable. If the pixel is not eliminable, go to step 5.

Table 2.2. Elimination rules

Amount	Elimination Rules
0	Never
1	Never
2	
3	
4	
5	
6	
7	
8	Never

3. Get the width of the pixel. If it is not two-pixel-width, delete it; otherwise, use the 3 x 4, 4 x 3 and 4 x 4 templates (Figure 2.6) to decide if the pixel should be preserved. If the pixel has not the requirement of preservation, delete it, or else preserve it.

4. Repeat 2-4 steps until no pixel can be eliminated.

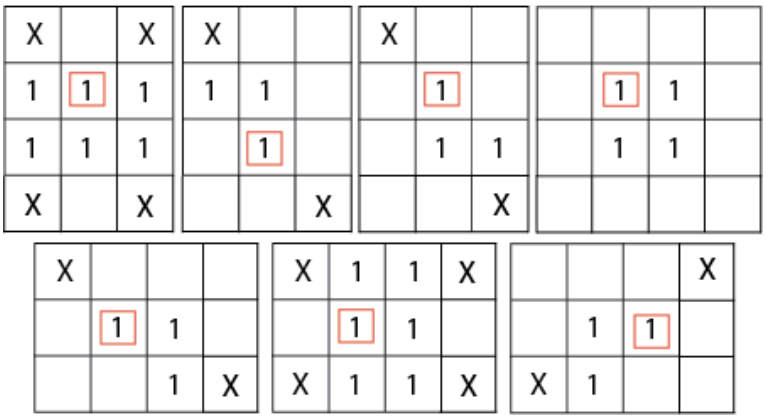


Figure 2.6 The preserved templates used in the algorithm.

Most of the times when a thinning algorithm is used, a pruning process follows it. Pruning is employed to remove branches produced during the thinning process that are not necessary to preserve the overall shape. These extra fragments appear when the line thickness is wide so, with the kind of lines it works with, this step is unnecessary. As an example Figure 2.7 a) is the input image in the thinning process and Figure 2.7 b) shows the obtained results.

At this time of the process, after the thinning is done, the density areas detected on the previous chapter (see section 2.2) are employed. On the new thinned image (Figure 2.7 b), the pixels forming these areas are going to become white, creating cuts on the lines (Figure 2.7 c), which later will have to be reconstructed. This process is necessary to have every line isolated, because the numbers connect the lines.

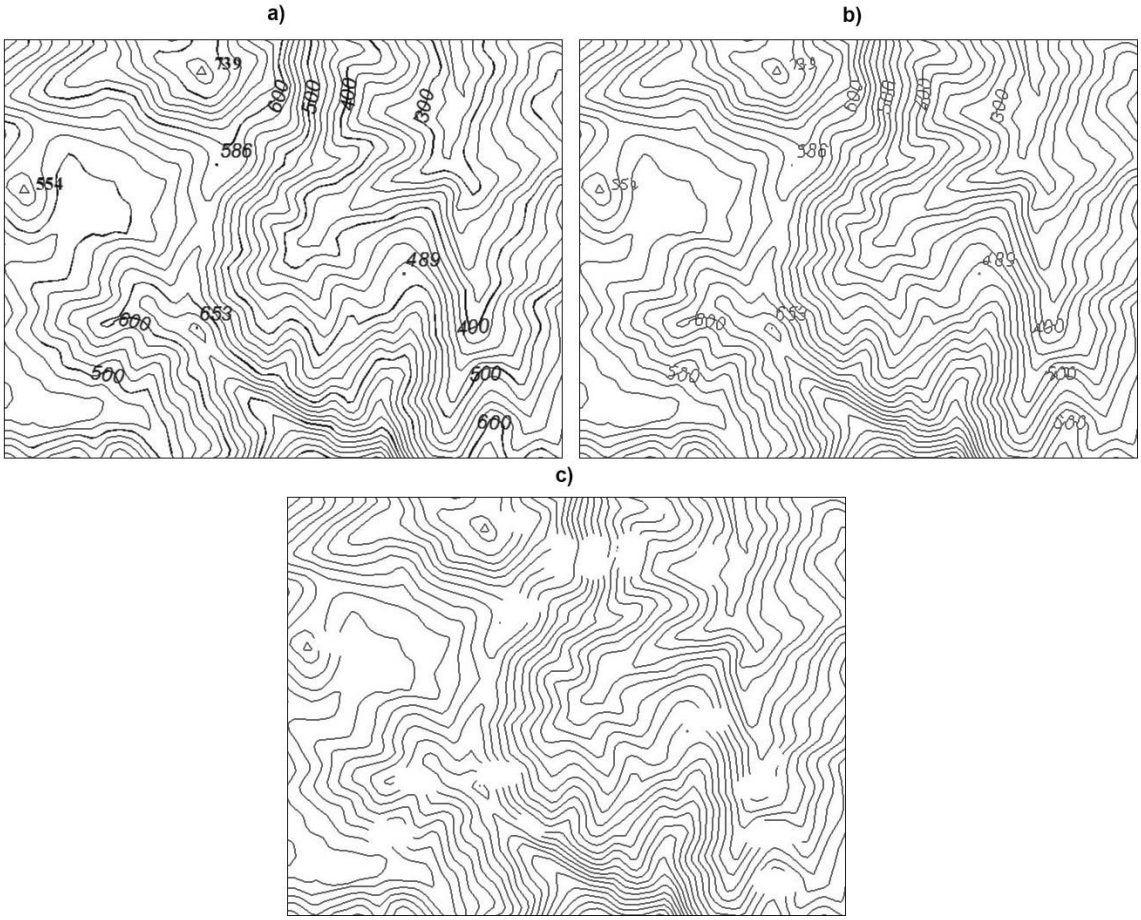


Figure 2.7: a) Original map fragment; b) Output of thinning algorithm; c) Extraction number

2.4. Contour line reconstruction

2.4.1. Description

Contour line reconstruction is one of most treated issues in contour map processing. Currently, it does not exist a general solution to solve it. To approach this subject, the reconstruction methods used in literature for fingerprint and contour lines have been analysed. This is because in some cases both areas present similar scenarios. Most of the proposed solutions consist of generating a directional field, although there also exist another ones. In this chapter there are going to be illustrated some of the solutions more widely used in this field and the best suited solution to this particular environment.

First of all, there are going to be exposed two scenarios to deal with the reconstruction issue: the most common one and the particular scenario presented in this document.

The first one consists of a colour image where a colour filter is needed to isolate just the contour lines. This filtering generates a binary image (Figure 2.3) with disconnections on the lines. These cuts have a non-uniform distribution.

On the other hand, the used image is a greyscale, which needs to be filtered to obtain a binary image. This one does not generate any isolation; numbers and contour lines compose it, so there are no cuts. Later, numbers have to be extracted and this generates disconnections on lines (Figure 2.7 c). Hereby, these ones are located on the place where the number was.

The main differences between both cases are the quantity of cuts and the difficulty to reconstruct the deleted line chunks. Generally, the filtering on the first scenario produces more cuts than the number extraction on the second one. However, taking into account the way the cuts are located, on the scenario used in this project, the matching options for each line is higher because the cuts affect to the lines surrounding the numbers. Moreover, the non-uniform distribution of the cuts on the first scenario exposed means that the matching is easier because the possibilities are lower.

2.4.2. Solution

Different solutions for contour lines reconstruction read in literature have been studied [1] [2] [7] [8] [9]. Next, two of the main techniques employed on contour maps reconstruction are going to be described.

2.4.2.1. Pouderoux and Spinello algorithm

Pouderoux and Spinello algorithm [9] consists of reconstructing the global gradient orientation field of the available contour lines. This information is used in order to find the pairs of end-points that should be reconnected. Three steps divide their algorithm: orientation field generation, connecting end-points and smooth gap reconstruction.

- *Orientation field generation:* The main goal in this procedure is to obtain the orientation in the normals at every point on the image. For this purpose, initially, the normal vector in each pixel of the input contour lines is computed and stored in F, an array $m \times n$ (input image size).

  Once this step is finished, it is needed to propagate the normal orientation in those points that are not initialized (white pixels). Field interpolation has been use to implement it. Function θ computes the mean orientation of a p point using a 3 x 3 window in the 8-neighbourhood ($N(p)$). The q points are those forming the 8-neighbourhood.

$$\theta(p) = \frac{\sum_{q \in N(p)} \Lambda(F(q), F(p))}{|N(p)|} \qquad \textbf{(2.3)}$$

Function Λ ensures the consistency of the orientation operations between two angles α and β in F:

$$\Lambda(\alpha,\beta)=\begin{cases}\alpha & if\ d1=\min(d1,d2,d3)\\ \alpha+\pi & if\ d2=\min(d1,d2,d3)\\ \alpha-\pi & if\ d3=\min(d1,d2,d3)\end{cases} \quad \textbf{(2.4)}$$

Main idea of this step can be illustrated in Figure 2.8, where through a streamline recreation it is possible to understand how the field interpolation is working.

Figure 2.8 Streamlines recreation of the generated orientation field in a contour map fragment.

- *Connecting end-points:* As the previous one, this step is divided in two subroutines: Matching's weight estimation, and perfect matching.

  At first, for each pair of end-points (e1, e2), a weight given by the following energy function is estimated:

$$w(e1,e2)=\sum_{t=0}^{1}\left|\overrightarrow{p-q}\cdot\overrightarrow{F(p)}\right| \quad \textbf{(2.5)}$$

  This energy is equivalent to go from one end-point to another one through the orientation field F. The energy is zero if the end-points directly meet together by following the flux line; otherwise the energy is maximal proportionally to the path length if the path to join them is everywhere normal to the gradient orientation field. Using the energy

values the matrix *W* is constructed. *W* and the Gabow's algorithm [3] implementation solve the perfect matching.

- *Smooth gap reconstruction:* Reconstruction takes into account the computed previous information (field orientation and matching). It makes a linear interpolation with the points that naturally continue the contour line in the generated field in each pair of end-points.

  Due to the good results shown in this paper and the capability of abstraction the employed method has, it was thought to be a good solution for our scenario. For the same reason, its implementation was begun. Once the orientation field generation stage was ended, it was evidenced that it wasn't able to resolve the reconstruction process in our kind of maps. In our case the cuts normally affect a high number of lines close to each other and, in most of the cases; these lines directions are very similar. These characteristics may result in a non-coincidence between the energy values obtained on the orientation field and the right path, obtaining a wrong matching in most cases.

2.4.2.2. N. Amenta algorithm

N. Amenta's proposal [7] is an algorithm based on medial axis[4] of a curve. She proved that *"the crust of a curve can be extracted from a planar set of points if the points that describe the curve are sampled densely enough"*. To achieve an approximation of the medial axis it uses the vertices of Voronoi diagram [5] (Figure 2.9 b). Then there is an application of Delaunay triangulation[6] among the original points and Voronoi's vertices (Figure 2.9 c). When analysing the obtained triangles, the edges that join two of the initial points are considered part of the curve (Figure 2.9 d).

Although this method has been studied, the kind of images our project works with doesn't fulfil the requirement of it. In most of the cuts, the distance between adjacent lines is lower than the distances to the possible end-points to be connected. This method is very useful when filtering method [1] is used.

---

[3] It is an algorithm for maximum matching on non-bipartite graphs. It means, that matching contains the greatest number of connections possible.

[4] A point in the plane belongs to the medial axis of a curve, if at least two points located on the curve are at the same distance from the point.

[5] It is a mathematical technique that enables the decomposition of a given space from a set of points.

[6] It is a triangulation for a set of points such that any point of these is inside the circumcircle of any triangle.

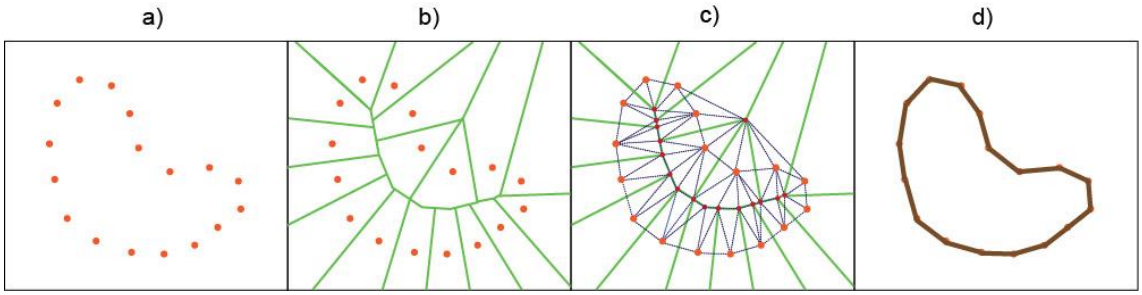


Figure 2.9: a) Initial set of points; b) Voronoi diagram; c) Delaunay triangulation; d) Reconstructed contour line

2.4.2.3. Proposed solution: self-designed algorithm

As the previous algorithms studied weren't suitable to the needs of our scenario, we decided to develop an own and specific solution for this kind of maps. It is not presented as a general solution to any type of reconstruction; it has been thought just for these specific reconstruction requirements.

Before describing the process, the two kinds of connections taking place are going to be explained in order to clarify the difference between them:

- The extracted information in section 2.3 is used for the line reconstruction.

- The reconstruction is carried out just taking into account the non-removed data, so the line has to be regenerated. To fill the gap, the simplest solution consists of joining the two end-points through a segment. The obtained results, in most cases, have poor appearance and can even cause topographic errors (Figure 2.10 a). A very used technique for getting smooth curves (Figure 2.10 b) in reconstruction is splines. In mathematics, splines are smooth polynomial functions divided in fragments.

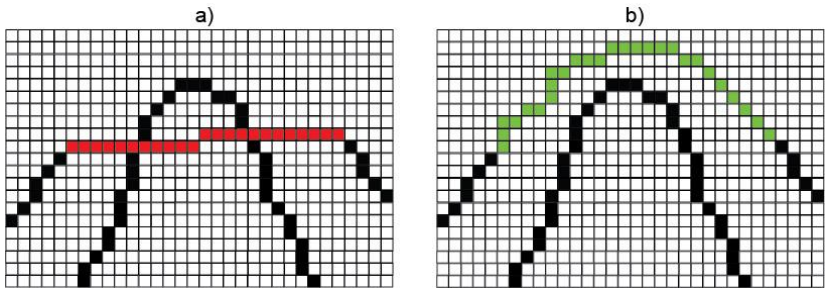


Figure 2.10: a) Join two end-points through a segment; b) Join two end-points through a spline

Some of the most commonly known splines are B-spline, Bézier spline and Hermite spline. Our problem has been solved using the cubic

Hermite spline, which has got better results in terms of more realistic curves. Hermite spline is defined in next equation, where p_0 and p_1 are the end-points to join and m_0 and m_1 are the tangents of these end-points.

$$p(t) = (2t^3 - 3t^2 + 1)p_0 + (t^3 - 2t^2 + t)m_0 + (-2t^3 + 3t^2)p_1 + (t^3 - t^2)m_1 \quad \textbf{(2.6)}$$

The reconstruction process has been divided in seven stages:

1. Looking for the end-points in the zones where the numbers have been extracted. As a consequence the end-points are already grouped (the matching will just be done between them).

2. Calculating the end-points direction, except those that are a single pixel (its direction will be calculated later when extra information is know).

   To compute the direction, the last 10 pixels from the end-point are used. The direction of each of them is calculated using the previous and following pixel and then the average is done.

3. Looking in the extracted information for line fragments that do not cross anything (in case there is a cross, a number detection is supposed). In the cases where this happens, the line fragment is directly returned to the map and the correspondent end-points to this fragment are removed or their position updated if the added fragment does not complete a line. Otherwise, two possible options are considered: those where one number only crosses one line and those where this affects more than one line.

   In case the line has any cross but it only joins two end-points that means there might be a number over it and it only affects one contour line. This line cannot be returned to the map, but its end-points will be joined with Hermite spline and removed from the end-points list. If it affects more than one line it will be considered in following steps.

4. Looking for possible noises (pixels with more than two neighbours) and small line fragments generated during the extraction process and managing them. These noises are number fragments as a consequence of an error in the number extraction process.

   In case the noise is detected, in most cases, there exist some branches that are sharing the same pixel. The solution to this problem is the removal of these branches until the point they share. This point is added to the end-points list and the end-points from the deleted branches are removed.

If a small fragment is found, two possibilities are treated: line fragments (have two end-points in the extracted area) or number fragments (just one end-point). Number fragments are directly deleted because they are not useful information from the terrain. Otherwise, although line fragments are useful information, they are deleted as well. This is because, as it will be seen in next steps, they could cause mistakes in the matching process.

5. Calculating the end-points direction, in those that were a single pixel, but as a consequence of the returned information they are not.

   Moreover, at this point, the end-point groups can be odd or even. For perfect reconstruction odd groups are needed so, when a group is made up of even end-points, at least one of them can be regarded as an error. To detect the error, the distance and the direction of all the end-points are considered. The one that differs from the rest of end-points will be removed.

6. Going to the extracted information again, but this time it is not treated end-point by end-point, it is analysed as a whole.

   Seeing from each end-point how many pixels can be returned to the map before the detection of a number. Once this information has been considered, the case with the minimum amount returned of pixels is the reference. This reference is used over the traversed extracted information to return this number of pixels from each end-point to the map. For example if the reference is five, from each end-point, five pixels are returned to the map. Then, the end-points location is readjusted.

   The reason for doing this procedure is to keep proportionally the distance of each end-point to the centre of the end-points in their group. The reconstruction lines will be shorter, reducing the risk of error in gap filling.

7. At this moment, it is assumed that the number of end-points in each matching group is even.

   Firstly, the end-points are clockwise sorted in an array respect the centre point of them. Then, the array is traversed using two pointers whose distance is half the array size. Hereby, to analyse all the array positions, the first pointer goes from the initial position to $\frac{array\ size}{2} - 1$ position. Using this, the algorithm computes the distance difference between each pointer and its next position in the array. Both differences are added. The pointer will be saved when this addition is the highest. The highest difference takes places when it is detected that both pointers are located in the last end-points of a subgroup (one side of the matching) before changing to the other subgroup. The first end-points subgroup will be formed by next

$\frac{array\ size}{2}$ positions from the first pointer (not included). When both subgroups are created, the matching consists of traversing them in opposite directions joint the end-points.

Figure 2.7 a) has been used as input in reconstruction algorithm and Figure 2.11 illustrates the output. Two types of connections can be differentiated by two colours (purple and green), depending on the available information.

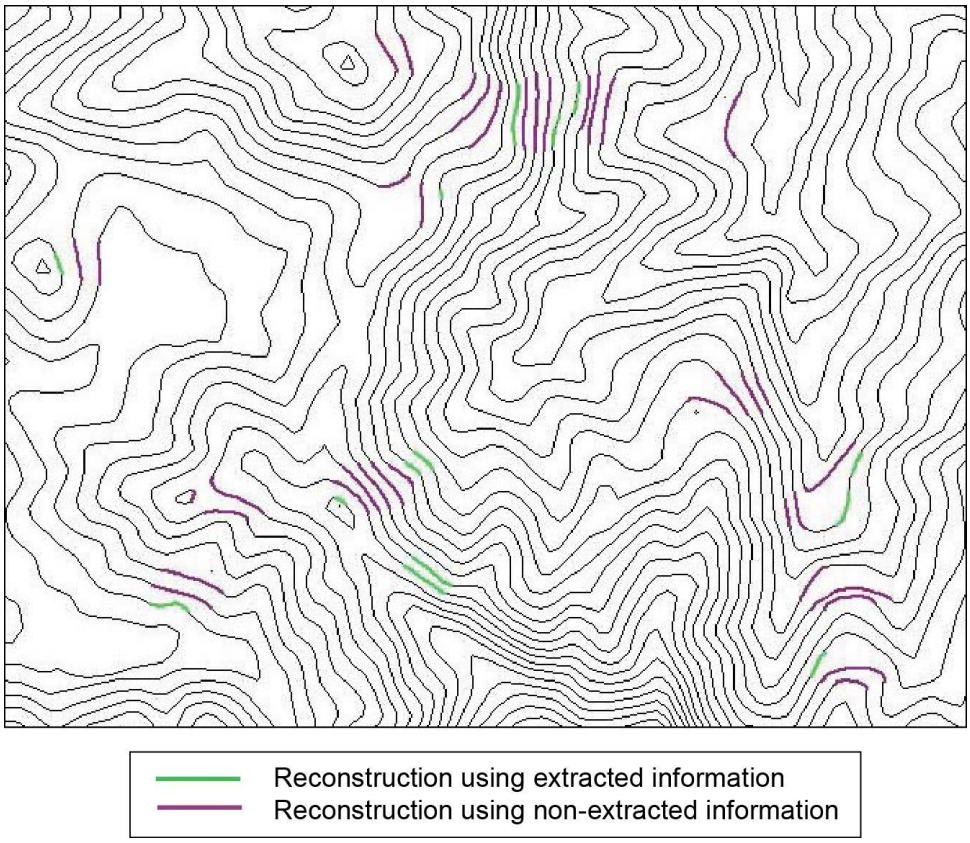


Figure 2.11 Output of reconstruction algorithm

2.5. Contour line acquisition

2.5.1. Description

After completing the step above, a binary image is obtained where the contour lines are closed. Next steps involve the processing of each contour line so a previous collection process is needed.

2.5.2. Solution

A simple algorithm has been designed to extract all the contour lines. Furthermore, taking advantage of the need of traversing the whole image, the end of contour lines will be collected for subsequent management.

The algorithm (Figure 2.12) consists of traversing the image while a black pixel is not found. When a black pixel is detected, its 8-neighbours are analysed looking for connected black pixels and computing the number of neighbours it has. Each time the pixel neighbourhood is analysed, it is marked as checked to avoid analysing it again.

When the number of neighbours is equal to one, it is checked if this pixel is part of the image framework and, if so, this point is saved. These steps are repeated in the connected black pixels and so on, until the contour line has completely been traversed.

Finally, the pixels forming the contour line and its corresponding end points (if they exist) are stored. Once a contour line is finished the algorithm continues traversing the image seeking for new contour lines.

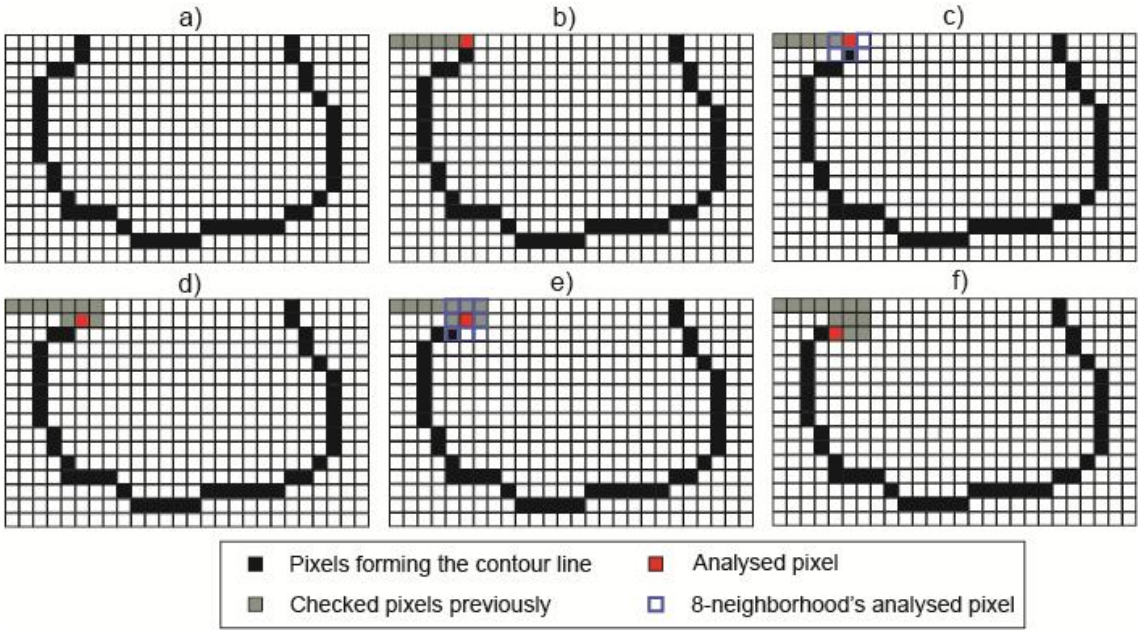


Figure 2.12 Contour line extraction algorithm

CHAPTER 3. CONTOUR LINE PROCESSING

The main objective of this chapter is to know the elevation value for each point of the contour map. In order to achieve the goal, the procedures described in this chapter will be carried out over the information obtained in the previous one.

The workflow in this chapter is represented in Figure 3.1.

Figure 3.1 Contour line processing diagram

3.1. Contour lines closing

3.1.1. Description

As in section 1.1 was explained, all contour lines are closed but sometimes the adjacent map is needed to complete it. The problem arises due to the fact that when a contour map is processed the adjacent information is not available. This results in many cut contour lines.

In most cases, despite having some elevation numbers from the map, the orientation of many contour lines is unknown. This is a key point because an error in a contour line can cause an inaccurate allocation of elevations in the closest contours to it.

3.1.2. Solution

As the information is not accessible, it is actually impossible to determine the curve trend. It is assumed that the contour line concavity or convexity about the segment that joins the two end-points will indicate which of the two sides close it.

The implemented solution to identify the contour concavity or convexity centres on computing the quantity of pixels above and below the segment. When the number of pixels is largest above than below the contour line, it is considered to be closed using the framework below. Otherwise, the contour line is closed in the opposite way. Furthermore, to avoid traversing the whole contour lines, a quantization factor equal to five has been used. Figure 3.2 shows how this solution is working.

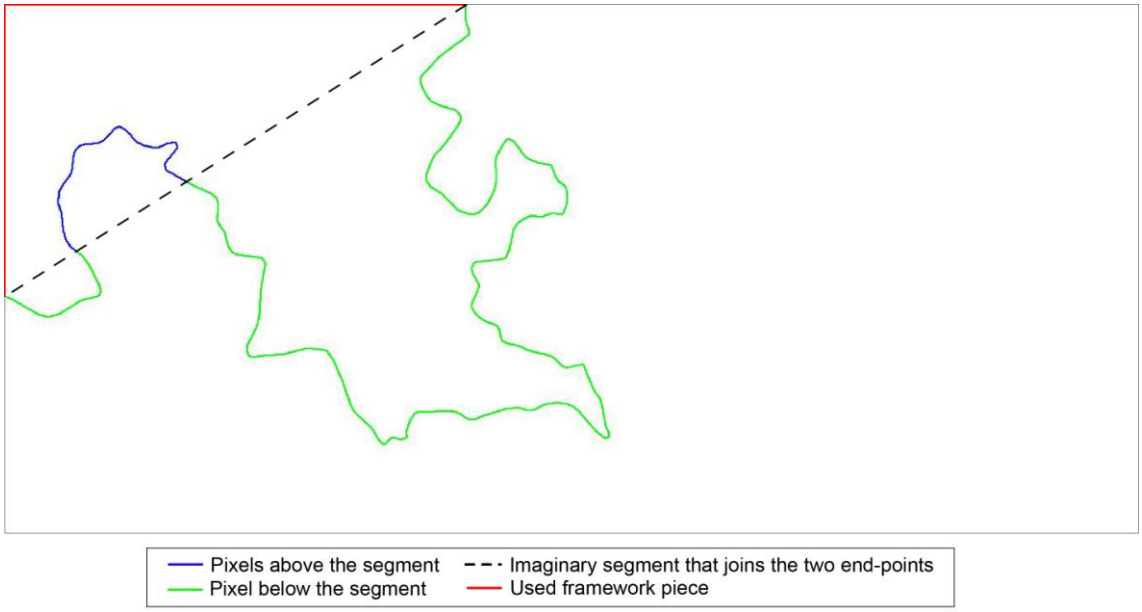


Figure 3.2 Number pixels below the segment is largest than above, so the used framework piece is the one who is above the segment

3.2. Interior points location

3.2.1. Description

Normally, when a person is using a graphic painting or graphic editing program and wants to fill a close figure, it does not matter if this person is using a simple software application as Microsoft paint or a more complex one as Photoshop; an interior point has to be indicated. However, in our scenario, the intervention of a person has not been considered so, through an algorithm, at least one interior point in each contour line has to be found. There exist some cases where more than one interior point is needed to completely fill the contour line: when the contour line is so close to itself and some pixels touch (Figure 3.3 a), and when the contour line touches the framework with some pixels dividing it in two (Figure 3.3 b).

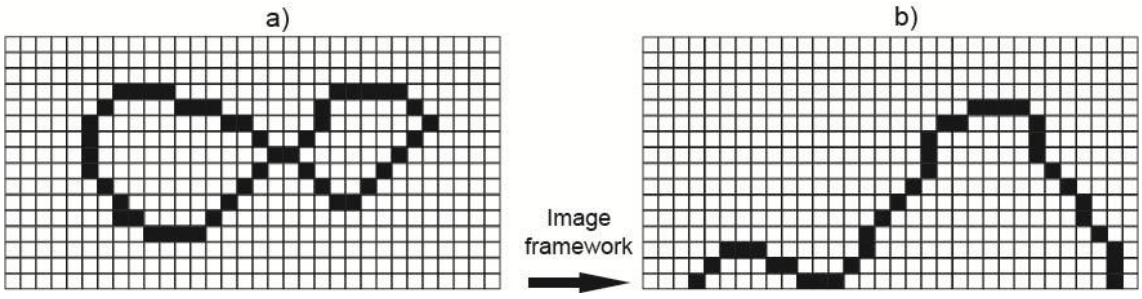


Figure 3.3: a) Contour line is divided itself b) Contour line is divided by framework

3.2.2. Solution

The proposed solution has to return as many points as necessary to fill the whole figure.

Firstly, the pixels forming the curve have to be sorted according to the horizontal axis and those sharing the same value according to the vertical axis. Then, each vertical line (same horizontal value) is analysed in order to find the number of segments it has. Each segment is formed by adjacent pixels. This vertical line will only be treated when its number of segments is equal to one or two. In case of having two, two options may be occurring: the pixels between both segments are within the figure (any pixel of these is an interior point) or we are analysing a vertical line where two points are protruding from the figure (Figure 3.4 a) (any pixel between of these is not inside the contour line, so are not useful). To avoid last option, besides using a vertical line with two segments, the vertical value in the possible interior point must have been checked previously in this contour line (Figure 3.4 b).

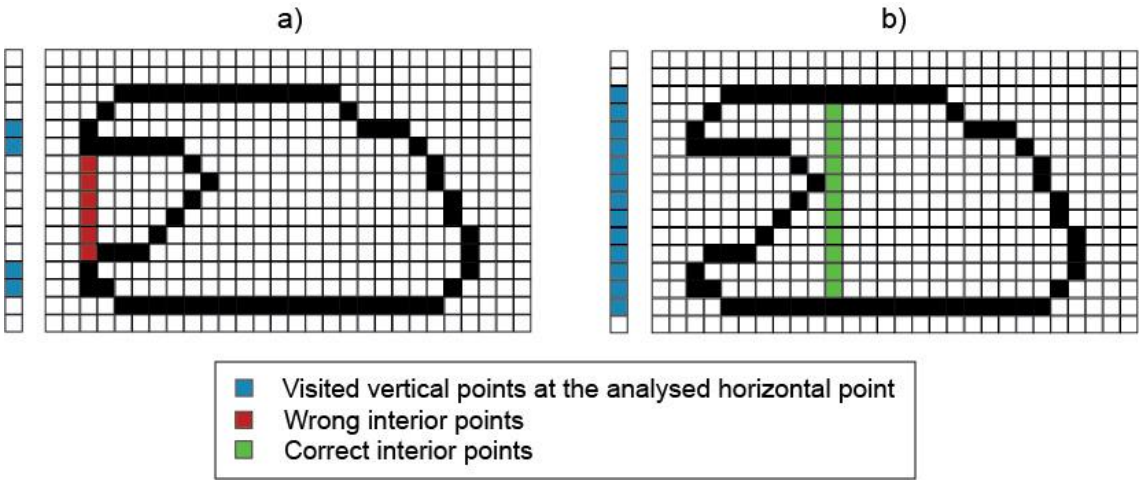


Figure 3.4: a) Wrong way to find an interior point, it is only taking into account the two segments premise b) Correct way to find an interior point, the vertical value in an interior point has to have been checked previously

When an interior point is found, the algorithm keeps repeating this process but, although new points fulfil the requirements, they will not be considered unless a single segment is detected in a vertical line. In this case, the contour line can be assumed as closed at this horizontal value, so a new interior point could be needed.

This procedure takes place a second time but, changing the way to sort the pixels forming the curve. They are firstly sorted according to the vertical axis, and then, those sharing the same values, according to the horizontal axis. So this time the pixel segments are composed by adjacent pixels on the horizontal value.

As a result, the obtained interior points in each contour line will be used in the filling algorithm (see section 3.3).

3.3. Contour lines filling

3.3.1. Description

In this stage of the process all the contour lines are isolated. It is essential to know which contour line is inside the others. With this information and the contour line heights, it will be able to estimate the height of each contour line.

The initial idea to solve this issue was to approximate each contour line to a rectangle. To do that, the lowest, the highest, the leftmost and the rightmost points of the contour line are taken into account. Through rectangles, in a simple way, it is possible to know when a contour is overlapping another one. The problem is that the contour line shape can be very irregular, so this approximation, in some of the cases, can lead to errors. Hereby, a filling algorithm has been implemented, rejecting the previous method. As a consequence, every pixel inside the contour line will be known. A contour line inside another will contain the same pixel permitting the overlapping detection.

To fill the contours, a blank canvas is used for each one. By means of this process, all the pixels forming the inside of the contour can be filled (there is no line within it). At first, a new blank canvas with the original image size was used every time. Although, it worked correctly, a lot of resources were wasted (Figure 3.5 a). Blank canvas with a customized size solves the problem (Figure 3.5 b).

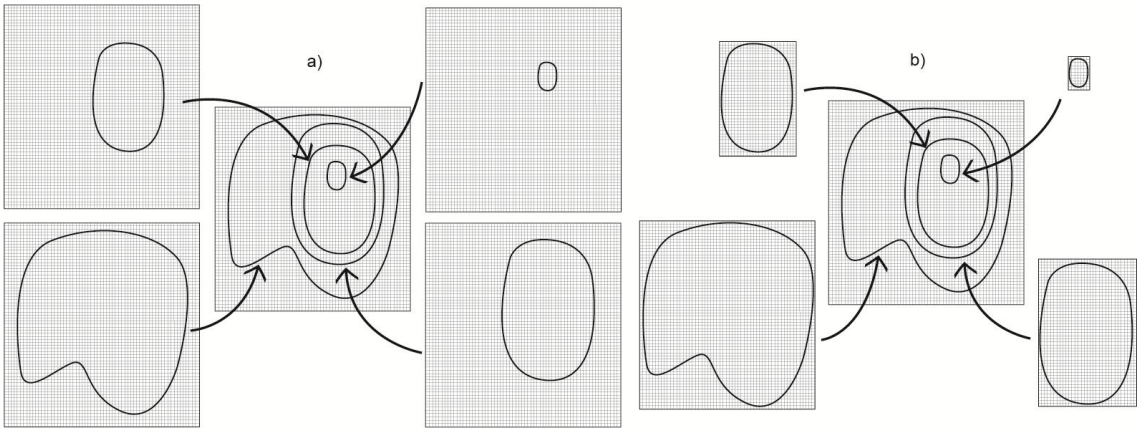


Figure 3.5 Generation of adaptive blank canvas to contours for a better use of resources

3.3.2. Solution

There exist different filling algorithms [10] [11]. Some of them were tested in order to determine which was better adapted to the characteristics of contour maps. Next paragraph briefly describes each of these: Recursive, Flood fill and Scan-line fill algorithms.

Firstly, a recursive method was used, because it is the simplest. This method consists of checking an interior point and its 8 pixel-neighbours. If any of the neighbours isn't coloured the method calls itself again with the centre in the non-coloured pixel. Recursive methods in Java use a default Java stack, whose size is limited. So, when the recursion exceeds a certain value, a "Java stack overflow" is obtained. The problem can be fixed substituting the java stack by a queue or stack.

For a better software performance, Flood fill algorithm was implemented. A queue is used and through an interior initial pixel, its left and right neighbours pixels are coloured until they arrive to a contour border. While these pixels are being traversed, their neighbour pixels above and below are added to the queue. On these queued pixels the process will be repeated.

Trying to reduce the filling process, an improvement has been implemented. Scan-line fill algorithm can be faster filling lines than Flood fill algorithm, because instead of pushing each potential pixel into the queue, it checks the previous and next line looking for the adjacent segments that might be filled in a future step. It adds to the queue either the start or the end pixel of the neighbour line.

Using the different algorithms on a same image, the velocity values obtained (pixels/second) demonstrate that Scan-line fill algorithm implies a better software performance (Figure 3.6).

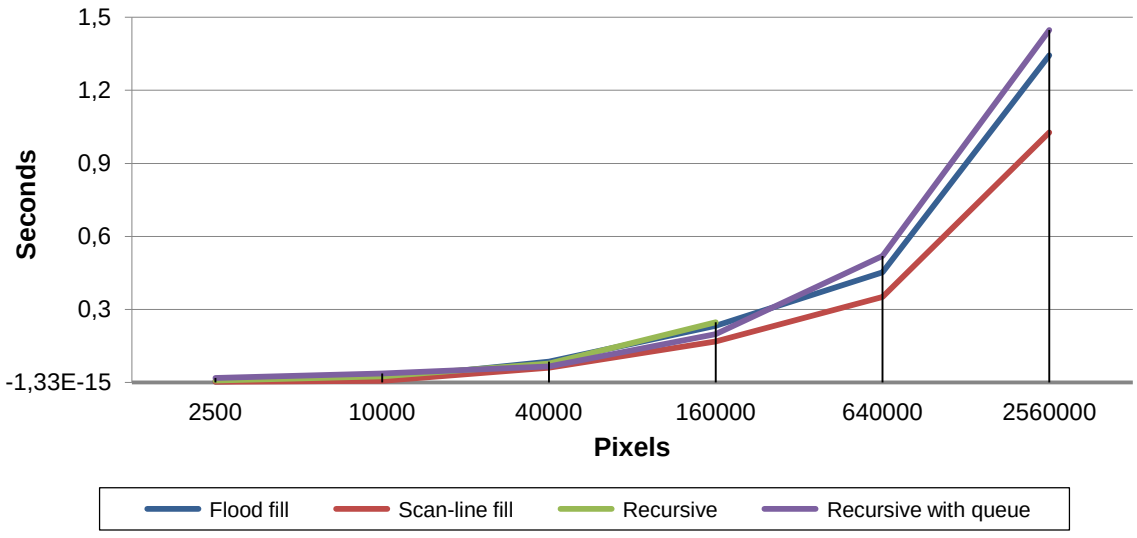


Figure 3.6 Benchmarking of filling algorithms

3.4. Contour lines matching

3.4.1. Description

In most cases, the elevation of the contour lines is unknown, but it is known that the difference between them is always the same. Once the elevation value of one of the contours is identified and, obtaining the existing relation among all the contour lines, it is possible to compute the elevation value of other contour lines. At this point, contour lines are going to be related.

3.4.2. Solution

In order to solve the trouble introduced above, a simple technique has been implemented. It adds the contours to a list and sorts them, in function of their size, in ascendant order. Then, the list is traversed to search for each contour line next on the list that includes it. It means, they share, at least, one pixel. After that, each contour knows the contours that surround it, which facilitates the elevation assignation.

The contour lines are sorted this way in order to check for each one only the ones with a larger area; otherwise it is impossible to be contained in it.

Figure 3.7 is the output of all the processes included in this chapter so far. The different colours show the elevation changes.

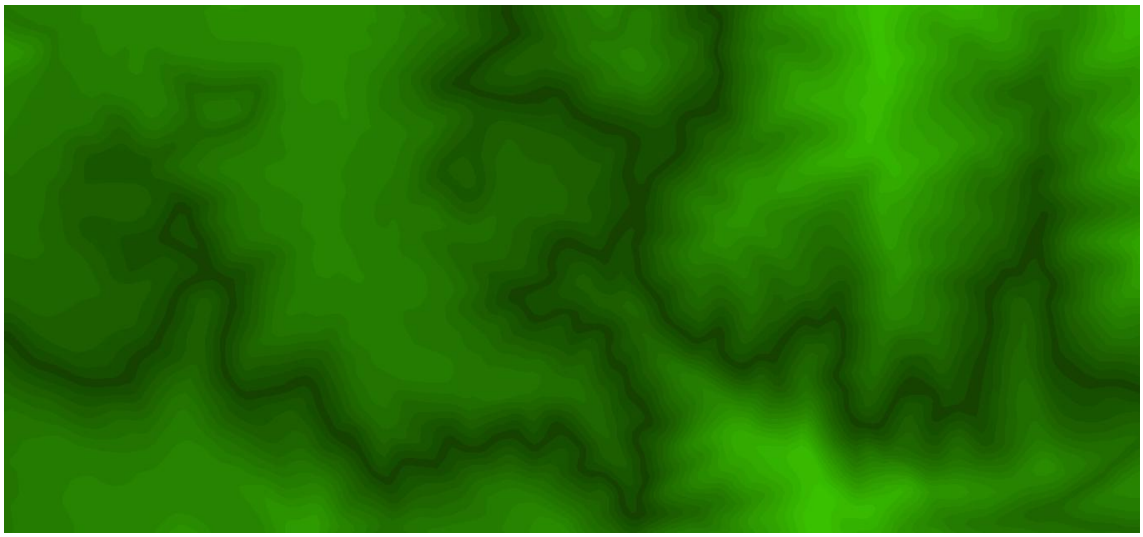


Figure 3.7 Filling and matching process output

3.5. Elevation data interpolation

3.5.1. Description

At this moment, the elevation value in each point is available so, the task to translating this information into a matrix is easy. Through the horizontal and vertical positions and the height values a 3D representation would be possible. This would display accurate information obtained from the map but would not seem a real terrain representation. As a result, terrain surface would show some steps indicating the elevation changes.

3.5.2. Solution

To get a smoother surface, it is needed to increase the number of height levels to reduce the distance between elevation values.

The first idea for achieving the objective explained above was to average the elevation values of the surrounding pixels for each one, obtaining a smoother elevation matrix.

Trying to improve the averaging results, a bilinear interpolation was implemented. Linear interpolation permits to obtain the points between two points, in other words, the straight line that joins it. Bilinear interpolation could be considered as an extension of the previous one to interpolate two variables function. Basically, the idea is to use the linear interpolation in one direction. Then, the obtained results are used in the linear interpolation in the other direction. For example, to calculate the interpolated value at the point $P = (x, y)$ points Q_{11}, Q_{12}, Q_{21} and Q_{22} are used (Figure 3.8).

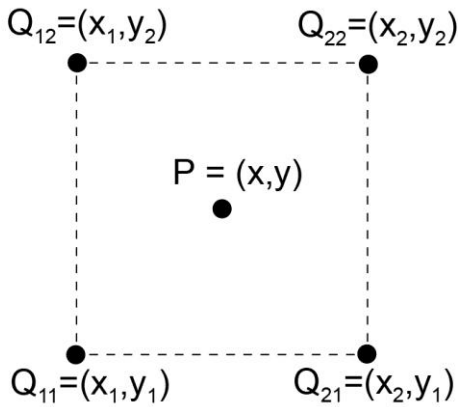


Figure 3.8 Bilinear interpolation example where points Q11, Q12, Q21 and Q22 are used to interpolate point P

Next equations show how the interpolation proceeds:

$$
\begin{aligned}
f(x,y) \approx & \frac{f(Q_{11})}{(x_2-x_1)\cdot(y_2-y_1)}\cdot(x_2-x)\cdot(y_2-y) \\
& + \frac{f(Q_{21})}{(x_2-x_1)\cdot(y_2-y_1)}\cdot(x-x_1)\cdot(y_2-y) \\
& + \frac{f(Q_{12})}{(x_2-x_1)\cdot(y_2-y_1)}\cdot(x_2-x)\cdot(y-y_1) \\
& + \frac{f(Q_{22})}{(x_2-x_1)\cdot(y_2-y_1)}\cdot(x-x_1)\cdot(y-y_1)
\end{aligned}
\qquad \textbf{(3.1)}
$$

To demonstrate why bilinear interpolation is better than averaging method, both solutions have been compared in terms of software performance and the smoothing level. Figure 3.9 illustrates how both methods work.

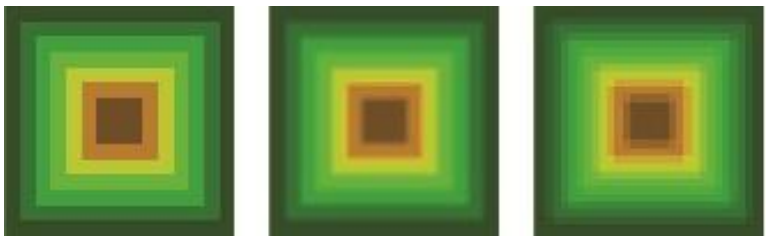


Figure 3.9: a) Sample image b) Averaging method c) Bilinear interpolation method

Averaging against interpolation methods presents a lowest smoothing level (perceptually, a lower quantity of elevation levels can be distinguish). Besides, taking into account the software performance (pixels/second), next graph shows a big difference in favour of interpolating method.

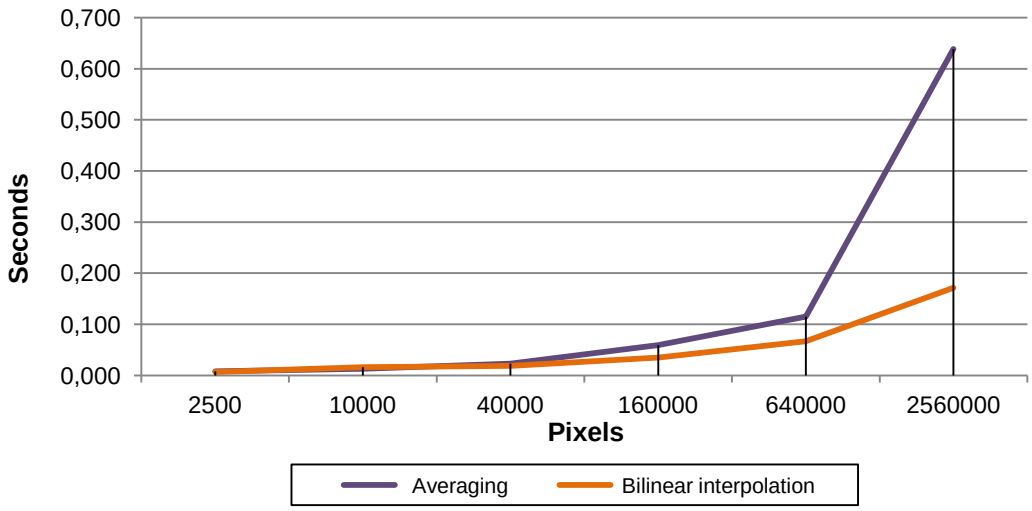


Figure 3.10 Benchmarking of smoothing methods

To sum up, comparing both characteristics it has concluded that interpolation method suits better. It provides a better software performance and higher number of elevation levels, so the terrain surface is represented in a more realistic manner.

CHAPTER 4. NUMBER PROCESSING

As a consequence of the number detection step, the pixels areas where numbers were located, are now stored. At this moment, it is needed to treat this information to know the number information included in the contour map. Optical character recognition (OCR) technology usually is implemented to solve this issue.

OCR libraries are designed to convert different types of documents, such as scanned paper documents, PDF files or images into text files. The problem is that, in most cases, they are not prepared to process the input information if the input is not properly displayed (rotated information, data with noise, etc.). So, some previously steps are proposed (Figure 4.1). Through these steps and number recognition stage, it is wanted to match the elevation values recognized from the image with the contour lines previously processed.

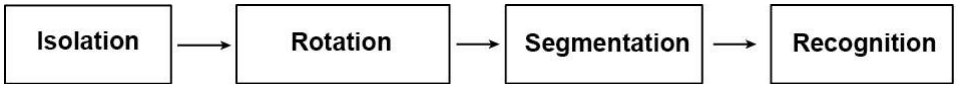


Figure 4.1 Number processing diagram

4.1. Number isolation

4.1.1. Description

On the basis of currently available number information, the scenario is very complex. Depending on the quality of the contour map and the font size, sometimes the number recognition is not even possible to the human eye. For this reason, as much as possible, it is required to isolate the number elevation from the background noise (contour lines). This way, their recognition will be easier.

4.1.2. Solution

In the different image fragments, where it could be a number, all the lines touch the edge of the area are contour lines (noise). Using this information and the summed-area table algorithm [4] to compute the density, most of the contour lines can be removed.

In order to do that, first the summed-area table algorithm has been computed. Then, as the location of the beginning of each contour line is known (black points in the edge of the area), the density in this position can be checked. This enables us to estimate the line density, so the line can be removed while the density is equal or lower to the estimated value. In case the density is higher, it is assumed that number has been detected. This way, this method dynamically

adapts to the line width, so it is not obligated to work with a single line thickness.

Once the surrounding noise has been deleted, next step consists of removing or reducing the noise, which crosses the numbers. For that, reusing the computed density and establishing a density threshold, good results can be obtained. The main disadvantage of using a threshold is that number information can be removed if its value is not accurate enough. So, the choice made has been to use a low value trying to remove the thinner lines and to reduce the other ones.

Next figure shows how the number isolation is working. It is important to note that the contour map used to show the elevation information has good quality, otherwise the results could be worse.

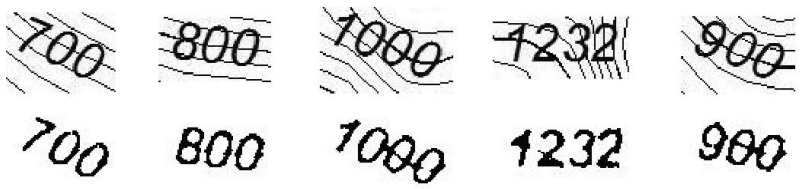


Figure 4.2 First row shows the extracted number from the number detection. Second row illustrates the results obtained in number isolation process.

4.2. Number rotation

4.2.1. Description

The number, in order to be recognized, has to be displayed horizontally. In most cases, the direction of the elevation numbers is the same or similar to those contour lines that cross them. Otherwise, it could be a number that reports elevation information about a significant point on the terrain. In this case, these values are not considered because they usually are located with a point or hyphen in the map, so their allocation is almost impossible (points or hyphens cannot be distinguished from noise in previous steps). Then, a method to obtain the angle in which the numbers are displayed is needed.

4.2.2. Solution

Taking into account the above information, the output obtained from reconstruction stage (see section 2.4) is used to get the slope of the reconstructed lines, which are the lines overlapped with the number. The angle formed by the horizontal axis and the slope can be computed. Next figure displays the results of applying a rotation equal to the calculated angle in the number images.

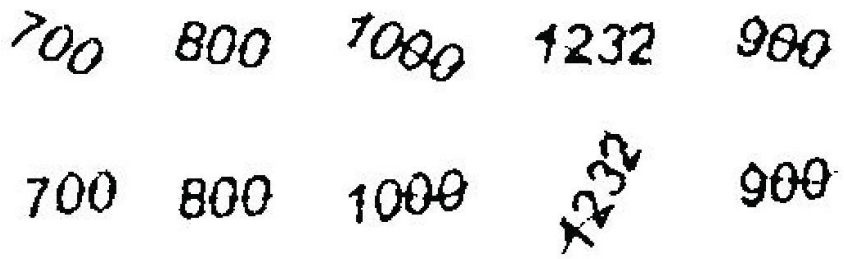


Figure 4.3 First row shows the number rotation input. Second row illustrates the output of number rotation

Most numbers have corrected their position pretty well. The exception in Figure 4.3 is the fourth number from left. As can be seen in Figure 4.2 for the same case, all the background lines under the number are not following the same direction so the angle obtained from reconstruction stage will be mistaken. In conclusion, the final results obtained for this type of number will be overlooked.

4.3. Number segmentation

4.3.1. Description

Before starting recognition stage, as can be seen in Figure 4.3, sometimes digits are connected by noise. In the recognition method, in order to segment the numbers, the pixels are analysed vertically looking for white vertical lines separating the digits. So, number slope or a not enough accurate number rotation complicates the separation between digits.

4.3.2. Solution

The solution consists of analysing vertically each of the lines on the detected numbers area. This way, it is possible to compute the quantity of black pixels in each vertical line. This enables us to detect the lines where there exists a lower density on black pixels. Lines with lower density may be assumed as the gaps between digits and the maximums are located when there is a number, although it is not always true.

The following steps are used to solve number segmentation:

- First step, as said before, calculates the quantity of black pixels in each vertical line. The information is saved in an array. Figure 4.4 shows the information stored in the array.

Figure 4.4 Statistics of the quantity of black pixels per line

- Secondly, the array is traversed to find the maximum density value. This permits us to delimitate the beginning and the end of the number. It is assumed that the vertical line where the number starts is the last traversed array point with value equal to 0 before checking the first point with value equal or higher than $\frac{maximum}{2}$. This process is carried out in the same way but traversing the array in the opposite direction in order to find the end of the number. Then, the noise surrounding the number is eliminated.

- There always exists a maximum value in each array and it will be similar to the other high values representing the digits location. So, the reason to establish a threshold equal to $\frac{maximum}{2}$ is to be sure of the detection of a number.

- Once the number is delimitated, the third step identifies the low peaks in the array. Two pointers, separated by 5 positions, are used to traverse the array calculating the difference value between them. When the array value on the first pointer is lower than the value on the second one, the previous value of the array on the first pointer is checked and if it is higher than the first pointer value, the first pointer value on the array is considered as a low peak. Sometimes some adjacent point fulfil the previous rule, in these cases the minimum values will be the only taken into account. Figure 4.5 illustrates the low peaks detected using this algorithm.

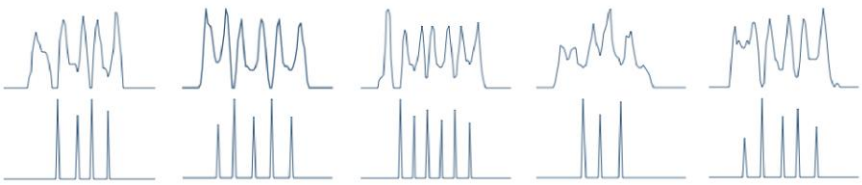
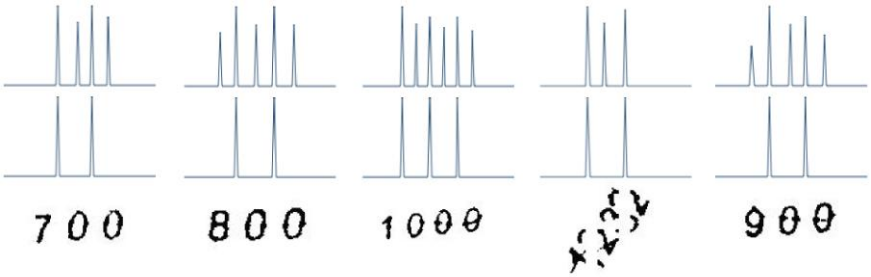


Figure 4.5 Low peaks detection from array information

- Last step tries to detect if the number is formed by 3 or 4 digits (which is the most common). Taking into account the points delimitating the number location, number length is known. For each number two patterns are generated: one for 3 digits and one for 4. The patterns divide the number length in 3 or 4 equal sections and they are compared with the peaks pattern obtained before. The quantity of digits this number has is

determined by the pattern that best matches. Those peaks that match or are close to the chosen pattern will be preserved and used to segment the number (Figure 4.6).



-

Figure 4.6 First row displays the resulting of low peaks detection. Second row shows the pattern that best matches. Third row illustrates the output of number segmentation

4.4. Number recognition

4.4.1. Description

Once the number information has been prepared, it is time to recognize the numbers. Although there exist a large amount of OCR libraries, some exceptions fulfilled the project requirements: must be written in Java, must not be proprietary and must be able to recognize different fonts, even if that means that it has to be trained for each new font.

4.4.2. Solution

Asprise OCR, GOCR and Java OCR, which has been implemented by Ron Cemer, have been tested to check how each one works.

The main idea was to test the libraries for different levels of difficulty; first clear information and then more difficult one. The problem arises due to the fact that many errors were obtained with this first level of difficulty so, these tests were enough to determine the best option. Asprise OCR demo achieved better results but it was discarded because is a proprietary solution. In my opinion, Java OCR achieves better and faster results than GOCR. That is why this library is used.

As can be seen in the project website[7], this library is based on image matching. Through automatic position and aspect ratio correction (using a least-square-error[8] matching), the isolated characters are compared with trained characters.

[7] http://www.roncemer.com/software-development/java-ocr

In this procedure different maps have been tested with varied level of complexity. In function of their characteristics, the results have been very diverse (Table 4.1). In most of them the numbers size is small so, as a consequence, the contour lines greatly affect to the number recognition. Furthermore, the proposed system is very dependent of a well number orientation that is why always using the direction of the contour line below the number is not always the good choice.

Furthermore, an elevation number is not correct if all the digits forming the number are not well recognized. This problem affects the statistics.

Table 4.1 Number recognition tests

	Quantity of digits	Success rate of digits	Quantity of Numbers	Success rate of numbers
Map0	36	19,4 %	12	0,0 %
Map1	32	56,3 %	10	30,0 %
Map2	33	36,3 %	11	9,0 %
Map3	7	42,9 %	3	0,0 %
Map4	45	15,0 %	15	0,0 %
Map5	9	33,3 %	3	0,0 %
Map8	30	30,0 %	10	0,0 %
Map9	12	41,7 %	4	25,0 %
Map12	18	33,0 %	6	0,0 %

The low obtained results have conducted me to avoid the implementation of number recognition in the software application. So, it has been replaced by user's intervention, who has to specify the height difference between contour lines.

[8] Algorithm to look for the smallest difference between information and the predictions of several models, which could be used to describe this information.

CHAPTER 5. CONTOUR MAP REPRESENTATION SYSTEM

This chapter treats different ways to graphically represent the previously processed information. There have been considered two types of representations. One of them used 2D graphics and the other one 3D graphics.

5.1. Coloured Contour Map

The simplest way to represent the terrain elevation is through coloured contour maps. Java 2D™ opens the possibility to display the coloured image in this program. To create a coloured contour map it has been initialized a wide range of colour and every one of them has been assigned to an elevation value. Figure 3.7 demonstrates how a processed contour map can be displayed.

5.2. Augmented Reality

Augmented reality (AR) is a live view of the reality where virtual objects (2D and 3D) are overlaid, besides the user's environment becomes interactive and manipulated. Some of the inputs used in order to show overlapped information are: sound, video, graphics and GPS data. AR term was coined en 1990 by Thomas Caudell.

To work with AR there are necessary some hardware and software components. The hardware includes a processor, a display, sensors and input devices. In the other hand, the software usually needs some methods of computer vision and image processing to locate the 2D or 3D objects over the display that shows the reality.

There exist different types of AR, such as: those projecting the information directly on the screen without taking into account the user's surrounding environment characteristics; those recognizing some kind of patterns and use the information in the pattern to allocate the graphics; and those that through GPS data indicate where the virtual information has to be illustrated. As can be seen, the difference between them lies in the way to locate the position where the information has to be situated.

Next, some fields where this technology is being used are going to be described. However, this is just a small fraction of the applications where AR is implemented.

- Tourism: in this field some of the different types described above can be found. Through GPS data some software applications display information about monuments and other points of interest in a city. Otherwise, other places where the location is complex, like museums and other indoor buildings, AR markers permit to obtain info about pieces of art.

- Navigation: Some of the main automobile manufacturers are developing navigation systems using AR to show forecast and traffic information in the car windshields.

- Entertainment: Everyday more, videogames industries use this technology trying to increase the interactivity between the user and the game. In this field, also in the broadcasting of different events, as sport events, channels use this technology to add extra information to the viewer (scores, adds, etc.).

Before implementing augmented reality, it is necessary to decide what the project requirements are. In this project, the main idea of using augmented reality is to provide the user an easy way to display anywhere a scaled 3D terrain surface from a contour map. Marker-based augmented reality can situate the render 3D objects more accurately than other options, so this is better adapted to the project scenario.

A great deal of libraries are designed to implement AR. ARToolKit, ARTag, NyARToolKit, FLARToolkit, are some examples. Although most of them are a complete port of ARToolKit written in different programming languages. NyARToolKit seems to be the most widely used and the best option for virtual machines (Java, Android, etc.) so this is the reason for its choice.

NyARToolKit needs two external libraries to ensure that it operates properly. One of them has to capture video from an input device and send it to NyARToolKit, and the other one has to render 3D objects. To produce a comprehensive understanding of NyARToolKit, first it is needed to speak about them:

- JavaCV: By default, NyARToolKit depends on Java Media Framework (JMF) to capture video from an input device. This is totally outdated and is only supported by Windows so it has been replaced by JavaCV library. It provides some wrappers to commonly used libraries in the field of computer vision, although only OpenCV (Open Source Computer Vision) has been used. This library permits programming functions for real time computer vision. So this is the tool used to provide data video captured from an input device to NyARToolKit.

- JOGL: It is a Java Binding for the OpenGL library and is designed to provide hardware-supported 3D graphics to software applications written in Java. It is in charge of generating a 3D mesh to render into video frame.

  To translate the 2D map into 3D object a mesh is used. The grid obtained on the previous processes is employed to build it. This mesh is formed by quadrilaterals, each one built up out by two triangles. Each side of the triangles joins two vertices (Figure 5.1). The vertices correspond to the pixels on the 2D image and the cells on the grid. Their vertical and horizontal coordinates are *x* and *y* on the mesh.

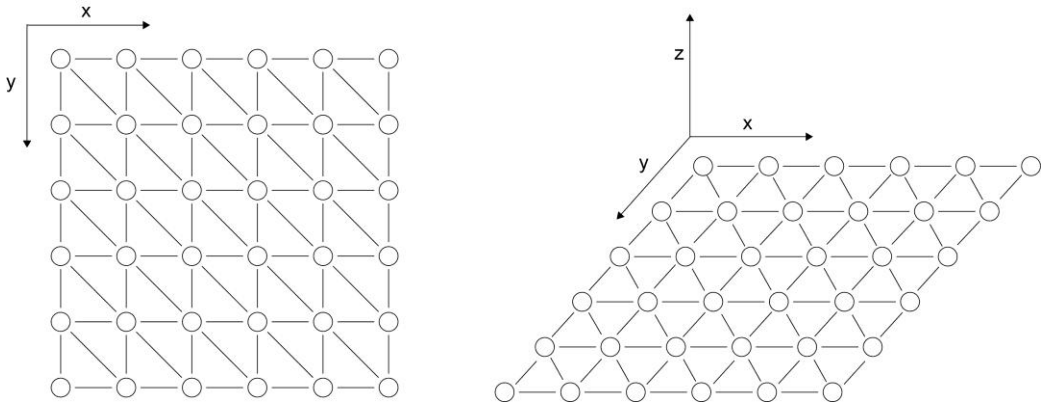


Figure 5.1 2D and 3D top-views of a mesh fragment, where circles are the vertices and segments represent the links between them

The *z* component gives the elevation value on each vertex, which is obtained from the grid cells. Figure 5.2 shows the mesh achieved from the sample image used in this document.

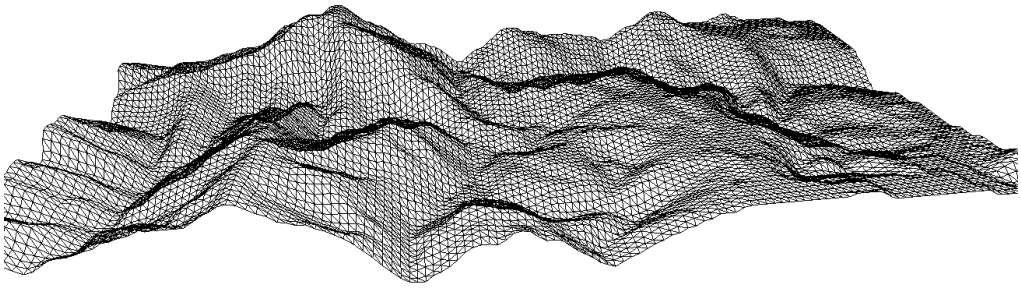


Figure 5.2 Terrain surface sample

Once library dependencies have been clarified, the description of NyARToolKit can be carried out. When JavaCV captures video, it sends the data frame to NyARToolKit. Then, this information is analysed to identify any square shapes. If a square is detected, it is checked if this square matches with markers in memory. If so, mathematical functions are used to calculate the camera position and orientation relative to the marker. Finally, the 3D mesh provided by JOGL is rendered and overlaid onto the capture image.

Next diagram, extracted from ARToolKit website[9], summarizes how most of marker-based AR libraries work.

[9] http://www.hitl.washington.edu/artoolkit/

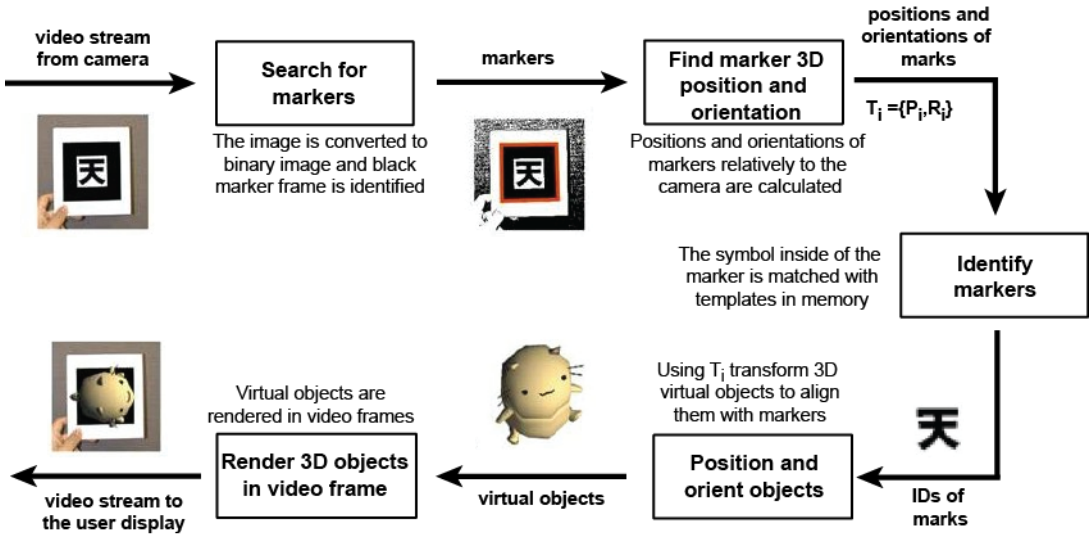


Figure 5.3 Marker-based augmented reality diagram

Figure 5.4 is an example of how augmented reality implementation overlays the processed surface with the image captured from a webcam.

Figure 5.4 Mesh representations using augmented reality

CHAPTER 6. RESULTS

The purpose of this chapter is to show the obtained results in this project. Unlike the previous chapter, these results are going to be studied as a whole in an analytical way.

For that, there have been used 15 contour maps with different properties, which can involve a higher complexity during the processing. Some of these properties are: contour lines and numbers quantity, image size, closeness between contours, number size respect the line thickness, etc.

Before analysing the results, it is needed to detail the testing scenario. Tests have been made on a MacBook Pro with 2.66 GHz Intel Core 2 Duo processor and 8 GB RAM.

Table 6.1 shows some of the main properties in examined contour maps, as for example: number of contour lines, number of gaps generated in number extraction, success rate in reconstruction stage and the total computing time to process the image.

Although errors in image can exist for many reasons, only the output will be affected in case these errors lead to wrong reconstruction. Basically, in reconstruction stage two types of errors can be found: those errors produced in matching step or, the most common ones, when the matching step works properly but the drawn Hermite spline overlaps with other (when there are steep slopes and contour lines are very close). Any of these errors implies a mistake in acquisition step because two contour lines are merged, so the elevation values in those contour lines within it could be affected. Despite this, the overall success rate (number of correct connections from the total number of gaps) is very high, 98.98 %.

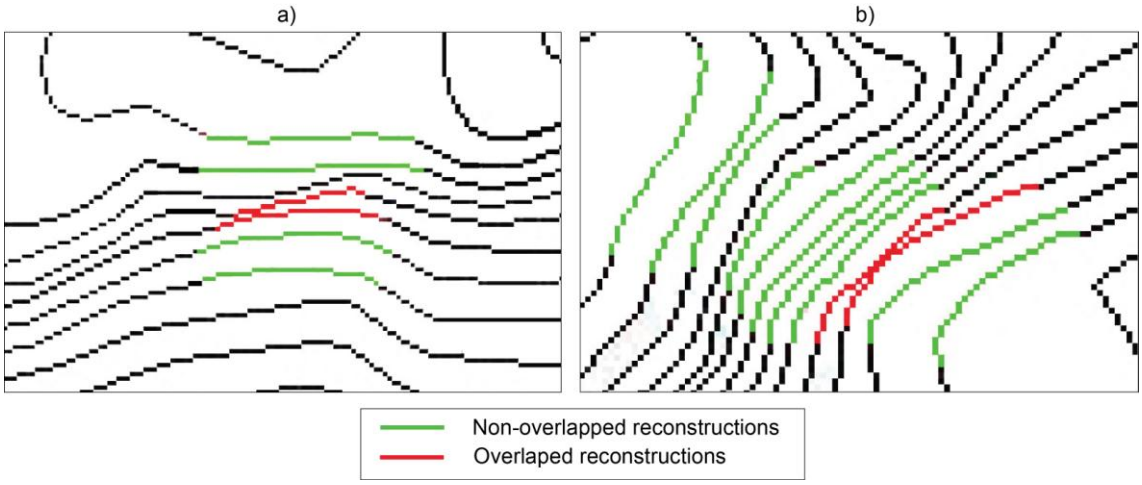


Figure 6.1 Wrong connections in reconstruction stage

As previously stated, the most likely reason for obtaining mistakes is when the proximity between contours is high, it translates into higher complexity level once the reconstruction has to be carried out. In Figure 6.1 can be seen the complexity found in some areas and the type of errors produced due to this. Sometimes contours are so close that only seems to exist a single valuable path for the connection; this is why overlapping is happening.

Moreover, using some of the contour maps, for example the first ten, where errors have not been produced (trying to not alter the times using wrong results), it has been analysed the computing time for each main process. As it can be seen in Figure 6.2, those procedures included in acquisition system are not very influenced by the image properties (the line trend is very similar in all cases) and do not imply much time. However, in procedures as interior point location, filling and matching contour lines (included in the contour map processing), the computed time can change a lot in function of the image properties. It is noted that filling algorithm is, by far, the procedure that contributes the most to the whole computing time.

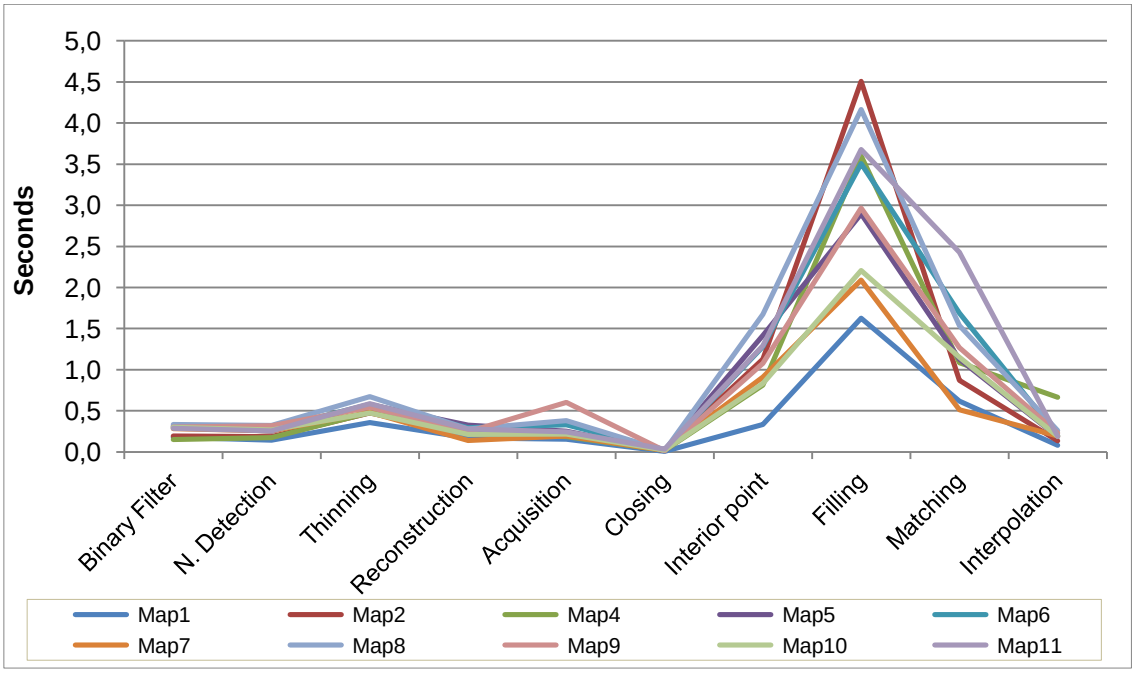


Figure 6.2 Computing time in main procedures

Table 6.1

	Image size	No. of contours	No. of gaps	Correct connections		Wrong connections		Success rate *	Computing time
				Returned Information	Regenerated Information	Matching errors	Overlapping errors		
Map 1	1362 x 634	94	141	78	63	0	0	100	3.67 s
Map 2	1362 x 634	116	194	156	38	0	0	100	7.97 s
Map 3	1362 x 634	118	203	146	55	0	2	96.4	4.64 s
Map 4	1362 x 634	82	113	92	21	0	0	100	7.41 s
Map 5	1899 x 927	129	139	89	50	0	0	100	7.36 s
Map 6	1899 x 927	113	178	127	51	0	0	100	8.49 s
Map 7	1899 x 927	63	26	16	10	0	0	100	5.07 s
Map 8	1899 x 927	135	150	87	63	0	0	100	9.64 s
Map 9	1875 x 905	145	110	56	54	0	0	100	7.57 s
Map 10	1875 x 905	97	92	68	24	0	0	100	5.88 s
Map 11	1875 x 905	203	203	121	82	0	0	100	9.28 s
Map 12	1872 x 905	140	187	144	40	0	3	93	7.89 s
Map 13	1872 x 905	97	189	128	59	2	0	96.7	6.81 s
Map 14	1872 x 905	97	86	60	26	0	0	100	6.33 s
Map 15	1875 x 905	80	142	97	45	0	0	100	5.85 s

(*) To compute the success rate there only have been considered those gaps reconstructed by means of the regeneration procedure; ‘Returned information’ column has not been taken into account.

CHAPTER 7. GRAPHICAL INTERFACE

To use the different procedures explained in chapters 2 and 3 and show the resulting information as in chapter 4, a simple Graphical User Interface (Figure 7.1) has been designed.

The design has been motivated by the user's ability to understand how the software application is working without needing previous information. The fact that the quantity of functionalities is low also helps for this objective.

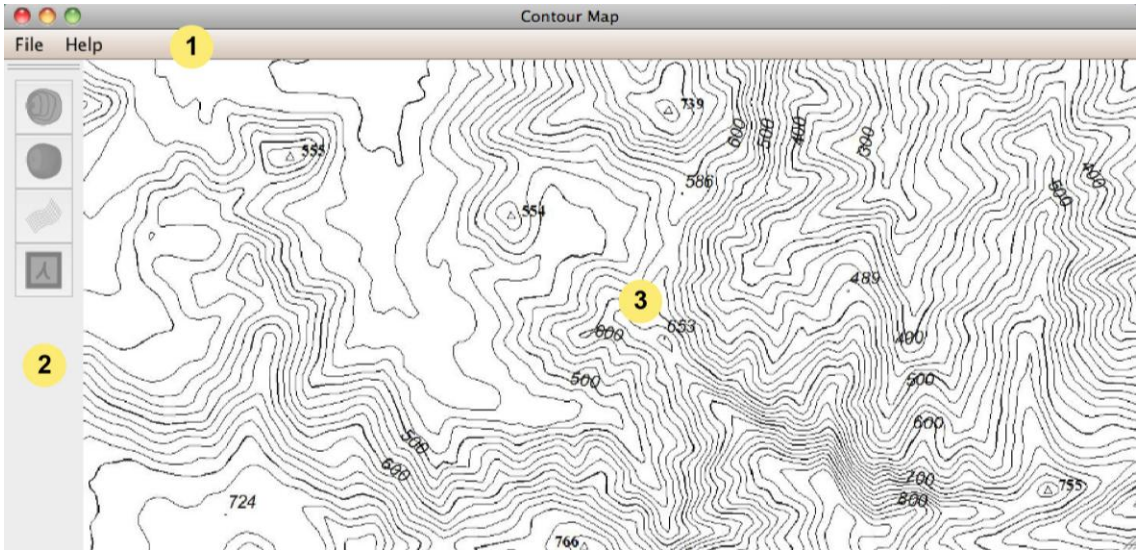


Figure 7.1 Contour Map GUI

The contour map application is made up of three main components:

1. Main menu → consist of two submenus: File Menu and Help Menu. File Menu allows both import files and close the software application. On the other hand, Help Menu shows information about the software (Help > About Project).

2. Toolbar → shows the different ways to represent the information according to the chosen file.

3. Terrain board → canvas where the terrain information is displayed based on the previous elections.

This software application has been thought to enable the user to import JPG images (Contour maps) and ASCII files (DTM). The procedure for processing both options will be exactly the same, although the software application capabilities (Toolbar) will change. To import files, the steps below have to be followed:

- In the main window, choose File > Import
- In file list, select the desired option: Contour Map or DTM file. The import dialog box appears.
- Double-click the file or select the file and click accept.

Although reading, interpreting and representing DTM information was not a goal in this project, this functionality has been added to allow the user managing more types of terrain information and, most important, providing the user with a tool for comparing the obtained results with the obtained ones from image processing.

Once any of the two possible options have been imported, next step consists of starting its processing. For that, in case a contour map is chosen, the terrain board will display the map next to a start/play button (Figure 7.2 a). Otherwise, it will work in the same way but the image shown will be a sample map. Obviously, image or file processing will start when the start button is pressed. An animation will appear while the processing is being carried out (Figure 7.2 b).

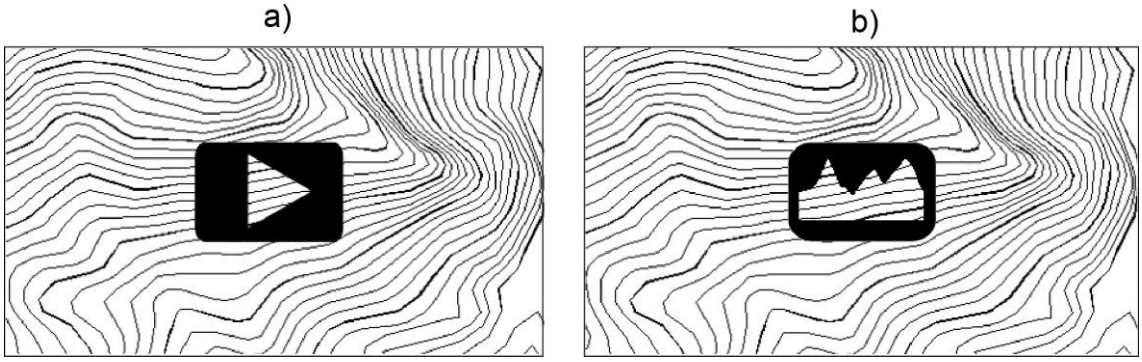


Figure 7.2: a) Screenshot from start button b) Screenshot from progress animation

As described above, the representing capabilities will change taking into account the type of imported file. Toolbar (Figure 7.3) is in charge of managing these capabilities or tools, enabling or disabling them.

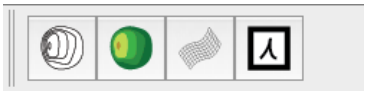


Figure 7.3 Contour Map Toolbar

Four buttons for the surface representation form the toolbar:

Contour map button: using Java 2D the contour map chosen is displayed.

Coloured contour map button: after the image has been processed the output is shown using Java 2D. See section 5.1.

3D surface mesh button: the DMT grid or the computed grid from the contour map is displayed using JOGL. See section 5.2.

Augmented reality button: the DMT grid or the computed grid from the contour map is overlaid on a real-time view of the reality using NyARToolKit. See section 5.2.

Two combinations can be possible on toolbar. If a contour map has been chosen all tools are available because all the information has been processed. Otherwise, when a DTM file is selected contour map and coloured contour map buttons are disabled. This is because the contour map information is not available, so the image has not been processed, then the coloured image has not been computed.

Moreover, in case a contour map is being used, the map and the coloured map images are loaded as textures. Then, when the terrain information is shown through a 3D mesh, a key listener is implemented for applying the different textures. The idea of using the map information is to check whether the 2D information matches with the calculated 3D information, but the main disadvantage is that the feeling of depth sometimes is lost. However, coloured map is intended to provide the user a more realistic way to check the elevation changes. To achieve this goal, three keys are going to be used:

- Pressing D key the textures are disabled, so the mesh is displayed. It is the default setting (Figure 7.4 a).
- Pressing M key the contour map texture is applied (Figure 7.4 b).
- Pressing C key the coloured contour map texture is applied (Figure 7.4 c).

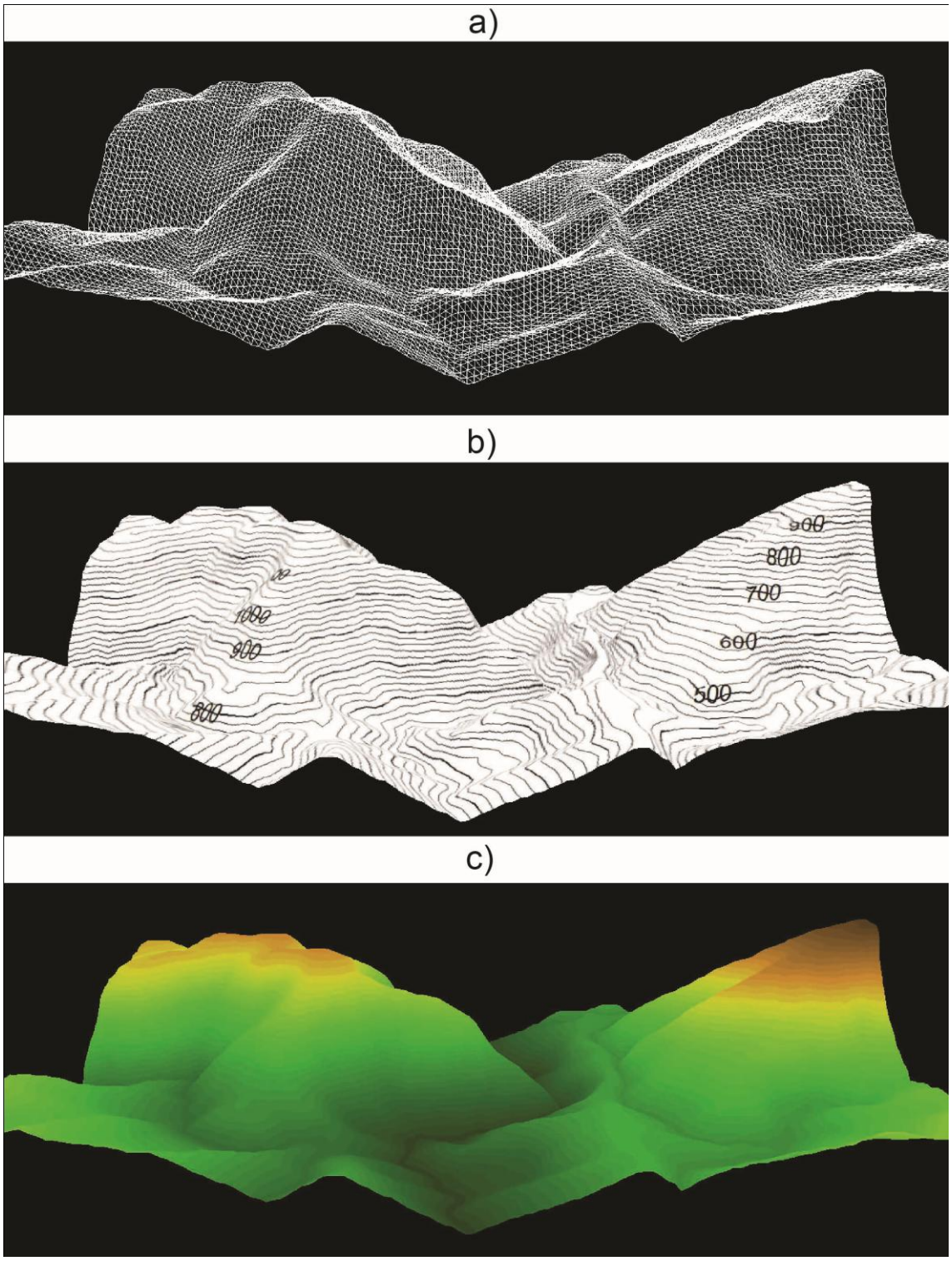


Figure 7.4 Different options for displaying meshes: a) disabled textures, b) contour map texture and c) coloured contour map

CONCLUSIONS

After some months working on this project I can affirm that I am very satisfied with the obtained results. It is true that looking back at the initials objectives, it is seen that some of them have suffered changes. These modifications have been a consequence to the different issues appeared in the way to achieve the purpose.

Initially, the main objective of this project was to process any type of topographic map to obtain the terrain information. The problem arises when it is seen that most contour maps show a great variety of information and some is printed on them in the same colour range so, isolating it is impossible.

The most striking case is that the colour range used to draw transportation routes (routes, motorways) and contour lines is very similar. Binary contour maps, which are less common, have been used in order to avoid this problem. They only contain topographic information (contour lines and elevation numbers).

Undoubtedly, the issues that led to more problems have been the reconstruction step, and the elevation numbers processing for their posterior recognition. A lot of literature has been studied about these topics.

For the reconstruction step, different solutions to specific cases have been found, but none of them suited our scenario. One interesting solution proposes a set of steps to solve the reconstruction issue in a general way. The treated concepts seemed to be capable of adapting to the project requirements. However, the obtained results shown in the paper are carried out over fake and extremely simple contour maps so, once it was tried over the maps used in my project, the results were not the estimated. As no solution was found which satisfied our specific scenario, an own solution was implemented.

As for the elevation number processing, it was needed to study the different ways the number is printed on the image. It was seen that it doesn't exist an only way to represent them: those that are drawn over the contour line (the contour line crosses the number), those that are printed over the contour line but there is a gap between the contour lines and number and those written on the contour line. For this reason, I contacted with *"Instituto Geográfico Nacional",* to know if any of this options was more widely used. I was recommended to use the first one, because the third is not frequently used and the two first are equal but in the second option the number information is overlaid in a second layer so, the contour lines is hidden. In mi opinion, the recommended option is the most complex one because it is so difficult to isolate the contour lines from the numbers. That is why, possibly, the results have not been the expected and have not been included on the software application. I believe number processing and recognition would need a whole project for itself, because of the complexity and required time to treat it.

All the problems explained above have introduced changes in the initial objective. However, the main objective has been fulfilled but numbers recognition. Pathways forward could lead to develop a better number recognition system, especially, the procedure in charge of detecting the number slope.

REFERENCES

[1] T. Ghircoias and R. Brad, “Contour lines extraction and reconstruction from topographic maps”.

[2] S. Salvatore and P. Guitton, “Contour line recognition from scanned topographic maps”.

[3] R. Pradhan, S. Kumar, R. Agarwal, M. P. Pradhan and M. K. Ghose, “Contour line tracing algorithm for digital topographic maps”.

[4] F. C. Crow, “Summed-area tables for texture mapping”.

[5] L. Huan, G. Wan and C. Liu, “An improved Parallel thinning algorithm”.

[6] T. Y. Zhang and C. Y. Suen, “A fast parallel algorithm for thinning digital patterns”.

[7] N. Amenta, M. Bern and D. Eppstein, “The Crust and The B-Skeleton: Combinational Curve Reconstruction”.

[8] A. Martínez and L. Gerardo, “Terrain Reconstruction from Contour Maps”.

[9] J. Pouderoux and S. Spinello, “Global Contour Lines Reconstruction in Topographic Maps”.

[10] T. J. Watson, “Area Fill Algorithms,” Binghamton University EnginNet.

[11] W. c. “Flood Fill,” Wikipedia, The Free Encyclopedia.

BIBLIOGRAPHY

- Cardani, Darrin. *Adventures in HSV Space*
http://visl.technion.ac.il/labs/anat/hsvspace.pdf

- *Computer Graphics- JOGL Tutorials and examples*. Laboratorio di Robotica. Università degli Studi di Pavia
http://robot.unipv.it/index.php/didattica/grafica-3d/54

- El mapa topográfico
http://olmo.pntic.mec.es/esam0009/Actividades/mapa%20topografico.pdf

- Free GIS Data- GIS Data Depot
http://data.geocomm.com/

- GOCR- Open Source Character Recognition
http://jocr.sourceforge.net/

- Instituto Geográfico Nacional
http://www.ign.es/ign/main/index.do

- Institut Cartogràfic Català
http://www.icc.cat/

- JavaCV. Java interface to OpenCV and more. Google Project Hosting
http://code.google.com/p/javacv/

- JOGL (Java OpenGL) Tutorial.
http://www.land-of-kain.de/docs/jogl/

- LAB Asprise! *Asprise OCR SKD v4.*0
http://asprise.com/home/

- NeHe Productions. Everything Open GL.
http://nehe.gamedev.net/

- NyARToolkit project
http://code.google.com/p/javacv/

- NyARToolkit- 2.3.1 for Java on Snow Leopard (using OpemCV/JNI)
http://edotprintstacktrace.blogspot.com.es/2009/09/nyartoolkit-231java-on-snow-leopard.html

- Pascale, Danny. (2003) *A Review of RGB Color Spaces*
http://www.babelcolor.com/download/A%20review%20of%20RGB%20color%20spaces.pdf

- Using a Topographic Map
http://geology.isu.edu/geostac/Field_Exercise/topomaps/topo_interp.htm

- What are the rules of contour lines
http://wiki.answers.com/Q/What_are_the_rules_of_contour_lines