3D RECONSTRUCTION OF VEGETATION FROM ORTHOPHOTOS

BACHELOR'S THESIS

Author

MARC COMINO TRINIDAD

Director:
CARLOS ANDÚJAR GRAN
Department of Software (LSI)

Co-Director:
ANTONIO CHICA CALAF
Department of Software (LSI)

June 21th, 2013

Computing Specialization
Bachelor Degree in Informatics Engineering

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Acknowledgements

In the first place, I would like to express my deep gratitude to Professor Carlos Andújar and to Professor Antonio Chica, my research supervisors. Professor Andújar was the one who welcomed me with his arms wide open, when I asked him about a final degree work, and also he proposed the project that I am presenting on this document. Professor Chica, after learning about the subject of the work, offered himself to help during the process, regardless that he was supervising few other projects. Also, he was the one who proposed the algorithm that we have used for the generation of 3D vegetation models. I must offer them my special thanks, for their patient guidance, enthusiastic encouragement and for their continued advice.

Also, I am particularly grateful for the assistance given by Professor Luís Belanche on topics related to machine learning. His willingness to give his time so generously has been very much appreciated.

Finally, I wish to thanks friends and family for their moral support, because with them by my side, there is no stressful situation that I can not overcome.

Abstract (English)

There is much work done in the field of extracting information, from remotely sensed images, for further analysis. These analysis range from differencing industrial and residential zones, in a city, and finding phytoplankton in a water mass, to differencing different kinds of vegetation. Also, the information extraction methodologies are are wide and diverse.

On a totally different topic, in the literature there is also a wide variety of techniques for the generation of 3D models of vegetation, that range from procedural algorithms, rule based algorithms, hybrid algorithms, ... to mass producing them, to generate the most utopic forests.

However, there is little work done in taking techniques from both fields of knowledge, and mixing them to elaborate an augmented reality application. In this study, we aim to use remote sensing techniques to analyze the information found in an orthophoto (concretely, the flora) and, then, use procedural techniques to generate 3D models that will represent the detected plants. If this is done correctly, in the end we want to obtain a feasible 3D reconstruction of the vegetation found in the orthophoto.

To attain this goal, several small steps will be done, starting by computing a segmentation of the orthophoto to separate the different vegetation classes. For this, we will rely upon the field of supervised learning, to perform a pixel based classification (which is a traditional methodology in the field of remote sensing). However, to carry out this kind of technique, we need a set of values, or features, to describe each pixel.

The most basic set features would be the pixel RGB values, nevertheless, with only this few information, the process is sure not to yield an acceptable accuracy. For this reason, different higher order features are computed to improve the classification performance. Among these can find statistical methods (grey level coocurrence matrix), filter techniques (energy filters), multiresolutional analysis techniques (wavelet decomposition), etc. Another key matter is to find the proper parameters that optimize a these descriptors for a specific application, like the neighbourhood size, the distance between pixels, etc. Local search to find some of these optimal parameter is also carried out.

On a second step, we aimed to program procedural techniques to generate the necessary plant models, to represent the detected vegetation during the classification. It was decided to only generate a reduce set of them, and generate the variability later.

In the final application, we integrated the results of the classification with the 3D models. We adapted each model to each detected plant, using basic transformations, and added the foliage and the coloring for the trunks. The whole process has proven to generate a plausible reconstruction of the vegetation found in the processed orthophoto.

Abstract (Catalan)

Hi ha molta feina feta en el camp de l'extracció d'informació, a partir d'imatges obtingudes per teledetecció, per al seu posterior anàlisi. Aquests anàlisi van des de diferenciar entre zones industrials i zones residencials, d'una ciutat, i la cerca de fitoplàncton en una massa d'aigua, fins a distingir diferents tipus de vegetació. A més, les metodologies d'extracció d'informació són àmplies i diverses.

Per altra banda, en la literatura també hi ha una àmplia varietat de tècniques de generació de models de vegetació 3D, que van des d'algoritmes procedurals, algoritmes basats en regles de producció, algorismes híbrids, ... fins a la producció massiva d'aquests, per a generar els boscos més utòpics.

No obstant això, hi ha poc treball realitzat en agafar tècniques d'ambdós camps i ajuntar-les per elaborar una aplicació de realitat augmentada. En aquest estudi, el nostre objectiu és utilitzar tècniques de teledetecció per analitzar la flora d'una ortofoto i, a continuació, utilitzar tècniques procedurals per generar models 3D per representar les plantes detectades. Si això es fa correctament, el resultat serà una reconstrucció 3D factible de la vegetació detectada en l'ortofoto.

Per assolir aquest objectiu, començarem segmentant l'ortofoto, per separar les diferents classes de vegetació. Per a dur a terme aquest procés, recorrerem al camp de l'aprenentatge supervisat, i realitzarem una classificació píxel a píxel (una de les metodologies tradicionals en el camp de la teledetecció). No obstant això, per dur a terme aquest tipus de tècnica necessitem un conjunt de valors o característiques que descriguin cada píxel.

El conjunt de característiques més bàsiques és els valors RGB del píxel, però, només amb aquesta poca informació, el procés segurament no tindria una precisió acceptable. Per aquesta raó, calculem diferents característiques d'ordre superior, millorant el rendiment de la classificació. D'entre aquests, hi han mètodes estadístics (matriu de coocurrència de nivells de grisos), tècniques de filtració (filtres d'energia), tècniques d'anàlisi multiresolucional (descomposició wavelet), etc. Una altra qüestió clau és trobar els paràmetres adequats que optimitzen aquests descriptors per a una aplicació específica, com podrien ser la mida del veïnatge, la distància entre píxels, etc. També hem dut a terme una cerca local per trobar l'òptim d'algun d'aquests paràmetres.

En un segon pas, hem programat tècniques procedurals per generar els models de plantes necessaris per representar la vegetació detectada durant la classificació. Es va decidir que només es produiria un conjunt reduït d'ells i que, més tard, es generaria la variabilitat.

En l'aplicació final hem integrat els resultats de la classificació amb els models 3D. Hem adaptat cada model a cada planta detectada, utilitzant transformacions bàsiques, i hem afegit el fullatge i l'acoloriment als troncs. Mitjançant aquest procés, acabem generant una reconstrucció plausible de la vegetació que es troba a l'ortofoto processada.

Abstract (Spanish)

Hay mucho trabajo hecho en el campo de la extracción de información, a partir de imágenes obtenidas por teledetección, para su posterior análisis. Estos análisis van desde diferenciar entre zonas industriales y zonas residenciales, dentro de una ciudad, y la búsqueda de fitoplancton en una masa de agua, hasta diferenciar diferentes tipos de vegetación. Además, las metodologías de extracción de información son muy amplias y diversas.

Por otra parte, en la literatura también hay una amplia variedad de técnicas de generación de modelos 3D de vegetación, que van desde algoritmos procedurales, algoritmos basados en reglas de producción, algoritmos híbridos, ... hasta la producción masiva de estos, para generar los bosques más utópicos.

Sin embargo, hay poco trabajo realizado en coger técnicas de ambos campos y juntarlas para elaborar una aplicación de realidad aumentada. En este estudio, nuestro objetivo es utilizar técnicas de teledetección para analizar la flora de una ortofoto y, a continuación, utilizar técnicas procedurales para generar modelos 3D con el fin de representar las plantas detectadas. Si esto se hace correctamente, el resultado será una reconstrucción 3D factible de la vegetación detectada en la ortofoto.

Para alcanzar este objetivo, empezaremos realizando una segmentación sobre la ortofoto, para separar las diferentes clases de vegetación. Para llevar a cabo este proceso, recurriremos al campo del aprendizaje supervisado, y realizaremos una clasificación píxel a píxel (una de las metodologías tradicionales en el campo de la teledetección). Sin embargo, para llevar a cabo este tipo de técnica necesitamos un conjunto de características que describan cada píxel.

El conjunto de características más básicas son los valores RGB del píxel, pero, sólo con esta poca información, el proceso seguramente no tendría una precisión aceptable. Por esta razón, calculamos diferentes características de orden superior, mejorando el rendimiento de la clasificación. Entre estos, hay métodos estadísticos (matriz de coocurrencia de niveles de grises), técnicas de filtrado (filtros de energía), técnicas de análisis multiresolucional (descomposición wavelet), etc. Otra cuestión clave es encontrar los parámetros adecuados que optimizan estos descriptores para una aplicación específica, como podrían ser el tamaño de la vecindad, la distancia entre píxeles, etc. También se ha llevado a cabo una búsqueda local para encontrar el óptimo de alguno de estos parámetros.

En un segundo paso, hemos programado técnicas procedurales para generar los modelos de plantas necesarios para representar la vegetación detectada durante la clasificación. Se decidió que sólo se produciría un conjunto reducido de ellos y que, más tarde, se generaría la variabilidad.

En la aplicación final hemos integrado los resultados de la clasificación con los modelos 3D. Hemos adaptado cada modelo a cada planta detectada, utilizando transformaciones básicas, y hemos añadido el follaje y el coloreado en los troncos. Mediante este proceso, acabamos generando una reconstrucción plausible de la vegetación apreciable en la ortofoto procesada.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Nowadays, many known applications offer the user the possibility to visualize a certain desired region of a virtual globe. Some examples of such applications are ICC's Vissir3[1], Google Earth[2], NASA World Wind[3] and Bing Maps[4]. The first one only displays orthophotos, whereas the other 3 can display 3D views of certain regions. The aim of this usually is to let the user explore those places of interest as "if they were there´´. Usually, relevant features such as buildings in a city are modelled in detail whereas other details such as vegetation are usually modelled poorly, as shown in Figure 1.1.

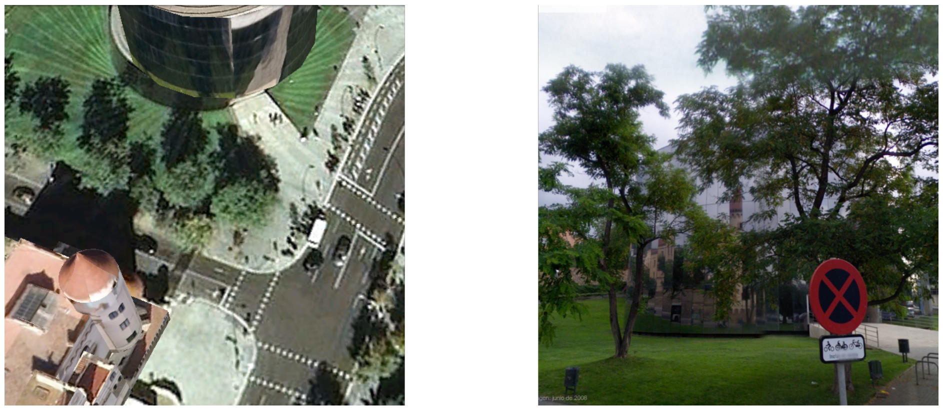

Figure 1.1: Left: Aerial view of trees near the Nexus building (UPC), extracted from Google Earth. Right: Frontal view of trees near the Nexus building (UPC), extracted from Google Street View. We can observe that there is an effort to model the building in high detail, whereas a plain zenithal view is used for the vegetation. Notice that the zenithal view does not contain enough information to explain the whole geometry of a tree.

The aim of this project is to find a solution for the reconstruction of 3D vegetation using just information extracted from orthophotos. Our goal is not to elaborate models that faithfully represent reality, since we are lacking important information of the real geometry (Figure 1.1). Instead, we aim to build plausible models compatible with the aerial view.

Such application could be used to improve the realism of existing virtual globe applications, such as the ones mentioned above. Also, it could in a future evolve to applications aimed to reproduce real vegetation zones with certain plausibility. For instance, 3D visualization of natural parks for divulging purposes or realistic reconstruction of real natural environments for games.

1.2 Context and Stakeholders

In this section we justify this project using the different motivations that each stakeholder holds. The main stakeholders of this project are the author of this project, the director and codirector of this project and the research group to which they belong, and the potential clients.

1.2.1 Author

The author of this project is one of the most important stakeholders because his aim is to finish a working system within the deadline, to finish his degree studies. This will motivate him to invest the required amount of time and effort to attain the previously set goals.

1.2.2 Director and Codirector

The director and codirector of the project and the research team to which they belong have developed a system to model roads and their surrounding landscape. For this, they focus on the reproduction of the shape of the terrain and on the reproduction of the surrounding vegetation.

Now, they are interested on further improving the realism of the landscapes they produce and, for this, they proposed this project. Our aim is to build an independent system to generate feasible 3D landscapes of the vegetation of a region, only using the information extracted from an orthophoto. However this will be done in a way that in a future it could be integrated in their system. Therefore, their needs and suggestions are to be taken into account during the whole process.

1.2.3 Potential Clients

We can identify three types of potential clients: those that might want to improve an existing system of their own, those that might want to turn our results into a new application and research groups.

First, we must remember that we are researching how to solve a certain problem, and we will try to implement our results in a prototype application. Because of this, our main clients will probably be other research groups on computer graphics or computer vision.

Second, applications such as Google Earth[2], NASA World Wind[3] or Bing Maps[4] can display 3D views of certain places of interest worldwide. However, in their views they usually only show 3D models of buildings or terrains and do not take much effort on displaying feasible realistic vegetation. We claim that the results of our project could be used to improve the realism of these applications because we aim to refine each model to fit each particular plant. This will produce a great variance of models, which appears more realistic than using the same model every time. Also, the process will be automatic and because of this its cost will be low.

In the third place, we envisage that our system would be of interest for entities wanting to advertise certain natural environments such as natural parks. Our results could evolve into an application able to reproduce certain vegetation zones with certain plausibility, to let the user get a first impression of what could be found there. Also, the game industry could take profit of our work to reproduce real environments.

1.3 Objectives

The result of this project will be an application that, given an orthophoto as an input, produces a feasible 3D landscape matching the vegetation represented on the orthophoto. Therefore, the main objective of this project is the development of the described system within the deadline, June 2013. Nevertheless, to attain this, the following goals will have to be achieved first:

- Develop a classifier able to predict the vegetation class of a pixel of an orthophoto, with relative **accuracy**.
- Build **feasible** 3D models for each vegetation class.
- Choose and refine a 3D model for a detected plant, in a way that the result looks feasible for the **human eye**.

Finally, it is an important objective to do all this in a way that every part, or the whole system, is **reusable and integrable** in other applications.

1.4 State of the Art

1.4.1 Similar Applications

Google Earth, after launching version 6, supports the visualization of 3D trees in certain places of the world[5]. However, what Google Earth can offer is far from what we expect to be results of this projects for a number of reasons:

- First of all, there are a couple of desired features that this application does not support, such as rendering other types of vegetation (like shrubs or grass) or using the orthophoto's color to determine the resulting trees color, as can be seen in Figure1.2.

- Second, we have no knowledge of how the recognition and placing of the trees has been done. Nevertheless, we want to use machine learning techniques to do this process in an automatic way.
- Third, Google Inc. is a private company, therefore it is unlikely that they will share any information on how they have elaborated this application. Therefore, we can not relay on this system if we want to do further research on the field.

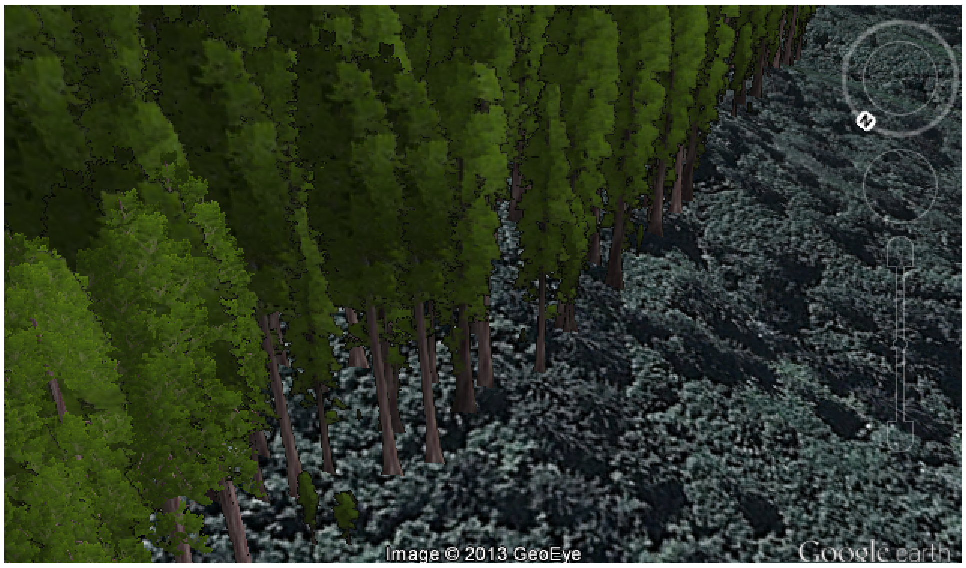

Figure 1.2: As can be seen in the picture, the color of the leaves of the trees is completly different to the color shown in the orthophoto. Image extracted from Google Earth.

1.4.2 Solution

This project has been designed as a pipeline. Different pieces of software have been designed to solve small problems specific problems and, combined together, they can solve the problem of rendering a feasible 3D landscape of the vegetation found in an orthophoto.

The first piece of the pipeline is piece of software that aims to classify each pixel of an orthophoto into a kind of vegetation. For the elaboration of the classifier, different Machine Learning methods have been tested. However, without proper features for each pixel the classifier sentenced not to work properly so we turned to the field of Computer Vision to extract those.

The second piece of the pipeline is aimed to process a cloud of classified pixels (or points) and identify individual plants or trees in them. For this process, other Computer Vision techniques have been used to remove part of the noise generated by the classification process, and Unsupervised Learning techniques have been use to group pixels into separated individual plants.

The third part of the pipeline will serve the purpose of procedurally generating 3D models of different kinds of plants, shrubs and trees. For this, different algorithms and techniques related to the field of Computer Graphics will be used.

Finally, the last part of the system will take the information generated by the second phase and the models generated in the third phase, and will build and display the final 3D view of the landscape.

1.4.3 Project Context: Texture descriptors

For the first part of our project we need to elaborate a vegetation classifier, therefore research on suitable texture descriptors has been done. In [6] T. Pouli, D. W. Cunningham and E. Reinhard have given an overview of the most relevant image statistics. These are divided as follows:

- First-Order Statistics, which involves the analysis of the properties of single pixels. For instance, pixel histograms.
- Second-Order Statistics, which involves studying the relationships of pairs of pixels.
  - Fourier Transform.
  - Power Spectra.
  - Gradients.
  - Others.
- Higher-Order Statistics, which involves searching for statistical regularities in larger groups of pixels.
  - Phase Structure.
  - Wavelets.
  - PCA.
  - Others.

Whereas statistics described in [6] are relevant in general in computer graphics, the MPEG-7 visual standard for content description[7] describes another set of statistics specifically designed to measure similarity in images or videos and, therefore, can be used to efficiently identify, filter or browse images. The descriptors are divided as follows:

- Visual Color Descriptors.
  - Color Spaces.

    - Scalable Color Spaces (SCD).
    - Dominant Color Descriptor.
    - Color Layout Descriptor (CLD).
    - CSD.

- Visual Texture Descriptors.
    - Homogeneous Texture Descriptor (HTD).
    - Non-Homogeneous Texture Descriptor (Edge Histogram or EHD).

- Visual Shape Descriptors.
    - Shape Descriptor - Shape Spectrum.
    - Region Based Descriptor (ART).
    - Contour-Based Shape Descriptor.
    - 2D/3D Shape Descriptor.

Finally, L. A. Ruiz, A. Fdez-Sarría and J.A. Recio [8] conduct a series of experiments to test another set descriptors and its utility to classify images similar to the ones we will be used in our project. They test usefulness of four sets of descriptors:

- Grey level coocurrence matrix (GLCM).
- Energy filters and edgeness.
- Gabor filters.
- Wavelet transform.

They also give information about the best estimated parameters to calculate each descriptor and their results are promising. Therefore we will be most likely using their descriptors, alongside other ones, during the elaboration of our project.

For general guidelines about how to implement the mentioned descriptors we will be using books such as Digital Image Processing[9] or Computer Vision[10]. Also documents, such as [11] describing the Haar Wavelets, will be used for this purpose.

Once the appropriate descriptors have been selected and calculated, then *machine learning* techniques will used to train the most appropriate classifier. We will experiment with different models, such as the Linear Discriminant Analysis, the Quadratic Discriminant Analysis, etc. These, and more, are explained in Pattern Recognition[12].

1.4.4 Project Context: Vegetation 3D Models

Quoted from the preface of O. Deussen and B. Lintermann's book[13] "... During the past four decades, computer graphics has evolved into an innovative multifaceted field of research and computing that affects many other sciences ... Within computer graphics, the area of modeling and rendering synthetic plants and landscapes that is covered in this book is a rather interdisciplinary field. Here a knowledge of botany and ecology is necessary, and aside from the mathematical and physical laws, artistic aspects are of utmost importance."

For a long time, botanists and computer engineers have worked together aiming to develop different methods for the generation of feasible vegetation models. Throughout [13] the authors try to summarize their work on the field, and three major topics are treated: methods for the modeling of single plants, methods for the modeling of plant communities and landscapes and rendering methods for such synthetic landscapes. As for now, we are only interested in the modeling of single plants.

The authors divide the methods for generating 3D models of single plants into *procedural* or *rule based.* The former ones consist on parameterized algorithms that are designed for the production of a certain type of plant, they can be seen chronologically throughout chapter 4, and the later ones are procedures that through the application of rules transform a simple initial states into a complex final state, and they can be seen throughout chapter 5. Also, an hybrid combination of both kinds called *rule-based object production* can be seen in chapter 6.

Another interesting fact about [13] is that a whole chapter is dedicated to give a brief introduction to the relevant botanical concepts for the generation of 3D models of plants, such as branching structures, budding, spatial division, leaves or plant communities. The result of all concepts explained can be seen in their web page [14]. An example is shown in Figure 1.3.

One of the concepts mentioned above, spatial division, states that a plant tries to optimize its growth. For this, during their growth process they tend to occupy all the available space where light reaches, this is the basic idea for another procedural method for the generation of 3D vegetation called *Space Colonization* described by A. Runions, B. Lane and P. Prusinkiewiez[15]. They describe the tree architecture in terms of competition for space, and steps of their algorithm can be intuitively seen in 1.4. Their algorithm takes as input different parameters, which result in producing a wide variety of models, as shown in Figure 1.5.

1.4.5 Innovation

So far, we have explained part of the work done in different fields such as vegetation classification or 3D rendering of vegetation. Our main contribution will be to use together techniques from both fields to automatically detect vegetation (on an orthophoto) and do a 3D rendering

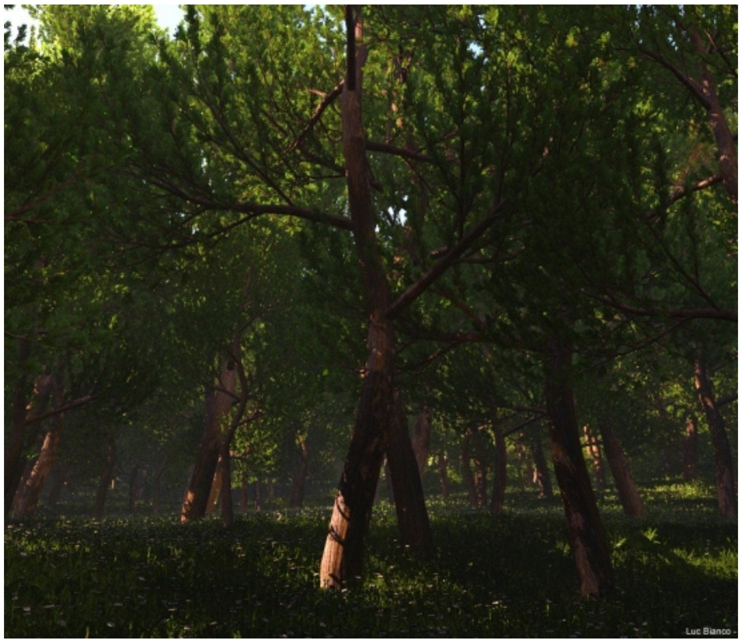

Figure 1.3: An image of artificial 3D model of tropical trees and vegetation, extracted from Xfrog.

of it.

We have seen that Google Earth[2] also renders in 3D the vegetation of some places, however it is not known how the placement of the models is done. Our goal is to do the whole process automatically. Also we do not aim to represent the real vegetation found in a place, but something that might seem feasible only taking into account the orthophoto. For this, we will also be using the orthophoto's color to render our models. Finally, we intend our models to be more realistic than those shown in the mentioned application.

1.4.6 Technologies

During the first and second phases of this project the programming language R[16] has been used. R gives many conveniences when treating with large amounts of data formatted as matrices, and this is the case of pictures. There are also some specific libraries for image processing such as EBImage[17]. Also, this language gives many conveniences when machine learning algorithms must be applied, as there are many libraries with implementations of these, such as MASS[18], nnet[18] or cclust[19]. Finally, R is a very high-level programming language and also strongly functional, making the development of scripts and code very fast, which is optimum for research.

The rest of the phases are strongly related to Computer Graphics and, therefore, C++, OpenGL and FreeGlut will be used. The reason behind C++ is that, for these phases, efficiency is an issue. OpenGL grants a large range of tools for the rendering of 3D scenes

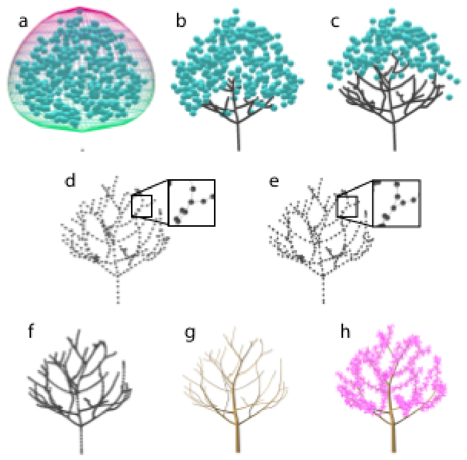

Figure 1.4: Graphic representation of the steps of the Space Colonization *algorithm. Picture extracted from [15].*

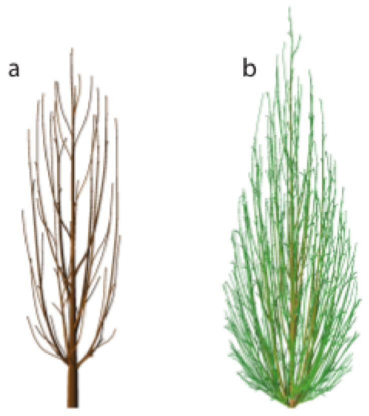

Figure 1.5: Different models generated by the Space Colonization *algorithm with different parameters as input. Picture extracted from [15].*

and FreeGlut has a very easy to use API for the generation and management of windows.

1.5 Project Scope

As mentioned in the previous section, our aim is to construct feasible models of vegetation which would fit the aerial view of a region. This encloses two tasks: extracting information from an orthophoto and using this information to generate a feasible model.

For the first task, machine learning techniques will be used to elaborate a classifier which, given an image, will classify every pixel into “no vegetation” or into a vegetation class. However, it is out of the scope of this project to elaborate a very thin grained classification, instead we will only distinguish between “grass”, “shrub” or “tree”. Also we will only focus on a narrowed set of pictures, since elaborating a classifier for any kind of vegetation in any picture taken worldwide is also out of the scope of this project. Once all pixels in the orthophoto have been classified, we will proceed by testing different algorithms to group the pixels that belong to a single plant.

For the second task, a number of algorithms will be tested to generate a narrowed number of feasible models for each vegetation class. Using basic transformations and additional information extracted from the orthophoto (for instance, colour information), these models will be refined to match the plants detected in the previous task.

Bearing in mind that the strong research component of this project, we are interested in finding solutions to attain our goals and, therefore, we are not much concerned about efficiency. This does not mean that we do not care about time constraints, because a possible future market application would be constrained, however there are more important matters that need our focus and, due to the complexity of the project, time efficiency will be the last thing to be treated.

1.6 Methodology

As it has been explained so far, this project has two big differentiated tasks, classification and model generation, which will be developed sequentially and independently. However, while developing both tasks, every week or every two weeks a meeting will be held with the directors of the project, to evaluate its state and introduce the opportune changes to solve the setbacks that might have arisen. Also, all the code will be kept in a repository with version control, such as GitHub.

During the elaboration of the classifier, at each step, we will be testing a new texture descriptor for the orthophotos or a different machine learning method, and then we will check the effectiveness of the change. We will repeat this process until we find a suitable set of descriptors and a suitable model that ensures the desired accuracy. Once this is done, we will test different approaches for the grouping of pixels into entities that would represent individual plants, and choose the most suitable approach in terms of effectiveness and complexity.

During the second phase, we will be programming an algorithm and testing different parametrisation for it, to produce different vegetation models. A subset of the produced models will be chosen to form the base set for each vegetation class. The last step will be finding a way to match the models in the base set with the plants found in the first phase.

1.7 Risks

As any other project, this one has its own risks, some of which are due to its complexity or to the continued research done in its field:

- **Not enough accurate classifiers:** If either it is really hard to find good descriptors or there is a big enough inherent variability, which can not be explained by a machine learning method, we might end up we no good methods to classify the vegetation. In order to avoid this, a rigorous search of texture descriptors will be held.
- **Variability of the vegetation:** A reduced base set of 3D models, with simple geometric transformations, may not be enough to match the great variety of plants found in a natural environment and, therefore, might not be enough to reproduce the desired landscapes in a feasible way. In order to avoid this, we will study enough flexible methods to transform the base models into the desired ones.
- **Competition:** Many companies are researching actively the field of this project, and some of them might find a better and more simple way to solve the same problem, even before this project is finished.
- **Time Management:** The final product might not be ready in time if many difficulties arise. To prevent this, we will periodically evaluate the state of the project during the weekly meetings.

2 Project Management

2.1 Project Planning

2.1.1 Project Phases

In the following sections the main phases of the project, and the changes they have suffered in comparison with the original planning, are described. Each phase encloses the developing of a software system which will become a part of the application pipeline. The Gantt diagram is available in Appendix A.

The only resources that were used in this project were the manpower of its author, his personal laptop, computation time and diverse bibliography, either papers or books.

2.1.1.1 Test Cases Selection

Requirements: None, this is the first phase of the project.
Expected Time: 1 day.

Our ideal final goal would be to build a system that could process othophotos capturing any kind of landscape, but because of its complexity we decided to develop only a first prototype. This is expected to deal with at least a reduced set of examples or with pictures with similar characteristic to these (This was pointed out in the Project Scope section). On a later project, this prototype could become a starting point for more powerful applications that could successfully handle a wider number of images.

Therefore, the first step of our project is to carefully select a set of orthophotos that will form our test dataset.

2.1.1.2 Classifier Development

Requirements: We need to have selected the test datasets, because samples extracted from them will be used to train the classifier.
Expected Time: 30 days.

During this phase, machine learning techniques were used to elaborate a classifier, the aim of which is to automatically detect vegetation regions on a picture and predict their grass/shrubs/tree class.

The following tasks were carried out to achieve this goal:

- **Literature Search (5 days):** This part implied searching and studying scientific documents to grasp an idea of which texture descriptors and which machine learning methods are useful for our purpose. Understanding how these could be implemented in our application was also work that was done here.

- **First Prototype Elaboration (2 days):** The first thing that was done was to manually classify the vegetation of one of our test images, using a photo editing application. Thanks to this, we could define a training set, to train our models, and a test set, to check their effectiveness, and know to which kind of vegetation they belong. With this, a very basic classifier was elaborated and its error rate was calculated to set a starting point.
- **Improve and Refine (29 days):** This part was carried out as an iterative process. At each step, a new descriptor was computed and its effectiveness was checked, or a different machine learning method was tested.
- **Test Case Processing (3 days):** Finally, the test cases were processed using the classifier.

2.1.1.3 Classified Vegetation Analysis

Requirements: We need the processed test cases that resulted from using the classifier on the original images.
Expected Time: 11 days.

The output of the classification phase is the class assigned to each pixel in the original picture. However, in this phase further information is extracted to ease the placement of vegetation in a further step.

The following tasks were carried out to achieve this goal:

- **Literature Search (1 days):** The most important step in this phase is to identify the pixels that belong to a single plant and to group them together. Because of this, a research on algorithms to solve this problem was done.
- **Pixel Grouping Algorithms Implementation and Testing (6 days):** The solution found for this problem was implemented and tested.
- **Further Analysis (3 days):** Other information about the processed plants might of interest (like the form of its boundary). During this step, code was developed for this purpose.
- **Test Case Processing (1 days):** Finally, the test cases were processed using the steps described above.

2.1.1.4 Vegetation Base Models Generation

Requirements: None, the base models can be generated independently from the rest of the

phases.
Expected Time: 22 days.

It was expected that generating a realistic 3D model of a type of vegetation would be very time consuming, therefore real-time production of the models was discarded. Instead, the aim of this phase was to program a piece of software to build a small set of 3D models for each vegetation type. Then, in a further step, these models would be slightly transformed to make a better match for a target plant.

- **Literature Search (5 days):** Research was conducted in order to find methods to generate realistic 3D models of vegetation.
- **Algorithms Implementation and Testing (15 days):** The selected approach was implemented and then tested.
- **Models Generation (2 days):** Finally, a small set of 3D models was generated and refined, for each kind of vegetation.

2.1.1.5 Development of the Final Application

Requirements: All previous phases must be completed. **Expected Time:** 11 days.

In this phase we built a piece of software that, using the information generated for each detected plant and the base set of 3D models, builds and shows the user the resulting 3D landscape.

- **Implementation and Testing (10 days):** The implementation was carried out testing different ways to render the vegetation with its foliage, the trunks' colour, the terrain, and other elements.
- **Performance Analysis (1 day):** Some optimisations were made in order to improve the performance of the application, and real-time rendering results were obtained.

2.1.1.6 Write the Project Document

Requirements: All previous phases must be completed. **Expected Time:** 20 days.

Although to finish the writing of the documentation it was needed to finish the project first, there are parts of the document which were written right after one of the phases was completed. However, the writing started during the late phases.

2.1.2 Plan Monitoring

One unexpected delay in one of the phases could mean a delay on the whole project since there were many dependencies between phases. To prevent this from happening we had agreed on a strict schedule of weekly meetings to evaluate the situation of the project.

There are parts of the project where different algorithms are tested to find out the best possible approach to solve a problem. For instance, reducing the error of the predictions of our models was an incremental process, our goal was to reduce it to a reasonable amount but if we found ourselves short in time, we could always consider accepting slightly worse results.

Furthermore, the final estimated time for each task was between 1 and 1.5 times more than our first estimations. This was another measure the prevent the negative effects of unexpected delays.

2.1.3 Plan Modifications

During the development of the firsts phases of the project, some setbacks arose making the achievement of the original goals more difficult than expected. To obtain the results specified in the beginning, more time had to be spent in some cases. This is something natural in the course of many projects and it did not imply any major problem because corrections on the original plan were introduced.

In the original planning, a dedication of 30 days was estimated for the phase "Classifier Development". However, the development of this phase implied sequential iterations in pursuing the improvement of an initial solution. More iterations than expected were needed to obtain a satisfactory results and this directly meant a delay in the original planning. The second major difficulty, that occurred during this phase, was related to the large amount of data that was being used (2048 x 2048 pixel images were being used, which means dealing with more than 4 million samples). Because of this, some processes proved too expensive in computation time and the original approach had to be changed to deal with less data (the size of the training set was reduced), which meant repeating many experiments. Nevertheless, several processes still needed to work with the original data set, which implied using up to twelve hours of non-stop computation for each one, while only one machine was available.

In the end, 9 more days than expected were invested in the phase "Classifier Development". During the next phase, "Classified Vegetation Analysis", no major setbacks happened, however the structure of the phase was slightly modified because it became more obvious how to confront the problem that was being solved. The changes narrowed the time spent in search of information and widened the time spent in testing.

The initial planning predicted the deadline of the project to be by the end of May. Nevertheless, in the end the deadline came in mid-June and, therefore, a cautious prediction was made and the deadline of the planning was extended until June 6. Also, five days were subtracted from the phase "Vegetation Base Models Generation" to balance the planning. This has not been done carelessly, to account for the reduced time, existing pieces of software were reused. More precisely, an incomplete implementation of an algorithm for this particular part provided by the codirector of the project and, therefore, there was no further need to do a search for other algorithms. The aim was focused on re-factoring and improving this piece of software.

2.2 Project Budget

2.2.1 Budget

The total cost of this project encloses the costs associated with the human resources and the costs associated with the hardware and software resources.

2.2.1.1 Human Resources

To calculate the human resources costs, different prices have been associated to each task of this project, based on the roles that people assume when carrying them out. These roles can be either "Analyst and Designer" or "Programmer", and their respective salaries can be seen in Table 2.1.

Roles	Salary
Analyst and Designer	45 €/h
Programmer	30 €/h

Table 2.1: Estimated salaries associated to each role needed for the development of this project.

During the planning phase, it was determined that the development of the project would be done between February 11 and June 6, and this implies that about 84 working days were needed to finish the project. However during the last 20 days two tasks were done at the same time, therefore the dedication was doubled.

If we consider our working days to be 4 hours long, and 8 hours long when two tasks are done simultaneously, that gives us a total of 416 hours, which is close to the expected dedication of 375 hours (15 ECTS).

2.2.1.2 Hardware and Software Resources

Hardware costs mainly consist on the price of the machine where the development of this project will take place, which is the author's laptop. The machine is a Sony Vaio VPC-EB1S8E, the cost of which is 829 €.

It is most likely that only free distribution software will be used during the project development, therefore the cost associated to software is expected to be 0. We plan to work with development tools such as the statistical language and environment R, the OpenGL libraries, etc.

Task	Role	Days	Hours	Cost
Test Cases Selection	Analyst/Designer	1	4	**180 €**
Classifier Development				**4980 €**
Literature Search	Analyst/Designer	5	20	900 €
First Prototype Elaboration	Programmer	2	8	240 €
Improve and Refine	Programmer	29	116	3480 €
Test Case Processing	Programmer	3	12	360 €
Classified Vegetation Analysis				**1380 €**
Literature Search	Analyst/Designer	1	4	180 €
Pixel Grouping Algorithms Implementation and Testing	Programmer	6	24	720 €
Further Analysis	Programmer	3	12	360 €
Test Case Processing	Programmer	1	4	120 €
Vegetation Base Models Generation				**2940 €**
Literature Search	Analyst/Designer	5	20	900 €
Algorithms Implementation and Testing	Programmer	15	60	1800 €
Models Generation	Programmer	2	8	240 €
Development of the Final Application				**1320 €**
Implementation and Testing	Programmer	10	40	1200 €
Performance Analysis	Programmer	1	4	120 €
Write the Project Document	Analyst/Designer	20	80	**3600 €**
Total		104	416	14400 €

Table 2.2: Cost of the human resources needed for each task.

Role	Total Days	Total Hours	Cost
Analyst and Designer	35	140	6300 €
Programmer	65	260	7800 €
Total	100	400	14100 €

Table 2.3: Cost of the human resources associated to each role.

2.2.1.3 Total Costs

The estimated total cost of the project can be calculated by adding the cost related to human resources and the cost related to hardware and software resources, as shown in Table 2.5.

2.2.2 Budget Monitoring

As can be seen in the previous subsection, most of the cost of this project comes from human resources and this is directly related to the price of the hours invested in the project. In the Plan Monitoring subsection we have already explained how setbacks and delays were dealt with, therefore as long as the plan was followed with no major delays the costs would not increase.

Resource	**Cost**
Hardware	829 €
Software	0 €
Total	829 €

Table 2.4: Cost of Hardware and Software Resources.

Resource	**Cost**
Human	14400 €
Hardware & Software	829 €
Total	15229 €

Table 2.5: Total Costs

2.3 Laws and Regulations

2.3.1 ICC's (Institut Cartogràfic de Catalunya) Conditions of use

In this section, we will outline the regulations over which, the images we use as test cases, are subject to. The orthophotos that are used in this project, were taken from ICC's geographic database. ICC[1] offers various services related to cartography among which we can find a large database of orthophotos. From all around Catalonia, from different years, and with various resolutions.

According to ICC's terms of use[20], in our case we are subject to conditions C.2.a which are related to academic research. These regulations specify that:

- ICC should be contacted using a pre-established form.
- The service is free as there is no commercial profit.
- Authorisation is required and, in certain cases, signing a contract.
- The text included in section B.8 General conditions must be mentioned in the footer.
- The result of the research (thesis, article) must be sent to the ICC.

However, upon contacting with the ICC we were informed that, even if the result of this project is to be public, this project actually was subject to regulations C.1.b that stands for personal and individual use of the data. Therefore, no special authorization is required.

Other relevant statements to our project are found in section B, where general conditions are described:

- 1. The ICC's geographic information is protected by the Law of Intellectual Property and other applicable national and international legislation.
- 2. The acquisition of the ICC's geographic information, granting of a licence to use ICC's geographic information or the authorisation to use ICC's geographic information does not involve transfer of property, as it completely belongs to ICC.
- 3. The use of the ICC's geographic information by individuals or legal entities involves full and unconditional acceptance of these Terms of Use. The ICC's geographic information can be used by individuals or legal entities subject to these Terms of Use or authorised by the ICC when it applies.
- 4. The ICC reserves the right to give special authorisation when it deems appropriate. The ultimate decision on the commercial nature of a specific use of the ICC's geographic information will always be up to the ICC.
- 8. In all cases except in those included in section C.1.b., the user have to mention the source of geographic information that is used, in a visible place, with the following text:

  ©*Cartographic base orthophoto property of the Institut Cartogràfic de Catalunya, available on www.icc.cat.*

  This quote must appear visibly in the original publication and in all those that are derived from it or in the viewer, if that is the case.
- 9. When using geographic information subject to these Terms of Use for printed publications, a copy must be sent to the ICC.
- 10. The ICC's geographic information can not be used for redistribution, dissemination, copy or public communication, with or without modification of the data, when this use is not under these Terms of Use.
- 14. The ICC reserves the right to take legal actions, and also to claim appropriate compensations, against those who do not use the geographic information properly or according to these Terms of Use. When detecting any case of violation of these Terms of Use, the ICC will be able to ask the user to request authorisation to use the information, or to take the appropriate legal actions. In both these cases, the ICC shall be able to interrupt or block the user's access to the ICC's geographic information until the situation is solved under the current Terms of Use.

2.3.2 Software Licenses

Here we list the software packages used in the project and their license types.

- **R: A Language and Environment for Statistical Computing[16]:** GNU General Public License, either Version 2, June 1991 or Version 3, June 2007..

- **R package EBImage[17]:** License GNU LGPL.
- **R package geometry[21]:** License GNU GPL-3.
- **R package ReadImages[22]:** License BSD.
- **R package png[23]:** License GNU GPL-2 o GPL-3.
- **R package cclust[19]:** License GNU GPL-2.
- **R package MASS[18]:** License GNU GPL-2 o GPL-3.
- **R package nnet[18]:** License GNU GPL-2 o GPL-3.
- **OpenGL (Open Graphics Library):** Open source license, for use of the S.I.. This is a Free Software License B closely modeled on BSD, X, and Mozilla licenses.
- **FreeGLUT (OpenSourced alternative to the OpenGL Utility Toolkit (GLUT) library):** License X-Consortium.
- **GLEW (The OpenGL Extension Wrangler Library):** The source code is licensed under the Modified BSD License, the Mesa 3-D License (MIT License), and the Khronos License (MIT License).
- **ANN (Approximate Nearest Neighbor Library)[24]:** The ANN Library (all versions) is provided under the terms and conditions of the GNU Lesser General Public Library.
- **GLM (OpenGL Mathematics)[25]: The source code is licenced under the MIT license.**
- **Qt:** Qt is licensed under a commercial and open source license (GNU Lesser General Public License version 2.1).

2.4 Sustainability and Social Responsibility

For this section, we will take into account both the impact caused by the development of this project and the impact that the results of this project will have in our society.

As we have seen in the previous sections, the resources for the development of this project are mostly manpower and computation time. Given that nowadays the energy consumption of computers is quite small, we can say that almost no waste will be produced during the project development.

On the other hand, among other uses our application could be used to reproduce natural parks, giving the user the opportunity to get a first impression of what can be found in these places. In a society extremely dependant on computers, using this tool is a powerful mean to advertise natural environments.

Another topic is that, as we will see, the vegetation 3D models that will be generated during phase 3, will be saved in a standard object format. This means that, eventually, a free repository of 3D vegetation models can be created, making a small contribution to society and helping whoever might need this kind of models.

3 Test Cases

In this section we will give a very brief description of the image that we will be using to exemplify every procedure developed during this project. We will be using a 2048 x 2048 pixel image taken from ICC[1], that has a resolution of 25 cm by pixel. This images belongs to a road in Collserola.

Note that the properties of the test case image are important because the results of our study will be bound to them. Nevertheless, this does not mean that the project pipeline could not be applied to other images however, if, for instance, we trained the classifier using pixels from this orthophoto, the classifier will learn to correctly classify pixels from images similar to our test case (with similar resolution, illumination, etc), and could perform poorly on others. If we wanted our classifier to perform satisfactorily in very different images, we should also feed it some pixels from these.

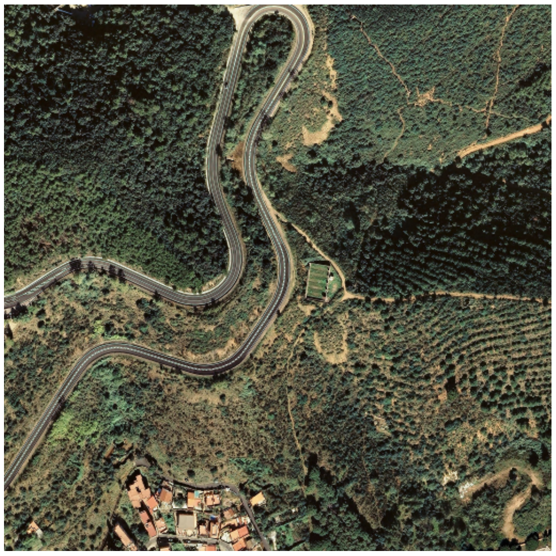

Figure 3.1: ©Cartographic base orthophoto property of the Institut Cartogràfic de Catalunya, available on www.icc.cat[1]

4 Phase 1: Orthophoto Segmentation

4.1 Introduction to the problem

Bearing in mind that the ultimate goal of the project is to reproduce a 3D natural landscape using only the information extracted from an orthophoto, the first obvious step is to identify the parts of the picture that contain vegetation and which kind of vegetation is represented. This task might seem quite easy for a human to perform, however it is desired to carry it out in an automatic way and, in a computer, this becomes quite challenging.

The branch of Computer Science concerned with endowing computers or other machines with vision, or the ability to see, is Computer Vision. This field comprises tasks such as analyzing, understanding, processing or extracting information from real world images or, in general, from high-dimensional data from the real world. Sometimes, to deliver such tasks the aim has been focused in duplicating the abilities of human vision by electronically perceiving and understanding an image.

For this phase, the goal is to analyse and extract information from the chosen images for the recognition of vegetation. We must keep in mind that there are limitations to what a computer can do and to the effectiveness of the techniques that will be used.

Nevertheless, we must also take into account the human visual system limitations because, after all, an orthophoto is missing a large part of the real-world information. Taking figure 4.1 as example; all four images correspond to the same tree represented in different resolutions. Whereas a tree can be easily recognised in the top images, for the bottom ones it is not as clear. Also, notice that one could argue if the plant represented in the bottom left image is either a tree or a shrub (more context information would be needed for this).

If the recognition is dubious even for the human-eye, there is even less guarantee that a computer classifier would act as one would expect. Because of this, our focus is to compute a feasible classification, not necessarily real one.

4.2 Approach and Methodology

4.2.1 Steps

According to [9], the fundamental steps in a Computer Vision application are:

- **Image Acquisition.**
- **Image Enhancement:** The idea is to bring out details that, in the original image, were obscured, or to highlight features of interest. The enhancement process tends to

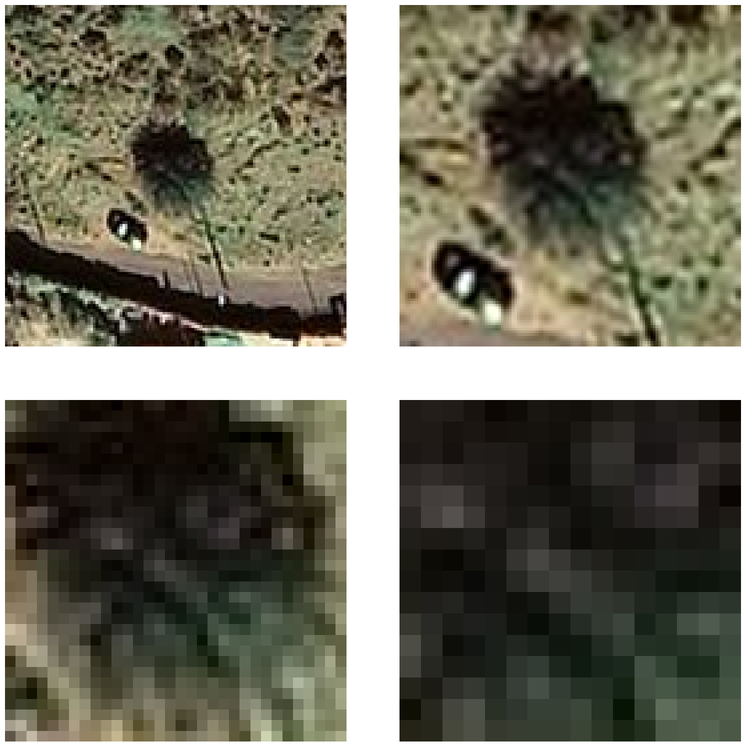

Figure 4.1: Illustrating the limitations of the human vision system. Top left: A 128 x 128 image of a tree. Top right: A 64 x 64 image of a tree with a 2X zoom. A 32 x 32 image of a tree with a 4X zoom. A 16 x 16 image image of a tree with a 8X zoom.

be very subjective and usually information is lost during the process (for instance, this happens when contrasting an image or when doing a histogram equalization).

- **Image Restoration:** Consists in improving the appearance of the image in an objective way, using mathematical and probabilistic models are used.
- **Color Image Processing.**
- **Wavelets and Multiresolutional Analysis:** Analysing an image in different degrees of resolution might lead to more useful information than doing the same analysis in just one resolution.
- **Compression:** Deals with reducing the storage required to save an image, or the bandwidth required to transmit it.
- **Morphological Processing:** Tools for extracting image components that are related to shape.
- **Segmentation:** Partitioning an image into the objects or parts represented on it.
- **Representation and Description:** Data can be represented as boundaries or complete regions, depending on the latter purposes. Description deals with extracting features of interest over objects.

- **Recognition:** Labelling the objects found in an image, according to its descriptors.

The last key element for a Computer Vision application is a Knowledge Base. This component might be as simple as basic knowledge on the image domain or could be more sophisticated. This knowledge base guides the operation of the other steps and also controls the interaction between these.

The organization described above is summarized in Figure 4.2. Nevertheless, this does not imply that every process must be applied to an image, this was just an overview of the methodologies that can be applied to images, for different purposes.

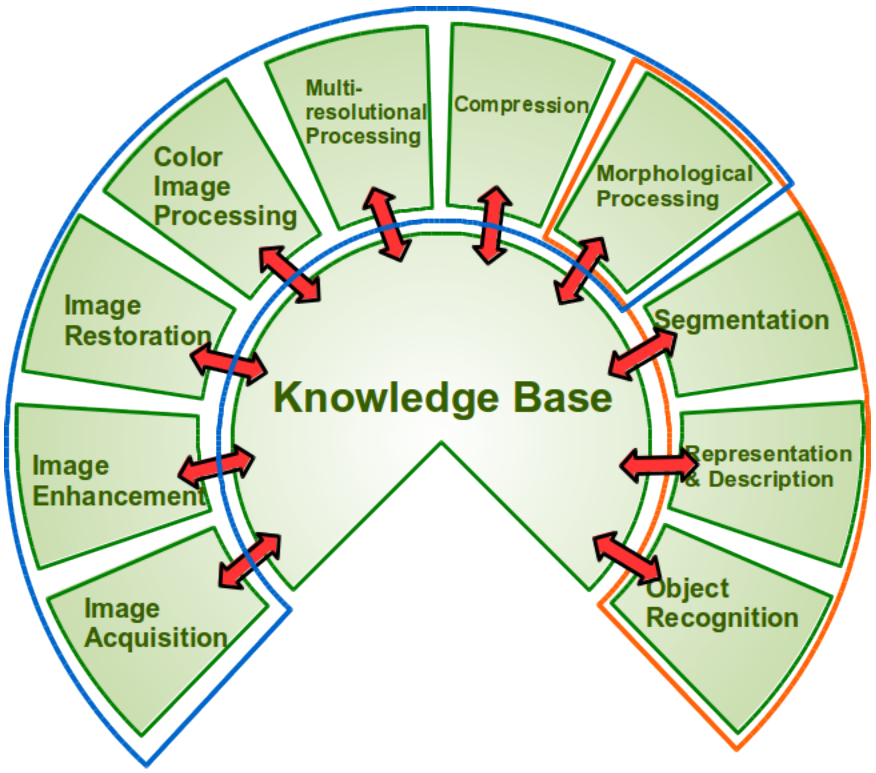

Figure 4.2: Fundamental steps in Digital Image Processing. The blue circled ones are those whose output generally is an image, whereas the orange circled ones are those whose output generally is attributes extracted from the images.

More specifically, our problem falls within one of the particular applications of computer vision, called remote sensing. This area that studies the use of aerial sensor technologies to detect and classify objects on Earth. Bearing this in mind, we can define which of the steps mentioned above will take part in our solution.

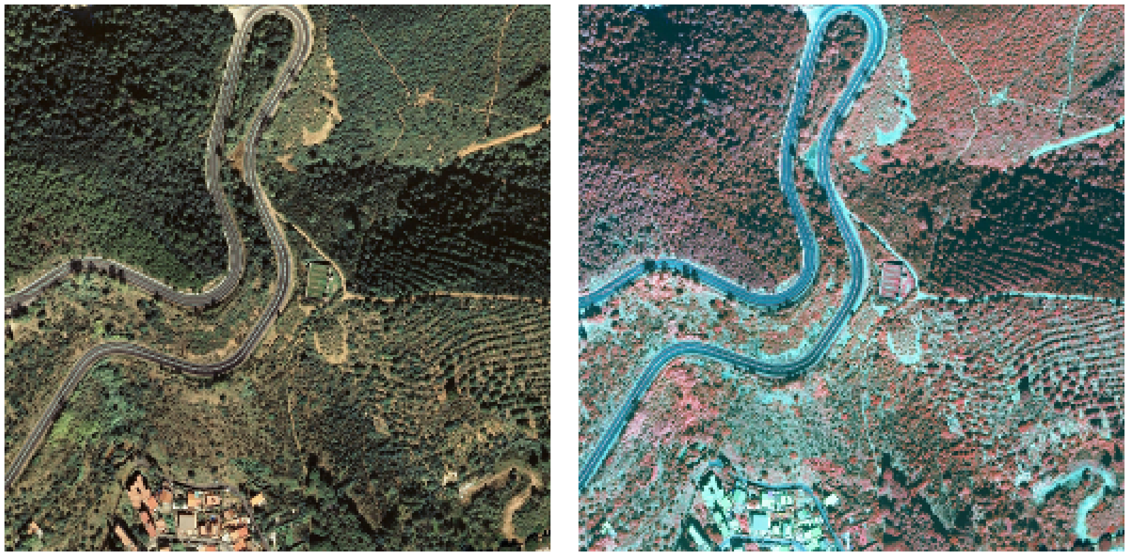

Figure 4.3: ©Cartographic base orthophoto property of the Institut Cartogràfic de Catalunya, available on www.icc.cat[1]. Left: RGB orthophoto. Right: Orthophoto with infrared information coded in the R channel, red information coded in the G channel and green information coded in the B channel..

4.2.2 Image acquisition

The first obvious step is *image acquisition*. For this study, images taken from the Institut Cartogràfic de Catalunya[1] will be used. We will be working with pairs of orthophotos, coded as three-channelled pixel matrices. Both pictures of each pair must correspond to the same geographical region. The first image of the pair must contain RGB information whereas the second one must contain infrared information coded in the R channel, red information coded in the G channel and green information coded in the B channel. An example in Figure 4.3.

ICC keeps a wide range of formats and resolutions for their images. Particularly, we will be using images coded using the Portable Network Graphics (png) format, with a resolution of four pixels per meter. It is important to recall that if images of other resolution are used, the results of this study might not be directly applicable.

4.2.3 Feature Extraction and Classification

From here on, a decision must be taken. There are two main approaches, described in remote sensing literature, to deal with our problem:

- **Per pixel or pixel-based classification:** This is the most traditional approach and aims to process the entire scene pixel by pixel.

- **Object-oriented classification[26]:** Developed relatively recently compared to traditional pixel-based image analysis, this procedure decomposes the scene into many relatively homogeneous image objects (patches or segments) using a multiresolution image segmentation process. These homogeneous image objects are then classified using their statistical characteristics.

For simplicity, and because vegetation usually lacks the structure needed for object-oriented classification, the pixel-based classification approach has been chosen. This implies that no segmentation step, previous to the recognition step, is needed. However, raw input images only contain information at a pixel level and, most likely, this will not be enough to carry out accurate classifications. Therefore, a higher-order feature extraction step must be carried out. The chosen features, and the methodology to extract them, are described in the next section.

Once the appropriate descriptors have been selected and calculated, we are left with the task of classifying into different classes. As it has been stated at some point, we only wish to do a coarse grained classification. Because of this, we have defined our set of classes or labels as "grass´´, "shrub´´, "tree´´, "road´´ or "no vegetation´´. We decided to include the label "road´´, even if it has not clear interest in the detection of vegetation, because roads have special properties (such as "they appear as long wide curves along the image´´) that other no-vegetation elements do not have, and that might be useful in a post-processing steps.

At this point we must select which *machine learning* technique will be used as a classifier and, for this, we must choose between two big families of algorithms:

- **Unsupervised Classification:** The computer or algorithm automatically group pixels with similar spectral characteristics, according to some statistically determined criteria. Then, clusters must be combined and re-labeled into information classes.
- **Supervised Classification:** A set of examples for each class must be identified a priori. The characteristics of these examples are used to train the classification algorithm and every pixel is then evaluated and assigned to the class of which it has the highest likelihood of being a member.

It has been decided to use Supervised Classification because it directly maps the image pixels into the desired classes. Nevertheless, within this family there are many methodologies and, due to the scope of this project, it is not possible to test them all. In the following section, it is explained which methodologies have been chosen and a brief overview on how they work is given. The pipeline for the whole software piece that will be developed in this phase is summarized in Figure 4.4.

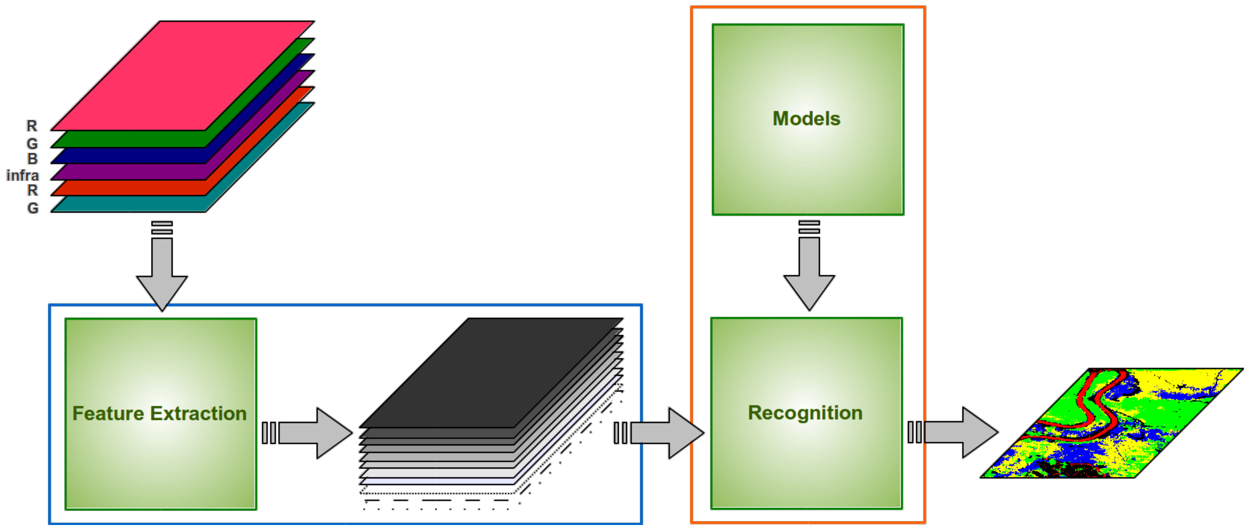

Figure 4.4: Pipeline for the software piece developed during phase 1. The blue framed steps are related with the use of computer vision techniques for feature extraction. The orange framed steps are related to machine learning techniques for the labelling of each pixel. The pipeline receives as input 6 image bands (or two three-channelled images) that the feature extraction step transforms into multiple feature bands. The recognition step uses a supervised classification algorithm, trained with a set of models, to classify every pixel into one of the predefined classes.

4.2.4 Implementation

Some reference must be made to the technology that will be used to carry out all those steps. As this is a research project, we are not much concerned with efficiency, instead we desire to use a fast development tool. R[16] is a semi-functional programming language for dealing with big amounts of data. This language has many libraries that support many machine learning methods and methodologies for image analysis, which makes developing very fast and easy, and, on top of that, it is open source (which is an advantage compared to other languages such as Matlab).

R must be executed in its on software environment and it will used as a scripting language. One script will be elaborated for each of the tasks related this phase. Finally, some of the libraries that will be used are EBImage[17], geometry[21], ReadImages[22], png[23], cclust[19], MASS[18] and nnet[18].

4.3 Feature Extraction

In the following sections it is described how different features can be calculated from the input images. These processes aim to make explicit certain information to improve the performance of the classification.

As we will see, the extraction of many of these features depends on different parameters (for instance, the size of a sliding window). Choosing which parameters are adequate may be as important as choosing which features to extract. Given the scope of this project, it is not possible to do an exhaustive search for the best ones. However, for each feature it will be highlighted which parameters are important for its computation, thus making clear where there is space for further improvement.

As can be seen, in the following sections, the maximum windows size that will be used to compute any of the features is 96 pixels, corresponding to 24m on a 25cm/px image. This means that a window of 96 x 96 pixels, centred at a particular pixel, will be considered. One should be mindful of that, if the pixel lays on the image border, part of this information will not be present. Usually a padding with 0 or 1 is used, but these pixel values are unreal and this is sure to lead to erroneous features and, consequently, to errors in the future classification.

Note that each feature is computed for every pixel in the image. This information can be represented in a matrix with the dimensions of the original image and, if normalized, it can also be displayed as an image. Because of this, sometimes it is said that the output of a feature extraction process is a multi-band image, where each band corresponds to the values of a feature for each pixel on the original image.

4.3.1 HSL Color Space

Most of the chosen features are usually computed over the lightness component of the colour of the pixel, therefore the first step is to separate this component from the rest. For this, we have chosen the HSL colour space (that stands for Hue, Saturation and Lightness). Also, the values for HSL will be used as classification features. In the following sections it is explained how to transform the RGB colour space into HSL.

4.3.1.1 Lightness

The Lightness is the value that tells us how bright a colour is. It is defined as the average of the largest and smallest colour components:

$$M = max(R, G, B) \tag{4.1}$$

$$m = min(R, G, B) \tag{4.2}$$

$$L = \frac{1}{2}(M + m) \tag{4.3}$$

4.3.1.2 Saturation

The Saturation is the value the tells us how pure a colour is. It indicates the range of grey in the colour space and takes values from 0 to 1. Saturation can be calculated as follows:

$$S = \begin{cases} 0 & \text{if M-m} = 0 \\ \frac{M-m}{1-|2L-1|} & \text{otherwise} \end{cases} \tag{4.4}$$

4.3.1.3 Hue

The Hue indicates the actual colour. It ranges from 0° to 360° and can be computed as follows:

$$H' = \begin{cases} undefined & \text{if M-m} = 0 \\ \frac{G-B}{M-m} & \text{if R} = \text{M} \\ 2 + \frac{B-R}{M-m} & \text{if G} = \text{M} \\ 4 + \frac{R-G}{M-m} & \text{if B} = \text{M} \end{cases} \tag{4.5}$$

$$H = H' * 60 \qquad if(H < 0), H = H + 360 \tag{4.6}$$

We will make the assumption that when $M - m = 0$ the Hue take the value 0°.

4.3.2 NDVI Descriptor

4.3.2.1 Overview

The Normalised Difference Vegetation Index (NDVI) is a graphical indicator which can assess whether the region that is being analysed contains live green vegetation or not. It can be calculated as:

$$NDVI = \frac{infrared - red}{infrared + red} \tag{4.7}$$

The pigment in green live vegetation leaves (commonly known as chlorophyll) strongly absorbs visible light to perform the photosynthesis, whereas it strongly reflects infrared light to prevent overheating. The NDVI ranges between 1 and -1, being closer to 1 when the infrared value and the red value diverge the most.

4.3.2.2 Feature Extraction

As we have mentioned earlier, one of the inputs of our system is a pixel array that have infrared information coded in the R channel, and red information coded in the G channel. With the information, the computation of the NDVI is straightforward, and this index can be used as a very useful low-dimensional classification feature to discriminate between vegetation and no vegetation.

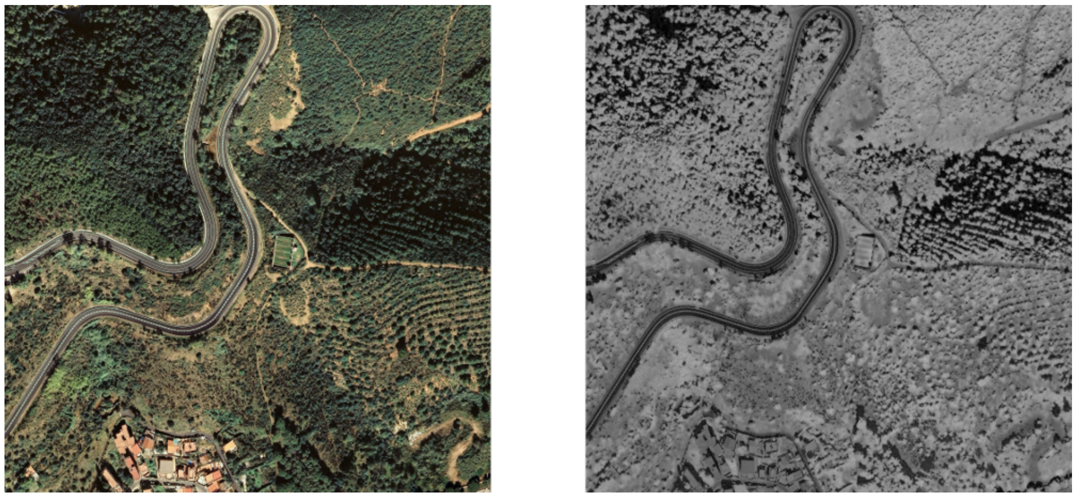

Figure 4.5: Left: ©Cartographic base orthophoto property of the Institut Cartogràfic de Catalunya, available on www.icc.cat[1]. Right: NDVI, normalised to [0,1], for the image on the left.

However, the NDVI also has its limitations. Note that this index is not helpful for the detection of vegetation when it is dry, deciduous or shadowed. To solve this problem, and others, and increase the overall accuracy of our classifiers other more complex texture features will be computed.

4.3.3 Grey Level Coocurrence Matrix (GLCM)

4.3.3.1 Overview

A co-occurrence matrix generally is a 2D array C in which each element C(i,j) indicates how many times the value i co-occurs with the value j in some designed spatial relationship. For the case of grey-scale images, these values are grey tones. The co-occurrence matrices are useful for the computation of second-order statistics (statistics that take into a account pairs of pixels) and usually, for this purpose, the grey-level histogram is discretized in M values instead of using the possible 256 values.

Another way to understand co-occurrence matrices, for grey-scale pictures, is seeing the element (i, j) of the co-occurrence matrix $C^{\delta\theta}$ as the joint probability that a grey tone i co-occurs with a grey tone j on a distance δ in a direction θ. Usually, small values for δ are used since most relevant correlation between pixels exists on small distances.

δ and θ can be modelled as a vector $d = (dr, dc)$ that specifies the displacement between the pixel with value i and the pixel with value j. dr can be thought as the displacement in rows (downwards) and dc the displacement in columns (to the right). Then, the grey-level co-occurrence matrix C^d for a discretized grey-scale image I is defined by:

$$C^d(i,j) = |\{(r,c)|I(r,c) = i \text{ and } I(r+dr, c+dc) = j\}| \tag{4.8}$$

One could also compute a variation of the standard grey-level co-occurrence matrix which is called *normalised* grey-level co-occurrence matrix N^d. It can be calculated as:

$$N^d(i,j) = \frac{C^d(i,j)}{\sum_i \sum_j C^d(i,j)} \tag{4.9}$$

Another variant is the *symmetric* grey-level co-occurrence matrix, which is defined by:

$$S^d(i,j) = C^d(i,j) + C^d(j,i) \tag{4.10}$$

L. A. Ruiz *et al.*[8] have used some features extracted from grey-level co-occurrence matrices for the classification of Mediterranean vegetation found in remote sensing images. To extract features for each pixel, they use normalised symmetric co-occurrence matrices calculated over a window around each of them. Experimentally they have found that increasing the window size rises the the classification accuracy, for the inner parts of the picture, but produces an overall progressive increase of the error due to the border effect. A windows size of 25 x 25 pixels was found to be the optimal, however this is bound to the resolution of the image. Also, they tested distances between pixels between 1 and 3 getting to the conclusion that these did not seem to affect the results.

The set of statistical features L. A. Ruiz *et al.*[8] use is a part of the ones proposed by Haralick[27], such as *contrast*, *uniformity*, *mean*, *variance* and *inertia moments* In Computer Vision[10], the following set of features, derivable from a normalised co-occurrence matrix, is proposed to characterise a texture:

$$\text{energy} \quad \sum_{i=0}^{n}\sum_{j=0}^{n}(N^d(i,j))^2 \tag{4.11}$$

$$\text{entropy} \quad -\sum_{i=0}^{n}\sum_{j=0}^{n}N^d(i,j)\log N(i,j) \tag{4.12}$$

$$\text{contrast} \quad \sum_{i=0}^{n}\sum_{j=0}^{n}(i-j)^2 N^d(i,j) \tag{4.13}$$

$$\text{homogeneity} \quad \sum_{i=0}^{n}\sum_{j=0}^{n}\frac{N^d(i,j)}{1+|i-j|} \tag{4.14}$$

$$\text{correlation} \quad \frac{\sum_{i=0}^{n}\sum_{j=0}^{n}(i-\mu_i)(j-\mu_j)N^d(i,j)}{\sigma_i\sigma_j} \tag{4.15}$$

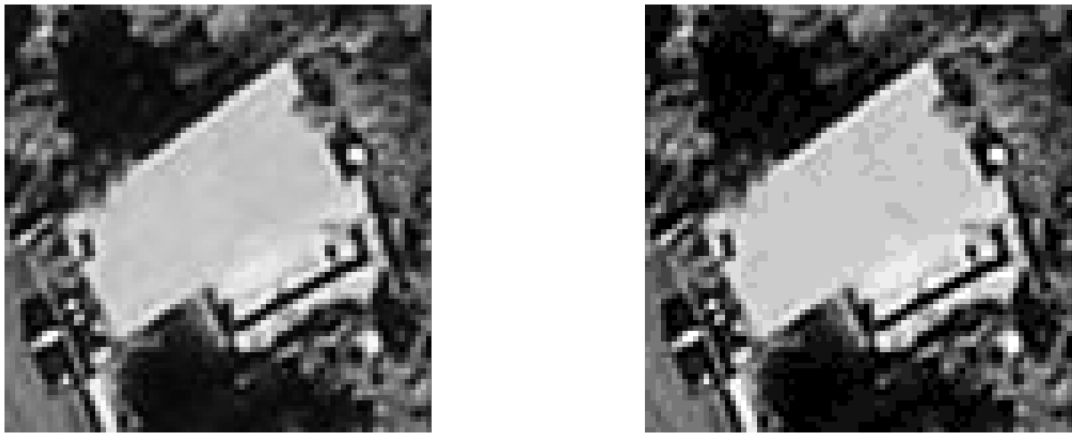

Figure 4.6: Left: Aerial image extracted from ICC[1]. Right: Image on the left discretized using 16 grey values.

where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column sums $N^d(i)$ and $N^d(j)$ defined by:

$$N^d(i) = \sum_j N^d(i,j) \tag{4.16}$$

$$N^d(j) = \sum_i N^d(i,j) \tag{4.17}$$

4.3.3.2 Feature Extraction

In this study, we will extract the 5 features proposed in Computer Vision[10] over 6 different normalised symmetric grey-level co-occurrence matrices. The vectors d which characterise each matrix will be $d_1 = (0,1)$, $d_2 = (1,0)$, $d_3 = (1,2)$, $d_4 = (2,1)$, $d_5 = (0,3)$ and $d_6 = (3,0)$. The matrices will be computed using a sliding window of 25 pixels. Finally, all the grey values will be disctretized into 16 values, as seen in Figure 4.6.

4.3.3.3 Parameters In this section we will summarise the parameters that will be used to compute the features based on grey-level co-occurrence matrices:

- **Sliding windows size:** 25 pixels, about 6.25m in a 0.25m/px image.
- **Distance Vectors:** $(0,1)$, $(1,0)$, $(1,2)$, $(2,1)$, $(0,3)$ and $(3,0)$.
- **Number of discretized values:** 16.

grey levels	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	473	207	61	32	30	14	11	6	1	1	2	1	0	0	0	0
2	235	439	113	50	31	17	8	6	1	2	1	2	0	0	0	0
3	49	115	134	78	46	27	11	13	8	5	2	0	0	0	0	0
4	37	53	76	73	66	42	30	16	6	4	4	5	2	2	0	0
5	16	32	40	66	70	56	41	32	10	14	7	4	1	2	1	0
6	12	26	20	42	70	89	53	28	15	9	9	5	1	0	2	0
7	10	13	24	27	47	59	73	71	35	12	10	4	1	2	1	0
8	2	7	8	21	24	39	71	87	42	22	5	7	8	4	5	0
9	2	10	5	11	11	15	27	47	27	17	5	9	3	7	4	0
10	0	2	3	5	3	10	20	17	23	15	15	7	9	7	1	5
11	0	2	2	3	3	9	10	10	10	19	9	9	7	10	8	3
12	0	1	3	9	2	3	8	4	3	8	15	83	88	21	7	2
13	0	0	1	1	2	4	2	2	2	4	14	88	838	62	11	3
14	0	0	0	1	2	1	4	4	8	4	6	22	57	98	35	6
15	0	0	0	1	1	1	1	2	5	5	7	7	15	28	42	9
16	0	0	0	0	0	0	1	0	3	0	4	4	4	5	7	11

Table 4.1: Co-occurrence values for the image in Figure 4.6, using $d = (0, 1)$*. Once made symmetric and normalised, the following values were computed:* energy*: 0.036,* entropy*: 6.118,* contrast*: 4.549,* homogeneity*: 0.630,* correlation*: 29409.44*

4.3.4 Edge Density Descriptor

4.3.4.1 Overview

The discrete approximation of the first derivative (or gradient) of the intensity function of an image, gives us information about the relationship between pairs of pixels. There are many ways to calculate this, such as the ones explained in [6]:

$$\text{Forward Differences:} \quad \begin{cases} D_x(i,j) = I(i+d,j) - I(i,j) \\ D_y(i,j) = I(i,j+d) - I(i,j) \end{cases} \tag{4.18}$$

$$\text{Backward Differences:} \quad \begin{cases} D_x(i,j) = I(i-d,j) - I(i,j) \\ D_y(i,j) = I(i,j-d) - I(i,j) \end{cases} \tag{4.19}$$

$$\text{Central Differences:} \quad \begin{cases} D_x(i,j) = I(i-d,j) - I(i+d,j) \\ D_y(i,j) = I(i,j-d) - I(i,j+d) \end{cases} \tag{4.20}$$

Between these three, the last one is the most robust and gives information centred exactly at pixel (i, j).

It is common to calculate the mean gradient at a given location using:

$$D(i,j) = \sqrt{(D_x(i,j)^2 + D_y(i,j)^2)} \tag{4.21}$$

It can be approximated by:

$$D(i,j) = |D_x(i,j)| + |D_y(i,j)| \tag{4.22}$$

According to [10], the edge density can be defined as the number of edge pixels in a given region, and it gives an indication of the busyness of edges in that region. One easy way to determine if a pixel is an edge pixel is to compute the gradients, explained before, and find the spots where the value of the derivative is a local maximum.

For instance, one could count each pixel, whose gradient is above a certain threshold T, as an edge pixel, and compute the edge density for the pixel (i,j) as:

$$EdgeDensity(i,j) = \frac{|\{p \in Neighbourhood(i,j) | D(p) \geq T\}|}{|Neighbourhood(i,j)|} \tag{4.23}$$

L. A. Ruiz *et al.*[8] propose to use the edge density, computed over a neighbourhood, as a descriptor for pixel-by-pixel classification of Mediterranean vegetation. For this, they compute the gradient of the images as a function of a distance parameter d, and for the pixel (i,j) they add these values over a certain neighbourhood N(i,j):

$$\begin{aligned} g(i,j,d) = \sum_{(x,y)\in N(i,j)} & |I(i,j) - I(i+d,j)| + |I(i,j) - I(i,j+d)| \\ & |I(i,j) - I(i-d,j)| + |I(i,j) - I(i,j-d)| \end{aligned} \tag{4.24}$$

Experimentally, they found that the optimal distance for the parameter d was 3, for their experiments. However, this is expected to be resolution-dependant and, therefore, for different resolution images this may vary.

Sometimes, pictures may contain noise and this can make false edges to be detected by the means explained before. This can be avoided using a gaussian smoothing before applying the edge detection operator. A more efficient option is to do both steps at the same time, using the derivative of a 2D gaussian, which can be seen in Figure 4.7.

The problem of this method is that detecting an edge using a threshold may not give good results. This is due to the fact that some edges are sharper than others. The common feature that all edges share is that the derivative has a maximum where the edges is located. A better way to localise this maximum is employing the second derivative, because it yields a 0 on the maximum of the first derivative.

As before, smoothing and doing the second derivative can be done in one step, using the Laplacian of gaussian operator, which can be seen in Figure 4.8. According to Marr and Hildreth[28], this operator can be approximated very fast by computing the difference

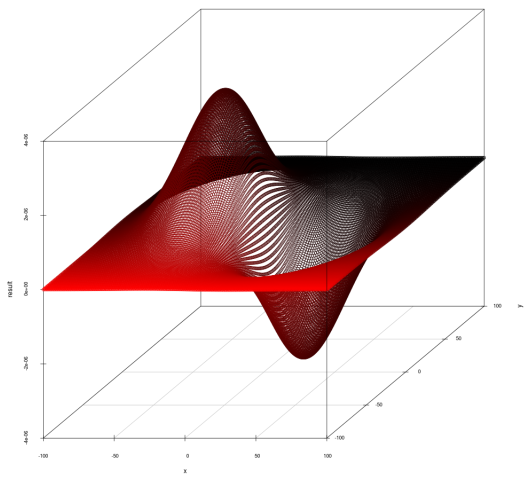

Figure 4.7: Derivate of a gaussian filter.

of two gaussians. The best approximation is obtained when the rate between the standard deviations of these gaussians is close to 1.6.

Once the image has been convolved with the laplacian of gaussian filter, all that is left to do is to detect zero crossings.

4.3.4.2 Feature Extraction

The edge density over a neighbourhood of 25 x 25 pixels will be computed for each pixel, and this value will be used as a feature for classification. For the edge detection process, the method of L.A. Ruiz *et al.*[8] and the laplacian of a gaussian method of Marr and Hildreth[28] will be tested.

L.A. Ruiz *et al.*[8] do not give any hint about the neighbourhood they use to compute the edge density. We have fixed this value to 25 x 25 pixels for simplicity. Different d will be tested because, as pointed out before, the optimal value for this parameter is expected to be dependend on the resolution of the pictures. The values 1, 2, 3, 5, 7, 12, 17 and 25 will be tested.

For the laplacian of a gaussian method, an approximation using a gaussian of sigma 1 and a gaussian of sigma 1.6 will be used. Sizes of 3, 5, 7, 11, 15, 25, 35 and 51 will be tested for the zero crossing detecting filter. The final edge values will be computed as the sum of the absolute values of the images filtered with a vertical and horizontal zero crossing detecting masks.

4.3.4.3 Parameters

In this section we will summarise the parameters that will be used to compute the features

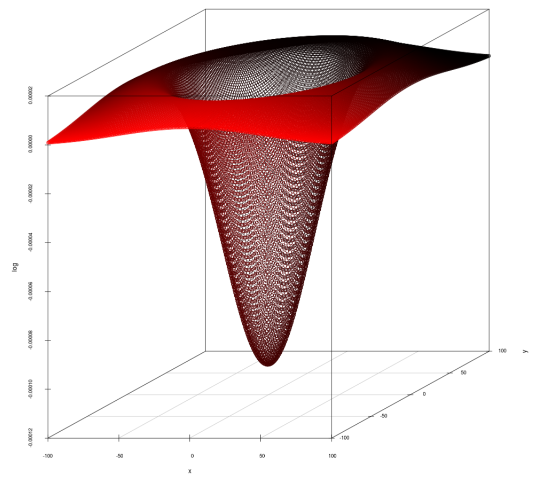
Figure 4.8: *Laplacian of a gaussian filter.*

based on the edge density:

- **Neighbourhood Size** 25 x 25 pixels.
- **Value for d (L.A. Ruiz *et al.* method):** 1, 2, 3, 5, 7, 12, 17, 25.
- **Sigmas (Laplacian of a Gaussian Method):** 1 and 1.6.
- **Zero crossing detection mask size (Laplacian of a Gaussian Method):** 3, 5, 7, 11, 15, 25, 35, 51.

4.3.5 Edge Histogram Descriptor

4.3.5.1 Overview

The Edge Histogram Descriptor (EHD) defined in the standard MPEG-7[7] is a descriptor that captures the spatial distribution of edges in textures. The distribution of edges is a good signature that is useful for image to image matching.

The extraction method is defined in [29], and goes as follows: first, the original image is divided in 4 x 4 sub-images, then the local edge histogram is computed over each of these blocks, grouping edges into five categories which are vertical, horizontal, 45^o diagonal, 135^o diagonal and isotropic. A total of 80 features (5 for each block) are extracted.

To compute the local edge histograms, each block is further subdivided into a constant number (application dependent) of sub-blocks. Then simple edge detectors (which can be seen in Figure 4.9) are applied to each sub-block, treating them as if they were 2 x 2 pixel images. The pixel intensities for the 2 x 2 partitions of the image sub-block are computed

$$\begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix}\quad\begin{bmatrix}1 & 1\\ -1 & -1\end{bmatrix}\quad\begin{bmatrix}\sqrt{2} & 0\\ 0 & -\sqrt{2}\end{bmatrix}\quad\begin{bmatrix}0 & \sqrt{2}\\ -\sqrt{2} & 0\end{bmatrix}\quad\begin{bmatrix}2 & -2\\ -2 & 2\end{bmatrix}$$

Figure 4.9: Filters for edge detection.

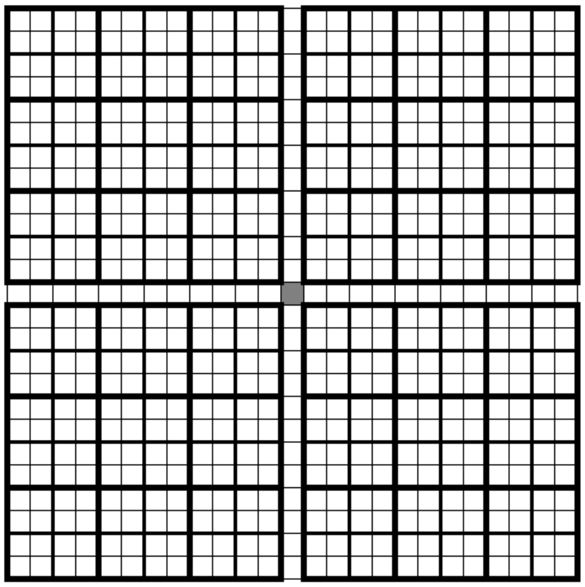

Figure 4.10: Window for the extraction of the local edge histogram descriptor. The grey pixel is where 25 x 25 is centred. The window has been divided in 36 regions of 4 x 4 pixels. Each region is treated as a 2 x 2 image therefore, each pixel of this image is the result of averaging 4 pixels.

by averaging the intensities of the corresponding pixels. The blocks whose edge strengths exceeds a certain minimum threshold are used in computing the histogram.

For an image sub-block we can compute five different edge strengths. If the maximum of these surpasses a certain threshold then the sub-block is considered an edge block with that strength as label. The edge blocks contribute to the appropriate bin of the histogram descriptor. Finally these values are normalised between [0,1].

4.3.5.2 Feature Extraction

The edge histogram descriptor is useful to extract features to classify a whole pciture, however we are interested in pixel-by-pixel classification and, because of this, some adaptation must be made. Our approach implies computing the local edge histogram over an sliding window, to extract features for each pixel. For this, a window of 25 pixels will be divided into 36 regions over which the edge detectors will be applied, as can be seen in Figure 4.10.

This way, a total of 5 features will be extracted for each pixel, which indicate the density of edges in its neighbourhood. The major advantage that this descriptor has, in comparison with the edge density descriptor explained before, is that this one also captures the directionality of the edges.

Different threshold values will be tested. Note that each pixel intensity ranges from 1 to 0 and, because of this, the result of applying the edge detectors will be a value between -1 and 1. Knowing this, the threshold values that will be tested are 0.15, 0.25, 0.35 and 0.45.

4.3.5.3 Parameters

In this section we will summarise the parameters that will be used to compute the features based on edge histogram descriptor:

- **Sliding windows size:** 25 x 25 pixels.
- **Number of blocks:** 36.
- **Threshold value:** 0.15, 0.25, 0.35 and 0.45.

4.3.6 Energy Descriptors (Laws, 1985)

4.3.6.1 Overview

L. A. Ruiz *et al.*[8] propose the use of Laws' texture energy measures as a descriptor for pixel-by-pixel classification of Mediterranean vegetation. For this, they use different energy filters, of size 7 x 7 pixels, to enhance some textural features of the images and then, for each pixel they computed the local energy over a neighbourhood. Finally, in order to reduce the error due to border effects, they apply a post-processing method proposed by Hsiao and Sawchuk (1989).

Kenneth I. Laws[30] defined a set of texture transforms aimed to generate pixel features to improve the segmentation accuracy for aerial images. Texture energy is defined as the amount of variation within a filtered window around a pixel. These measures tend to be constant across any perceptually homogeneous texture region and distinct for distinct textures.

The first step in Laws' procedure is moving a small window around the image and substracting the local average from each pixel. This process removes the effects of illumination. A window of 15 x 15 pixels was used for natural scenes.

The second step in Laws' is to convolve the preprocessed image with different small masks. These masks are obtained by computing outer products of pairs of these vectors which result from convolving recursively the vectors [1 1] and $[-1\ 1]$, as seen in Figure 4.11. In Computer Vision[10] length-5 vectors L5, E5, S5 and R5 are used. One mask generation example can be see in Figure 4.12.

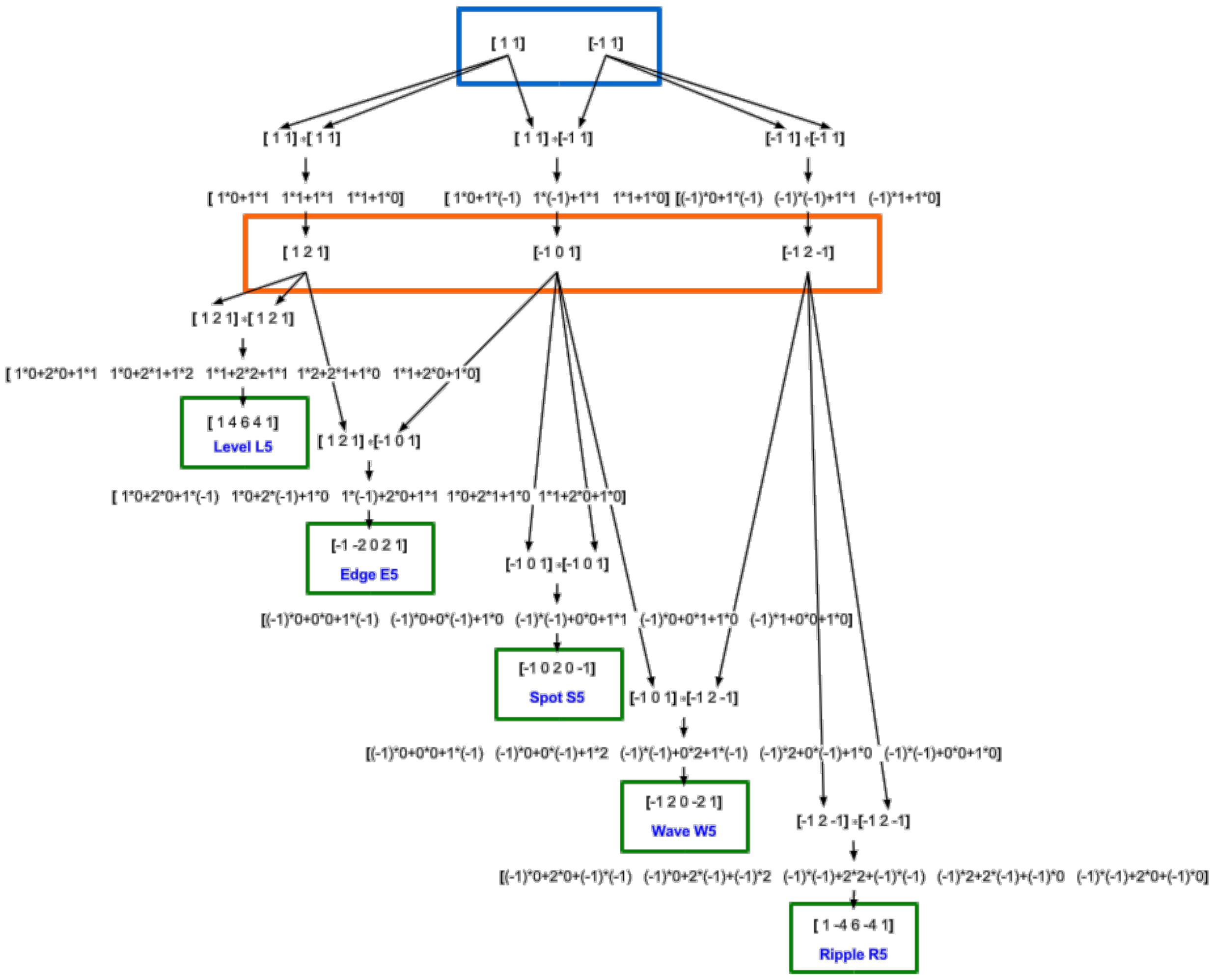

Figure 4.11: Generation of 5-pixels vectors, doing recursive convolutions, used in the generation of 2D convolution masks. 2-pixel vectors are framed as blue, 3-pixel vectors are framed as orange and 5-pixel vectors are framed as green.

The energy at a point can now be computed as the variance of the filtered images over a fixed size window. Standard deviation has been found just as effective as the variance, and can be approximated by an average of the filtered image magnitudes. This works because all the filtered images, except for the L5L5 band, have mean equal to zero. The energy is computed as the sum of the filtered image magnitudes over a neighbourhood of 15 x 15 pixels, around a certain pixel.

Finally, certain symmetric pairs of energy maps are combined, replacing each pair with its average, to produce the nine final maps used as features for classification. The resulting energy maps are E5E5, S5S5, R5R5, L5E5/E5L5, L5S5/S5L5, L5R5/R5L5, E5S5/S5E5, E5R5/R5E5 and S5R5/R5S5.

$$\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix} \times [1\ 4\ 6\ 4\ 1] = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figure 4.12: Outer product between vectors E5 and L5, to obtain mask E5L5.

4.3.6.2 Features Extraction

In this case, to extract the energy features, the procedure exposed in Computer Vision[10] will be followed. A preprocessing window of 15 x 15 pixels will be used to remove the effects of illumination and then the image will be filtered using the 16 convolution masks generated using the vectors L5, E5, S5 and R5 (Note that vector W5 is not used).

However, for the last step, different sizes energy gathering windows will be used. Particularly, sizes of 15, 25, 35 and 45 pixels will be tested. Note that, in this study, we are not concerned about border effects and, therefore the postprocessing method, that L. A. Ruiz *et al.*[8] suggested, will not be used.

4.3.6.3 Parameters

In this section we will summarise the parameters that will be used to compute the features based on Laws' energy texture measures:

- **Preprocessing window size:** 15 pixels.
- **Energy Maps:** E5E5, S5S5, R5R5, L5E5/E5L5, L5S5/S5L5, L5R5/R5L5, E5S5/S5E5, E5R5/R5E5 and S5R5/R5S5.
- **Energy gathering window sizes:** 15, 25, 35 and 45 pixels.

4.3.7 Power Spectra

4.3.7.1 Overview

The Fourier theory shows how most real functions can be represented in terms of basis of sinusoids. The 2D Fourier Transform transforms a spatial function $f(x, y)$ into the (u, v), frequency domain. However, images (picture functions) are discrete, we can represent them as images I[x,y] with N^2 samples. For these, we can use the Discrete Fourier Transform (DFT), which transforms an image of N x N spatial samples I[x,y] into an N x N array F[u,v] of coefficients used in its frequency representation.

To compute the Fourier Transform, we need to define a set of sinusoids of varying frequencies as an orthogonal basis. Note that, if m, n are any two different integers, then the two cosine waves with these frequency parameters are orthogonal over the interval $[-\pi, \pi]$. For 2D picture functions, the following set of complex functions can be used:

$$E_{u,v}(x,y) \equiv e^{-i2\pi(ux+vy)} = \cos(2\pi(ux+vy)) - i\sin(2\pi(ux+vy)) \tag{4.25}$$

Using this set of sinusoidal basis images (Fourier basis functions) $E_k \approx E_{u,v}(x,y)$, any picture function can be represented as $I[x,y] = \sum_{k=0}^{N^2-1} a_k E_k[x,y]$. The coefficients a_k are a measure of similarity between $I[x,y]$ and $E_k[x,y]$. When one of these basis functions correlates highly with a picture function, it means that the picture function has high energy in frequency (u,v). The Fourier Transform converts a picture function into an array of these correlations.

The Discrete Fourier (DFT) can be directly applied to digital images. Each of the N^2 basis vectors needed to represent N x N images with real intensity values is determined by a pair of frequency parameters (u,v) which range from 0 to N-1. The calculation of the coefficients F[u,v] can be done as follows:

$$F[u,v] \equiv \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I[x,y] e^{\frac{-i2\pi}{N}(ux+vy)} \tag{4.26}$$

Each coefficient F[u,v] is computed as the dot product between the image I[x,y] and $E_{u,v}[x,y]$. We can also define the inverse transform to convert frequency domain representations F[u,v] into a spatial image I[x,y]:

$$I[x,y] \equiv \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F[u,v] e^{\frac{i2\pi}{N}(ux+vy)} \tag{4.27}$$

The power spectrum combines the energy from sine and cosine waves of the same frequency components (u,v) and can be calculated as follows:

$$P(u,v) = \sqrt{Real(F(u,v))^2 + Imaginary(F(u,v))^2} \tag{4.28}$$

From its definition, it is clear that $P(-u,-v) = P(u,v)$, therefore the power spectrum is symmetric about the origin $u = 0$, $v = 0$. Because of this, the power can be sampled in rings to extract features that may be useful for classification purposes as explained in Computer Vision[10].

According to [6], the power spectrum of natural images tends to follow a power law that can be modelled as $P = 1/f^{\beta}$, where P is the power as function of frequency f and β is the

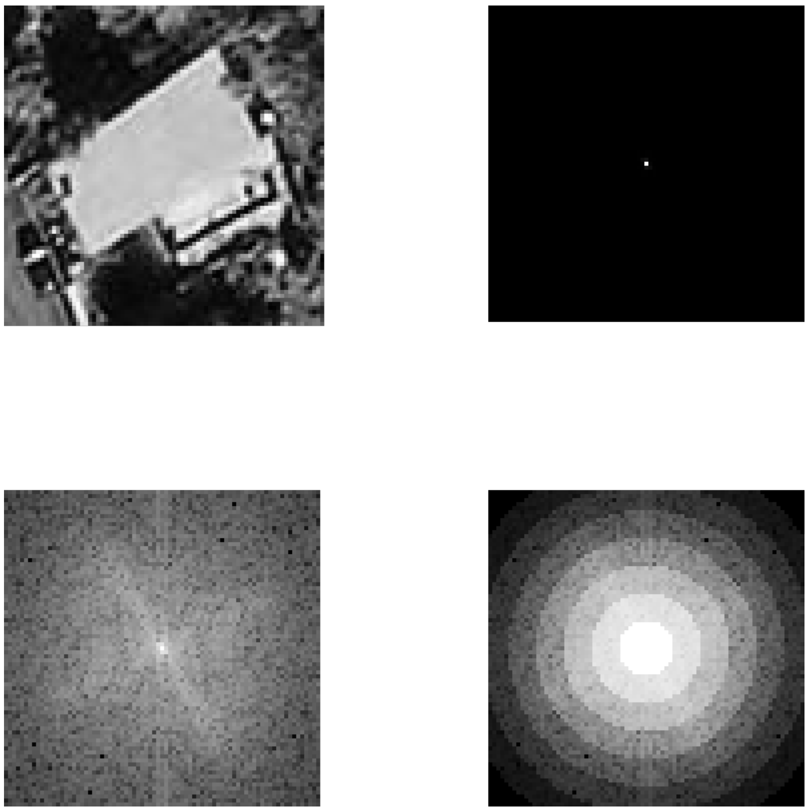

Figure 4.13: Top left: Aerial 80x80 image of a house, extracted from ICC[1]. Top right: Power spectra of the image on the left, normalised between 0-1. Bottom left: Logarithm of the power spectra (for a clearer visualisation). Bottom right: Rings over which the texture features are computed.

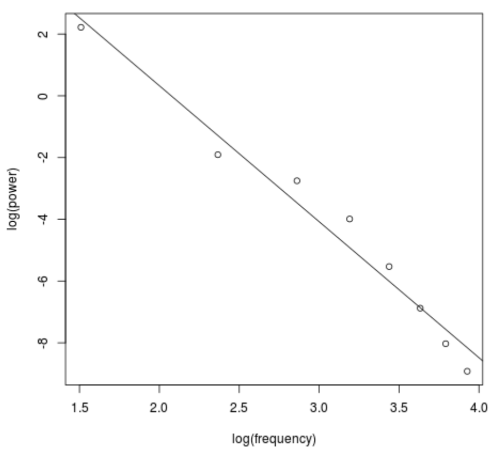

Figure 4.14: Power spectra of the picture in Figure 4.13 plotted as function of frequency in a log-log scale.

spectral slope. That means that, on a log-log scale, the power as function of frequency lies in a approximately on a straight line, as can be seen in Figure 4.14.

Spectral slope is directly related to the roughness of the image texture and can be used a low-dimensional feature for classification. Huang and Mumford[31] found that there were systematic differences in the slopes of images of different categories. 2.3, 1.8, 1.4 and 1 were the slopes they found for images in the categories of man-made, vegetation, road and sky pixels, respectively.

4.3.7.2 Feature Extraction

In this study, we will extract features of the power spectra calculated over a sliding window. Windows sizes of 35, 45, 55 and 65 pixels will be tested. To compute the total power spectrum over rings, the frequency domain will be partitioned into a fixed number ring-shaped regions around the origin. There will be one feature as the result of adding the power values over each region. 8 and 16 partitions will be tested, giving place to either 8 or 16 features for each pixel.

After this, linear regression will be used on the logarithm of these values as function of the logarithm of the mean frequency they represent. With this we will obtain information about the spectral slope and the intersect value. These 2 values will also be used as features.

4.3.7.3 Parameters

In this section we will summarise the parameters that will be used to compute the features based on power spectra:

- **Sliding windows size:** 35, 45, 55 and 65 pixels.
- **Number of Frequency Rings:** 8 or 16.

4.3.8 Wavelets

4.3.8.1 Overview

The wavelet transform, first introduced to the field of texture analysis by Mallat[32], maps an image onto a low resolution image (low-pass filtering) and a series of detail images (high-pass filtering). This process is useful to analyze a texture at different scales, which is why it is called multiresolution analysis.

Van de Wouwer *et al.*[33] proposed an approach for texture characterisation using information extracted from the wavelets detail coefficients. For this, they compute a set of first-order statistics from the wavelet detail image histogram, which they call *histogram signatures*, and a set of second-order statistics from the detail image co-occurrence matrices, which they call *co-occurrence signatures.*

L. A. Ruiz *et al.*[8] compared the use of *fluctuations* and *details* for the classification of image pixels into different vegetation classes. They define *fluctuations* as the different scale wavelet detail images, whereas *details* are the reconstructed images obtained by applying the inverse transform to the *fluctuations.*

Based on the work of Van de Wouwer *et al.*, L. A. Ruiz *et al.* computed four co-occurrence features over the *fluctuations* and *details.* The chosen features were *variance*, *inverse difference moment*, *contrast* and *correlation.* They obtained the best results using the Coiflet-24 wavelet and its reconstructed details from 3 levels, however, they do not mention which window size was used.

4.3.8.2 Feature Extraction

In this study, first-order statistics will be computed over the detail wavelets coefficients. For this, three levels of details will be built applying the haar wavelet transform (which is explained in [11]) over a sliding window. Windows sizes of 96, 80, 64 and 48 pixels will be tested. For each pixel 18 features will be calculated, these include the 2 *histogram signatures* extracted for each of the 3 detail images (one for horizontal detail, one for vertical detail and one for diagonal detail) on each resolution level.

Second-order statistics over the detail wavelets coefficients contain more information that could help to further improve the classification accuracy. However, computing them would be very time consuming and, due to the scope of this project, it has not been possible. Therefore it is left as future work.

Notice that calculating x co-occurrence statistics over y different co-occurrence matrices for 3 levels of wavelets detail coefficients implies building $9 * Npixels * y$ co-occurrence matrices

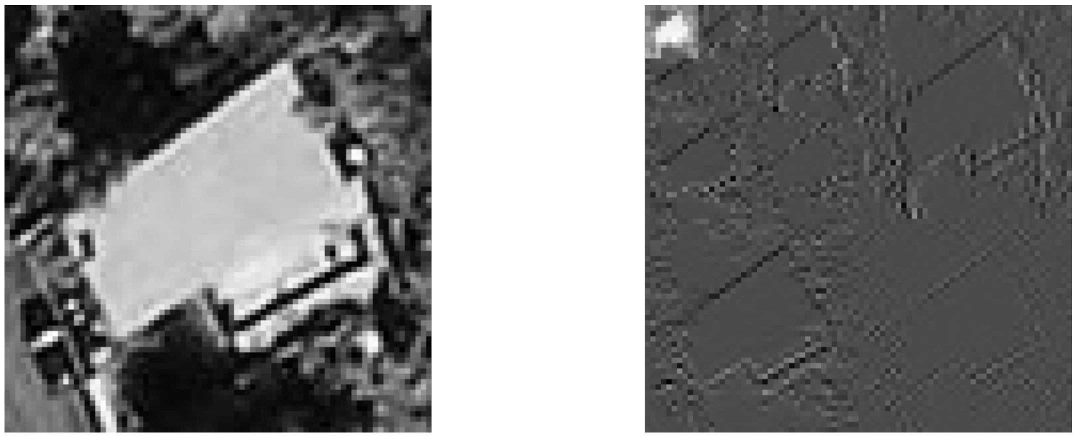

Figure 4.15: Left: Aerial 80x80 image of a house, extracted from ICC[1]. Right: Haar wavelet decomposition with 3 scales of the image on the left.

and extracting $9 * Npixels * y * x$ features. This means, if we want to compute 5 statistics over co-occurrence matrices $C^{1,0^\circ}$ and $C^{1,90^\circ}$, extracting 90 features for each pixel.

4.3.8.3 Histogram Signatures

Mallat[32] found experimentally that the detail histograms of natural textured images can be modelled by a family of exponentials:

$$h(u) = Ke^{-(|u|/\alpha)^\beta} \tag{4.29}$$

The model parameters α, β, and K of a wavelet detail image D can be computed from:

$$m_1 = \int |u|h(u)\mathrm{d}u \quad \textit{estimated by} \quad MD = \frac{1}{N}\sum_{j,k}|D(b_j, b_k)| \tag{4.30}$$

$$m_2 = \int |u|^2 h(u)\mathrm{d}u \quad \textit{estimated by} \quad E = \frac{1}{N}\sum_{j,k}(D(b_j, b_k))^2 \tag{4.31}$$

Note that the mean of the wavelets detail coefficients equals zero, therefore E is exactly the variance and MD is exactly the mean deviation. With this, the parameters can be calculated as:

$$\beta = F^{-1}\left(\frac{m_1^2}{m_2}\right) \quad \text{where} \quad F(x) = \frac{\Gamma^2(2/x)}{\Gamma(3/x)\Gamma(1/x)} \tag{4.32}$$

$$\alpha = m_1 \frac{\Gamma(1/\beta)}{\Gamma(2/\beta)} \tag{4.33}$$

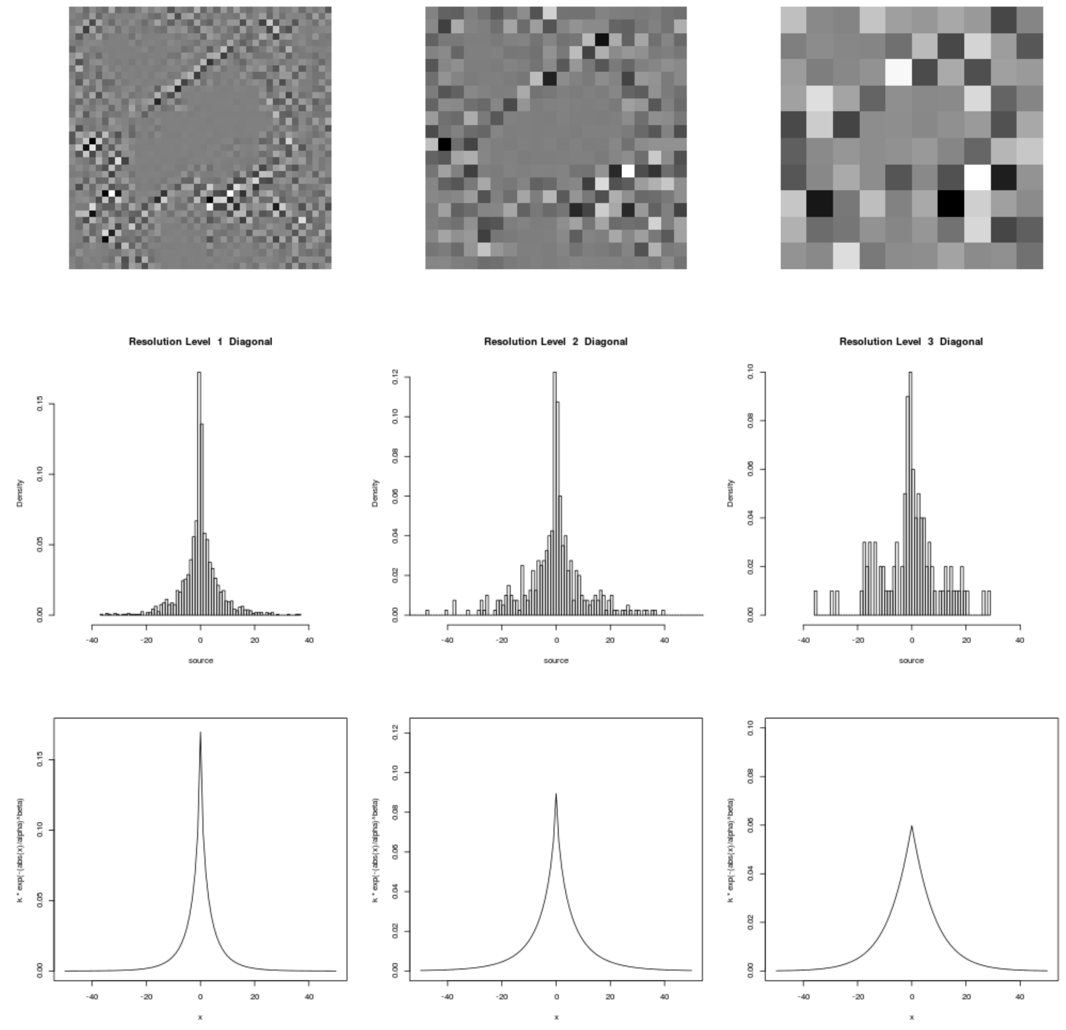

Figure 4.16: Wavelet detail images, histograms and models for the diagonal detail of the 3 scales in the haar decompoistion in Figure 4.15.

$$K = \frac{\beta}{2\alpha\Gamma(1/\beta)} \tag{4.34}$$

This transformation maps the correlated features E and MD into the *wavelet histogram signatures* α and β. Examples on this process applied to the wavelet decomposition in Figure 4.15 can be found in Figures 4.16, 4.17 and 4.18.

4.3.8.4 Parameters

In this section we will summarise the parameters that will be used to compute the features based on wavelets transforms:

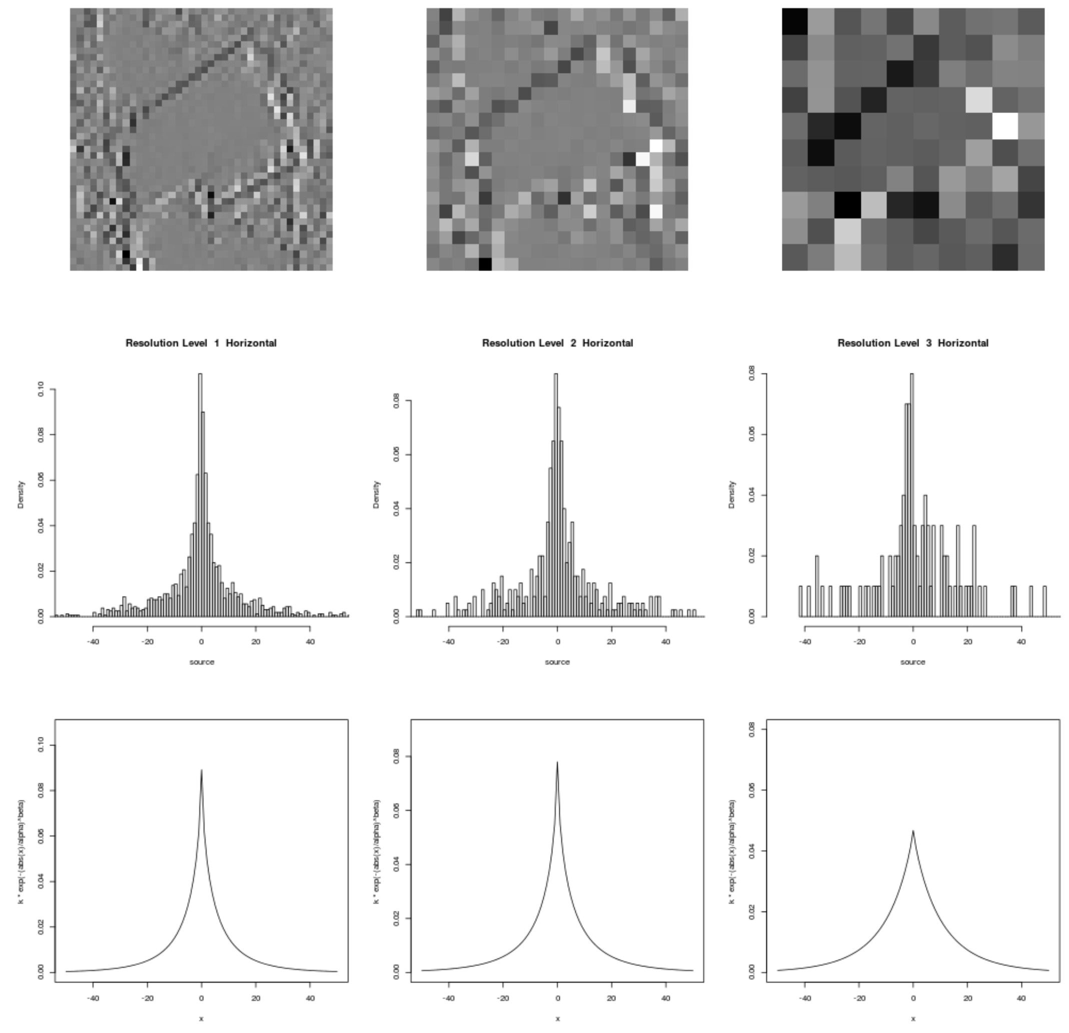

Figure 4.17: Wavelet detail images, histograms and models for the horizontal detail of the 3 scales in the haar decompostion in Figure 4.15.

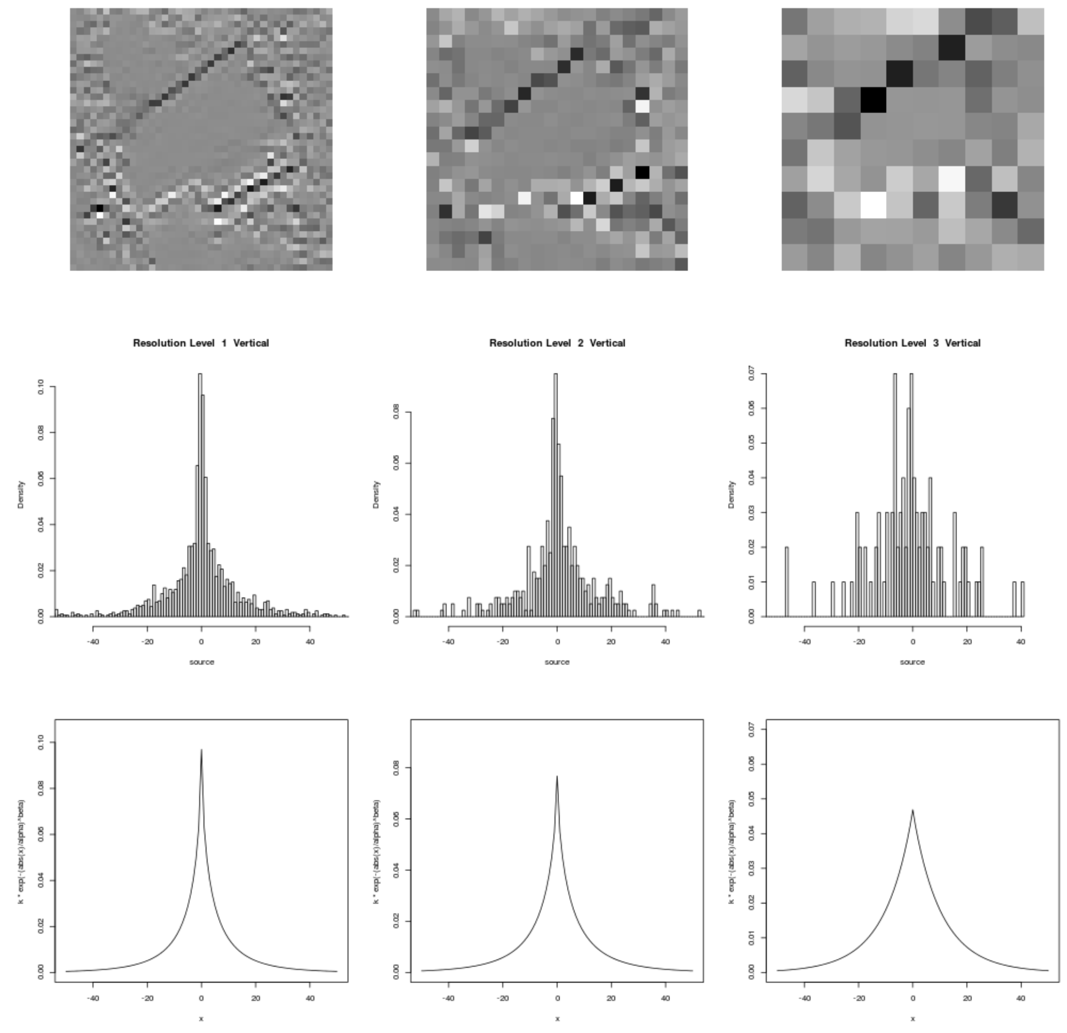

Figure 4.18: Wavelet detail images, histograms and models for the vertical detail of the 3 scales in the haar decompostion in Figure 4.15.

- **Sliding windows size:** 48, 64, 80 and 96 pixels.
- **Wavelet Family:** Haar.
- **Resolution Levels:** 3.

4.4 Classification Methods

In this section, a quick overview of the *machine learning* used in this study is presented. Only few general ideas will be given on how each method works, for more detailed information check Appendix B.

4.4.1 Linear Discriminant Analysis

The Linear Discriminant Analysis (LDA) is a Generative Probabilistic Model, which means that it tries to establish linear decision boundaries that arise from simple assumptions about the distribution of the data. This method assumes that class-conditional densities are Gaussian and then explores the resulting form for the posterior probabilities.

The LDA method for multiple classes is implemented in the R library MASS[18].

4.4.2 Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA), like linear discriminant analysis (LDA), assumes that class-conditional densities are Gaussian. However, LDA also makes the assumption that the covariance of each each class-conditional density $p(x|C_k)$ is identical, whereas no such assumption is made in QDA. This gives place to quadratic functions of x.

The QDA method for multiple classes is implemented in the R library MASS[18].

4.4.3 Multinomial Regression

The Multinomial (or Multiclass) Logistic Regression is a Discriminative Probabilistic Model, which means that it uses the functional form of the generalized linear models explicitly and determines its parameters using maximum likelihood. There is an efficient algorithm finding such solutions known as iterative reweighted least squares, or IRLS.

The Multinomial Logistic Regression method for multiple classes is implemented in the R library nnet[18].

4.4.4 Model Selection

In the features section, we explained that many of the features that are being calculated depend on multiple parameters. In this section we explain how the optimal set of parameters will be selected. The same methodology will be used to determine which of the models, resulting from each one of the machine learning algorithms proposed, is the best.

In this study, the model selection problem is presented as an optimization problem in search of the lowest error rate. As stated before, the result of the different supervised classification methods is a labelling. If this labelling is compared to the expected class, a confusion table can be built and an error rate can be computed.

To compute the error rate, the performance on the training set is not a good indicator of predictive performance on unseen data. This is because reducing the error on the training set might lead to over-fitting the model to the training data. If data is plentiful, one could train a model with a training set and test it on independent data, called a validation set. However, over-fitting to the validation data might also occur.

Cross-validation consists in partitioning the training set into k chunks, using every combination of $k-1$ chunks to predict the remaining one, and finally computing the *cross-validation* error as the average error over all the executions. This process ensures that the resulting error rate is not dependant on a concrete set of data and, thus, it becomes a robust measure. The *cross-validation* error can be used for between-model comparison however, to assess the predictive performance on unseen data, another independent data set (which is usually referred as test set) is needed.

An important issue of the *cross-validation* method is that the number of training runs that must be performed is increased by a factor of k, and this can prove problematic for models in which the training is itself computationally expensive. Moreover, exploring combinations of settings of parameters (for instance, for feature parameters), in the worst case, requires a number of training runs that is exponential in the number of parameters.

Considering everything that has been explained so far, a training set and a test set must be chosen. We have performed a manual classification of our case study image, which can be seen in Figure 4.19, and 40,000 pixels have been chosen for the train set and 100,000 pixels have been chosen for the test set. The choice has been done randomly, nevertheless no pixel closer than 50 pixels of any image border has been chosen. This is because we do not want to take into account the error due to border effects. This decision is justified since we assume that we have access to the images that surround our case study (which in fact is true, because we could take those from ICC) so, if commercial application was to be developed, border effects could be avoided.

There are seven pixel descriptors which are not parametrized, namely *red*, *green* and *blue*, which are taken from the original image, and *ndvi*, *hue*, *saturation* and *lightness*, which can be computed as explained in the previous section. These will be referred as the basic set of

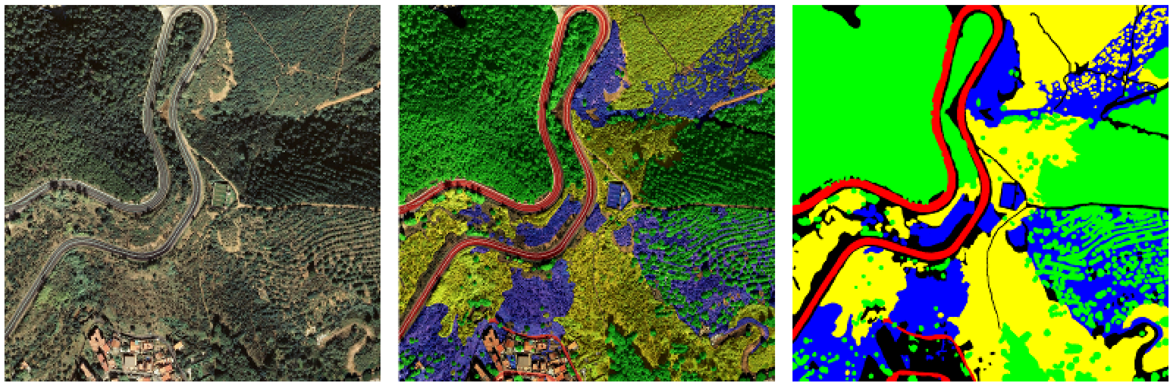

Figure 4.19: ©Cartographic base orthophoto property of the Institut Cartogràfic de Catalunya, available on www.icc.cat[1]. Left: Original image. Right: Manual labelling into 5 classes: trees (green), shrubs (yellow), grass (blue), road (red), no-vegetation (black). Center: Manual labelling overlayed over the original image.

features. Realize that the five remaining descriptors are parametrised and that they must be tested for the three selected classification algorithms. With these premises, exploring all the possible combinations using *cross-validation* becomes unfeasible and, therefore, an heuristic approach will be taken.

Each feature will be tested independently, in search of the best possible parameter setting. This test will be carried out only using the feature being tested and the base set of features, since they always remain constant. A parameter is considered optimal when it minimizes the *cross-validation* error. Local optimal parameters will be the starting point for a future heuristic optimization.

During a second step, all the descriptors computed with their optimal parameters will be used. One feature will be chosen at a time and, then, each of its parameters will be tested while keeping constant the parameters for the rest of the features. If a better parameter setting is found for the feature that is being tested, it will replace the original setting and then the algorithm will proceed testing another feature. This step will be repeated until no reduction on the error is observed.

Once the optimal model has been computed for each of the three classification algorithms (LDA, QDA and Multinomial Regression), the one that holds the lowest *cross-validation* error will be selected as our final model. Computing the test error using this model will give us a measure of the predictive performance on unseen data.

4.5 Summary

Now that we have seen how the extraction of features and the classification process will be performed, we are going to do a fast summary of all the process before presenting the results.

The whole process is illustrated in Figure 4.20.

1. The test and training sets are selected independently, choosing random points over our case study. There is no overlapping between these sets.
2. All the described features, with all their possible parametrization, are computed over the training set.
3. The model selection process is conducted, and the best model and the optimal set of parameters are chosen.
4. The features, with their optimal parametrization, are computed over the test set.
5. The test set is classified using the optimal model and the test error is computed comparing the results of the classification with the manual classification. With this error we can asses the predictive performance on unseen data, of the optimal model.
6. The features, with their optimal parametrization, are computed over all the pixels on our case study.
7. Our case study is classified, to provide an input for the following phase.

4.6 Results

The script *indexes.R* contains the code used to randomly generate the training and test sets. The first thing that must be done is to check that the rate between classes, in these sets, is similar to the original image. The training and test sets that will be used in this study are illustrated, as points over the original image, in Figure 4.21, and the results for the class distribution test can be seen in Table 4.2.

	Trees	Shrubs	Grass	Road	No Vegetation
Original Image	38.85%	27.39%	17.59%	4.99%	11.18%
Training Set	38.06%	27.66%	18.33%	4.85%	11.1%
Test Set	37.86%	28.22%	18.11%	4.96%	10.85%

Table 4.2: Percentage of each class in relation to the total number of pixels in each set.

Following, the results of individually testing each descriptor are going to be exposed:

First of all, we are showing the classification errors for the base set of features in Table 4.3. The base set of descriptors it can be calculated using the code in the script *features.rgbnhsl.R* and its tests are written in the script *test.rgbnhsl.R.*

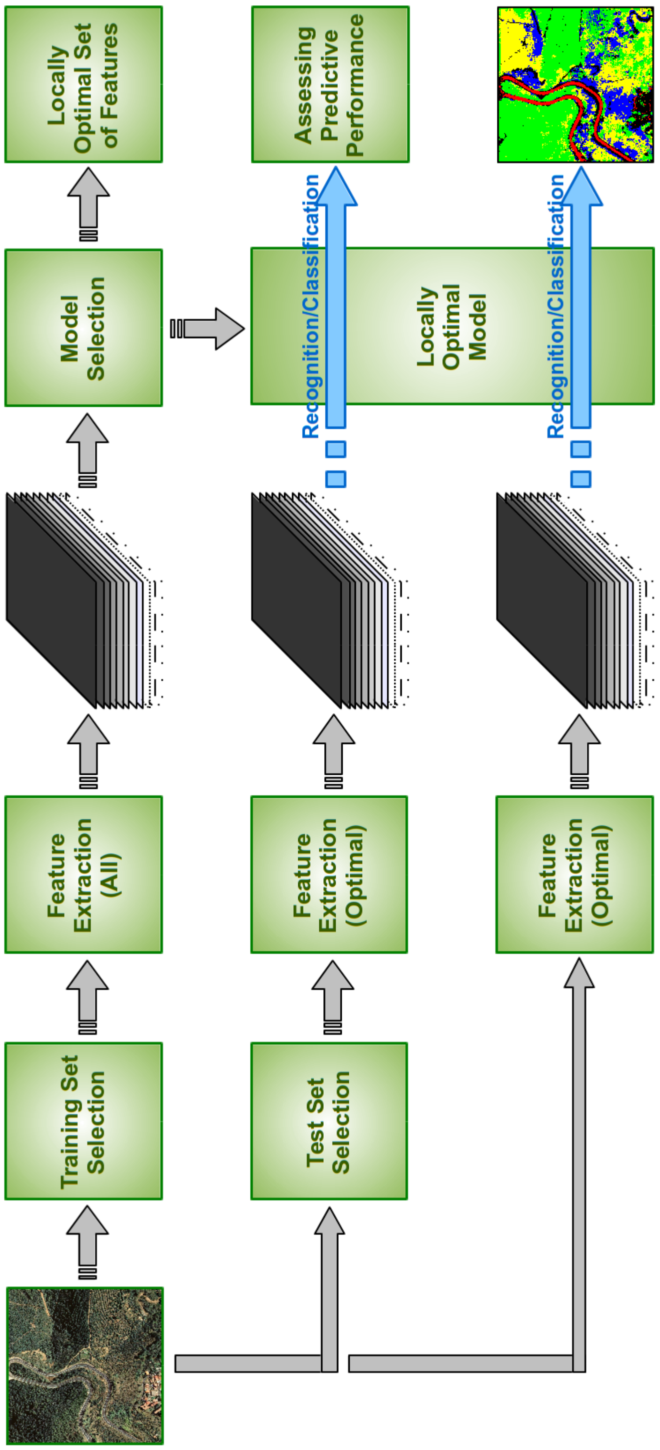
Training Set Selection
Feature Extraction (All)
Model Selection
Locally Optimal Set of Features
Test Set Selection
Feature Extraction (Optimal)
Recognition/Classification
Locally Optimal Model
Assessing Predictive Performance
Feature Extraction (Optimal)
Recognition/Classification

Figure 4.20: Illustrating the feature extraction and classification process.

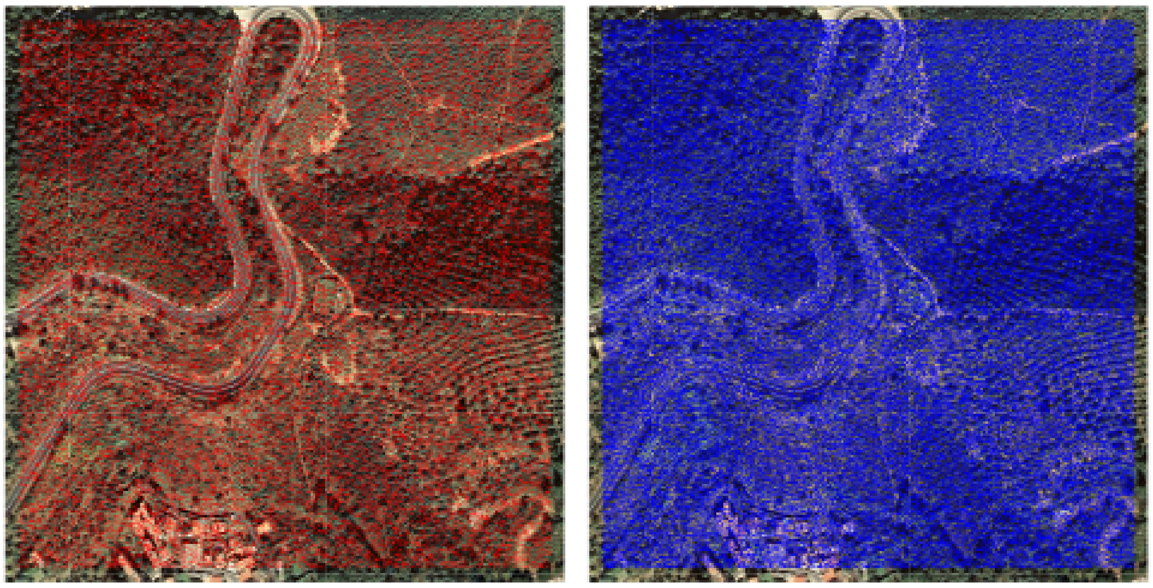

Figure 4.21: Left: Training set illustrated as red points over the original image. Right: Test set illustrated as blue points over the original image.

It can be observed that the error rate varies between the 45% and the 47%. Even if this number might seem quite bad, taking a look at the confusion matrix, shown in Table 4.4, one could notice many interesting details. Actually, the biggest part of the error comes from mistaking kinds of vegetation (35.7525%) whereas a smaller amount is due to a mismatch between vegetation and no vegetation or road (10.1275%). This was expected because, as stated before, the NDVI descriptor is a very powerful discriminator when it comes to detect green vegetation, which is the kind mostly present in our case study.

The effort, from now on, will be focused on further reducing the global error. However, it could happen that while the between kinds of vegetation error is reduced, the other errors are increased. We wish to reduce all kinds of errors at the same time but, specially, we want to keep low the error due to mistaking road pixels for vegetation pixels. This due to the fact that rendering a tree where in the place of a shrub is way more tolerable than rendering a plant is the middle of a road.

	LDA	**QDA**	**Multinomial Regression**
Cross-Validation Error	46.495%	47.335%	45.255%

Table 4.3: Cross-validation error for the classification process using the base set of descriptors, namely RGB, NDVI and HSL.

The script *features.glcm.train.R* contains the code needed to compute the grey-level coocurrence matrix descriptors for the training set, whereas the script *features.glcm.R* contains the code needed to compute these descriptors for the test set and for those pixels that

	Trees	**Shrubs**	**Grass**	**Road**	**No Vegetation**
Trees	29.4625%	5.8775%	2.1100%	0.3900%	0.2225%
Shrubs	14.9950%	7.8025%	4.0850%	0.1700%	0.6100%
Grass	3.9325%	4.7525%	7.3925%	0.0700%	2.1825%
Road	1.6075%	0.0275%	0.2025%	2.7300%	0.2825%
No Vegetation	1.9575%	0.3750%	2.3125%	0.3325%	6.1175%

Table 4.4: Confusion matrix for the cross-validation classification process, using the LDA algorithm with the base set of features. Rows correspond the expected labels and columns correspond to the classification results. This means that, for instance, a 1.6075% of the error is caused by classifying road pixels as trees whereas 0.39% of the error is a consequence of classifying tree pixels as road.

one would wish to classify.

Henceforth, this code division will be made for each descriptor. The reason behind this is that the purpose of **.train* scripts is to compute the desired features with all their possible parametrisation. Once we know which is the optimal parametrisation and, consequently, the classification model can be computed, if it is wished to classify a new pixel there is no need to calculate all possible parametrisation, only the optimal one is needed. Thus, the other scripts are for this purpose.

Going back to the grey-level coocurrence matrix descriptors, we can see the cross-validation errors for its parametrisation in Table 4.5, which were determined using the *test.glcm.R* script. Note that there is not much difference between using different distance vectors individually and, because of this, using some of them together was tried. The result is that for LDA and Multinomial Regression, the best result is obtained when using the 6 set of parametrised descriptors together, while in the contrary, for QDA it is better to use just one of them.

Distance Vector	**LDA**	**QDA**	**Multinomial Regression**
(0,1)	41.1625	40.69	39.2275
(1,0)	40.53	**40.215**	38.955
(0,3)	40.8025	41.0825	39.06
(3,0)	40.8775	40.4575	39.2525
(1,2)	40.905	40.4225	39.1025
(2,1)	40.7775	40.2525	39.0975
(0,1) & (1,0)	40.0425	42.135	38.5675
(0,3) & (3,0) & (1,2) & (2,1)	39.815	44.42611	37.5725
(0,1) & (1,0) & (0,3) & (3,0) & (1,2) & (2,1)	**38.6575**	44.21611	**36.5425**

Table 4.5: Cross-validation error for the classification process using the base set of descriptors plus the grey-level coocurrence matrix descriptors, computed using different values for the ***distance vector*** *parameter.*

The edge density descriptor is computed using two pairs of scripts, since two different methodologies are used. In the scripts *features.edges.train.R* and *features.edges.R* the descriptor proposed by L.A. Ruiz *et al.*, while in the scripts *features.log.train.R* and *features.log.R* the laplacian of a gaussian approach is taken. Using the script *test.edges.R* we have estimated the cross-validation errors, and these are shown in Table 4.6.

d	**LDA**	**QDA**	**Multinomial Regression**
1	42.63%	42.8625%	41.6825%
2	**42.4675%**	**42.7925%**	41.5900%
3	42.505%	43.2225%	**41.5100%**
5	42.71%	43.5225%	41.8550%
7	43.0875%	43.8425%	42.3925%
12	44.685%	44.82%	43.4575%
17	45.135%	44.96%	43.9125%
25	44.905%	44.715%	43.3625%

Table 4.6: Cross-validation error for the classification process using the base set of descriptors plus L.A. Ruiz et al.*[8] edge density descriptor, computed using different values for the parameter* ***d***.

Mask Size	**LDA**	**QDA**	**Multinomial Regression**
3	42.59%	43.1925%	41.6725%
5	42.8075%	43.495%	41.8625%
7	42.5775%	43.2525%	41.7325%
11	42.655%	43.31%	41.8975%
15	42.53%	43.225%	41.7175%
25	41.9675%	42.4625%	41.3550%
35	**41.935%**	**42.35%**	**41.0425%**
51	42.21%	42.37%	41.1775%

Table 4.7: Cross-validation error for the classification process using the base set of descriptors plus the laplacian of gaussian edge density descriptor, computed using different values for the ***zero crossing detection mask size*** *parameter.*

The script *features.ehd.train.R* contains the code needed to compute the edge histogram descriptors for the training set, whereas the script *features.ehd.R* contains the code needed to compute these descriptors for the test set and for those pixels that one would wish to classify.

Using the script *test.ehd.R* we have calculated the cross-validation errors using different values for its parameters, and these are shown in Table 4.8.

Notice that, unexpectedly, the error rates are not much better than the ones obtained for the edge descriptors. Looking at the image representation of these descriptor, in Figure 4.22 and Figure 4.23, one can grasp an intuition of what is happening.

Intuitively, the edge density descriptor and the edge histogram descriptor show high response in roads and man made structures. In fact this makes much sense because it is in

Threshold	LDA	QDA	Multinomial Regression
0.15	**41.7775%**	43.565%	**41.0725%**
0.25	42.4625%	43.13%	41.855%
0.35	43.21%	42.8675%	42.3275%
0.45	43.405%	**42.5425%**	42.4375%

Table 4.8: Cross-validation error for the classification process using the base set of descriptors plus the edge histogram descriptor, computed using different values for the ***threshold*** *parameter.*

this kind of elements that present very strong edges. Before, we have seen that the major part of the classification error comes from mismatching the type of plant and, therefore, a descriptor that, at first sight, it seems does not discern between these, does not have much place for improvement.

As an example, we present, in Table 4.9, the confusion matrix for the edge histogram descriptor computed with a threshold of 0.45. The decrease in the total error, when adding this descriptor to the base set of features, accounts for a 3%. Looking at the table we can see that a third of this is due to the reduction of the error of road pixel classified as plants. Recall that we are very interested in the reduction of this kind of error (because we do not want to render plants in the middle of a road) so, even if this descriptor does not contribute much to the reduction of the total error, it might prove useful.

	Trees	Shrubs	Grass	Road	No Vegetation
Trees	29.0725%	6.4850%	1.9325%	0.3600%	0.2125%
Shrubs	12.3075%	10.4925%	4.0425%	0.2150%	0.6050%
Grass	3.5950%	4.9225%	7.5900%	0.1550%	2.0675%
Road	0.6925%	0.2400%	0.1750%	3.5525%	0.1900%
No Vegetation	1.6700%	0.4450%	2.3500%	0.7425%	5.8875%

Table 4.9: Confusion matrix for the cross-validation classification process, using the LDA algorithm with the base set of features plus the edge histogram descriptor computed with a threshold of 0.45. Rows correspond the expected labels and columns correspond to the classification results.

The script *features.energy.train.R* contains the code needed to compute the Law's energy descriptors for the training set, whereas the script *features.energy.R* contains the code needed to compute these descriptors for the test set and for those pixels that one would wish to classify.

Using the script *test.energy.R* we have calculated the cross-validation errors using different values for its parameters, and these are shown in Table 4.10. These features are easy and fast to compute and also yield good classification results.

The script *features.spectra.train.R* contains the code needed to compute the power spectra descriptors for the training set, whereas the script *features.energy.R* contains the code needed

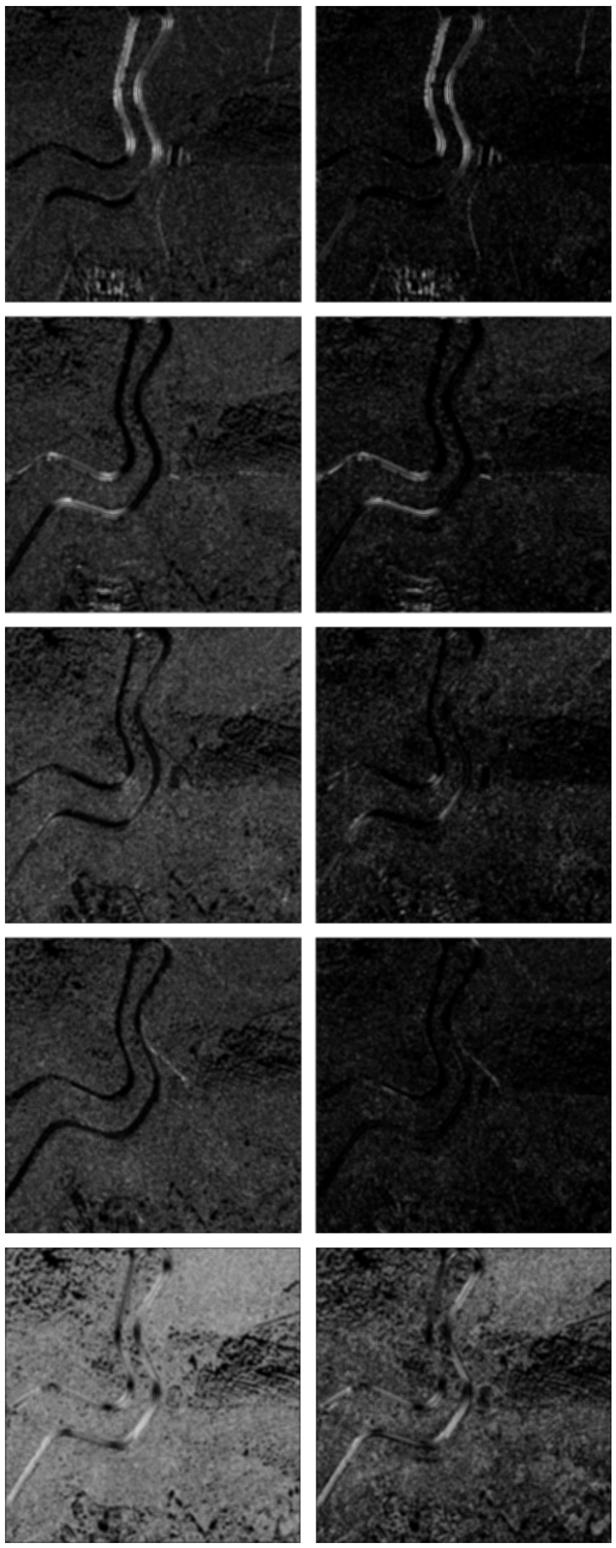

Figure 4.22: Image visualization of the Edge Histogram Descriptor computed with different thresholds. Left: threshold of 0.15. Right: threshold of 0.45. From top to bottom: vertical edges, horizontal edges, 45^o edges, 135^o edges and isotropic edges.

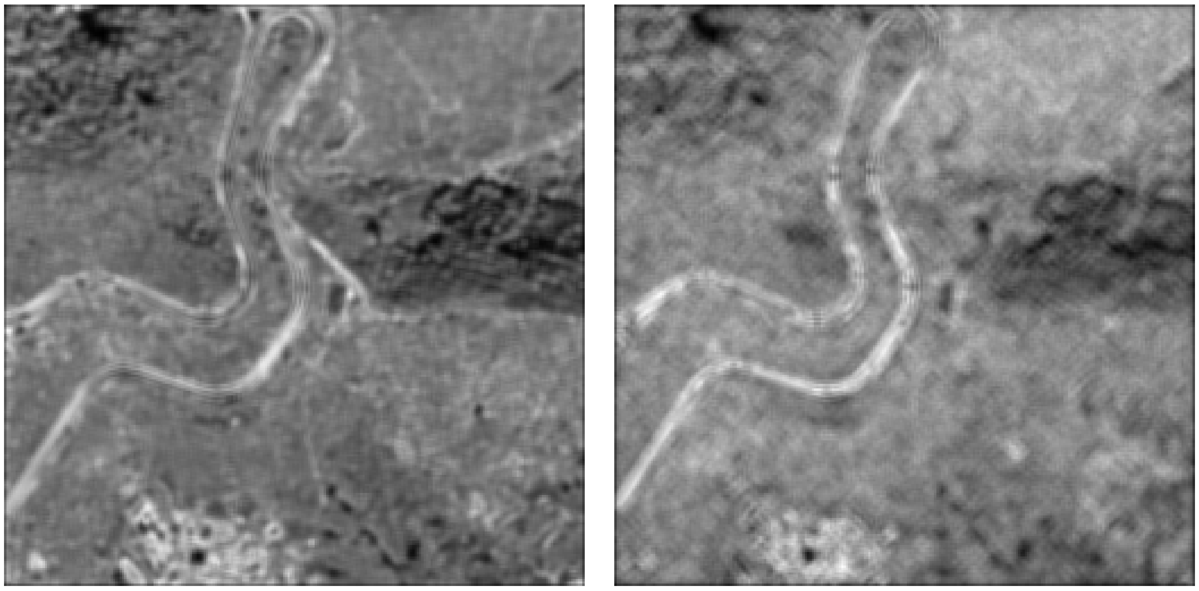

Figure 4.23: Left: Image visualization of L.A. Ruiz et al.*[8] edge density descriptor for d = 17. Right: Image visualization of the laplacian of gaussian edge density descriptor for a zero crossing detection mask size of 35.*

Window size	**LDA**	**QDA**	**Multinomial Regression**
15	43.0075%	41.73%	41.9075%
25	41.3325%	39.24%	40.31%
35	40.2875%	37.3725%	38.7525%
45	**39.64%**	**36.56%**	**37.895%**

Table 4.10: Cross-validation error for the classification process using the base set of descriptors plus the energy descriptors, computed using different values for the ***energy gathering window size*** *parameter.*

to compute these descriptors for the test set and for those pixels that one would wish to classify.

Using the script *test.spectra.R* we have calculated the cross-validation errors using different values for its parameters, and these are shown in Table 4.11.

The script *features.wavelets.train.R* contains the code needed to compute the wavelets descriptors for the training set, whereas the script *features.wavelets.R* contains the code needed to compute these descriptors for the test set and for those pixels that one would wish to classify.

Using the script *test.wavelets.R* we have calculated the cross-validation errors using different values for its parameters, and these are shown in Table 4.12.

Rings	**Windows Size**	**LDA**	**QDA**	**Multinomial Regression**
8	**35**	39.0375%	39.36%	37.01%
8	**45**	38.5475%	38.53%	36.37%
8	**55**	38.0225%	37.54%	35.91%
8	**65**	37.95%	37.025%	35.9%
16	**35**	38.9125%	40.125%	36.565%
16	**45**	37.67%	38.72%	35.27%
16	**55**	37.0725%	38.1175%	34.83%
16	**65**	**36.46%**	**36.9675%**	**34.2125%**

Table 4.11: Cross-validation error for the classification process using the base set of descriptors plus the power spectra descriptors, computed using different values for the ***sliding window size*** *parameter and for the* ***number of frequency rings*** *parameter.*

Windows Size	**LDA**	**QDA**	**Multinomial Regression**
48	37.9625%	37.505%	36.1275%
64	37.1%	36.2625%	35.375%
80	36.28%	35.33103%	34.415%
96	**35.535%**	**34.5975%**	**33.835%**

Table 4.12: Cross-validation error for the classification process using the base set of descriptors plus the energy descriptors, computed using different values for the ***energy gathering window size*** *parameter.*

The general trend is that the more descriptors, or the more complex the descriptor is, the best accuracy we obtain. Because of this, the final step will be to combine all descriptors together, to reduce the classification error as much as possible. From now on, we will be treating each classification method independently.

We shall first present the results for the Linear Discriminant Analysis. To begin with, we are summarising what parameters were found to individually make each feature optimal:

- **Grey-level Coocurrence Matrix:** The optimal setting was using all the features computer over the matrices with direction vectors $d_1 = (0, 1)$, $d_2 = (1, 0)$, $d_3 = (1, 2)$, $d_4 = (2, 1)$, $d_5 = (0, 3)$ and $d_6 = (3, 0)$.
- **Edge Density Descriptor:** The optimal feature was found to be the Laplacian of Gaussian descriptor computed using a zero crossing detection mask mask of size 35.
- **Edge Histogram Descriptor:** A threshold of 0.15.
- **Law's Energy Descriptors:** An energy gathering window size of 45 pixels.
- **Power Spectra Descriptors:** Using 16 frequency rings over a window of 65 pixels.
- **Wavelets Descriptor:** Using a 96 pixels sliding window.

This setup holds a cross-validation error rate of 30.9175%. Note that this is the lowest rate obtained so far, however, until now only independent optimizations have been carried out, for each descriptor. Thus, parameter optimizations taking into account the rest of descriptors could further reduce this mark. The idea is to locally optimise one feature at a time, but taking into account the effects that using the other descriptors may have. The iterations of this process are summarized in Table 4.13.

Iteration Step	Descriptor	**New-found Optimal Parameter**	**Error rate**
1	Edge Density	L.A. Ruiz *et al.* with d = 17	30.71%
2	EHD	Threshold = 0.25	30.6075%
3	Energy	Energy Gathering Window Size = 45	30.6075%
4	Wavelets	Sliding Window Size = 96	30.6075%
5	Spectra	Rings = 16, Window Size = 65	30.6075%
7	Edge Density	L.A. Ruiz *et al.* with d = 17	30.6075%

Table 4.13: Steps for the iterated local optimisation process.

The resulting error rate for this supervised classification procedure is 30.6075%.

Now we shall present the results for the Quadratic Discriminant Analysis. As in the previous case, first we will highlight the parameters the were found to individually make each feature optimal:

- **Grey-level Coocurrence Matrix:** The direction vector $d = (1, 0)$.
- **Edge Density Descriptor:** The optimal feature was found to be the Laplacian of Gaussian descriptor computed using a zero crossing detection mask mask of size 35.
- **Edge Histogram Descriptor:** A threshold of 0.45.
- **Law's Energy Descriptors:** An energy gathering window size of 45 pixels.
- **Power Spectra Descriptors:** Using 16 frequency rings over a window of 65 pixels.
- **Wavelets Descriptor:** Using a 96 pixels sliding window.

With this setting, we obtain an error rate of 0.328125%. However, as in the previous case, this is the result of individually optimizing each feature. We can attempt to improve this mark by doing a local optimization but taking into account all the descriptors at the same time. To do this, we will carry out the same procedure as for LDA, choosing a feature at a time and iteratively trying to find the parametrisation that gives better results when also using the rest of descriptors. The iterations of this process can be seen in Table 4.14.

The resulting error rate for this supervised classification procedure is 31.8275%.

Iteration Step	Descriptor	**New-found Optimal Parameter**	**Error rate**
1	GLCM	Direction Vector = (3,0)	32.73%
2	Edge Density	L.A. Ruiz *et al.* with d = 17	32.365%
3	EHD	Threshold = 0.15	32.205%
4	Energy	Energy Gathering Window Size = 45	32.205%
5	Wavelets	Sliding Window Size = 96	32.205%
6	Spectra	Rings = 8, Window Size = 65	31.9775%
7	GLCM	Direction Vector = (1,2)	31.87%
8	Edge Density	L.A. Ruiz *et al.* with d = 12	31.8275%
9	EHD	Threshold = 0.15	31.8275%
10	Energy	Energy Gathering Window Size = 45	31.8275%
11	Wavelets	Sliding Window Size = 96	31.8275%
12	Spectra	Rings = 8, Window Size = 65	31.8275%
13	GLCM	Direction Vector = (1,2)	31.8275%

Table 4.14: Steps for the iterated local optimisation process.

Finally, the last algorithm that must be reviewed is the Multinomial Regression. As follows, we are exposing the parameter setting that was found to individually make each descriptor optimal:

- **Grey-level Coocurrence Matrix:** The optimal setting was using all the features computer over the matrices with direction vectors $d_1 = (0,1)$, $d_2 = (1,0)$, $d_3 = (1,2)$, $d_4 = (2,1)$, $d_5 = (0,3)$ and $d_6 = (3,0)$.
- **Edge Density Descriptor:** The optimal feature was found to be the Laplacian of Gaussian descriptor computed using a zero crossing detection mask mask of size 35.
- **Edge Histogram Descriptor:** A threshold of 0.15.
- **Law's Energy Descriptors:** An energy gathering window size of 45 pixels.
- **Power Spectra Descriptors:** Using 16 frequency rings over a window of 65 pixels.
- **Wavelets Descriptor:** Using a 96 pixels sliding window.

This setup holds a cross-validation error rate of 28.5575%, which is really the best mark obtained so far. However, as in the previous cases, parameter optimizations taking into account the rest of descriptors could further improve this result. Table 4.15 shows the results of applying the iterated local optimization process to this setting.

The resulting error rate for this supervised classification procedure is 28.2875% which gives the best model found up until now. We have conducted a local optimization process and, therefore, this value belongs to a local minimum. As argued before, exploring all the possible combinations is unfeasible and, because of this, for this study we are satisfied with this result.

Iteration Step	Descriptor	**New-found Optimal Parameter**	**Error rate**
1	Edge Density	L.A. Ruiz *et al.* with d = 25	28.355%
2	EHD	Threshold = 0.35	28.3325%
3	Energy	Energy Gathering Window Size = 45	28.3325%
4	Wavelets	Sliding Window Size = 96	28.3325%
5	Spectra	Rings = 16, Window Size = 65	28.3325%
6	Edge Density	L.A. Ruiz *et al.* with d = 17	28.2875%
7	EHD	Threshold = 0.35	28.2875%
8	Energy	Energy Gathering Window Size = 45	28.2875%
9	Wavelets	Sliding Window Size = 96	28.2875%
10	Spectra	Rings = 16, Window Size = 65	28.2875%

Table 4.15: Steps for the iterated local optimisation process.

Yet, to assess the predictive performance of this model on unseen data, of this model, we must compute the error over the test set data, which is implemented in the script *test.error.R*. This give us a 28.428% error rate which, in fact, is quite close to this model's cross-validation error. The reason behind this is that the training set was plenty big and, thanks to this, the model is able to do a good generalization.

A 28% of error might seem a quite big rate, however, it might be good to take a look at the confusion matrix for the test classification, shown in Table 4.16. To make some measures a bit more explicit, we have grouped some kinds of errors in Table 4.17.

	Trees	**Shrubs**	**Grass**	**Road**	**No Vegetation**
Trees	32.405%	3.593%	1.234%	0.064%	0.565%
Shrubs	7.490%	17.256%	2.902%	0.007%	0.563%
Grass	2.059%	4.714%	10.232%	0.019%	1.089%
Road	0.083%	0.025%	0.020%	4.414%	0.416%
No Vegetation	0.938%	0.578%	1.665%	0.404%	7.265%

Table 4.16: Confusion matrix for the test set classification, using the final model. Rows correspond the expected labels and columns correspond to the classification results.

Road as Vegetation:	0.128%
Vegetation as Road:	0.09%
No Vegetation as Vegetation:	3.181%
Vegetation as No Vegetation:	2.217%
Vegetation Confusion:	21.992%

Table 4.17: Test error divided in categories.

The error that comes from mistaking road pixels for vegetation pixels is the one that concerns us most, nevertheless, we can see that it only represents a 0.128% of misclassified

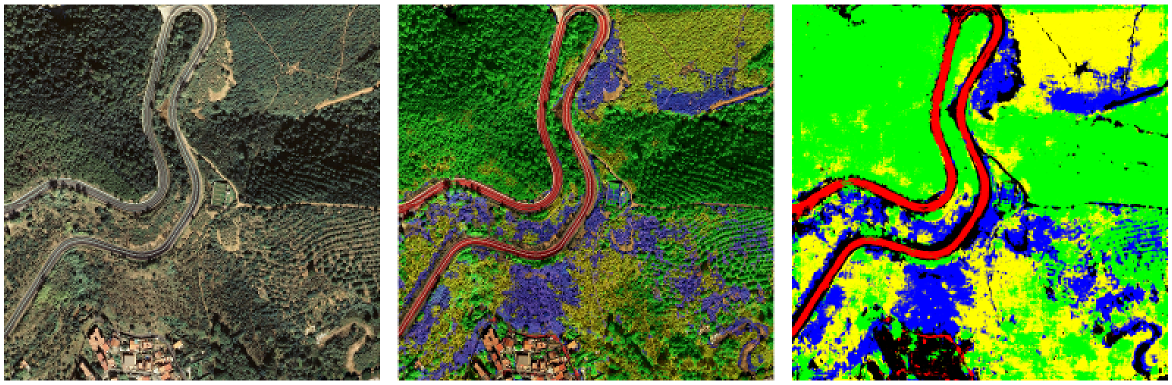

Figure 4.24: ©Cartographic base orthophoto property of the Institut Cartogràfic de Catalunya, available on www.icc.cat[1]. Left: Original image. Right: Results of the classification using the elaborated model. Center: Classification overlapped over the original image.

pixels over the total. Actually, this means that 128 pixels are being mistaken by plants, which stands for just a 2.65% out of 4958 road pixels.

The biggest contribution to the error rate comes from misclassifying vegetation into other types of vegetation. We could say that this is error is "benign", in the sense that rendering grass in the place of shrubs, or rendering shrubs in the place of trees, is visually more tolerable than rendering trees in the middle of a road.

The final step of this phase is to classify every pixel of our case study, so this information can become the input of the following phase. This classification has been done using the script *classifier.R* and can be seen in Figure 4.24.

4.7 Conclusions and Future Work

The first thing that might surprise the reader is that the improvement produced by using all the descriptors together is not much, in comparison with using some of them individually. For instance, with the wavelet descriptor a 33.835% of error is achieved (a reduction of an 11.39% over the error obtained using just the base set of descriptors), which differs just a 5.407% from the best error achieved.

We have given some insights about why some descriptors may not yield a great reduction of the error. For instance, we highlighted that the Edge Density descriptor and the Edge Histogram descriptor seem to help improving the classification of road pixels, however most of the error comes from vegetation type mismatching and, therefore, there is no much place for improvement. Moreover, these two descriptors are guaranteed to contain redundant information and, because of this, using them together might not bear much better results than using them individually.

Redundancy is a curse that probably affects all the used descriptors. To further illustrate this, note that Law's energy descriptors[30] also contain edge energy information and that high frequencies on the power spectra descriptor also measure the presence of edges.

Nevertheless, the cause of the low efficacy for the Edge Histogram Descriptor might be a very different one. Through all the experiments, a major trend is observed, the bigger the window the more accurate the classification is. The possible reason behind this was already pointed out at the beginning of the phase (Figure 4.1), even for the human eye, context is needed to accurately discern between some kinds of objects. A clear indicator of this is that the best results are obtained when descriptors are computed using big windows (such as energy, spectra or wavelets) whereas when using small windows the performance decreases (that is also the case of the Edge Histogram).

The major drawback of using big sizes for the windows is the possible increase of error due to border effects (missing information at the border of the image), however, we are not taking those into account. It is clear that there is much place for future work finding the optimal window sizes because, in general, in this study we have ran short.

There might be three other reasons as to why it has not been possible to further reduce the error. For beginnings, we are basing the whole process on a classification done manually, based on what an human could visually perceive and, even if it was not like this, there might be a significant underlying variance that can not be explained. In the case of digit recognition, it would be the same as being unable to discern if a handwritten digit is either a 1 or a 7. Indeed a second manual segmentation of the orthophoto by another user showed significant differences regarding the classification of vegetation pixels.

Another reason might be that, even if our descriptors were optimal, there might be some kind of information that we are not using. So computing other features might be needed. Finally, it is also possible that, even if using other descriptors, the models we are using are not powerful enough to make more accurate classifications. We have been mostly using linear models (except QDA that uses a quadratic model) which are fast and easy to compute, because the whole search for optimal parameters is already costly. However, a more power model (such as Neural Networks or Support Vector Machines) could lead to lower errors.

The reader should take into account that the trained model is expected to perform well on data examples similar to the ones that it has been trained with. This means that it will perform well when classifying images similar to our test case, and it does not have to perform well on others. The reason behind this is obvious, the classifier learns from its training examples, if we want to classify very different images, we must include few pixels from those in the training set.

The other major issue that this phase had is the computation time. As stated before, this is a research study, we are interested in finding a suitable solution for our problem and, as long as this happens, time should not be a problem. However, thinking in a future possible application, much work must be done in optimising the overall process. Some descriptors have taken up to 12 hours to be computed for each pixel of the 2048 x 2048 case study image, which is not feasible for a market application, since doing the manual labelling is faster.

In conclusion, we are very satisfied with the results of this phase because we have fulfilled our goals, within a reasonable bound. All that remains is to give few guidelines on what could be done as future work:

- A wider set of parameters can be tested, in search of improving the existing descriptors. For instance, it seems that there would be good results if the window sizes are increased.
- Those descriptors that show high redundancy with others should be removed. This way the final model could be simplified, reducing the overall computation time.
- To reduce the computation time, an implementation on a dedicated language (such as C++ using OpenCV) could be done. Also, another option is using libraries such as OpenMP (that offers parallelism for shader memory architectures), Cuda, or OpenCL (which are languages that take profit of the GPU's potential parallelism to speed-up applications).
- Finally, more powerful models, such as Neural Networks or Support Vector Machines, could be tested.

5 Phase 2: Classified Vegetation Analysis

5.1 Introduction to the problem

In the former section it has been shown how to detect vegetation pixels in an orthophoto, using both Computer Vision and Supervised Learning techniques. Therefore, at this point, we have computed an array of values that indicate if the pixel (x, y), on the original image, corresponds to a tree, a shrub, grass, road or something else.

In this phase, two main problems will be addressed:

- First, as seen previously the classification process is not flawless. Taking this into account, we can assume that the result of the classification is noisy. We conjecture that it might be possible to remove this noise partially, if domain specific knowledge is used.
- Second, the reader should notice that pixel-level classification is not directly useful for our goals. This is because a given large plant (such as trees or shrubs) most likely does not map to a single pixel of the original image, but to a group of them. As a consequence of this, a method must be described to form groups of pixels that can be associated with individual plants. As stated before, this study is aimed to generate a feasible reproduction of a natural landscape and this justifies the use of heuristic methods to solve this problem, generating an approximation of the reality.

5.2 Approach and Methodology

Aiming to partially remove the noise generated during the classification process, we turn to the field of Morphology Image Processing. Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image. These operations rely on the relative ordering of pixel values, rather than on their particular numerical values, and therefore they are especially fit for binary images.

Morphological techniques probe an image with a small binary image (which represents a shape) called a structuring element. The structuring element is placed at all possible locations in the image and it is compared with the corresponding neighbourhood of pixels. Morphological operations on binary images create a new binary image in which the pixel has a non-zero value only if the test is successful at that location in the input image.

On the structuring element, the pattern of ones and zeros specifies its shape. When one of those is placed over a pixel of an image the pattern is centered around its origin. The origin of a structuring element is usually one of its pixels, although it could also be outside of the matrix. However, a common practice is to choose an odd dimension for the element, and define the origin as the center of the matrix.

A structuring element is said to fit in the neighbourhood around a pixel if, for each of its pixels set to 1, the corresponding pixel in the neighbourhood is also 1. The structuring element is said to hit a pixel if, at least for one of its pixels set to 1, the corresponding pixel in the neighbourhood of this pixel is also 1. Zero-valued pixels are ignored, indicating that those points are irrelevant.

Now we will described the operations in which we are interested (an implementation of those can be found in the R library EBimage[17]):

- **Dilation:** The result of dilating a binary image with a particular structuring element, is a new binary image with 1 on those pixels where, on the corresponding pixel of the original image, the structuring element hits. Dilation adds layers of pixels to the inner and outer boundaries of the original image.
- **Erosion:** The result of dilating a binary image with a particular structuring element, is a new binary image with 1 on those pixels where, on the corresponding pixel of the original image, the the structuring element fits. Erosion removes layers of pixels from the inner and outer boundaries of the original image.
- **Opening:** Is the result of first applying an erosion followed by a dilation. Opening is an idempotent operations as long as the structuring element is not changed. An Opening erases from the original image those regions which are not big enough for the structuring element to fit.
- **Closing:** Is the result of first applying a dilation followed by an erosion. Closing is an idempotent operations as long as the structuring element is not changed. A Closing adds to the original image those regions which are not big enough for the structuring element to fit.

Thus, if we are dealing with a noisy binary image, we can erase stains, outside a region, using an opening, with an adequate structuring element, and we can fill holes, within a region, applying a closing, with an adequate structuring element. Nevertheless, for this we need binary images so we must separate the result of the classification into binary bands, as seen in Figure 5.1.

So particularly, we will be using morphological image processing for the following purposes:

- To remove the noisy pixels inside the road region. We will give priority to this processing because, as we have pointed out before, placing plants in the middle of roads must be avoided as much as possible.
- To remove the noise inside tree regions, shrub regions and grass regions.
- To remove those pixel groups which are not big enough to be considered trees or shrubs.

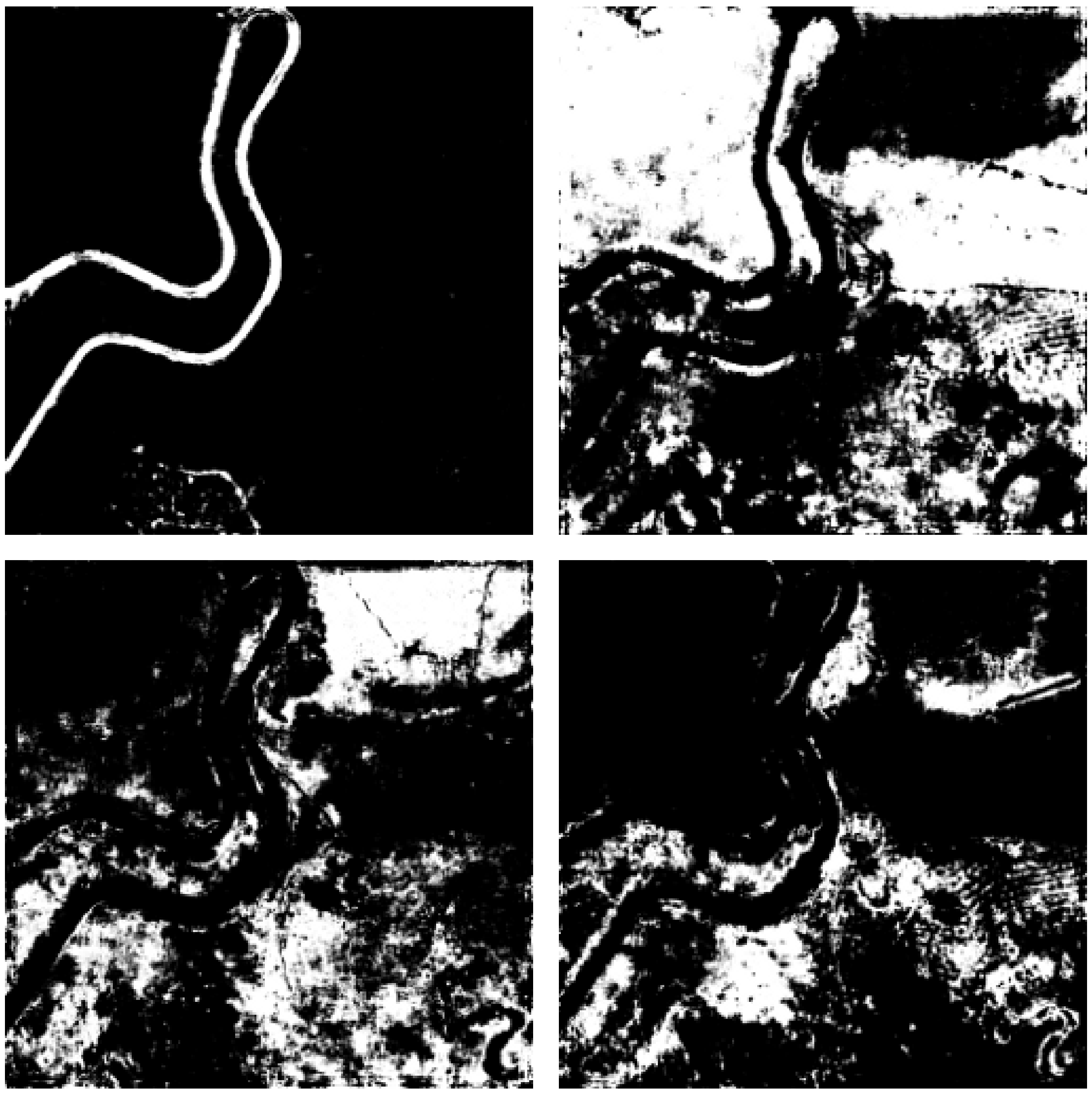

Figure 5.1: Top left: Binary band for the detected road pixels. Top right: Binary band for the detected tree pixels. Bottom left: Binary band for the detected shrub pixels. Bottom right: Binary band for the detected grass pixels.

Once the noise removal process is done, we are left with the task of identifying individual trees withing the processed tree band and identifying individual shrubs within the processed shrub band. For this, we turn to the field of Unsupervised Learning, particularly to the Clustering field. Our aim is to detect clusters of pixels which we can identify as an individual tree or shrub.

For this purpose we will use the *k-means* clustering algorithm (implemented in the R library cclust[19]), which takes as input a set of points in an n-dimensional space (in our case in a 2-dimensional space) and a number of clusters k, and outputs an assignations for each of the input points into one of the k clusters. To do this, the following steps are executed:

- **1:** Each cluster center is randomly initialized.
- **2:** Each point is assigned to its closest cluster. Different distance metrics might be used.
- **3:** Each cluster center is recomputed as the mean of the points assigned to it.
- **4:** Repeat 2-3 until no change is detected.

Note that the main issue of this algorithm is to choose the right number of clusters. In our case, we know how many pixel we want to process and we can estimate the area of the average tree or shrub. Doing an straightforward division between those two measures can give us an estimation of the number of clusters.

So the only thing we must do is to transform the pixels, in either the processed tree band or the processed shrub band, into their (x, y) coordinates in the image, and use this and the estimation of the number of clusters to run the *k-means* algorithm.

After this process, we have an assignation for each pixel in the processed image bands into one clusters that identifies what will become an individual plant. However, two problems might arise:

- Two pixels, that belong to two different regions that are not connected, might end up in the same cluster. If this happens, the center of the cluster might end up in a part of the image where there is no vegetation, like a road.
- A cluster might end up being too big, resulting in an unfeasible plant.

To solve this problem we purpose the following post-process:

- **1:** Choose as the new center of a cluster, the point belonging to the cluster that is closer to the old center.
- **2:** Erase from the cluster those points that are too far away from the new center.

- **3:** Compute the new center of the cluster, using the remaining points.
- **4:** Repeat 1-3 until no change is detected.

Now we have the definitive clusters of points which will become individual plants, and all that is left is to calculate some information about them. Each cluster can be represented as an irregular-shaped figure, therefore, its orientation and major and minor axis can be computed. Assuming that C_i is the set of points assigned to the cluster i, the orientation can be computed as:

$$\bar{x}_i = \frac{\sum_{(x,y)\in C_i} x}{|C_i|} \tag{5.1}$$

$$\bar{y}_i = \frac{\sum_{(x,y)\in C_i} y}{|C_i|} \tag{5.2}$$

$$a_i = \sum_{(x,y)\in C_i} (x - \bar{x}_i)^2 \tag{5.3}$$

$$b_i = 2 * \sum_{(x,y)\in C_i} (x - \bar{x}_i) * (y - \bar{y}_i) \tag{5.4}$$

$$c_i = \sum_{(x,y)\in C_i} (y - \bar{y}_i)^2 \tag{5.5}$$

$$\Theta_i = \frac{atan2(b_i, (a_i - c_i))}{2} \tag{5.6}$$

Once the orientation is known, we can rotate the points of each cluster according to this value and, then, the major and minor axis can be found as:

$$minx_i = min(x|(x,y) \in C_i) \tag{5.7}$$

$$miny_i = min(y|(x,y) \in C_i) \tag{5.8}$$

$$maxx_i = max(x|(x,y) \in C_i) \tag{5.9}$$

$$maxy_i = max(y|(x,y) \in C_i) \tag{5.10}$$

$$majoraxis_i = max(maxx_i - minx_i, maxy_i - miny_i) \tag{5.11}$$

$$minoraxis_i = min(maxx_i - minx_i, maxy_i - miny_i) \tag{5.12}$$

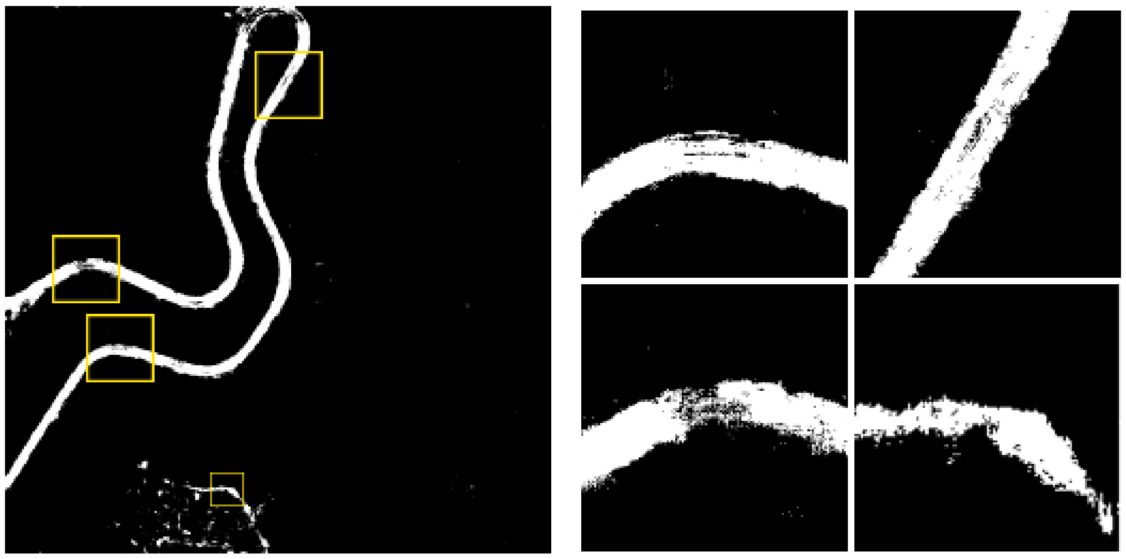

Figure 5.2: Zooming of certain road zones where the results of a noisy classification can be clearly observed.

5.3 Results

In Figure 5.2 we can clearly observe the need for applying a morphological processing to road pixels. If we were to use the original image as it is, we would find ourselves rendering plants in the middle of the road.

For the morphological processing of roads pixels, we have estimated that any road is at least 11 pixels (2.25 meters) wide. Taking this into account, we have applied the following processes over the road band:

- A closing using a disc of 11 pixels of diameters as structuring element. With this we can fill any hole in the road of size smaller than the structuring element. However, this process also makes more dense the noisy stains of road pixels.
- An opening using a disc of 11 pixels of diameters as structuring element. With this we can erase any stain of road pixels smaller than the structuring element (we consider that there is not any road narrower than 2.25 meters).

Finally, we decided that a pixel is considered road pixel if it is either labeled as road pixel in the original image or on the result of the morphology processing. We do this to be cautious because, as we have mentioned in the past, we prefer to label a vegetation pixel as road than doing otherwise. The results of this process can be seen in Figure 5.3.

Note that the irregularities of the road on the top and left of the image are due the border effects mentioned in the classification section. It would not be wise to adjust our process

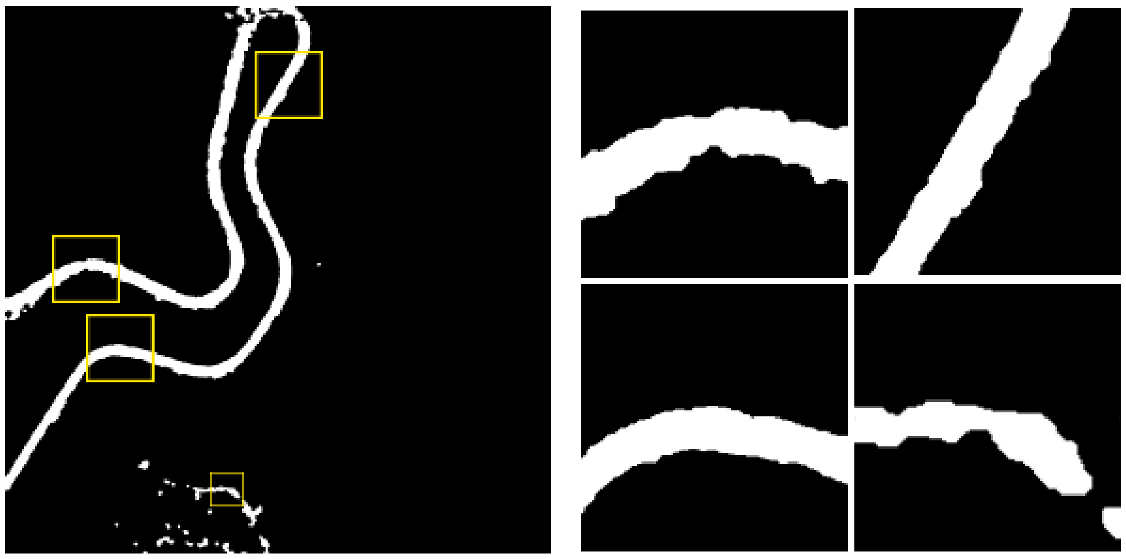

Figure 5.3: Zooming of certain road zones where the results of a morphology processing can be clearly observed.

to correct them because we have not even considered them during the previous phase, and doing so might cause negative effects on the normal road pixels.

Now, we show the process of removing the noise between different types of vegetation. We will apply a similar process as for the road (first a closing and then an opening), however in this case we do not know a priori the width of any kind of vegetation (there is a wide difference of widths between trees, shrubs or grass).

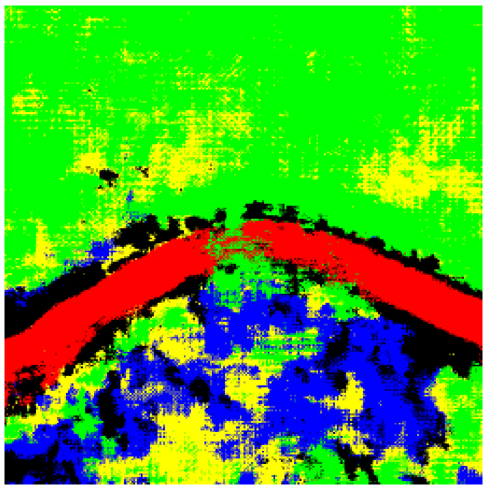

Figure 5.4: Example of noise.

If we look closely, the distance between noisy pixels in a noisy cloud is not big (between 1 and 9 pixels) as can be seen in Figure 5.4. For this reason, structuring elements of small size will be tested.

We wanted to illustrate the effect of using sizes 3 (Figure 5.5), 5 (Figure 5.6) and 7 (Figure 5.7) for a disc-shaped structuring element. We measure these effects in terms of leaving empty spaces where there was vegetation in the original image and in terms of overlaying kinds of vegetation.

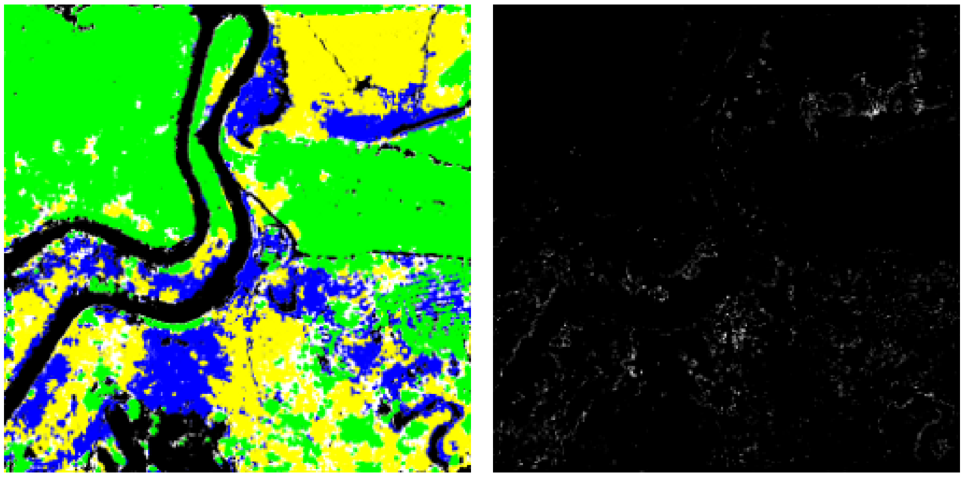

Figure 5.5: Results of applying a disc-shaped structuring element with diameter 3. Left: Overlaying of noisy-filtered vegetation bands. The regions that appear as white had vegetation on the original image. Right: As white, zones where the filtered vegetation bands are overlapped.

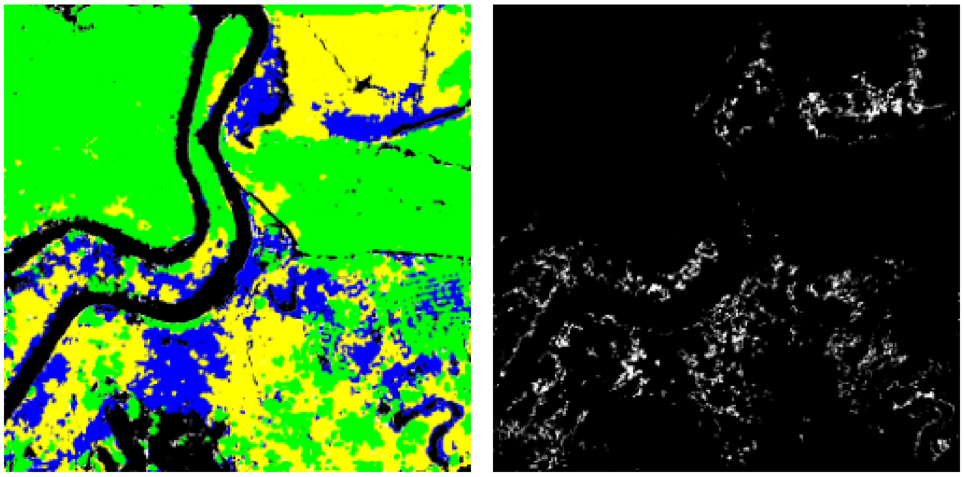

Figure 5.6: Results of applying a disc-shaped structuring element with diameter 5. Left: Overlaying of noisy-filtered vegetation bands. The regions that appear as white had vegetation on the original image. Right: As white, zones where the filtered vegetation bands are overlapped.

To understand what is happening in these images we must give a brief introduction to the next filtering process, which is only applied to the tree and shrub band. After filtering the noise, there are still groups of pixels that are not big enough to be considered either a tree or a shrub. For instance, a 2 x 2 group of pixels (50 x 50 cm) is not very likely to be identified as a tree, from a zenithal view, and therefore it must be discarded. The next filtering process will take care of this.

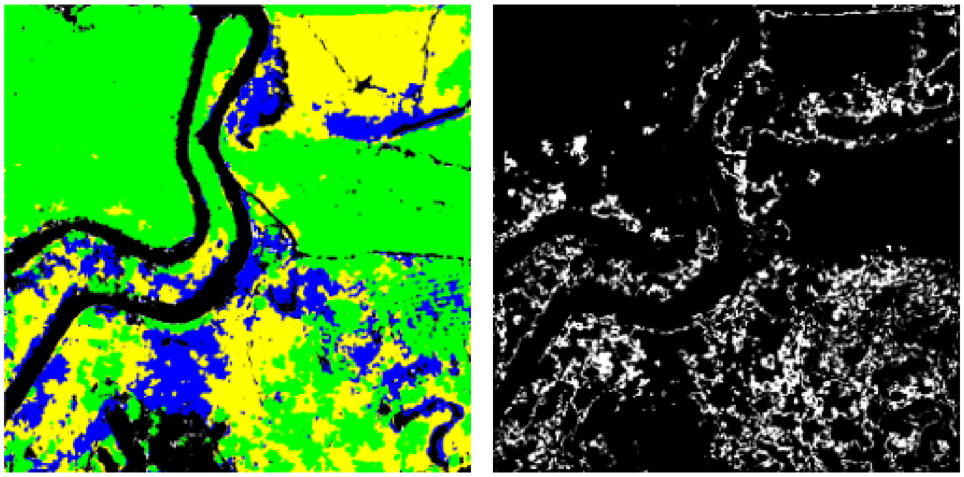

Figure 5.7: Results of applying a disc-shaped structuring element with diameter 7. Left: Overlaying of noisy-filtered vegetation bands. The regions that appear as white had vegetation on the original image. Right: As white, zones where the filtered vegetation bands are overlapped.

In Figure 5.5 the discs used to remove the noise are too small and, because of this, the noisy clouds do not become dense enough as to be detected as trees or shrubs in a second filtering. That is what is causing empty vegetation regions to appear. The only possible smaller size disc is one of diameter 1 which, in fact, would leave the bands as they originally were.

In Figure 5.7 the discs used to remove the noise are too big and, because of this, the noisy clouds become too dense and trees and shrubs are being detected where they shouldn't be. As a consequence, too much overlapping of vegetation bands is happening. Using bigger sizes would make this effect worse.

Using a disc of diameter 5 as structuring element is a trade-off between using a disc of diameter 3 and using a disc of diameter 7. In Figure 5.6 almost all vegetation regions in the original image are also filled in the filtered image and less overlapping than in figure 5.7 is happening.

In Figures 5.8, 5.9 and 5.10 we illustrate the effects of applying this noise removal process over the tree, shrub and grass bands, respectively.

As introduced before, the last morphological processing is only applied over the tree band and the shrub band, and aims to removes those isolated groups of pixels which are not big enough to be considered trees or shrubs, respectively. Note that a single pixel labelled as grass represents an area of 25 x 25 cm which, in fact, is already big enough to be considered grass.

For this process, we have estimated that the smaller feasible tree must have a diameter of at least 15 pixels (3.75 meters) and that the smaller feasible shrubs must have a diameter of at least 9 pixels (2.25 meters). The rest of the process is to apply an opening to the preprocessed

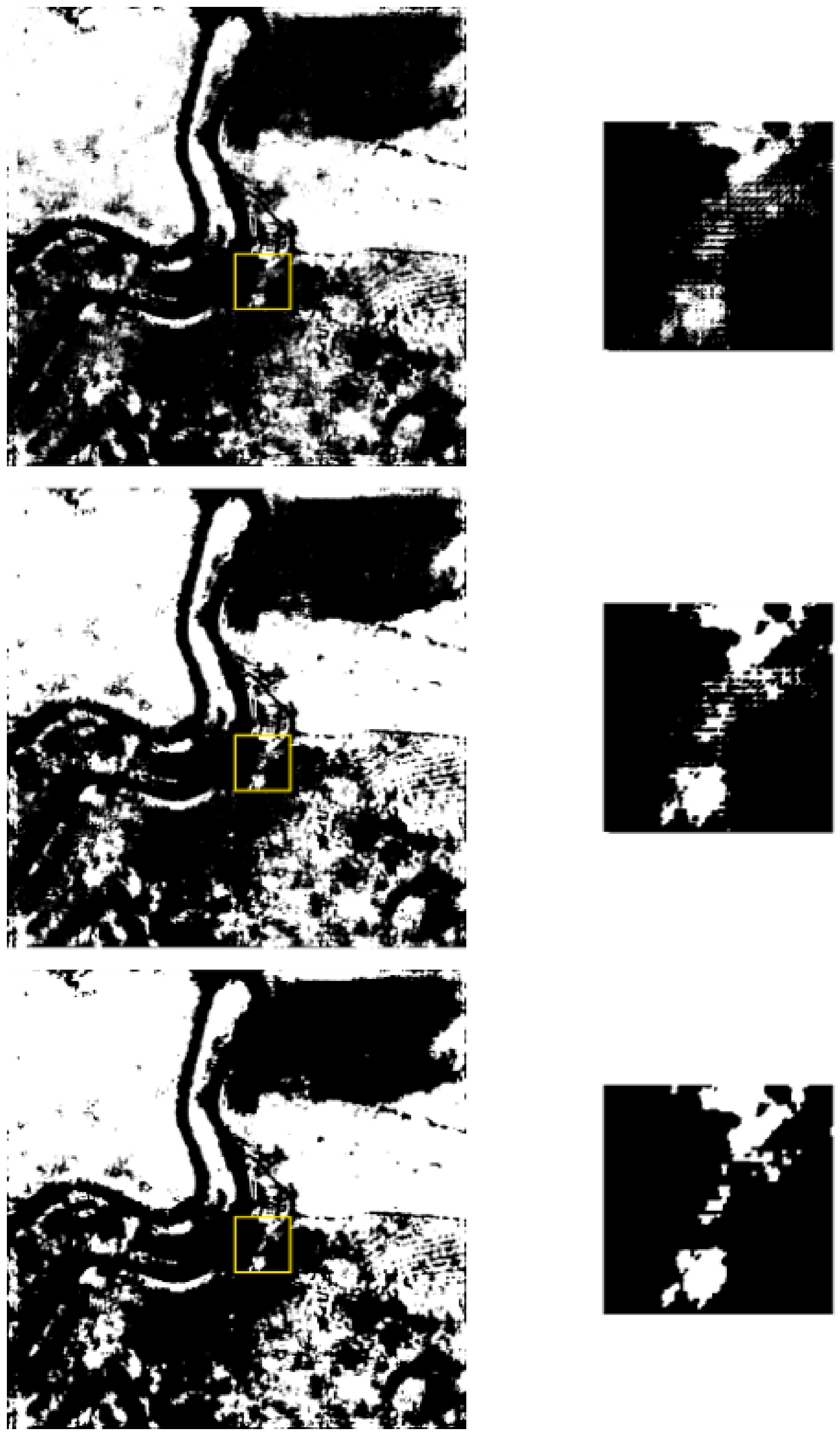

Figure 5.8: Result of noise removal using a disc-shaped structuring element of diameter 5 over the tree band. Top: Original band. Middle: Result of using a closing over the original image. Bottom: Result of using a closing followed by an opening over the original image.

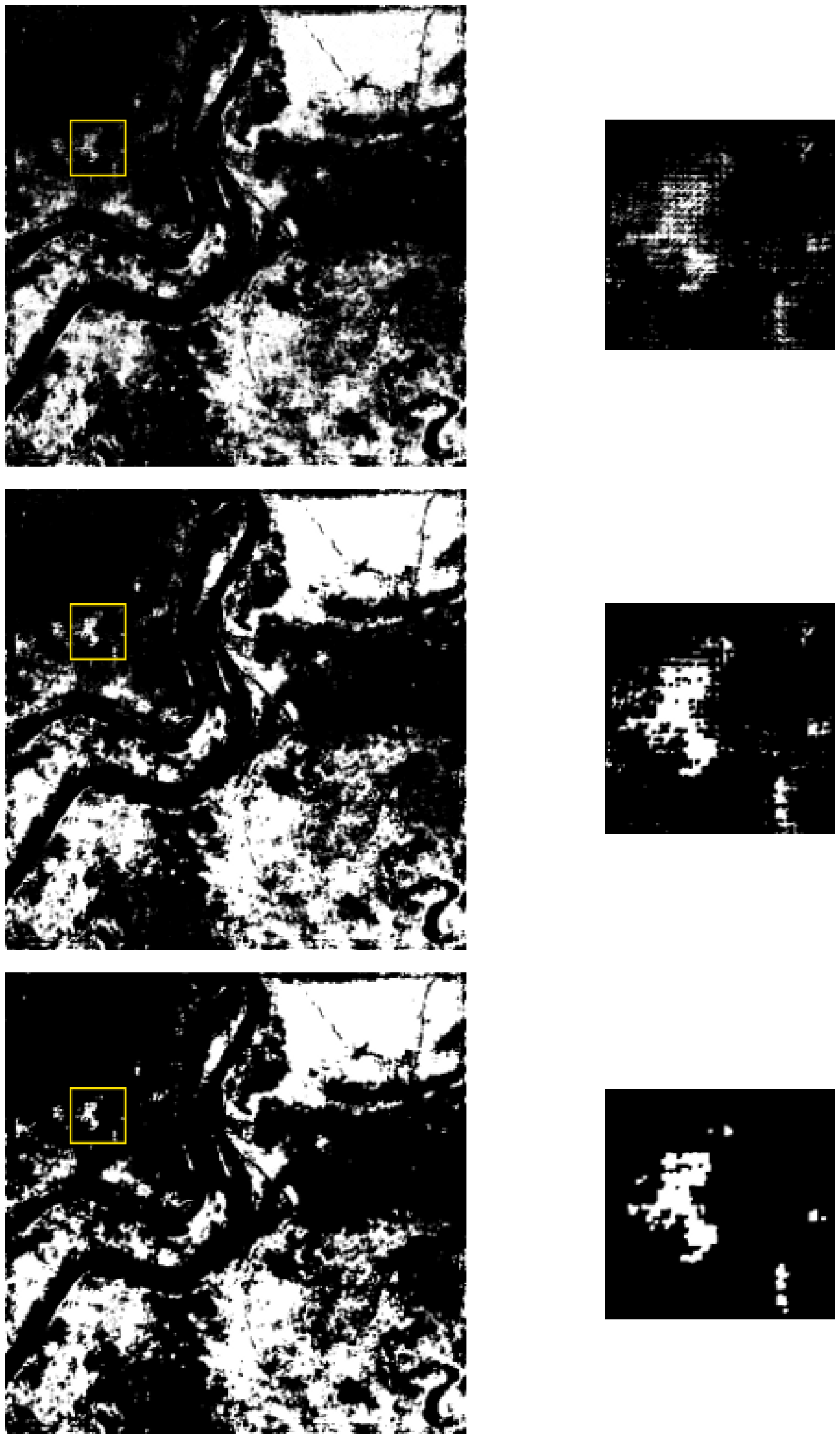

Figure 5.9: Result of noise removal using a disc-shaped structuring element of diameter 5 over the shrub band. Top: Original band. Middle: Result of using a closing over the original image. Bottom: Result of using a closing followed by an opening over the original image.

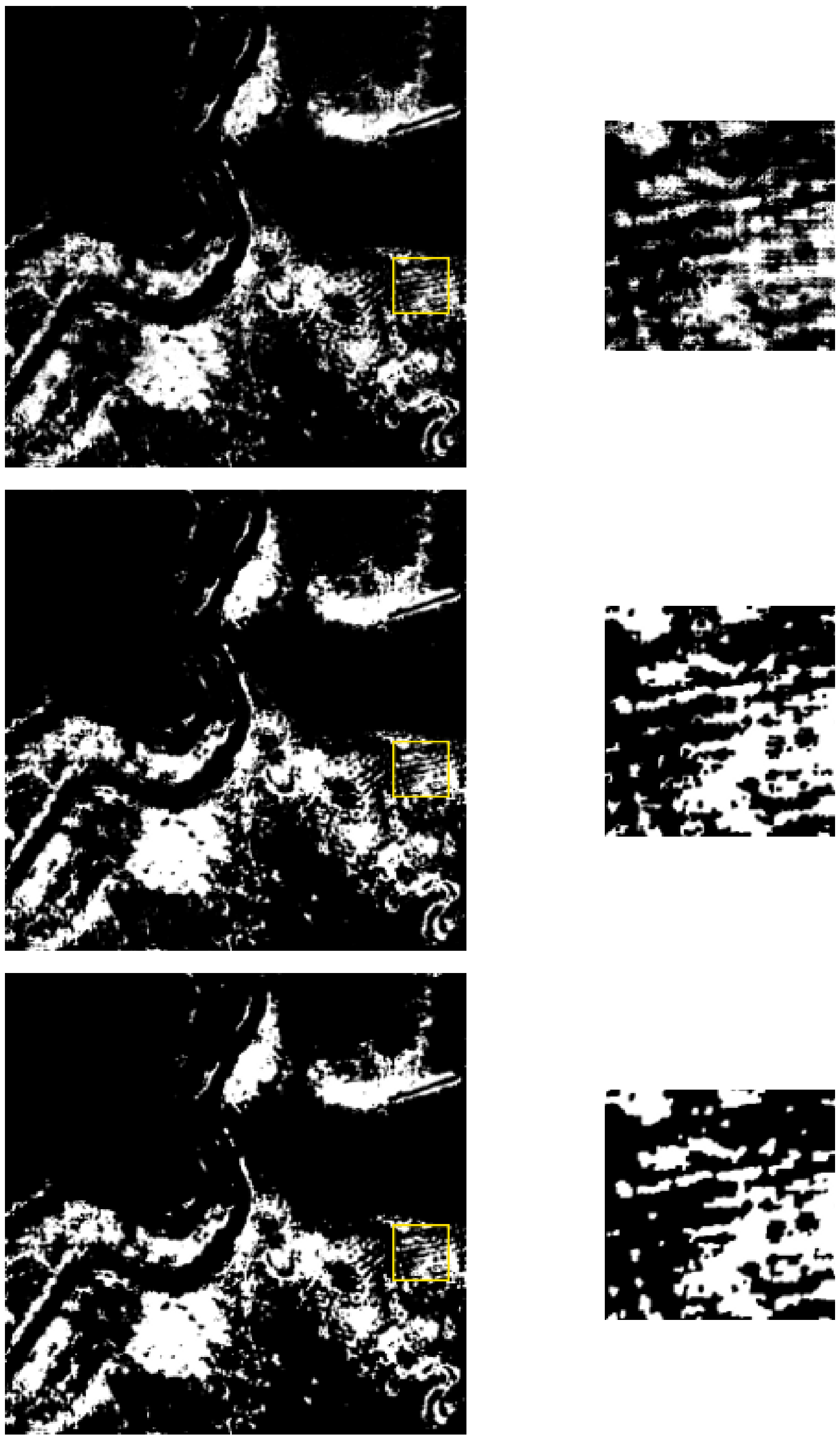

Figure 5.10: Result of noise removal using a disc-shaped structuring element of diameter 5 over the shrub band. Top: Original band. Middle: Result of using a closing over the original image. Bottom: Result of using a closing followed by an opening over the original image.

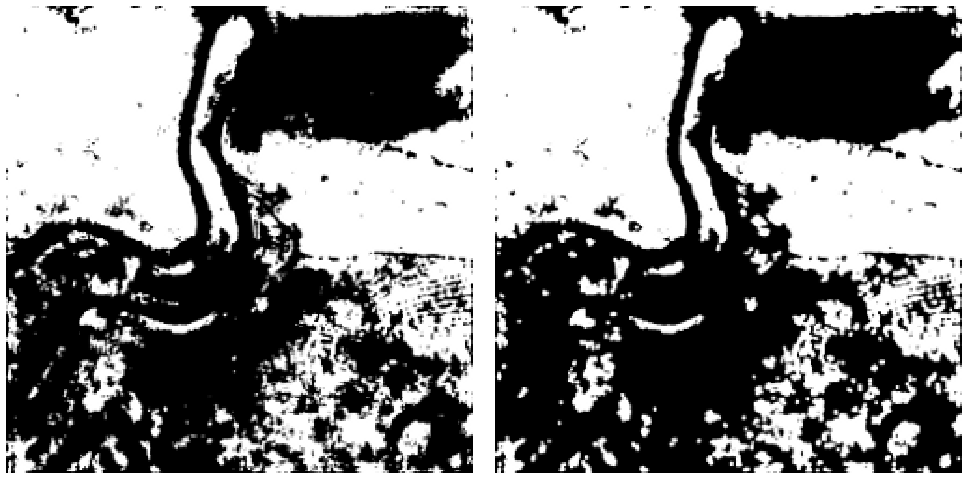

Figure 5.11: Right: Tree band after the noise removing process. Left: Result of applying an opening with a disc-shaped structuring element of diameter 15 on the image on the right.

tree and shrub bands using disc-shaped structuring elements with these diameters. The results of this process can be seen in Figure 5.11 for the tree band and in Figure 5.12 for the shrub band.

Finally, the last thing we have done to these bands is to remove the pixels that previously were labelled as road or no vegetation, as a precaution, and to merge the different bands. To decide which band dominated when two pixels from two different bands were overlapped, we took a look at the classification error. A 14.41% of tree pixels are misclassified whereas a 38.85% of shrub pixels are misclassified and a 43.51% of grass pixels are misclassified. Bearing this numbers in mind, we have given priority to the tree band, followed by the shrub band, followed by the grass band.

The morphological processing has been implemented in the script *morph.R* and its results, in comparison with the results of the classification, can be seen in Figure 5.13.

Now, we shall proceed to present the results for the second part of this phase. After the morphological processing, there are a total of 1,855,835 pixels labelled as trees, estimating that a normal tree area would be of 176 pixels (15 pixels of diameter), that yields 10502 trees or clusters. After applying the k-means algorithm to the tree band, we have post-processed the resulting clusters as explained in the methodology section. The result can be seen in Figure 5.14.

After the morphological processing, there are a total of 1,053,674 pixels labelled as shrubs, estimating that a normal shrub area would be of 95 pixels (11 pixels of diameter), that gives yields 11087 shrubs or clusters. After applying the k-means algorithm to the shurb band, we have post-processed the resulting clusters as explained in the methodology section. The result can be seen in Figure 5.15.

Finally, we have computed the orientation and major and minor axis of each cluster. This

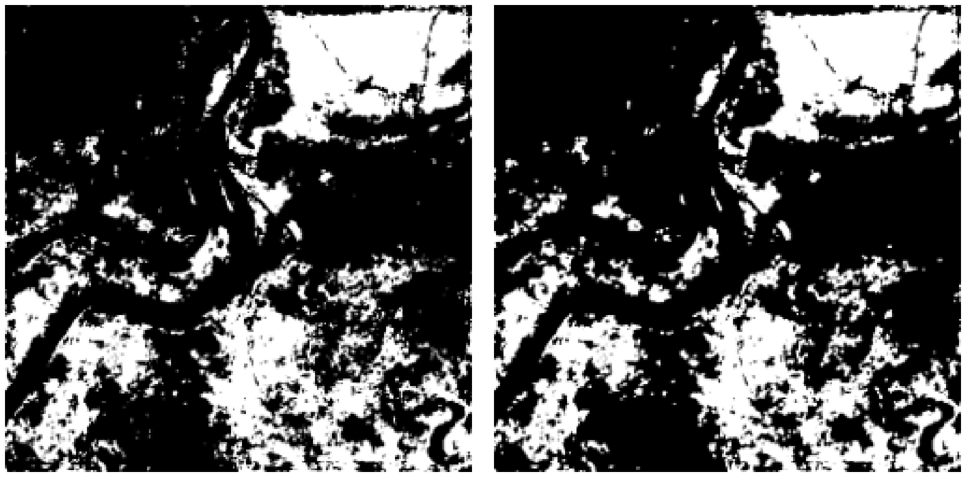

Figure 5.12: Right: Shrub band after the noise removing process. Left: Result of applying an opening with a disc-shaped structuring element of diameter 9 on the image on the right.

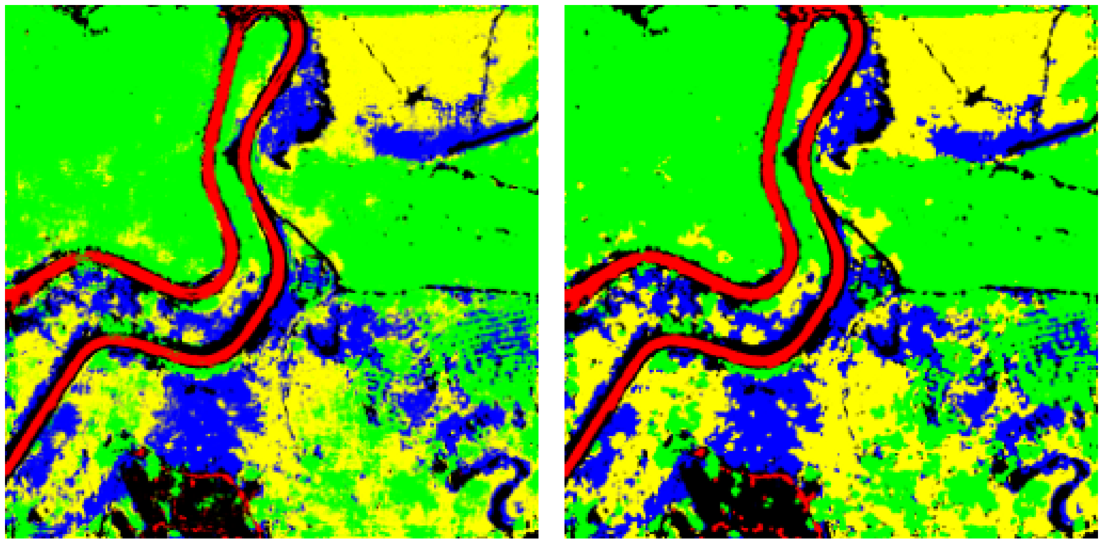

Figure 5.13: Right: Result of the classification process. Left: Result of the morphological processing.

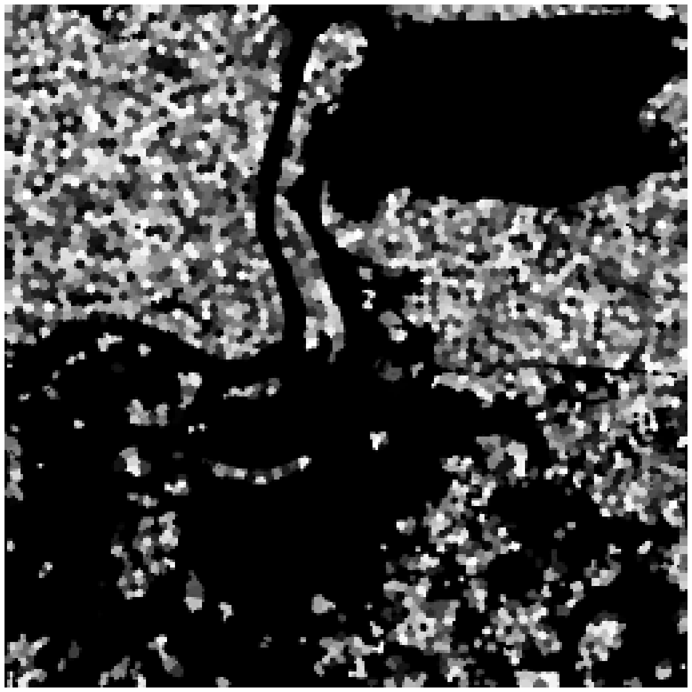

Figure 5.14: Clusters resulting from post-processing the result of applying the k-means *clustering algorithm to the tree band, using 10502 clusters.*

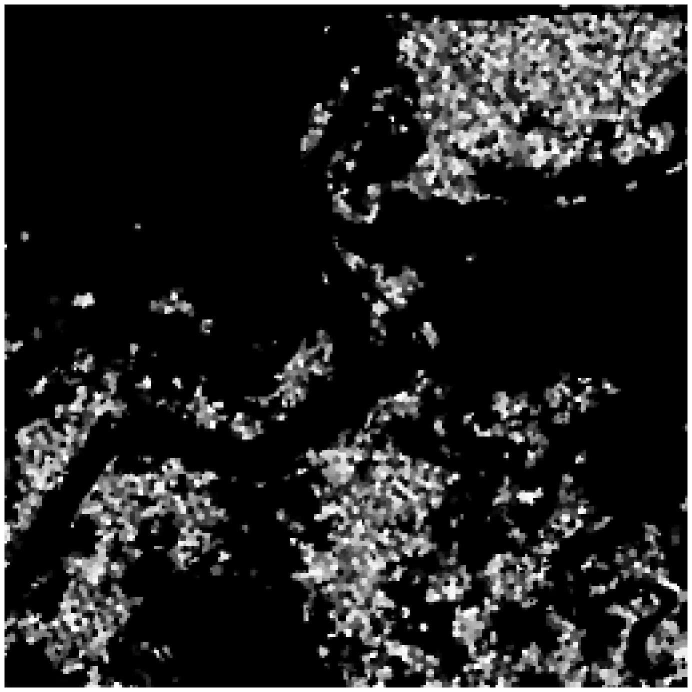

Figure 5.15: Clusters resulting from post-processing the result of applying the k-means *clustering algorithm to the shrub band, using 11087 clusters.*

information together with the center of each cluster has been saved into a file that will serve as input for the final application.

The clustering process and the information extraction process are implemented in the script *postprocessing.R*.

5.4 Conclusions and Future work

The first remark that must be done for this phase is that the overall process has been quite heuristic. All the estimations about road or plants sizes have been done doing an approximation based on common sense, and the experiments lack a bit of rigour, in the sense that, perhaps, wider number of parameters and approaches should have been tested to further assess the quality of the results.

However, everything that has been done in this phase has a strong theoretical base and the results has been proven to be quite satisfactory. Because of this, we can say that there is no urge to further improve this phase and that time can be spent on other more important tasks.

The reader must bear in mind that everything related to the morphological processing is very dependant on the previous phase results. If we were to change the classification procedure in a way that it would affect the resulting classification, there is no guarantee that the morphological process that we have described would work. On the other hand, if we use the same classification model to process images similar to our case study, we expect the described morphological process would produce good results.

On the same line as it was discussed in the previous phase, it might prove useful to spend some time doing a specific purpose implementation of the algorithms used. The morphological filtering operations have proven to be quite fast but the same can not be said for the clustering. The clustering of shrubs has taken around 4 hours to complete, while the clustering for trees has taken around 10. This is caused by the massive number of points and clusters in each case. In conclusion, doing an implementation of these algorithms in language, like C++, is an mandatory if a commercial application wants to be created.

Another issue that concerns us a little is that the *k-means* clustering algorithms, for this problem, results in very similar clusters (all of them have more or less the same size) that tend to be aligned. This layout mgiht appear very artificial and might provoke that the rendered 3D landscapes do not look natural. The only solution for this is to change the clustering algorithm or add a bit of random noise in the placement of the plants, in a later step.

Finally, we could say that we are rather extracting few information to characterize each cluster. There is place for future work on extraction more complex characteristics than the orientation and the major and minor axis for a cluster. For instance, one approach would be computing the convex hull of each cluster. However, at this point we are satisfied with just this information.

In summary, the result for this phase is good and meets our needs. We have been able to extract the basic information we need for the final application to render individual plants in their proper place. Nevertheless, as in any other project, there is always place for further investigation, refinement and improvement.

6 Phase 3: Vegetation Base Models Generation

6.1 Introduction to the problem

Up to this point, we have described how to process an orthophoto in search of different kinds of vegetation and how to artificially approximate the distribution of trees or shrubs, among a cloud of pixels classified as these. As a result, we have information on where the individual plants, of each kind of vegetation, must be placed and their artificially-determined properties.

The problem that will be addressed in this phase stands clear; it is known where to render each plant but we lack what to render. Especially for large plants (such as trees or shrubs) their complex branch structure needs to be reproduced to generate a realistic model.

Two options come up for the generation of such branching structures: designing these manually with the help of tools such as Blender, 3D Studio or Maya, or programming a piece of software to automatically generate them. We have opted for the former because we wish to automate the overall process as much as possible. Also, if done right, the final software should be able to generate as many models as wanted and this, together with the fact that no expert in designing tools will be needed, would decrease the cost of the final application, if it was wished to commercialise it someday. Nevertheless, it also has its setbacks because programming the software piece is an issue itself; the appropriate method must be selected and programmed in a way that it proves enough reusable and able to generate an enough variety of models.

Finally, the following issues over the generated model must be considered:

- They must prove realistic.
- They must be produced in a reasonable time.
- They must not be too complex (in number of polygons), in anticipation of what may happen in the final application. This means that, even if they prove feasible and realistic, it might be impossible to do a constrained-time visualisation of them. In consequence, a trade-off between complexity and realism must be found.

6.2 Approach and Methodology

For this phase, we propose to implement the Space Colonization algorithm[15] to solve the problem of generating 3D vegetation models. The idea behind this algorithm is explaining tree architecture in terms of competition for space. For this, an iteratively procedure is carried out where, at each iteration, new nodes are added to a tree structure formed in previous steps. This process is guided by the proximity of points marking the availability of free space. Several user-set parameters provide control over the process, which makes it possible to generate a wide variety of forms.

The main steps carried out by this algorithm are summarized below:

- 1: A three-dimensional envelope of the tree crown is given as an input. It should specified in a way that makes it easy to test if a certain point lies inside the volume.
- 2: The specified volume is filled with a predefined number of *attraction points*. The points occupy the space according to a user-defined distribution. These points mark the availability of free space for growth, and are eliminated when a branch reaches them.
- 3: A single node is defined at the base of the tree.
- 4: An attraction point is considered to influence a its closest node if it lies within an user-defined distance. If a node is influenced by at least one attraction point, a new node is created in the direction defined as the average of the normalized vectors towards all the attraction points that influence the original node, at an user-defined distance from this.
- 5: Once the new nodes have been computed, all the attraction points within an user-defined distance from any new node are erased.
- 6: Steps 4-5 are repeated until either no attraction point is left, no node is within the radius of influence of any attraction point or a number of user-defined iterations have been reached.
- 7: Different post-processes are applied over the tree skeleton. First, the skeleton structure might be simplified to reduce the amount of data needed to represent the tree geometry.
- 8: Each remaining node is moved in parallel to its more basal neighbour. Reducing branching angles can significantly improve the overall tree appearance.
- 9: Curve subdivision, extended to branching structures, might be applied to create more smoothly curved limbs.
- 10: The radius of each branch is computed in the following way: an initial width is assigned to the most outer branches. Proceeding from the branch tips to the tree base, when two branches, with radius r_1 and r_2, merge the radius r of the resulting branch can be computed using the formula:

  $$r^n = r_1^n + r_2^n \tag{6.1}$$

  Where n is a parameter for the algorithms that usually goes between 2 and 3.

- 11: Generalized cylinders, centered at the tree skeleton axis, are used to generate the tree geometry (the triangle mesh). Cylinders are oriented using a parallel transport frame[34] to minimize the twist between consecutive cylinder sections.

We part from a basic implementation of the algorithm in C++. Work will be done over this to improve its result and add desired functionalities but, to explain these, first we must outline which are the features of this system:

- Generating a basic tree skeleton using steps 1 to 6 of the Space Colonization algorithm.
- In step 1 a cubic revolution is used to define the three-dimensional envelope for the crown. In fact, the cubic curve is constructed as a Bézier curve. The enevelope is then filled with randomly generated point, keeping between them an user-defined distance.
- In step 3 a brute force search is conducted to find which attraction points influence a certain node.
- In step 4 a brute force is done to determine which attraction points should be removed.
- Computing the branches radius using step 10 of the Space Colonization algorithm.
- Visualizing the resulting tree skeleton using an approximation of the tree geometry, which is done using normal cylinders for the axes and spheres on the corners.

Even if not all the steps of the Space Colonization algorithm are implemented, the major issue that this implementation yields is that no triangle mesh is generated for the skeletons and, therefore, there is no way to save this information into a file that the final application could use. This and other matters will be solved by adding functionalities to the existent ones.

First of all, we are concerned with the manner in which the searches for close neighbours are performed during steps 3 and 4. To improve this, we aim to use the C++ library Approximate Nearest Neighbors[24] to improve this searches. ANN offers an implementation of the search data structure *kd trees*, which supports functions such as *fixed radius k-nearest neighbor search.* These can be used to find all the points (in an *n-dimensional* space) that are within certain distance from a query point.

If we use the search structures implemented in ANN, we conjecture that we can greatly improve the performance of finding all the attraction points that influence a certain node, in step 3. The same also applies to finding the attraction points that must be eliminated, in step 4.

Then, we will implement some of the other steps of the Space Colonization algorithms. We will begin by implementing step 7, which means implementing strategies to simplify the branching structures. For this two approaches will be considered; the first one contemplates eliminating a node if the two branches that that it connects form an angle smaller than a certain threshold. The second one will erase a node if the area between the two branches that it connects is smaller than a threshold.

Once the simplifying process is done, we will proceed to implement step 8, which stands for reducing the branching angles. For this, also two approaches will be considered; the first one will be the one described in the algorithm, namely, moving each node halfway to its more basal neighbor. The second one will test if the angle between two branches is bigger than a certain threshold and, if this happen, will reduce that angle to the threshold value.

Next, a we will implement a process that was not contemplated on the original algorithm. Taking into account that the final goal of the project is to render a massive number of these models (around 10000 trees and 11000 shrubs, as we have seen in the previous phase) we consider the use of impostors to reduce the quantity of geometry to be rendered.

Impostors are lower quality models, derived from the original ones, that are rendered far from the camera, where the differences between them and the higher quality models can not be observed. To generate impostors, from the tree skeletons that we have generated, we will simply prune the tip branches. This way, we will be keeping three different resolutions for each model.

Following, each model geometry must be generated. As mentioned before, we will use generalized cylinders for this, which means that the bases of these cylinders will be oriented according to the direction of the skeleton axes. The major issue with this approach is that one degree of freedom is left. In other words, there is no pre-established manner to control the torsion of the cylinders and this might result in unaesthetic twists.

The Parallel Transport Frame[34] is a very simple way to solve twist problems on curves. Let us define the T vector as the Tangent vector to a certain point of a curve (in our case, the tangent to a certain node n_i can be computed by normalizing $n_{i+1} - n_{i+1}$) and the vector N as the normal of this point. Note that, there is an unique tangent vector for a given point in a continuous curve, but there are infinite normal vectors.

Imagine that we are trying to approximate a generalized cylinder by a 6 faced-tube; we can generate one the bases of the cylinders (hexagons in this case) using only the vector T and N. To exemplify this, let p be a point of the curve, rotating the vector N around the vector T, 60 degrees at a time, we can generate the points $p' = p + N * radius$ which correspond to the vertices of the hexagon. Therefore, all we need to is to define the N vectors in a coherent way.

The solution that the Parallel Transport Frame proposes is to start with a randomly chosen N_0, and when advancing through the points of the curve (in our case, the nodes of the tree) compute the rotation between T_i and T_{i-1} and apply this same rotation to the vector N_i to obtain the vector N_{i+1}. In Figure 6.1 we can see a graphical example of a Parallel Transport Fram applied over the skeleton of a tree.

While we are computing the model geometry, we can also compute the texture coordinates

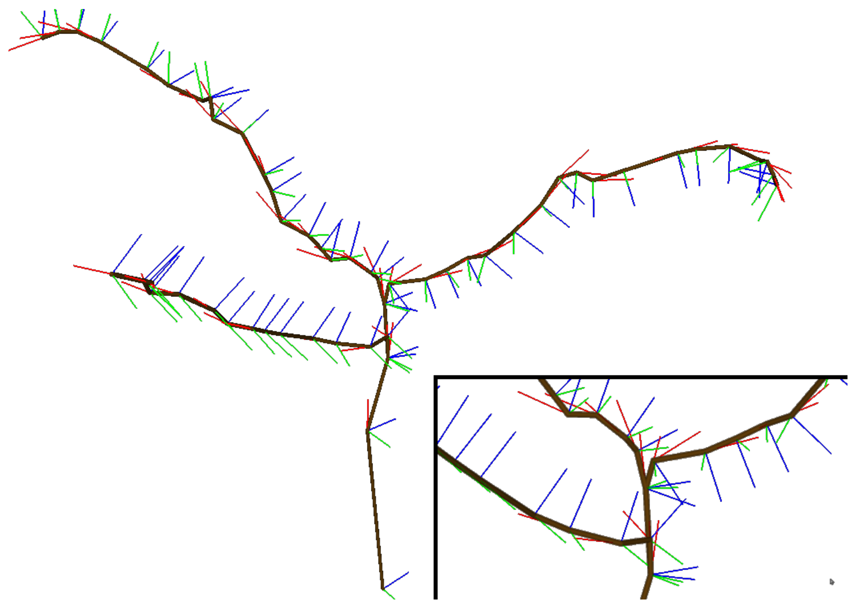

Figure 6.1: Paralle Transport Frame for the skeleton of a tree. Red lines represent tangents vectors whereas blue lines are normal vectors and green lines are the cross-product between tangents and normals.

for the model. Texture coordinate map a vertex of an object into a texel in a texture, the s coordinate is the horizontal coordinate for the texture, whereas the t coordinate is the vertical one. A quick intuition to assign texture coordinates to each vertex is that the horizontal or s coordinate should increment along the rotation around a branch, whereas the vertical or t coordinate should increment along the length of the branch.

Once we have computed the geometry and the texture coordinates for a model, the only task that is left is to save it into a file. For this, the .obj[1] format has been chosen. The advantages of using a wide spread format instead of a custom made one is that our models could be used in other applications, such as Blender.

Following, we are going to summarize the parameters that we have defined as input for our system, and their functions:

- **random_seed:** Parameter that sets the random seed for the execution of the program. It is important to control this parameter if one wants to reproduce the process of generating a specific tree.
- **tree_or_shrub:** Should take the value 1 if the user wishes to generate a tree, or 0 if the user wishes to generate a shrub. The difference lies in either generating an initial trunk or not.
- **initial_trunk_height:** If a tree is being generated, this sets the height for the initial

[1]A specification of the .obj format can be found in http://www.martinreddy.net/gfx/3d/OBJ.spec

trunk.

- **crown_height**, **crown_distance1** and **crown_distance2** define the shape of the cubic revolution for the crown envelope.
- **crown_height_displacement:** Parameter that displaces the height of the crown envelope.
- **attraction_points:** Number of attraction points to be generated.
- **min_attraction_points_distance:** Distance that must be kept between any pair of attraction points.
- **max_branch_gen_iterations:** Parameter that determines the maximum number of iterations for the step 6 of the Space Colonization algorithm.
- **new_node_distance:** Distance used in the step 4 of the Space Colonization algorithm, to generate new nodes.
- **node_kill_distance:** Distance used in the step 5 of the Space Colonization algorithm, to detect which attraction points must be erased.
- **max_attraction_distance** Distance used in the step 4 of the Space Colonization algorithm, to detect which attraction points influence a certain node.
- **min_attraction_vector_size:** Parameter used to prune too small branches, during the step 5 of the Space Colonization algorithm.
- **branch_tip_radius:** Initial radius for the tip branches of the tree skeleton.
- **branch_radius_exponent:** Parameter used in the step 10 of the Space Colonization algorithm, to compute the radius of a branch resulting of the merge of two others.
- **simplifying_by_area:** Should take the value 1 to apply the simplification by area process, whereas it should take the value 0 to apply the simplification by angle process.
- **simplifying_angle:** Threshold value for the simplification by angle process.
- **simplifying_area:** Threshold value for the simplification by area process.
- **max_simplifying_iterations:** The simplification process might be applied more than one time over the whole tree skeleton, since simplified branch might be further simplified. This parameter indicates how many iterations this process should carry out..
- **reducing_bifurcation_to_half** Should take the value 1 if the user desires to halve the branching angle, whereas it should take 0 if the user wants to reduce the branching angles over a certain threshold.
- **max_bifurcation_angle:** Threshold that indicates which branching angles should be reduced.

- **cylinder_edges:** Number of faces for the cylinder approximations.

Finally, mention that we will be using the library OpenGl Mathematics[25] because it facilitates dealing with vectors and matrices and operating with them. Therefore, parts of the algorithm, like the Parallel Transport frame, become easier to program.

6.3 Results

To begin with, we are presenting the class diagram of the application in Figure 6.2, only showing the attributes and relations between each class, because the aim is to give a general idea on how the code has been structured. For the windows management we have relied upon the library Freeglut[2]. The library GLM[25] is used to ease some operations needed to compute the parallel transport frame. The class Application is the one in charge to manage the callbacks generated by the window and also manage the interaction between the classes Statistics, Tree and TreeGenerator.

The class TreeGenerator encapsulates all the operations related to the Space Colonization algorithm, whereas the class Tree is a mere container for the generated branching structure. The class SCParameters manages parameter reading and storage, the class Statistics stores different statistics computed during the model generating process and, finally, the class VBO manages the rendering of the generated branching structures, so that the user can check the form of the generated plant before saving it into a file.

In Figure 6.3 we present an example of crown envelope filled with attraction points and the tree skeleton resulting from applying the Space Colonization algorithm.

To analyze the gain of using ANN searching structures against brute force, we have chosen 5 different sets of parameters (same as the ones used to generate the tree models that will be shown at the end of the section) and we have computed the average time over 10 executions, for the steps 4 and 5 of the Space Colonization algorithm. The results can be seen in Tables 6.1 and 6.2, where each row represent the average for one parameter setting.

Even if a statistical test should be conducted to ascertain with precision that there is an improvement, taking a glance at these numbers this fact appears obvious. However, the reader should note that the improvement is bound to the parameters *max_attraction_distance* and *node_kill_distance*.

For step 4, in the worst case, if the *max_attraction_distance* were very large making all attraction points fall within this distance, it is quite possible that doing the search for neighbors using the *kd tree* would become more expensive than the brute force search. The same would apply for step 5 and the *node_kill_distance*.

Simplifying the tree geometry has proven to incredibly reduce the overall model geometry

[2]http://freeglut.sourceforge.net/

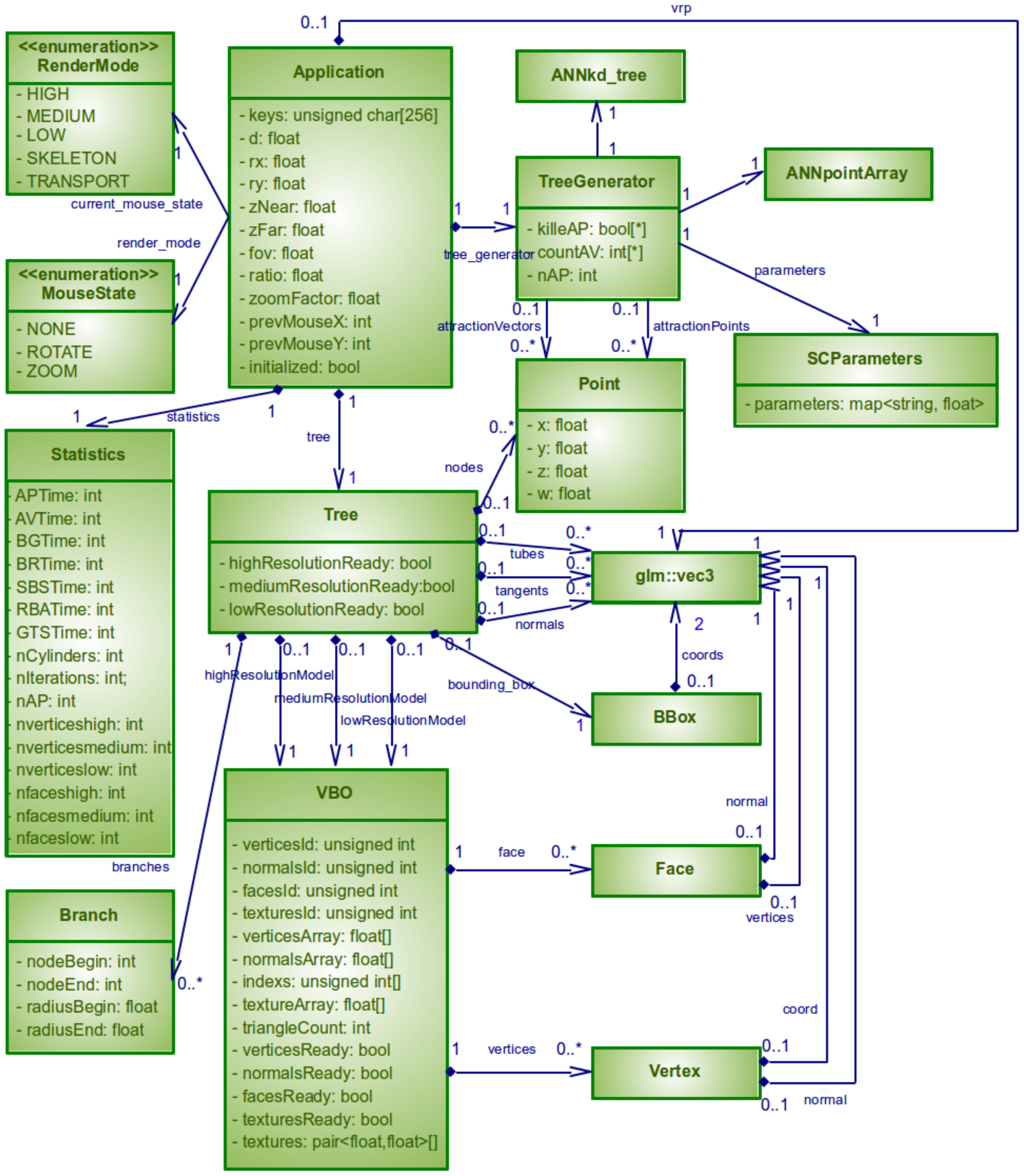

Figure 6.2: Class diagram for the application developed in phase 3.

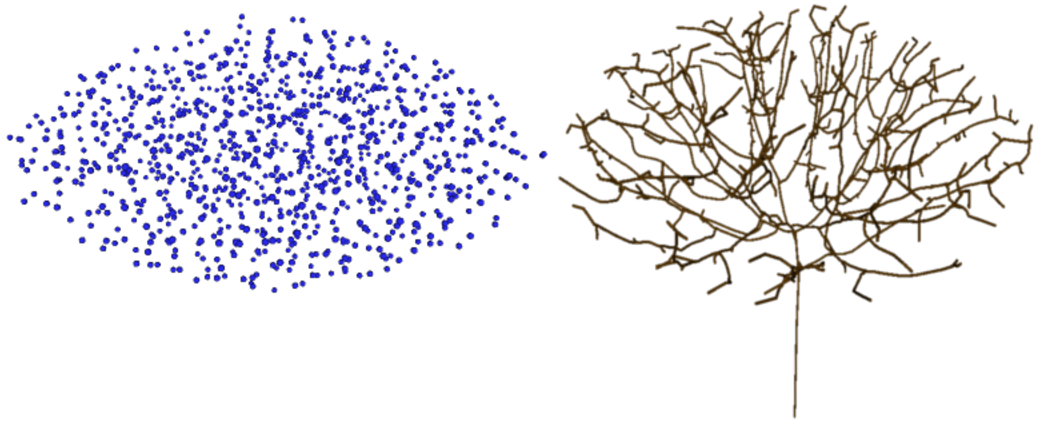
Figure 6.3: Left: Crown envelope generated with parameters crown_height = 30, crown_distance1 = 23 and crown_distance2 = 25. Right: Tree skeleton resulting from the Space Colonization Algorithm.

even if low threshold values are used. For the general case, the impact that this process has on the optical properties of the simplified models is not big, the resulting models usually appear very similar to the original ones. However, in some cases some parameter tuning might be needed to prevent some unaesthetic features to appear. We want to illustrate the results of this process in Figure 6.4, where the skeleton and the final geometry of a tree are shown, before and after applying the simplification process. In table 6.3 we quantify the geometry of the original model and the simplified one, in terms of vertices and faces.

As can also be seen in Table 6.3, the reduction of geometry produced by switching the level of detail is also quite notable. In this case, the visual effect of pruning tip branches is larger, as can be seen in Figure 6.5. However, one should take into account that lower-resolution models will be rendered far away from the observer, where those differences are less noticeable. Also, all the trees will be covered in leaves in the final application, so the visual effect of branches is only important in a close range.

Finally, we want to show the result of the geometry generation process, using the Parallel Transport Frame and generalized cylinders, the result of generating texture coordinates and the result of saving the model into an .obj file. For this, we refer to Figure 6.6 where the tree that has been used to illustrate the results of this phase, is shown with its final geometry textured, in the prototype for the final application.

Using the described process, we have generated 5 tree models and 3 shrub models that will be used in the final application. The files related to this model can be found together with the source code of the project, however, in Appendix C we are showing them together with their characteristics and the parameters that were used to generate them.

These models have been generated taking into account their resulting geometry. More complex models mean more computation time spent rendering them and, because of this, we have constrained their number of vertices and faces. Through parameter tuning, we have

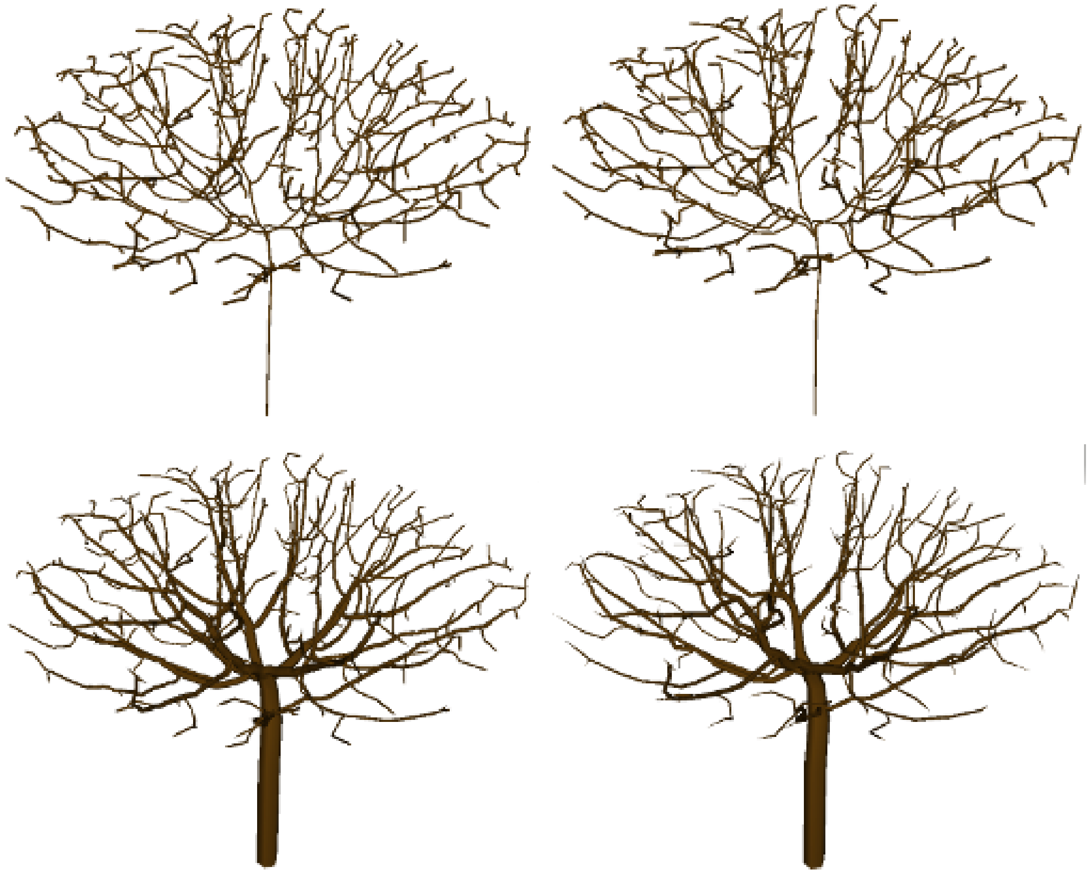

Figure 6.4: Example for the skeleton simplification process. Left: Original skeleton and final geometry for the tree generated using the cloud of points in figure 6.3 as an input for the Space Colonization Algorithm. Right: Simplified tree skeleton and simplified final geometry resulting from applying the area simplification process, with threshold 0.4, to the structures on the left.

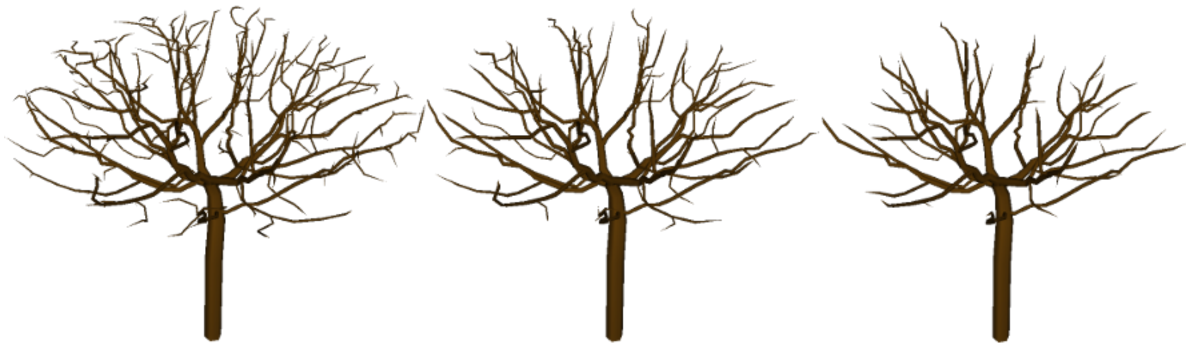

Figure 6.5: Three different resolutions for the final geometry of the simplified tree model in Figure6.4.

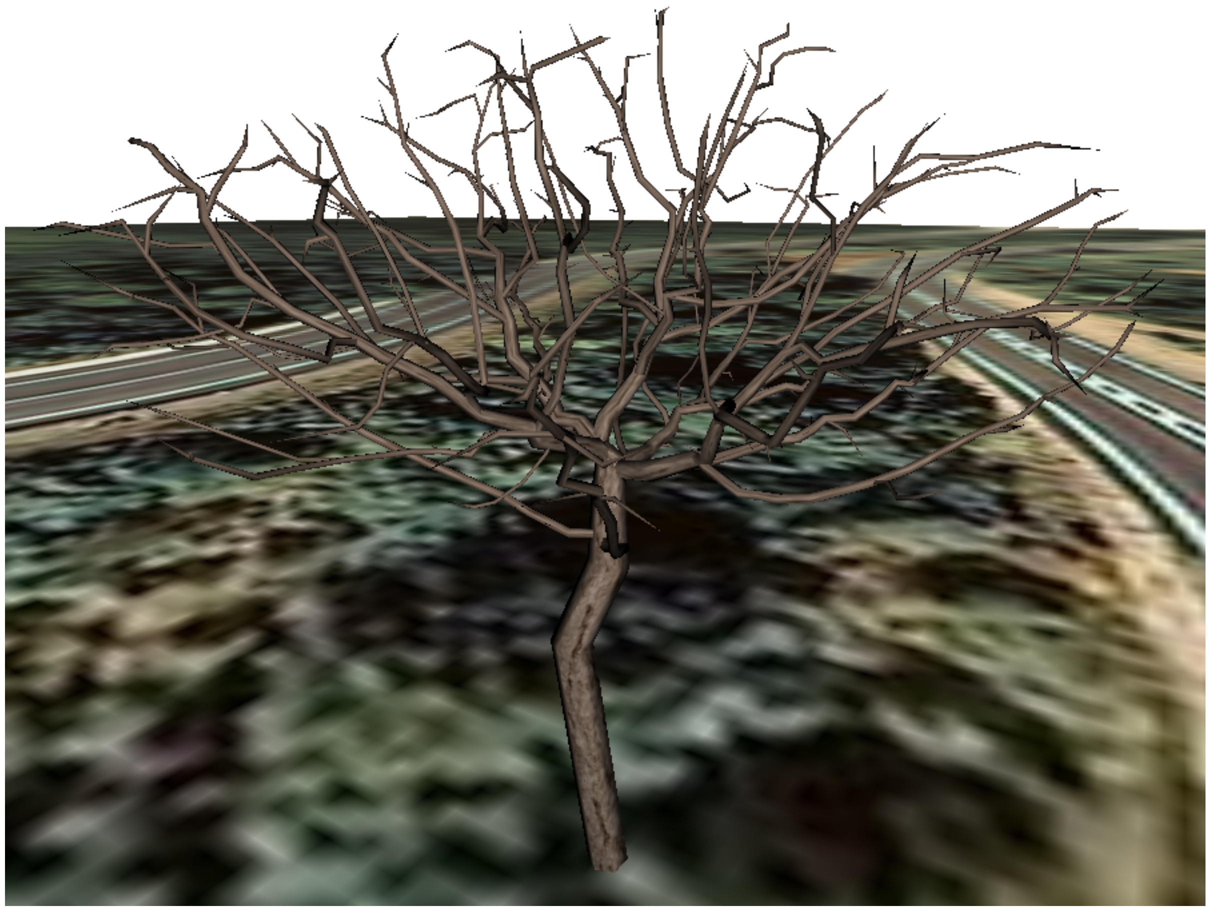

Figure 6.6: Textured tree generated with the software piece programmed in this phase.

Brute Force	473.2 ms	345.5 ms	434.2 ms	474.1 ms	4245.6 ms
ANN	292.9 ms	120.9 ms	234.3 ms	123.7 ms	2341.5 ms
crown_height	30	30	50	50	60
crown_distance1	25	23	15	30	30
crown_distance2	20	25	30	24	30
attraction_point	2000	1000	500	1700	4000
min_attraction_points_distance	2.3	2.3	2.3	4.0	2.0
max_branch_gen_iterations	150	150	150	150	150
new_node_distance	0.5	0.5	0.5	0.5	0.5
node_kill_distance	2.3	2.5	1.0	2.5	3.7
max_attraction_distance	12	8	11.5	8.0	20
min_attraction_vector_size	0.5	0.5	0.5	0.5	0.5

Table 6.1: Average time over 10 executions for the step 4 of the Space Colonization algorithm. Each column corresponds to executions using a different parameter setting. Parameters that might influence these times are also shown.

reduced the maximum resolution models for trees, to around 6000 vertices and 9000 faces and the maximum resolution models for shrubs, to around 4000 vertcies and 6000 faces. Nevertheless, this has been done trying not to lose their aesthetic properties.

6.4 Conclusions

We shall start this section summarizing the advantages and disadvantages that our approach holds in comparison with other methodologies. First of all, one should bear in mind that, for our case study, we detected around 11000 trees and 10000 shrubs that must be rendered in the final application. As we have seen, each model generation process takes few hundred milliseconds to complete and, therefore, it would take few hours to generate a model for each detected plant. Also, the volume of data that would suppose having one model for each detected plant could be costly for a normal graphic card memory.

Using only 5 base models for trees and 3 base models for shrubs we both save in memory and in generation time. Also, elaborating the models separately from the main application holds various advantages; If we were to generate the models inside the main application, its start-up time would be larger. Another issue would be that sometimes the results of the Space Colonization algorithm are anaesthetics and parameter tuning is needed to refine the models, this can not be done is the models are generated in the main application.

But not everything are advantages, using a reduced set of base models means that we must render each model many times. If not done carefully, this could lead to unfeasible landscapes because repetitions might be detected by the human eye. In the following phase we will extensively explain how this matter has been approached, but the basic idea is to generate variety among the models by rotating and scaling them, and by painting them using

Brute Force	35.6 ms	25.8 ms	19.6 ms	22.6 ms	199.9 ms
ANN	10.1 ms	8.0 ms	6.2 ms	8.8 ms	35.8 ms
crown_height	30	30	50	50	60
crown_distance1	25	23	15	30	30
crown_distance2	20	25	30	24	30
attraction_point	2000	1000	500	1700	4000
min_attraction_points_distance	2.3	2.3	2.3	4.0	2.0
max_branch_gen_iterations	150	150	150	150	150
new_node_distance	0.5	0.5	0.5	0.5	0.5
node_kill_distance	2.3	2.5	1.0	2.5	3.7
max_attraction_distance	12	8	11.5	8.0	20
min_attraction_vector_size	0.5	0.5	0.5	0.5	0.5

Table 6.2: Average time over 10 executions for the step 5 of the Space Colonization algorithm. Each column corresponds to executions using a different parameter setting. Parameters that might influence these times are also shown.

	Original Model	Simplified Model
Faces for the high resolution version	30654	8682
Vertices for the high resolution version	16331	5345
Faces for the medium resolution version	25986	5292
Vertices for the medium resolution version	13589	2982
Faces for the low resolution version	23028	3744
Vertices for the low resolution version	11946	2080

Table 6.3: Summary for the geometry generated for the original and simplified trees on Figure6.4

different textures.

In general, we are quite satisfied with the result of this phase. We have been able to improve the existing code in a satisfactory way, reducing the computation time, needed for some steps, and adding more procedures to refine the generated models. Moreover, we have successfully elaborate the base set of 3D models we needed for the final application and, through parameter tuning, we have been able to make them look aesthetic while keeping their geometry low (for information on these models can be found in Appendix C). Note that, as the models are saved in a standard format, they can be used in a wide range of other applications.

Finally, all that is left is giving some guides on what could be done as future work. There is one step of the Space Colonization algorithm that has not been implemented, namely, curve subdivision. This step aims to transform the axes of the tree skeleton in curves to smooth the limbs and, although it is not strictly necessary, it could be implemented on a future work. Also, we only offer one approach to generate the crown envelope, which means that we are restricted to a reduced number of shapes. Work could be done on improving the definition of the crown envelope, either adding new geometrical forms or using other more complex approaches.

7 Phase 4: Development of the Final Application

7.1 Introduction to the problem

During phase 1 and phase 2 we have obtained information about where different kinds of plants must be placed, to generate a feasible match of the vegetation found in an orthophoto. During Phase 3 we have seen how to build a base set of models for the branching structures needed to represent trees and shrubs. In this phase we will describe how the results of the previous phases have been integrated to finally produce the desired 3D landscape.

Even if, at first sight, this phase may seem trivial, there are few important issues to consider. In the first place, it has been described how to render the branching structures of major plants, but nothing has been stated about their foliage. Without any branch a tree might prove not realistic, but without foliage it might not even appear natural. For the leaves, we must consider how to generate their geometry and how to place them around the branches.

So far, major plants have been considered but nothing has been mentioned about the grass. This kind of vegetation does not need any complex geometry, however a simple textured quad might prove unfeasible. Grass, along with leaves and trunks, also bear another important issue which is how to determinate their colour. Nothing has been stated up until now, but the colour of the rendered vegetation should match the colour of the orthophoto.

Finally, the last matter to be addressed is efficiency. Bearing in mind that this project only holds research purposes, real-time rendering is not mandatory. Previous phases, like feature extraction and classification, have not been optimized, however some easy optimizations can be done to the visualizing process. If the navigation through the scene can be done in a reasonable time, the user interaction with the resulting 3D landscape could be widely improved.

7.2 Approach and Methodology

For beginnings, if the reader is not familiar with the the OpenGl pipeline, it is recommended to read the brief description done in Appendix D, since very specific concepts of the Computer Graphics field will be used during this phase. For further reading on OpenGl we refer the reader to the book The Official Guide to Learning OpenGL[35] and for further reading on shaders we refer the reader to the book OpenGL Shading Language[36].

In this section, we will begin talking about vegetation representation, followed by explaining how we are aiming to color the plants and foliage, and we will end talking about efficiency.

Starting with trees and shrubs, recall that we have information about where to place each of them, their orientation and their major and minor axis (this is the information obtained during phases 1 and 2). Therefore, to represent each of them, we first select randomly one of the base models (generated during phase 3), then we rotated it according to the plant's orientation and scale it proportionally to the major axis. Finally, the only thing left to do is to translate them to their proper place in the scene.

To solve the problem of representing grass and foliage, we will use oriented, textured polygons (usually quads) called *billboards*. The orientation of a billboard is defined by two vectors: the normal vector and the axis vector. The normal vector of a billboard is the vector that points away from the billboard plane, usually towards the viewpoint (camera), and its axis vector is a vector that lies in the billboard plane, and defines the billboard relative rotations. We want the normal vector to always look at the camera whereas we want the axis vector not to change, since we do not desire that the textured quads rotate over themselves.

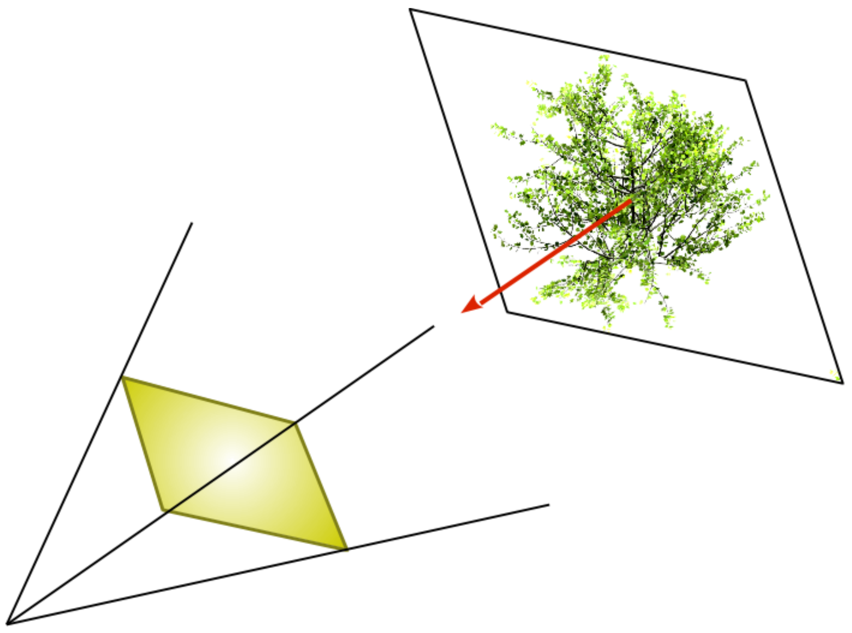

Figure 7.1: Exemplifying the billboard system.

In every place that, during phase 2, it has been determined that grass must be rendered, we will place a billboard that will constantly be rotated, to make it always look at the camera. However, the rotation will only be done around the y axis, making the polygon always remain orthogonal to the ground. It has also decided to render grass under the trees, using the same system, to simulate the undergrowth.

To place the leaves around the trees and shrubs crowns, we have decided to generate 100 random points, using, in each case, the same envelope that was used to generate the attraction points, that resulted in each model. Later, a billboard is placed centered at each of these points. The billboard size is set to around one third of the plant diameter and, unlike the grass, the billboards for the leaves are always directly looking towards the camera.

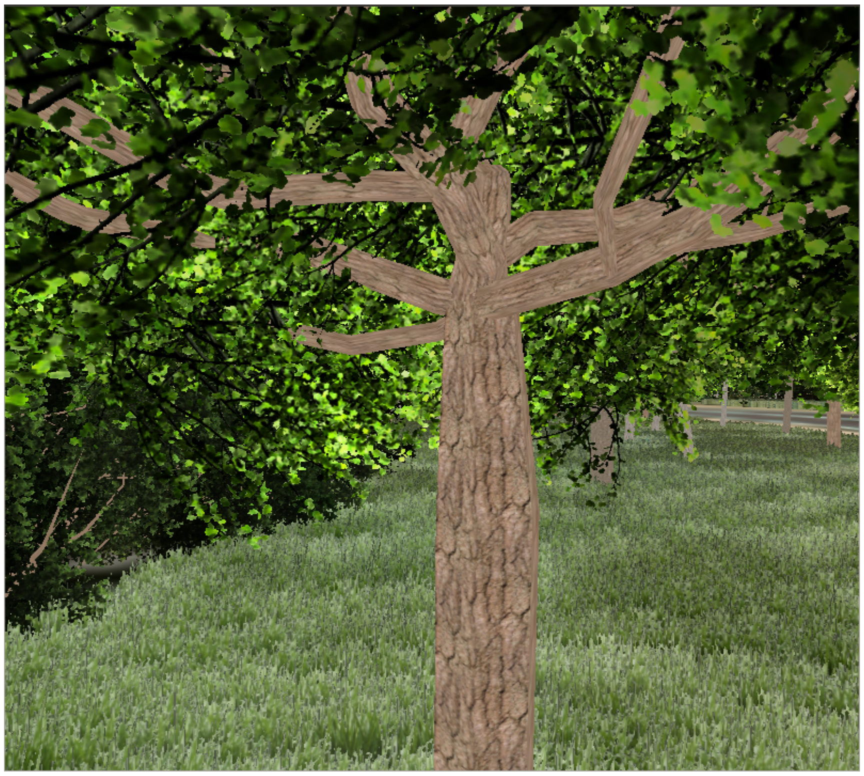

Figure 7.2: A tree trunk textured using a tileable texture.

Even if it might seem that, the strategy of using billboards (2D polygons) to represent something that should have volume, should produce implausible results, in the result section it will be shown that, using a big number of them, the generated landscapes are perfectly feasible.

So far, we have talked about the scene geometry but nothing has been said about how to paint each model. The major issue that we find in this process is that it is desired that the color of the vegetation match es the color of the orthophoto. The coloring of all the scene elements will be done using different fragment shaders for each purpose.

Recall that, during phase 3, we have generated texture coordinates for the 3D models of trees and shrubs. These coordinates are the base for texturing the trunks of these models. The basic texturing method would be repeating a tileable texture of a real trunk along the whole model, as can be seen in Figure 7.2. However the repetitive patterns are easily spotted by the human eye and, because of this, the result becomes unfeasible. An easy way to fix this is to add low and high frequency noise using other textures.

The other set of polygons that must be textured are the billboards. For all of them, either the ones that represent grass or the ones that represent leaves, we will use a basic texture modulated with the orthphoto color. This way, even if we use the same texture for all the leaves or for the grass, each polygon will have its own color and will be different to the rest.

Considering all the geometry that is going to be rendered (an approximation of this is given in the results section, for our case study) it is advisable to use some kind of strategy to reduce the rendering time.

The first thing to be done is to use *vertex buffer objects*, which are a very efficient way to render static geometry (vertices that always keep their relatives positions in the model). One vertex buffer object will be used for each of the base models and for each of their resolutions. Also, all the billboards with similar properties (all the leaves, all the grass billboards, etc) will be rendered using a single vertex buffer object because rendering each quad separately would take too much effort. However, a vertex shader will have to be used to correctly orientate the billboards, this can be done if its center is inputted into the shader as an attribute.

Next, we will try to speed-up the application reducing the geometry rendered at each frame. Some steps have already been taken in this direction. Recall that, during phase 3, lower resolution versions of each vegetation model were generated and now is the moment to use them. Therefore, our application must be able to decided which level of detail must be rendered, depending on the distance between the place where it should be placed and the camera. The bigger distance the lower resolution.

However the way in which we formulate this approach will, at most, halve the geometry (taking into account the data we have on the different resolution levels of the base set of models). Because of this, we are going to introduce the View Frustum Culling technique, which is also useful to speed-up the rendering.

First of all, let us define what is the view frustum. The view frustum is the volume that contains everything that is potentially visible on the screen. The camera settings (the Model View matrix and the Projection matrix) gives form to this volume that, when using a perspective projection, take the shape of a pyramid truncated by the near and far planes, as can be seen in Figure 7.3.

During the phase previous to the *Rasterization*, in the OpenGL pipeline, all the primitives outside this frustum are discarded to speed-up further processes. However, this means that all previous processes are applied over these discarded primitives. The goal of View Frustum Culling is to identify and discard these primitives before the render process starts. Furthermore, we will not be applying this test on primitives but on trees, which means that less operations will be performed. Note that, as all leaves and all grasses are rendered as a whole, they can not be partially discarded by this test.

The first thing that must be done for this procedure is to extract the six planes (near, far, bottom, top, right and left) that define the frustum. We will extract them in *world space*, adopting a simple geometric approach. The planes must be defined in a way that the normals point towards the inside of the view frustum, because testing if an object is inside the frustum will be done computing the signed distance between itself and each plane. If the signed distance is positive, this means that the object is on the side the normal is pointing and, therefore, if all six distance are positive, then the object is inside the view frustum and must be rendered.

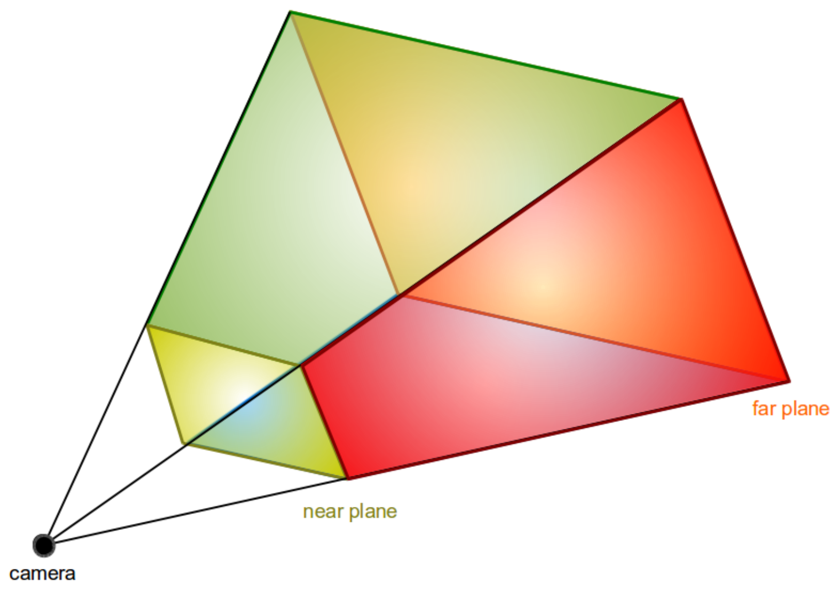

Figure 7.3: Pyramid of vision resulting from the use of a perspective projection.

For beginning, we must compute the six corners of the view frustum, because these can be used to define the six planes. For this, we need to know the following parameters:

- The position of the camera p in *world space*.
- The direction of the view ray d in *world space*.
- The *up* vector in *world space*. These three parameters can be derived from the transformations made to set the Model View matrix.
- The *right* vector, which can be computed as the cross-product between the *up* and d vector. *right*, *up* and d must be unitary vectors.
- The z_near and z_far distances, which are parameters of the Projection Perspective.
- The height and width of the near and far planes. To calculate this we can use the formula $height = 2*tan(field_of_view/2)*distance$ and $width = height*aspect_relation$, where $field_of_view$ and $aspect_relation$ are parameters for the Perspective Projection and $distance$ should take the value of either z_near or z_far.

With this information, the corner points can be computed as follows (the process is illustrated in Figure 7.4):

```
far_center = p + d*z_far;
half_up_far = up * height_far/2;
half_right_far = right * width_far/2;
```

```
ftr = far_center + half_up_far + half_right_far;
ftl = far_center + half_up_far - half_right_far;
fbr = far_center - half_up_far + half_right_far;
fbl = far_center - half_up_far - half_right_far;

near_center = p + d*z_near;
half_up_near = height_near/2 * up;
half_right_near = width_near/2 * right;

ntr = near_center + half_up_near + half_right_near;
ntl = near_center + half_up_near - half_right_near;
nbr = near_center -half_up_near + half_right_near;
nbl = near_center - half_up_near - half_right_near;
```

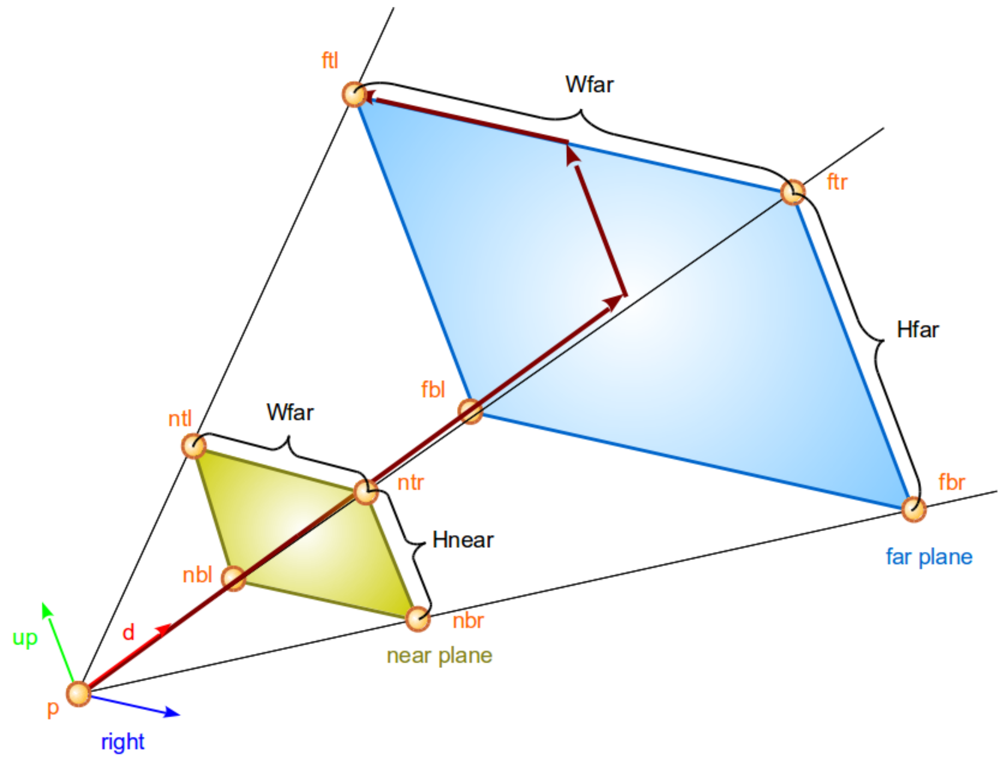

Figure 7.4: Different parameters needed for the View Frustum Culling procedure.

Once to we have the 8 corners of the frustum, we need to calculate the equations $Ax + By + Cz + D = 0$. If p_1, p_2 and p_3 are points of one of these planes, its equation can computed with the following procedure:

```
v = $p_2$ - $p_1$;
u = $p_3$ - $p_1$;
```

```
n = normalize(cross(u,v))

A = n.x
B = n.y
C = n.z
D = -($p_1.x$ * A + $p_1.y$ * B + $p_1.z$ * C)
```

Finally, we must check if our trees or shrubs are partially or totally inside the frustum. First we will compute the signed distance between the plant center c and each of the planes. For this we can use the formula $distance = c.x * p.A + c.y * p.B + c.z * p.C + p.D$, for a given plane p. However, a part of the tree might be inside the view frustum even if the center is not, therefore we must take its radius in consideration. This yields the test $distance + radius < 0$, that if proves true for all the planes, then the plant is inside the view frustum.

7.3 Results

To begin with, we are presenting the class diagram of the application in Figure 7.5, only showing the attributes and relations between each class, because the aim is to give a general idea on how the code has been structured. As in the previous phase, we are using the library GLM[25] to ease some computations, however the class glm::vec3 has been erased to simplify the diagram. For the windows management we have relied upon the library Freeglut[3].

The class Visualizer is the one in charge to manage the callbacks generated by the window, such as pressed key buttons, and relay them upon the Camera and the Frustum. Also, it is the responsible of initializing the Scene and the OpenGl context (for instance, activating the *z-test*, the *face culling*, etc).

The class Camera implements the flight camera. It keeps information about the position and rotation of the camera and updates this when receives new information from the class Visualizer. Upon an update, it resets the OpenGL Model View matrix with the appropriate matrix transformations.

The class Frustum implements the view frustum. It keeps information about the field of view, measured in degrees, and about the aspect relation of the window. Upon an update received from the class Visualizer, it resets the OpenGl Projection Matrix with the appropriate matrix transformation. Also, this class is the one that implements the Frustum View Culling test.

The class Scene is the responsible for managing all the application geometry. It takes upon the responsibility of calling each element initialization function and some initializations of itself, such as reading the trees information, the shrub information and the grass locations

[3]http://freeglut.sourceforge.net/

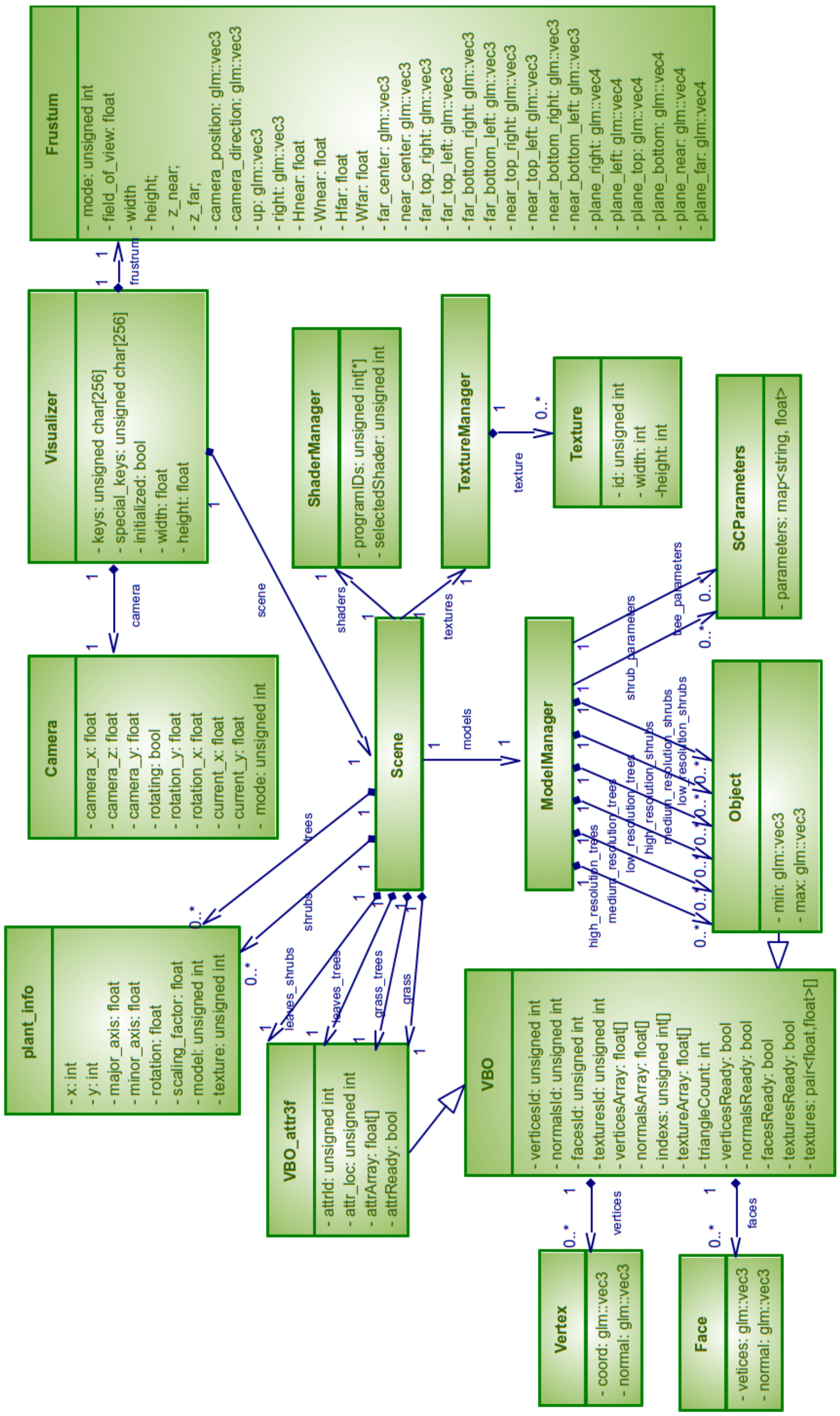

Figure 7.5: Class diagram for the application developed in phase 4.

from files. Finally, it is also responsible for calling each element rendering function when the whole scene need to be rendered.

The class VBO implements Vertex Buffer Objects with position, normal, and texture attributes. VBO_attr3f adds to this implementation an attribute formed by three floats, whereas Object adds a function to read a model from a .obj file. Vertex Buffer Objects are an efficient way to render geometry, for more information check Appendix E.

ShaderManager is the class responsible for reading, compiling and linking the shaders that will be used in the application. TextureManager and ModelManager are classes that manage textures and vegetation models, respectively. Finally, the class Texture reads an image from a file, using the class QImage from the library Qt, sets its properties and generates a texture from it.

Following we will present the results of texturing the trunks of trees ans shrubs, using the explained process. For each trunk, one of the texture in Figure7.6 is randomly selected as base texture. Higher and lower frequency noise is added using the other two. The results of this process can be observed in Figure 7.7, with this method trunks look much more realistic than using the tileable texture alone.

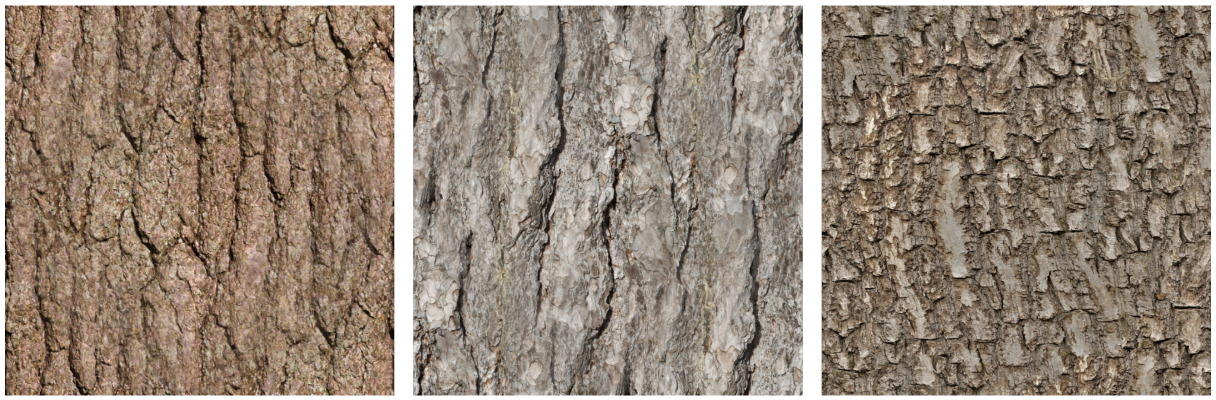
Figure 7.6: Textures used for coloring the trunks of trees and shrubs.

Next, we are presenting the results of representing grass using billboards. As stated before, there are two kinds of grass, the one that is rendered under trees to simulate undergrowth and the one that is rendered because it was detected on the orthophoto. The textures used to paint them are shown in Figure 7.8. The color of this textures was modulated with the color of the orthphoto to increase the realism and to match the orthophoto's view. Some grass fields can be seen in the Figure 7.9.

Continuing with the billboards, we are presenting the results of using this technique to represent the leaves of trees and shrubs. Same as for the grass ones, the color of a texture is modulated with the color of the orthophoto to make them look more realistic. The textures used for the shrubs and trees leaves can be seen in Figure 7.10 100 of these are rendered around the crown of each tree or shrub, and are oriented towards the observer through a

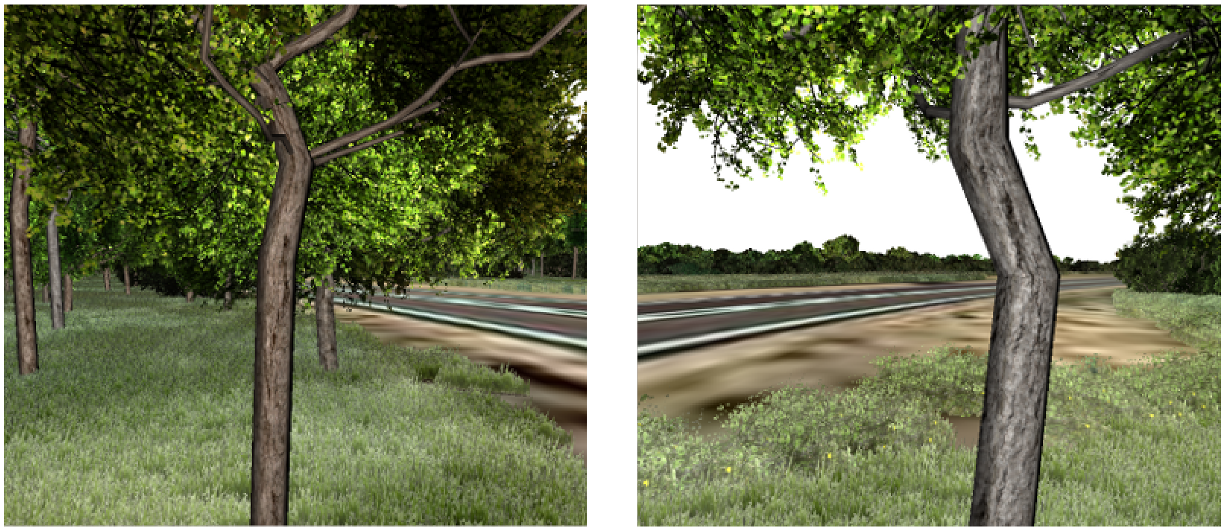
Figure 7.7: Textured trunks of trees in the final application.

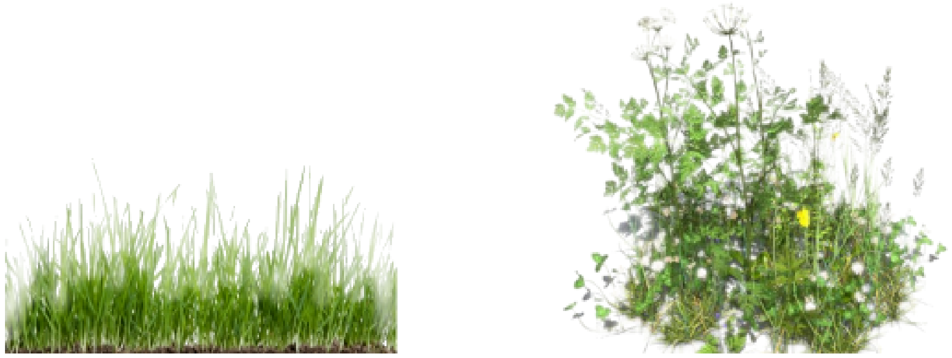
Figure 7.8: Left: texture used for the grass around trees. Right: texture used for the grass detected through the classification process.

vertex shader, producing the effect seen in figures 7.11. Even if each individual billboard is a mere 2D polygon, the combination of a big quantity of them generates a volume optical effect.

Finally, we are presenting the results of aiming to speed-up the rendering process. First, we analyze the effects of using levels of details. As can be seen in the images that we have shown up until now, most of the branching structure of trees and shrubs is occluded by their foliage. Because of this, the visual impact of using different resolution models is smoothed.

Even if the branches were not covered by leaves, all the level of detail are structurally similar, and only the small branches are pruned when reducing the resolution. In consequence, having low levels of details being rendered far away from the observer makes the change between levels almost impossible to notice. We can conclude that, visually, there is no difference between using different resolution models or not.

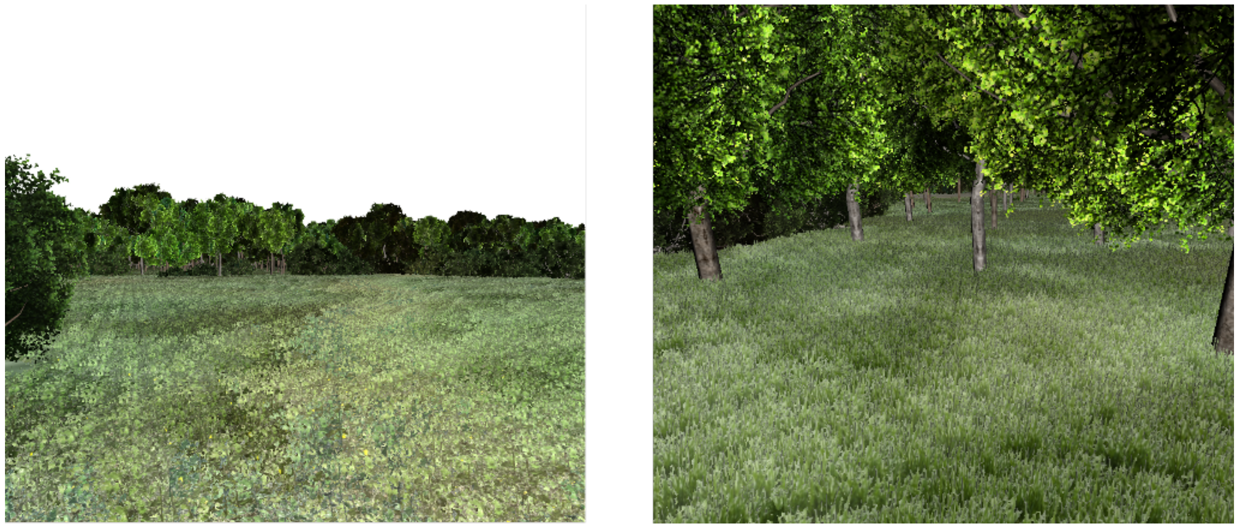

Figure 7.9: Some grass fields from the final application

However, using levels of detail have a great repercussion on the geometry being rendered. On Table 7.1 we show an approximation on the geometry, in terms of vertices and faces, that is present on the scene of our case study, without using different resolution models. Based on the information shown in Appendix C, we make the assumption that the geometry of the average high resolution tree is of 6000 vertices and 9000 faces, and the geometry of the average the high resolution shrub is to be of 3900 vetices and 5700 faces.

On the other hand, using levels of resolution, we can approximate the average geometry of a tree or a shrub with the geometry for their medium resolution models (3000 vertices and 5200 faces for the trees and 1600 vertices and 2750 faces for the shrubs). In Table 7.2, we approximate the geometry of our case study using these numbers. The difference stands clear, the geometry is halved using levels of detail which, in an ideal case, would mean rendering twice as faster.

	Number of Objects	Vertices per Object	Faces per Object	Total Vertices	Total Faces
Trees	10502	6000	9000	63012000	94518000
Shrubs	11087	3900	5700	43239300	63195900
Grass	2392955	4	2	9571820	4785910
Leaves	2158900	4	2	8635600	4317800
Total				**124458720**	**166817610**

Table 7.1: Geometry rendered at each frame, in our case study scene.

Finally, to assess in a qualitative way the efficacy of the view frustum culling, we have realized 1000 renders in random positions around the scene, computing the average time spent per frame. Without the view frustum culling, this accounts for 220.089 ms per frame (ranging from 93 ms per frame to 758 ms per frame), whereas with it, the average time per frame becomes 184.55 ms (ranging from 44 ms to 498 ms). The tests were carried out on a

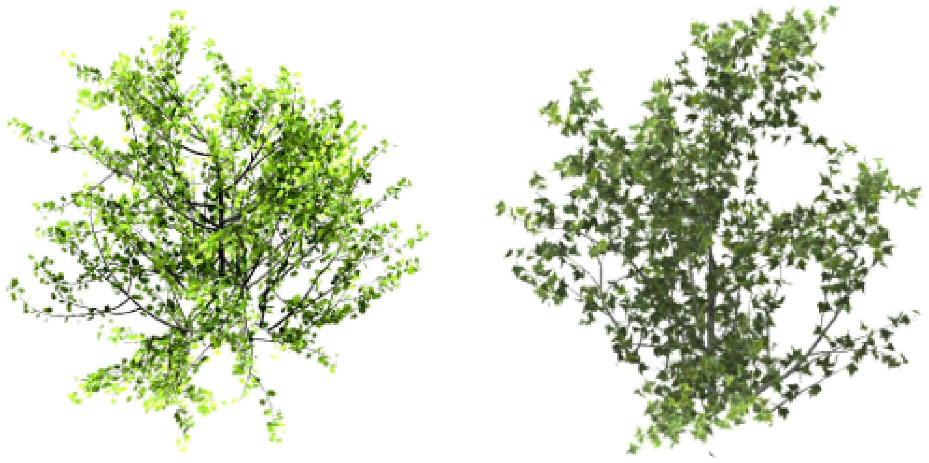

Figure 7.10: Left: texture used for the tree leaves. Right: texture used for the shrub leaves.

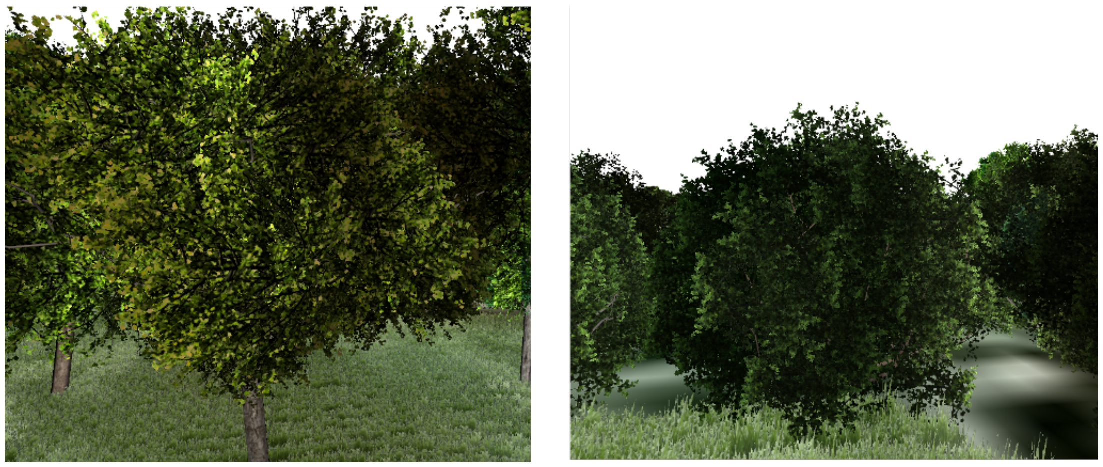

Figure 7.11: Left: A tree crown generated using 100 billboards. Right: A shrub crown generated using 100 billboards. Both image are from the final application.

computer with the features shown in Table 7.3.

7.4 Conclusions and Future work

Even if the end we have been able to successfully integrate all the results from the other phases into the final application, there were many features that could not be implemented due to the lack of time. For instance, using a wider and better set of textures for trunks, leaves and grass would cause a wider diversity of plants, improving the realism of the landscape.

Also, we possess a 3D model representing the relief of the terrain in that area, which is wider than the part we analyzed. Because of this, it should to be cut to fit in our application but there was no time to determine which part precisely corresponded with our orthophoto.

	Number of Objects	Vertices per Object	Faces per Object	Total Vertices	Total Faces
Trees	10502	3000	5200	31506000	54610400
Shrubs	11087	1600	2750	17739200	30489250
Grass	2392955	4	2	9571820	4785910
Leaves	2158900	4	2	8635600	4317800
Total				**67452620**	**94203360**

Table 7.2: Geometry rendered at each frame, in our case study scene, using levels of detail.

CPU	Intel(R) Core(TM) i5 CPU M 520 (2 x 2.40GHz)
RAM	4 GB
GPU	ATI Madison [Radeon HD 5000M Series]
Video card memory	256MB
Operating System	Ubuntu 12.04 (precise) 32-bit

Table 7.3: Geometry rendered at each frame, in our case study scene, using levels of detail.

A very different topic is the class architecture for the application. Even if the class hierarchy is quite clear and robust, there are some functionalities, such as the rendering of objects, which should be encapsulated in separated classes. More or less that same can be applied to the software piece developed during phase 3. Because of time constraints, this was not possible to perform and is left as future work.

Nevertheless, we have been able to implement one of the features that, in the beginning, was discarded which is real time rendering. Taking in average 184.55 ms per frame, this yields a frame rate of 5.4 fps. Even if it is a bit low for a completely satisfactory user experience, it is quite more than what we would expect when the project was devised. Probably, the worst issue that the rendering time has is that the frame rate is not stable, this is caused by how the frustum view culling works. Depending on where you are positioned in the scene, a bigger quantity of geometry might be contained inside the view frustum, causing the frame rate to fall.

Finally, to conclude this section, it must be said that we value very positively the result of this phase. Landscapes and plants appear real to the human eye, which is what we were aiming for since the very beginning.

8 Final Conclusions

This project was conceived as a very greedy one, probably too much. Because of this, we lacked time to polish some of the phases. Nevertheless, during the development of the project we have attained all the objectives that were set in the beginning.

Throughout the first phase, we have elaborated a classifier able to detect different kinds of vegetation with a more than an acceptable accuracy (around the 71%). Distinguishing between vegetation and the rest of classes is performed better than distinguishing between kinds of vegetation, nevertheless this is actually good because the visual impact of rendering vegetation where it should not be is expected to be much higher than the visual impact caused by mistaking the vegetation kind.

Throughout the second phase, the biggest achievement has been to successfully use morphological processing to remove the noise that appeared along the road, cause by an imperfect classification process. This way, we prevented the appearance of very grave visual effects, such as rendering vegetation in the middle of roads. Furthermore, partially removing this noise from vegetation classes produces the clouds of pixels to condense. This eases the process of grouping the pixels that belong to a single tree or shrub, because less unfeasible clusters are likely to appear. On a later step, unfeasible clusters, such as one formed by two pixel groups in different sides of the road, are also treated. With all this process we have been able to extract the information needed to render each plant.

Throughout the third phase, we have implemented a method that generates feasible 3D models of trees and shrubs, attaining the objective for this phase. Moreover, we also generate texture coordinates for them, which lets us color them in a very realistic way, we also save them in an standard format, enabling their use in other applications, and we also generate different levels of detail for each model.

Finally, in the fourth phase we have been able to integrate the information generated during the second phase and the models generated in the third phase, to reproduce the vegetation found in our case study image. Different shaders have been used to color the vegetation trunks and to reproduce the foliage, of trees and shrubs, and the grass, in a way that the final result looks feasible to human eye. Most important, as we present in the Figure 8.1, the cenital vision of the generated vegetation perfectly match the orthophoto's view, therefore we have attained another of our major objectives.

Along all the project, many guidelines have been given for future work. Among these, the most interesting ones are relate with reducing the computation time for the described algorithms. Specially, this is mandatory if a market application wants to be developed. We have discussed that the software pieces developed during the third and fourth phases are surprisingly fast, in comparison to what was expected in the beginning. Therefore, the efforts must be first focused on speeding-up the software pieces developed during first and second phases. Some approaches that have been pointed out, for this, are optimizing the number and quality of the descriptors, and using implementations on faster languages.

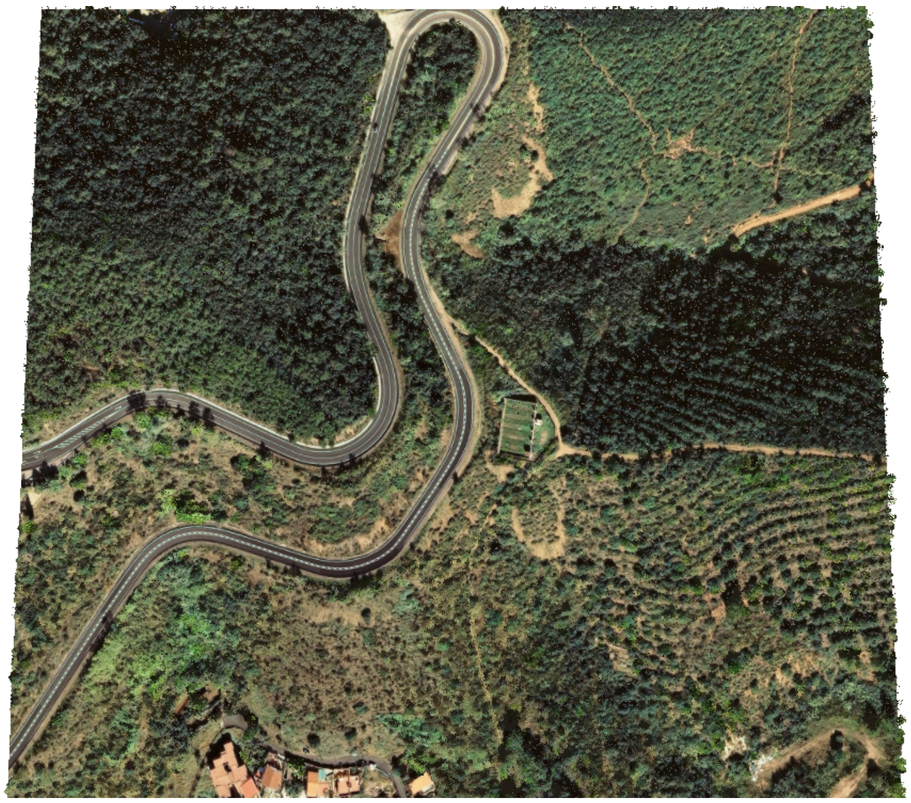

Figure 8.1: Cenital view of the rendered landscape in the final application.

A totally different approach to improve the performance of the first phases is related to changing the specification of the test cases. During phase 1 we have seen that bigger windows to compute descriptors, yield better results than small windows. The idea behind this is that makes this work is that bigger windows capture more context information. However another way to obtain these results could be, instead of using bigger windows, using lower resolution images. The reasoning behind this is that reducing the resolution of an image means capturing the information of a neighborhood, in a single pixel.

On another topic, we think that using another image, with similar characteristics to our case study, to assess the results of each phase would have enriched very much our study. However, because of the lack of time, this was not possible. This does not mean that our results are not solid, recall that, for instance, during the classification process even if the training and test sets came from the same picture, they were selected independently (therefore, the input data was partitioned in two independent sets, as it is usually done in any machine learning process). This yields that the resulting test error is a robust measure.

In summary, the results of this project fulfills all our goals satisfactorily. However, its elaboration became much more complex than we conceptualized in the first place, meaning that there was not enough time to perform some extensions. And, as in any other project, this study opens the door for much future work.

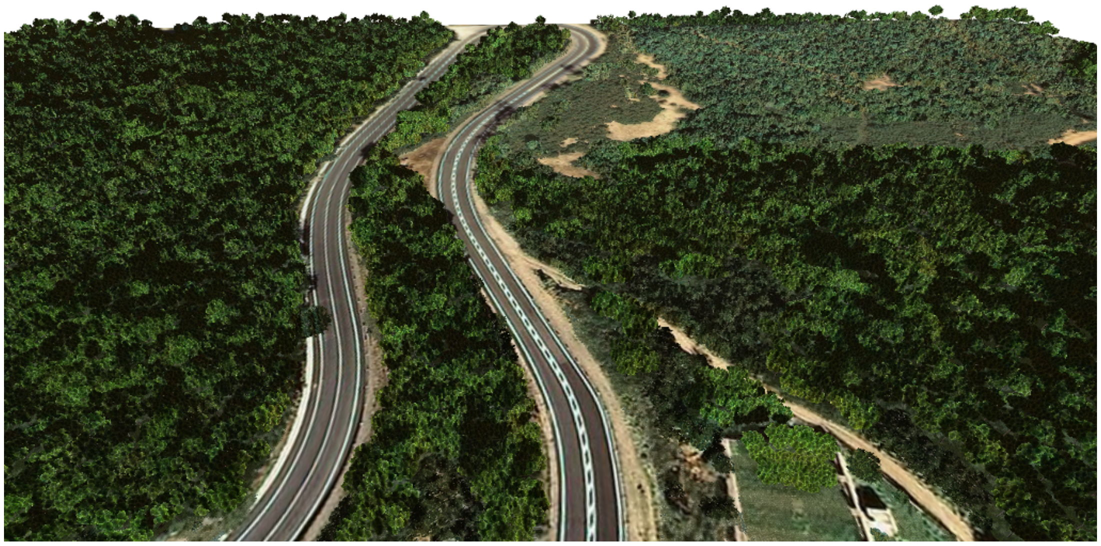

Figure 8.2: Landscape extracted from the final application.

To conclude, we are attaching some more images extracted from the final application, for the reader's enjoyment.

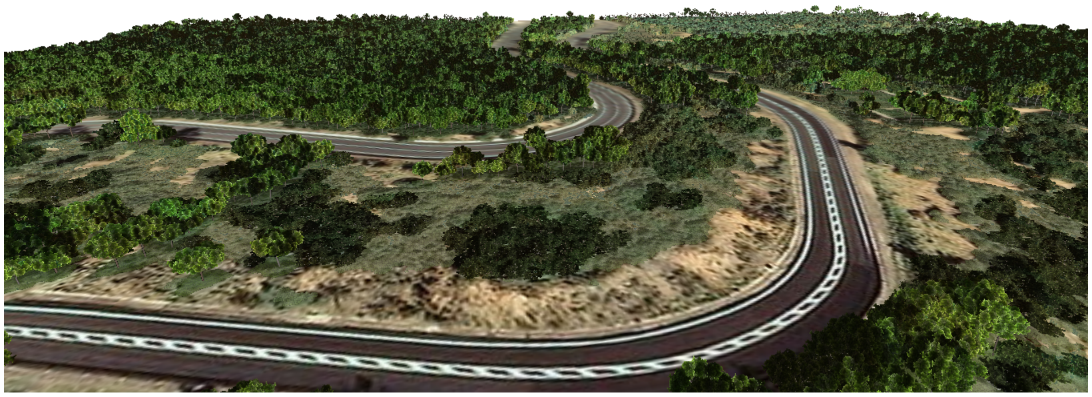

Figure 8.3: Landscape extracted from the final application.

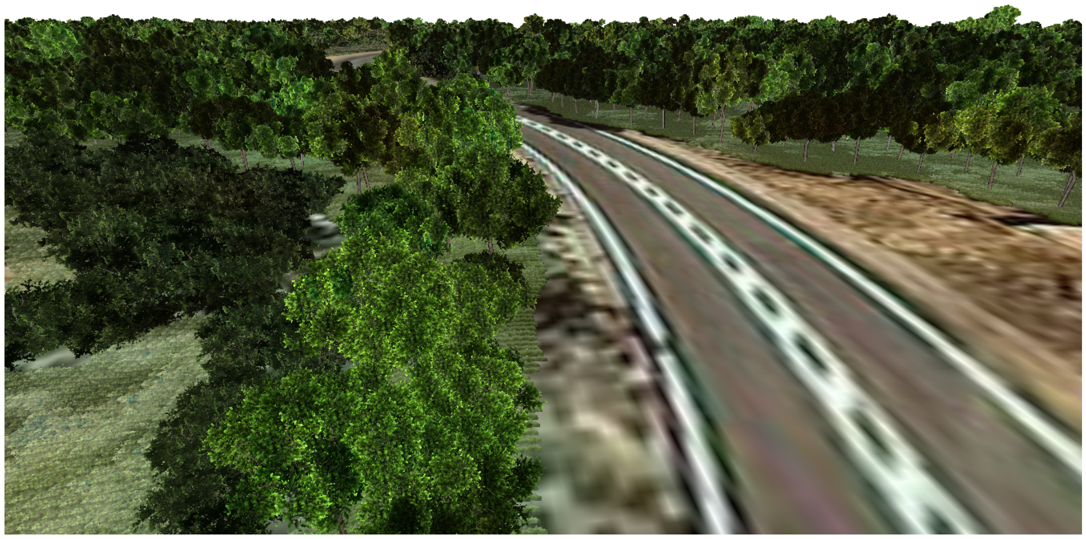

Figure 8.4: Landscape extracted from the final application.

9 References

[1] Institut cartografic de catalunya, . URL `http: //www.icc.cat/vissir3/`.

[2] Google earth. URL `http: //www.google.com/earth/index.html`.

[3] Nasa world wind. URL `http://worldwind.arc.nasa.gov`.

[4] Bing maps, microsoft. URL `http://www.bing.com/maps/`.

[5] Google earth 3d trees. URL `http://www.google.com/earth/explore/showcase /trees.html`.

[6] Tania Pouli, Douglas W. Cunningham, and Erik Reinhard. A survey of image statistics relevant to computer graphics. *Computer Graphics Forum*, 30(6):1761–1788, 2011. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2011.01900.x. URL `http://dx.doi.org/10.1111/j.1467-8659.2011.01900.x`.

[7] T. Sikora. The mpeg-7 visual standard for content description-an overview. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):696 –702, jun 2001. ISSN 1051-8215. doi: 10.1109/76.927422.

[8] L. A. Ruiz, A. Fdez-sarria, and J. A. Recio. Texture feature extraction for classification of remote sensing data using wavelet decomposition: a comparative study. In *International Archives of Photogrammetry and Remote Sensing. Vol.XXXV, ISSN*, pages 1682–1750, 2004.

[9] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2001. ISBN 0201180758.

[10] George Stockman and Linda G. Shapiro. *Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001. ISBN 0130307963.

[11] Colm Mulcahy Ph. D. Image compression using the haar wavelet transform.

[12] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[13] Oliver Deussen and Bernd Lintermann. *Digital Design of Nature: Computer Generated Plants and Organics*. SpringerVerlag, 2004. ISBN 3540405917.

[14] Xfrog inc. URL `http: //xfrog.com/`.

[15] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Eurographics Workshop on Natural Phenomena*, 2007.

[16] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. URL `http://www.R-project.org/`. ISBN 3-900051-07-0.

[17] Gregoire Pau, Andrzej Oles, Mike Smith, Oleg Sklyar, and Wolfgang Huber. *EBImage: Image processing toolbox for R*, 2012. R package version 4.0.0.

[18] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL `http://www.stats.ox.ac.uk/pub/MASS4`. ISBN 0-387-95457-0.

[19] Evgenia Dimitriadou. *cclust: Convex Clustering Methods and Clustering Indexes*, 2012. URL `http://CRAN.R-project.org/package=cclust`. R package version 0.6-17.

[20] Institut cartografic de catalunya, . URL `http://www.icc.cat/eng/Home-ICC/Digital-geoinformation/ About-ICC-geoinformation/Terms-of-use`.

[21] C. B. Barber, Kai Habel, Raoul Grasman, Robert B. Gramacy, Andreas Stahel, and David C. Sterratt. *geometry: Mesh generation and surface tesselation*, 2012. URL `http://CRAN.R-project.org/package=geometry`. R package version 0.3-2.

[22] Markus Loecher, Nikon Systems Inc, Nikon Digital Technologies Co., and Ltd. *ReadImages: Image Reading Module for R*, 2012. URL `http://CRAN.R-project.org/package=ReadImages`. R package version 0.1.3.3.

[23] Simon Urbanek. *png: Read and write PNG images*, 2011. URL `http://CRAN.R-project.org/package=png`. R package version 0.1-4.

[24] S Arya and D Mount. *ANN: library for approximate nearest neighbor searching*, 1997. URL `http://www.cs.umd.edu/ mount/ANN/`. Version 1.1.2 (Jan 2010).

[25] *OpenGL Mathematics (GLM)*. URL `http://glm.g-truc.net/`. Version 0.9.4.4 (May 2013).

[26] C Burnett and Thomas Blaschke. A multi-scale segmentation/object relationship modelling methodology for landscape analysis. *Ecological Modelling*, 168 (3):233 – 249, 2003. ISSN 0304-3800. doi: 10.1016/S0304-3800(03)00139-X. URL `http://www.sciencedirect.com /science/article/pii/S030438000300139X`. ¡ce:title¿Landscape Theory and Landscape Modelling¡/ce:title¿.

[27] R.M. Haralick, K. Shanmugam, and Its"Hak Dinstein. Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-3(6):610–621, 1973. ISSN 0018-9472. doi: 10.1109/TSMC.1973.4309314.

[28] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980. doi: 10.1098/rspb.1980.0020. URL `http://rspb.royalsocietypublishing.org/content/207/1167/187.abstract`.

[29] B.S. Manjunath, J.-R. Ohm, V.V. Vasudevan, and A. Yamada. Color and texture descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6): 703–715, 2001. ISSN 1051-8215. doi: 10.1109/76.927424.

[30] Kenneth I. Laws. Goal-directed textured-image segmentation. pages 19–26, 1985. doi: 10.1117/12.948402. URL + `http://dx.doi.org/10.1117/12.948402`.

[31] Jinggang Huang and David Mumford. D.: Image statistics for the british aerospace segmented database. Technical report, Division of Applied Math, Brown Univeristy, 1999.

[32] S.G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7):674–693, 1989. ISSN 0162-8828. doi: 10.1109/34.192463.

[33] G. Van de Wouwer, P. Scheunders, and D. Van Dyck. Statistical texture characterization from discrete wavelet representations. *Image Processing, IEEE Transactions on*, 8(4): 592–598, Apr. ISSN 1057-7149. doi: 10.1109/83.753747.

[34] Andrew J. Hanson and Hui Ma. Parallel transport approach to curve framing. Technical report, 1995.

[35] D. Shreiner and B.T.K.O.G.L.A.R.B.W. Group. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*. Pearson Education, 2009. ISBN 9780321669278. URL `http://books.google.es/books?id=xPu3mN2FPl4C`.

[36] Randi J. Rost, Bill Licea-Kane, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. *OpenGL Shading Language*. Addison-Wesley Professional, 3rd edition, 2009. ISBN 0321637631, 9780321637635.

A Gantt Diagram

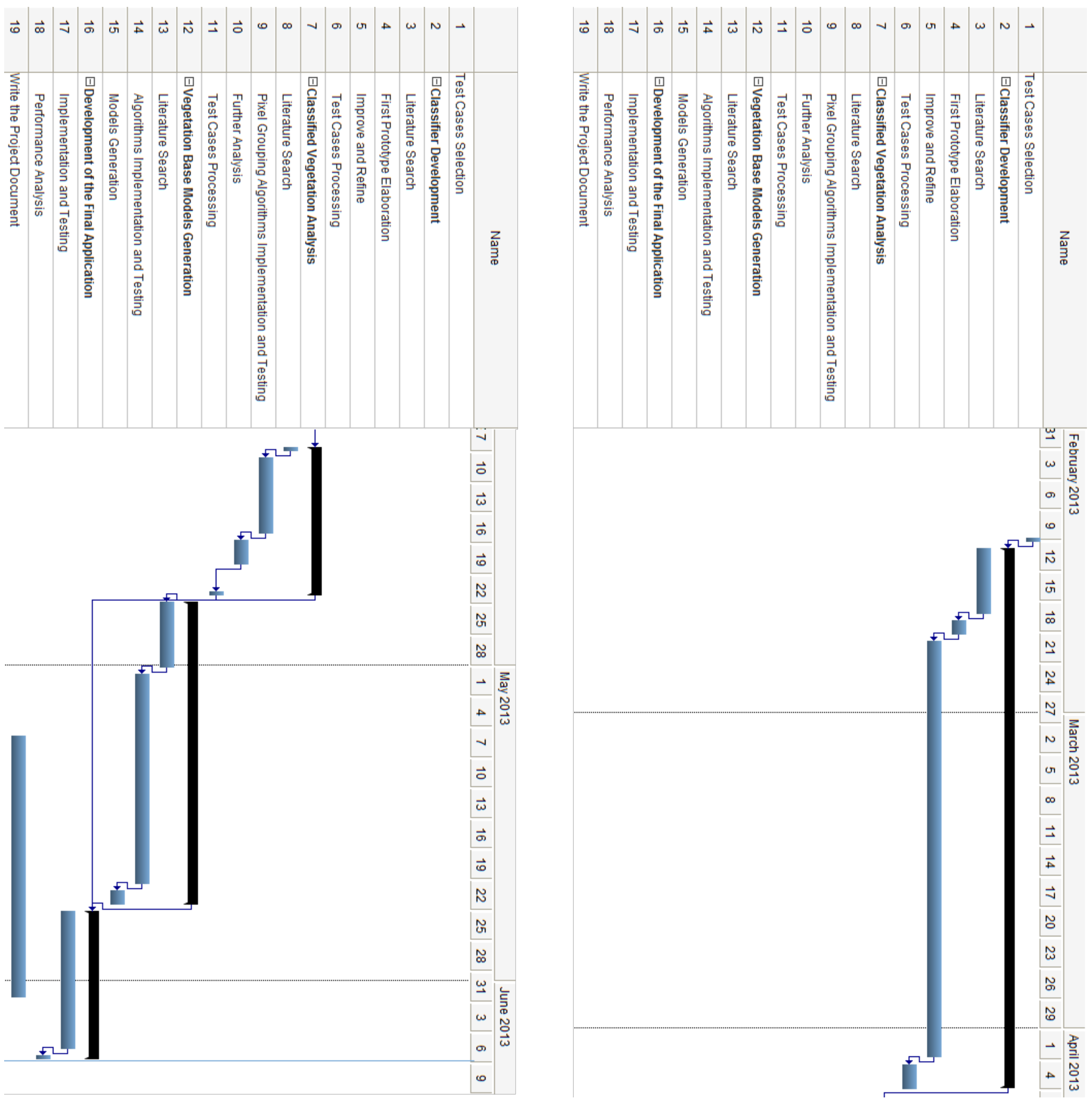

Figure A.1: Gantt diagram for the project panning.

B Classification Methods

This overview is a summary of what is explained in the book Pattern Recognition[12] plus some handmade demonstrations. Notice that some algorithms will be only mentioned as they are not relevant for this study and they are explained in the cited book.

B.1 Fisher's Linear Discriminant for two classes

Let's start introducing the Fisher discriminant for binary classification ($K = 2$). The input of our system is a D-dimensional vector x of features and our objective is to project it down to one dimension using $y = w^T x$. Choosing the adequate threshold w_0 we can classify as C_1 those samples for which $y \geq -w_0$ and the rest as C_2.

This way, the whole issue of classification is reduced to finding the parameters w which maximize the class separation. Consider a two-class problem where N_1 points belong to class C_1 and N_2 belong to class C_2, this way we define the mean vectors m_1 and m_2 as:

$$m_1 = \frac{1}{N_1} \sum_{n \in C_1} x_n, \quad m_2 = \frac{1}{N_2} \sum_{n \in C_2} x_n \tag{B.1}$$

Maximizing the separation between the the projected class means could be one criteria to follow, however it does not always work as expected. For instance, this does not work when the covariances of the class distributions are strongly non-diagonal. The idea proposed by Fisher is to maximize the separation between the projected class means while minimizing there variance within each class, thereby minimizing the class overlap.

Using the projection formula, we can transform a set of data points x into a set of points in the one-dimensional space y. Using this we can computed the within-class variance of the transformed data for the class C_k as:

$$y_n = w^T x_n \tag{B.2}$$

$$\mu_k = w^T m_k \tag{B.3}$$

$$s_k^2 = \sum_{n \in C_k} (y_n - \mu_k)^2 \tag{B.4}$$

The total within-class variance of the whole data is defined as $s_1^2 + s_2^2$. The Fisher criterion is defined to be the ratio of the between-class variance to the within-class variance:

$$J(w) = \frac{(\mu_2 - \mu_1)^2}{s_1^2 + s_2^2} \tag{B.5}$$

We can rewrite this to make the dependence on "w" explicit:

$$J(w) = \frac{w^T S_B w}{w^T S_W w} \tag{B.6}$$

where the between-class covariance matrix is:

$$S_B = (m_2 - m_1)(m_2 - m_1)^T \tag{B.7}$$

and the total within-class covariance matrix is:

$$S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum n \in C_2 (x_n - m_2)(x_n - m_2)^T \tag{B.8}$$

Applying some mathematics, we can find that $J(w)$ is maximized when:

$$w \propto S_w^{-1}(m_2 - m_1) \tag{B.9}$$

This result is known as *Fisher's linear discriminant*, although it is not strictly a discriminant but a direction for the projection of the data to one dimension. To find the adequate threshold we could model the class-conditional densities $p(y|C_k)$ as Gaussians using maximum likelihood and then find the optimal threshold.

B.2 Fisher's Linear Discriminant for multiple classes

To generalize the Fisher discriminant to classify into more than two classes ($K > 2$) we shall assume that the dimensionality D of the input space is greater than the number K of classes. Next, we introduce $D' > 1$ linear "features" $y_k = w_k x$, where $k = 1, ..., D$. The vector y is the result of the grouping of this features and the matrix the weight vectors w_k can be considered to be the columns of tre matrix W, thus:

$$y = W^T x \tag{B.10}$$

$$S_W = \sum_{k=1}^{K} S_k \tag{B.11}$$

where

$$S_k = \sum_{n \in C_k} (x_n - m_k)(x_m - m_k)^T \tag{B.12}$$

$$S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T \tag{B.13}$$

where

$$m = \frac{1}{N} \sum_{n=1}^{N} x_n = \frac{1}{N} \sum_{k=1}^{K} N_k m_k \tag{B.14}$$

In this case, there are many possible choices for the criterion to maximize. We want a criterion that is large when the between-class covariance is large and when the within-class covariance is small. One standard criterion is:

$$Tr(WS_W W^T)^{-1}(WS_B W^T) \tag{B.15}$$

B.3 Linear Discriminant Analysis for two classes

Adopting the Generative approach, we will model the class-conditional densities $p(x|C_k)$, as well as the class priors $p(C_k)$, and then these will be used to compute posterior probabilities $p(C_k|x)$ through Bayes' theorem.

$$\begin{aligned} P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x)} &= \frac{P(x|C_1)P(C_1)}{P(x|C_1)P(C_1) + P(x|C_2)P(C_2)} = \frac{1}{1 + \frac{P(x|C_2)P(C_2)}{P(x|C_1)P(C_1)}} = \\ &= \frac{1}{1 + \exp(-a(x))} \quad \text{where} \quad a(x) = \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} \end{aligned} \tag{B.16}$$

If we define logistic function as $g(z) = \frac{1}{1+\exp{(-z)}}$ we can rewrite $P(C_1|x)$ as $P(C_1|x) = g(a(x))$. From now on we will work on $a(x)$:

Assuming that the class-conditional densities are Gaussian:

$$P(x|C_k) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{B.17}$$

$$\begin{aligned} a(x) &= \ln \frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)} \\ &= \ln \frac{(2\pi)^{-\frac{d}{2}}|\Sigma_1|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\} * P(C_1)}{(2\pi)^{-\frac{d}{2}}|\Sigma_2|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\} * P(c2)} = \\ &= \ln \frac{|\Sigma_1|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\} * P(C_1)}{|\Sigma_2|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu_2)^T\Sigma_2^{-1}(x-\mu_2)\} * P(C_2)} \end{aligned} \tag{B.18}$$

Assuming that all classes share the same covariance matrices:

$$\begin{aligned} a(x) &= \ln \frac{|\Sigma|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\} * P(C_1)}{|\Sigma|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2)\} * P(C_2)} \\ &= \ln \frac{\exp\{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\} * P(C_1)}{\exp\{-\frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2)\} * P(C_2)} \\ &= -\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2) + \ln\frac{P(C_1)}{P(C_2)} \end{aligned} \tag{B.19}$$

Note 1):

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu - \mu^T\Sigma^{-1}x + \mu^T\Sigma^{-1}\mu \tag{B.20}$$

Note 2): Σ is a symmetric matrix, therefore:

$$x^T\Sigma^{-1}\mu = \mu^T\Sigma^{-1}x \tag{B.21}$$

Note 3): Taking into account 1) and 2) we can rewrite $(x-\mu)^T\Sigma^{-1}(x-\mu)$ as:

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = x^T\Sigma^{-1}x - 2 * \mu^T\Sigma^{-1}x + \mu^T\Sigma^{-1}\mu \tag{B.22}$$

Finally:

$$\begin{aligned}a(x) &= -\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1) + \frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2) + \ln\frac{P(C_1)}{P(C_2)} = \\ &= -\frac{1}{2}(x^T\Sigma^{-1}x) + (\mu_1^T\Sigma^{-1}x) - \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1) + \frac{1}{2}(x^T\Sigma^{-1}x) - (\mu_2^T\Sigma^{-1}x) + \frac{1}{2}(\mu_2^T\Sigma^{-1}\mu_2) \\ &+ \ln\frac{P(C_1)}{P(C_2)} = \\ &= (\mu_1^T\Sigma^{-1}x) - \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1) - (\mu_2^t\Sigma^{-1}x) + \frac{1}{2}(\mu_2^T\Sigma^{-1}\mu_2) + \ln\frac{P(C_1)}{P(C_2)} = \\ &= (\mu_1^T\Sigma^{-1} - \mu_2^T\Sigma^{-1})x - \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1) + \frac{1}{2}(\mu_2^T\Sigma^{-1}\mu_2) + \ln\frac{P(C_1)}{P(C_2)} = w^Tx + w_0\end{aligned} \tag{B.23}$$

$$w_0 = -\frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1) + \frac{1}{2}(\mu_2^T\Sigma^{-1}\mu_2) + \ln\frac{P(C_1)}{P(C_2)} \tag{B.24}$$

$$w^T = \mu_1^T\Sigma^{-1} - \mu_2^T\Sigma^{-1} = (\mu_1^T - \mu_2^T)\Sigma^{-1} \tag{B.25}$$

$$w = ((\mu_1^T - \mu_2^T)\Sigma^{-1})^T = (\Sigma^{-1})^T(\mu_1^T - \mu_2^T)^T = \Sigma^{-1}((\mu_1^T)^T - (\mu_2^T)^T) = \Sigma^{-1}(\mu_1 - \mu_2) \tag{B.26}$$

Therefore: $P(C_1|x) = g(w^Tx + w_0)$.

The same process could be carried out for $P(C_2|x)$.

B.4 Linear Discriminant Analysis for multiple classes

$$P(C_k|x) = \frac{P(x|C_k) * P(C_k)}{P(x)} = \frac{P(x|C_k) * P(C_k)}{\sum\limits_{j=1}^{K} P(x|C_j)P(C_j)} \tag{B.27}$$

Assuming that class-conditional densities are Gaussian with the same covariance matrices:

$$\begin{aligned}
P(C_k|x) &= \frac{(2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}\exp\{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\} * P(C_k)}{\sum\limits_{j=1}^{K}(2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}\exp\{-\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\} * P(C_j)} \\
&= \frac{(2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}\exp\{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\} * P(C_k)}{(2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}\sum\limits_{j=1}^{K}\exp\{-\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\} * P(C_j)} \\
&= \frac{\exp\{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\} * P(C_k)}{\sum\limits_{j=1}^{K}\exp\{-\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\} * P(C_j)}
\end{aligned} \tag{B.28}$$

Note 1):

$$(x-\mu)^T\Sigma^{-1}(x-\mu) = x^T\Sigma^{-1}x - 2*\mu^T\Sigma^{-1}x + \mu^T\Sigma^{-1}\mu \tag{B.29}$$

Following:

$$\begin{aligned}
P(C_k|x) &= \frac{\exp\{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)\} * P(C_k)}{\sum\limits_{j=1}^{K}\exp\{-\frac{1}{2}(x-\mu_j)^T\Sigma^{-1}(x-\mu_j)\} * P(C_j)} \\
&= \frac{\exp\{-\frac{1}{2}(x^T\Sigma^{-1}x) + (\mu_k^T\Sigma^{-1}x) - \frac{1}{2}(\mu_k^T\Sigma^{-1}\mu_k)\} * P(C_k)}{\sum\limits_{j=1}^{K}\exp\{-\frac{1}{2}(x^T\Sigma^{-1}x) + (\mu_j^T\Sigma^{-1}x) - \frac{1}{2}(\mu_j^T\Sigma^{-1}\mu_j)\} * P(C_j)} \\
&= \frac{\exp\{-\frac{1}{2}(x^T\Sigma^{-1}x)\} * \exp\{(\mu_k^T\Sigma^{-1}x) - \frac{1}{2}(\mu_k^T\Sigma^{-1}\mu_k)\} * P(C_k)}{\exp\{-\frac{1}{2}(x^T\Sigma^{-1}x)\} * \sum\limits_{j=1}^{K}\exp\{(\mu_j^T\Sigma^{-1}x) - \frac{1}{2}(\mu_j^T\Sigma^{-1}\mu_j)\} * P(C_j)} \\
&= \frac{\exp\{(\mu_k^T\Sigma^{-1}x) - \frac{1}{2}(\mu_k^T\Sigma^{-1}\mu_k)\} * P(C_k)}{\sum\limits_{j=1}^{K}\exp\{(\mu_j^T\Sigma^{-1}x) - \frac{1}{2}(\mu_j^T\Sigma^{-1}\mu_j)\} * P(C_j)} \\
&= \frac{\exp\{(\mu_k^T\Sigma^{-1}x) - \frac{1}{2}(\mu_k^T\Sigma^{-1}\mu_k) + \ln(P(C_k))\}}{\sum\limits_{j=1}^{K}\exp\{(\mu_j^T\Sigma^{-1}x) - \frac{1}{2}(\mu_j^T\Sigma^{-1}\mu_j) + \ln(P(C_j))\}} \\
&= \frac{\exp\{a_k(x)\}}{\sum\limits_{j=1}^{K}\exp\{a_j(x)\}}
\end{aligned} \tag{B.30}$$

If we define *softmax* function as $g(z_k) = \frac{\exp(z_k)}{\sum\limits_{j=1}^{K}\exp(z_j)}$ we can rewrite $P(C_k|x)$ as $P(C_k|x) = g(a_k(x))$. From now on we will work on $a_k(x)$:

$$a_k(x) = (\mu_k^T \Sigma^{-1} x) - \frac{1}{2}(\mu_k^T \Sigma^{-1} \mu_k) + \ln(P(C_k)) = w_k^T x + w_{k0} \tag{B.31}$$

$$w_{k0} = -\frac{1}{2}(\mu_k^T \Sigma^{-1} \mu_k) + \ln(P(C_k)) \tag{B.32}$$

$$w_k^T = (\mu_k^T \Sigma^{-1}) \tag{B.33}$$

$$w_k = (\mu_k^T \Sigma^{-1})^T = (\Sigma^{-1})^T (\mu_k^T)^T = \Sigma^{-1} \mu_k \tag{B.34}$$

Therefore: $P(C_k|x) = g(w_k^T x + w_{k0})$.

A curious fact is that when the LDA assumptions are satisfied, Fisher's method and this one become equivalent.

B.5 Quadratic Discriminant Analysis

If LDA method we relax the assumption of a shared covariance matrix and allow each class-conditional density $p(x|C_k)$ to have its own covariance matrix quadratic functions of x will be obtained, giving rise to a quadratic discriminant.

B.6 Multinomial Regression

The Multinomial (or Multiclass) Logistic Regression is a Discriminative Probabilistic Model, which means that it uses the functional form of the generalized linear models explicitly and determines its parameters using maximum likelihood. There is an efficient algorithm finding such solutions known as iterative reweighted least squares, or IRLS.

The posterior probabilities are given by a softmax transformation of linear functions of the feature variables as follows:

$$P(C_k|\phi) = y_k(\phi) = g(a_k) = \frac{exp(a_k)}{\sum_j exp(a_j)} \quad where \quad a_k = w_k^T \phi \tag{B.35}$$

Now maximum likelihood can be used to directly determine the parameters w_k for this model. To do this, the derivatives of y_k with respect to all of a_j are needed. These are given by:

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j) \tag{B.36}$$

Where I_{kj} are the elements of the identity matrix. The likelihood function is given by:

$$P(T|w_1, ..., w_k) = \prod_{n=1}^{N}\prod_{k=1}^{K} P(C_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_{nk}} \tag{B.37}$$

Where t_n for a feature vector ϕ_n belonging to a class C_k is a vector of zeros that contains a 1 at position k, $y_{nk} = y_k(\phi_n)$ and T is an N x K matrix of target variables with elements t_{nk}. The negative logrithm gives the known as *cross-entropy* function:

$$E(w_1, ..., w_k) = -\ln P(T|w_1, ..., w_k) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ln y_{nk} \tag{B.38}$$

With this we can compute the gradient and the hessian of the error function:

$$\nabla_{w_j} E(w_1, ..., w_k) = \sum_{n=1}^{N}(y_{nj} - t_{nj})\phi_n \tag{B.39}$$

$$\nabla_{w_k}\nabla_{w_j} E(w_1, ..., w_k) = -\sum_{n=1}^{N} y_{nk}(I_{kj} - y_{nj})\phi_n\phi_n^T \tag{B.40}$$

The Hessian matrix for the multiclass logistic regression model is positive definite and so the error function has a unique minimum. The IRLS algorithm can find this minimum using the gradient and the hessian found before.

C 3D Vegetation Models

C.1 Tree 1

C.1.1 Parameters

random_seed	100	node_kill_distance	2.3
tree_or_shrub	1	max_attraction_distance	12
initial_trunk_height	17	min_attraction_vector_size	0.5
crown_height	30	branch_tip_radius	0.1
crown_distance1	25	branch_radius_exponent	2.2
crown_distance2	20	simplifying_by_area	1
crown_height_displacement	-3	simplifying_angle	7
attraction_points	2000	simplifying_area	0.57
min_attraction_points_distance	2.3	max_simplifying_iterations	3
max_branch_gen_iterations	150	reducing_bifurcation_to_half	1
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.1.2 Visual Result

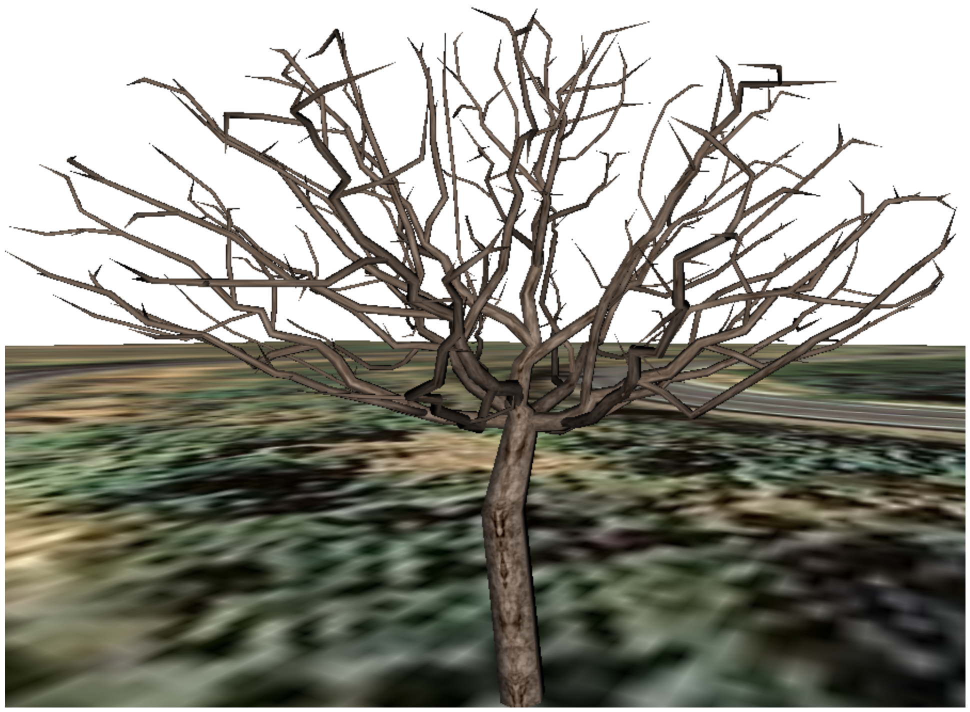

Figure C.1: A 3D model of a tree, used for the final application.

C.1.3 Model Information

```
####################Statistics###################
138     ms spent generating attraction points.
314  ms spent computing attraction vectors
14  ms spent computing new branches
0  ms spent computing the branches radius
1  ms spent simplifying the branch structure
1  ms spent reducing branch angles
4  ms spent generating the triangle strip
Total generating time: 472

1286 attraction points were generated
67 iterations were performed
3117 generalized cylinders.

5947 vertices for the High Resolution Model.
9198 faces for the High Resolution Model.

2978 vertices for the Medium Resolution Model.
5268 faces for the Medium Resolution Model.

2013 vertices for the Low Resolution Model.
3570 faces for the Low Resolution Model.
##################################################
```

C.2 Tree 2

C.2.1 Parameters

random_seed	150	node_kill_distance	2.5
tree_or_shrub	1	max_attraction_distance	8
initial_trunk_height	25	min_attraction_vector_size	0.5
crown_height	30	branch_tip_radius	0.1
crown_distance1	23	branch_radius_exponent	2.2
crown_distance2	25	simplifying_by_area	1
crown_height_displacement	-5	simplifying_angle	10
attraction_points	1000	simplifying_area	0.4
min_attraction_points_distance	2.3	max_simplifying_iterations	3
max_branch_gen_iterations	150	reducing_bifurcation_to_half	1
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.2.2 Visual Result

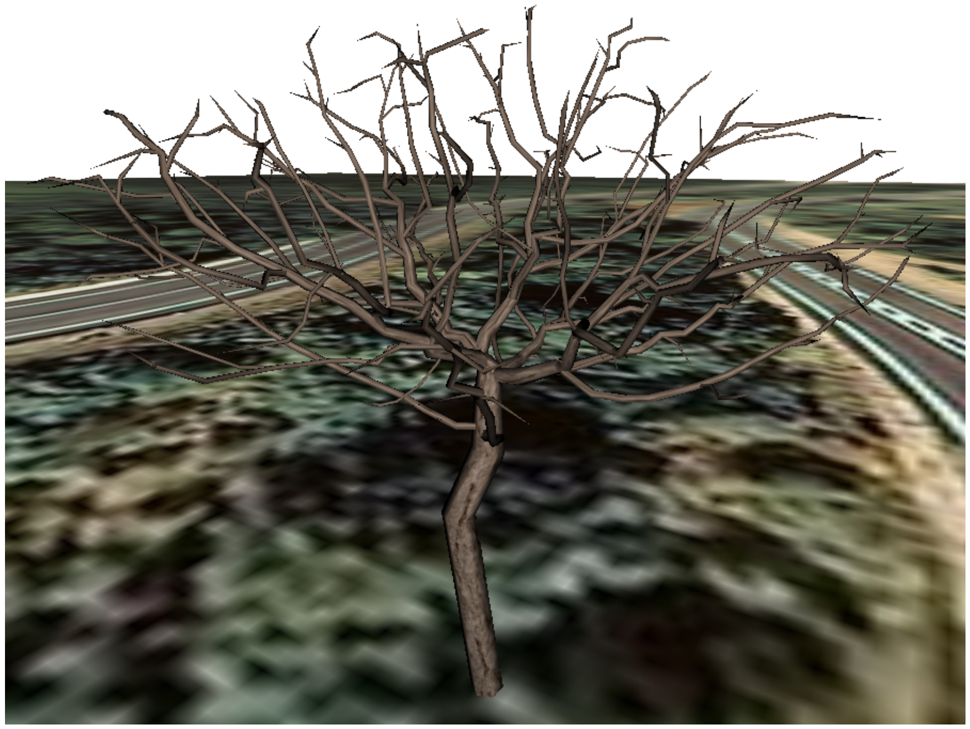

Figure C.2: A 3D model of a tree, used for the final application.

C.2.3 Model Information

```
####################Statistics###################
30     ms spent generating attraction points.
129  ms spent computing attraction vectors
15  ms spent computing new branches
0  ms spent computing the branches radius
1  ms spent simplifying the branch structure
1  ms spent reducing branch angles
3  ms spent generating the triangle strip
Total generating time: 179

1000 attraction points were generated
70 iterations were performed
2680 generalized cylinders.

5345 vertices for the High Resolution Model.
8682 faces for the High Resolution Model.

2982 vertices for the Medium Resolution Model.
5292 faces for the Medium Resolution Model.

2080 vertices for the Low Resolution Model.
3744 faces for the Low Resolution Model.
##################################################
```

C.3 Tree 3

C.3.1 Parameters

random_seed	178	node_kill_distance	1.0
tree_or_shrub	1	max_attraction_distance	11.5
initial_trunk_height	40	min_attraction_vector_size	0.5
crown_height	50	branch_tip_radius	0.1
crown_distance1	15	branch_radius_exponent	2.2
crown_distance2	30	simplifying_by_area	1
crown_height_displacement	-15	simplifying_angle	10
attraction_points	500	simplifying_area	0.4
min_attraction_points_distance	2.3	max_simplifying_iterations	3
max_branch_gen_iterations	150	reducing_bifurcation_to_half	1
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.3.2 Visual Result

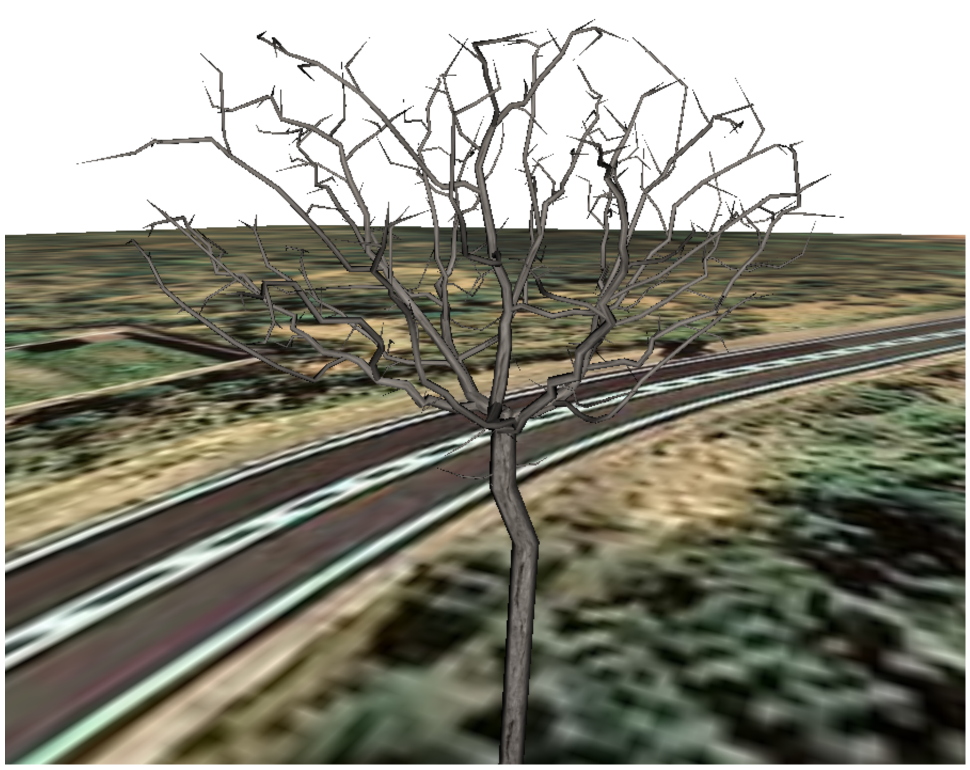

Figure C.3: A 3D model of a tree, used for the final application.

C.3.3 Model Information

```
####################Statistics###################
3     ms spent generating attraction points.
236  ms spent computing attraction vectors
8  ms spent computing new branches
1  ms spent computing the branches radius
1  ms spent simplifying the branch structure
1  ms spent reducing branch angles
2  ms spent generating the triangle strip
Total generating time: 252

500 attraction points were generated
93 iterations were performed
3254 generalized cylinders.

5413 vertices for the High Resolution Model.
8418 faces for the High Resolution Model.

2711 vertices for the Medium Resolution Model.
4806 faces for the Medium Resolution Model.

1902 vertices for the Low Resolution Model.
3468 faces for the Low Resolution Model.
##################################################
```

C.4 Tree 4

C.4.1 Parameters

random_seed	11	node_kill_distance	2.5
tree_or_shrub	1	max_attraction_distance	8.0
initial_trunk_height	35	min_attraction_vector_size	0.5
crown_height	50	branch_tip_radius	0.1
crown_distance1	30	branch_radius_exponent	2.2
crown_distance2	24	simplifying_by_area	1
crown_height_displacement	-10	simplifying_angle	10
attraction_points	1700	simplifying_area	0.7
min_attraction_points_distance	4.0	max_simplifying_iterations	4
max_branch_gen_iterations	150	reducing_bifurcation_to_half	1
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.4.2 Visual Result

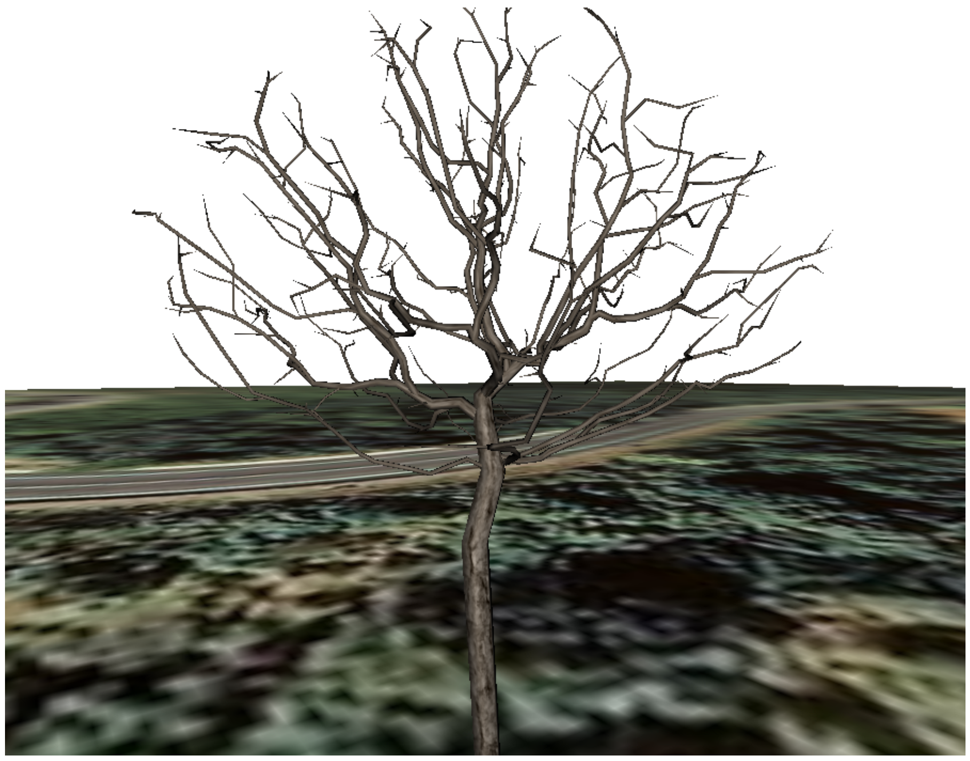

Figure C.4: A 3D model of a tree, used for the final application.

C.4.3 Model Information

```
###################Statistics###################
68     ms spent generating attraction points.
132  ms spent computing attraction vectors
10  ms spent computing new branches
0  ms spent computing the branches radius
1  ms spent simplifying the branch structure
2  ms spent reducing branch angles
3  ms spent generating the triangle strip
Total generating time: 216

686 attraction points were generated
93 iterations were performed
3541 generalized cylinders.

5506 vertices for the High Resolution Model.
8724 faces for the High Resolution Model.

2985 vertices for the Medium Resolution Model.
5370 faces for the Medium Resolution Model.

2133 vertices for the Low Resolution Model.
3858 faces for the Low Resolution Model.
##################################################
```

C.5 Tree 5

C.5.1 Parameters

random_seed	11	node_kill_distance	3.7
tree_or_shrub	1	max_attraction_distance	20
initial_trunk_height	50	min_attraction_vector_size	0.5
crown_height	60	branch_tip_radius	0.1
crown_distance1	30	branch_radius_exponent	2.2
crown_distance2	30	simplifying_by_area	1
crown_height_displacement	-5	simplifying_angle	10
attraction_points	4000	simplifying_area	2.7
min_attraction_points_distance	2.0	max_simplifying_iterations	4
max_branch_gen_iterations	150	reducing_bifurcation_to_half	1
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.5.2 Visual Result

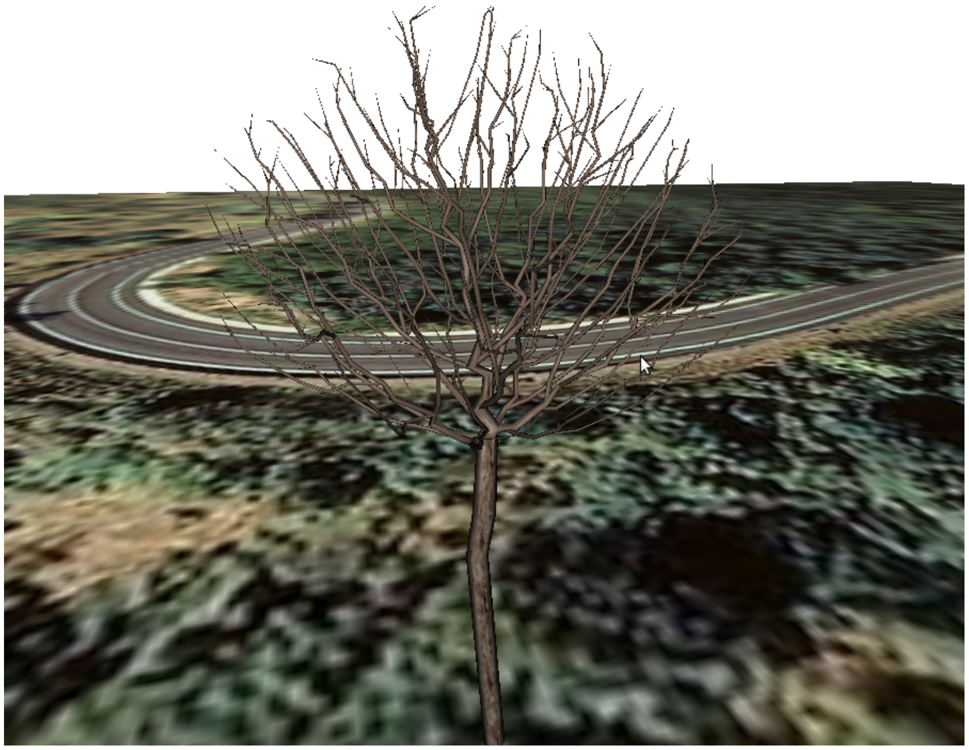

Figure C.5: A 3D model of a tree, used for the final application.

C.5.3 Model Information

```
####################Statistics###################
328     ms spent generating attraction points.
2373  ms spent computing attraction vectors
35  ms spent computing new branches
0  ms spent computing the branches radius
2  ms spent simplifying the branch structure
2  ms spent reducing branch angles
3  ms spent generating the triangle strip
Total generating time: 2743

4000 attraction points were generated
111 iterations were performed
5957 generalized cylinders.

6595 vertices for the High Resolution Model.
9822 faces for the High Resolution Model.

3075 vertices for the Medium Resolution Model.
5406 faces for the Medium Resolution Model.

2015 vertices for the Low Resolution Model.
3510 faces for the Low Resolution Model.
##################################################
```

C.6 Shrub 1

C.6.1 Parameters

random_seed	100	node_kill_distance	2.0
trees_or_shrubs	0	max_attraction_distance	8
initial_trunk_height	0	min_attraction_vector_size	0.5
crown_height	30	branch_tip_radius	0.1
crown_distance1	20	branch_radius_exponent	1.7
crown_distance2	17	simplifying_by_area	1
crown_height_displacement	-2.0	simplifying_angle	10
attraction_points	2000	simplifying_area	1.8
min_attraction_points_distance	2.3	max_simplifying_iterations	4
max_branch_gen_iterations	150	reducing_bifurcation_to_half	0
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.6.2 Visual Result

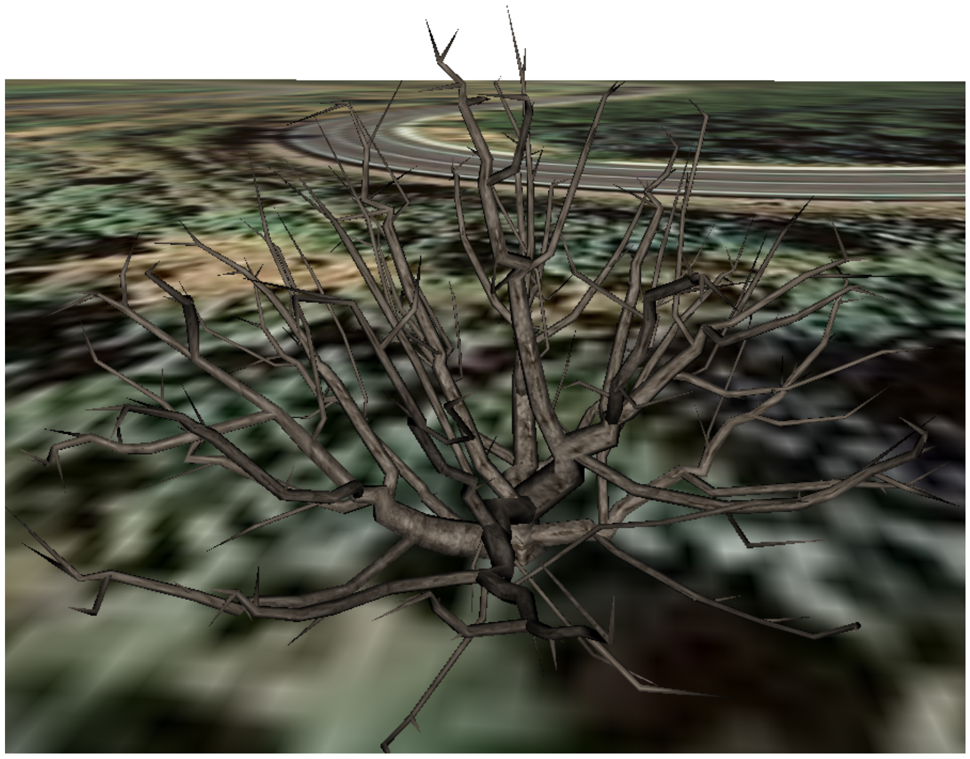

Figure C.6: A 3D model of a shrub, used for the final application.

C.6.3 Model Information

```
####################Statistics###################
112     ms spent generating attraction points.
116  ms spent computing attraction vectors
4  ms spent computing new branches
1  ms spent computing the branches radius
1  ms spent simplifying the branch structure
0  ms spent reducing branch angles
2  ms spent generating the triangle strip
Total generating time: 236

963 attraction points were generated
65 iterations were performed
2499 generalized cylinders.

3847 vertices for the High Resolution Model.
5670 faces for the High Resolution Model.

1628 vertices for the Medium Resolution Model.
2808 faces for the Medium Resolution Model.

1009 vertices for the Low Resolution Model.
1770 faces for the Low Resolution Model.
#################################################
```

C.7 Shrub2

C.7.1 Parameters

random_seed	100	node_kill_distance	2.7
trees_or_shrubs	0	max_attraction_distance	14
initial_trunk_height	0	min_attraction_vector_size	0.5
crown_height	40	branch_tip_radius	0.1
crown_distance1	20	branch_radius_exponent	1.8
crown_distance2	20	simplifying_by_area	1
crown_height_displacement	-4.0	simplifying_angle	10
attraction_points	2000	simplifying_area	2.4
min_attraction_points_distance	2.3	max_simplifying_iterations	4
max_branch_gen_iterations	150	reducing_bifurcation_to_half	0
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.7.2 Visual Result

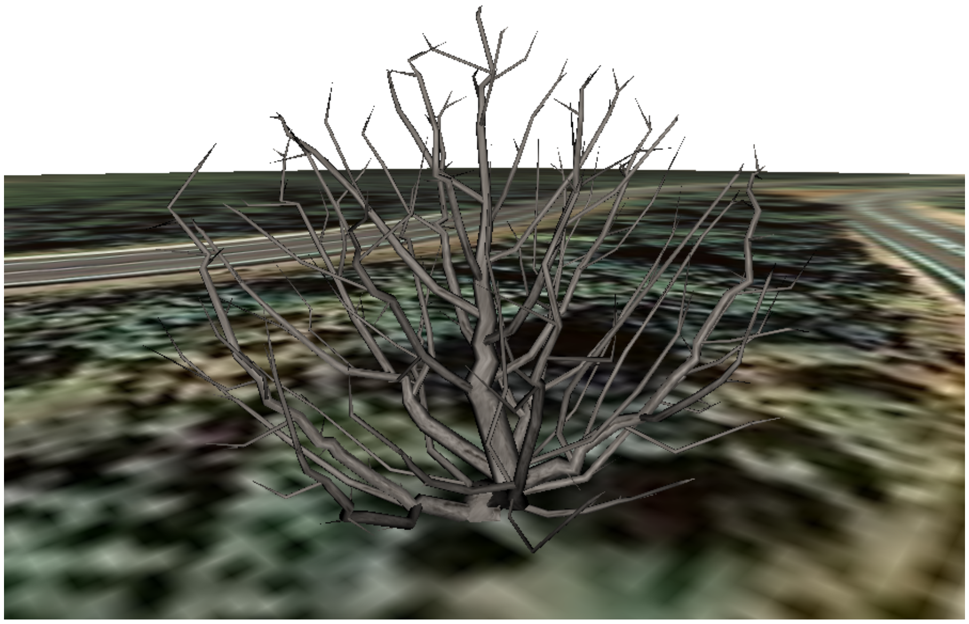

Figure C.7: A 3D model of a shrub, used for the final application.

C.7.3 Model Information

```
####################Statistics###################
160     ms spent generating attraction points.
376  ms spent computing attraction vectors
14  ms spent computing new branches
0  ms spent computing the branches radius
1  ms spent simplifying the branch structure
1  ms spent reducing branch angles
2  ms spent generating the triangle strip
Total generating time: 554

1368 attraction points were generated
74 iterations were performed
2761 generalized cylinders.

3692 vertices for the High Resolution Model.
5400 faces for the High Resolution Model.

1570 vertices for the Medium Resolution Model.
2724 faces for the Medium Resolution Model.

1003 vertices for the Low Resolution Model.
1758 faces for the Low Resolution Model.
##################################################
```

C.8 Shrub 3

C.8.1 Parameters

random_seed	100	node_kill_distance	2.05
trees_or_shrubs	0	max_attraction_distance	4.1
initial_trunk_height	0	min_attraction_vector_size	0.5
crown_height	30	branch_tip_radius	0.1
crown_distance1	30	branch_radius_exponent	1.7
crown_distance2	30	simplifying_by_area	1
crown_height_displacement	-4.0	simplifying_angle	10
attraction_points	2000	simplifying_area 12	
min_attraction_points_distance	2.5	max_simplifying_iterations	10
max_branch_gen_iterations	150	reducing_bifurcation_to_half	0
new_node_distance	0.5	max_bifurcation_angle	30
cylinder_edges	6		

C.8.2 Visual Result

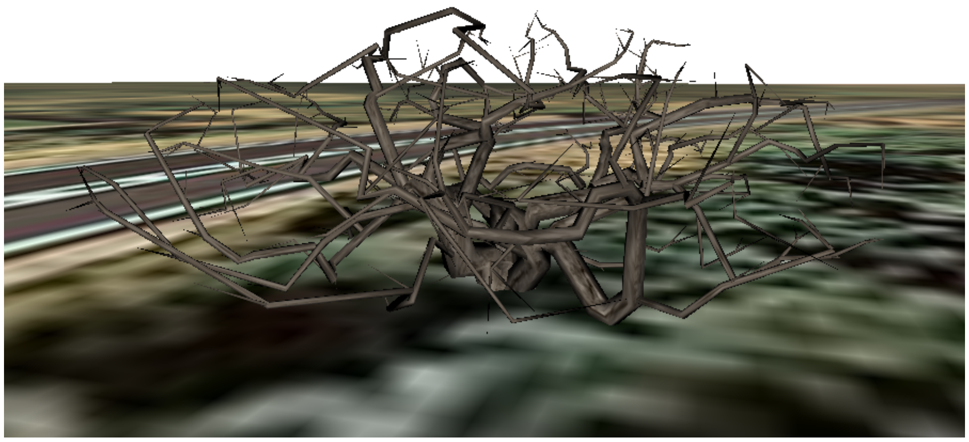

Figure C.8: A 3D model of a shrub, used for the final application.

C.8.3 Model Information

```
####################Statistics####################
197     ms spent generating attraction points.
254  ms spent computing attraction vectors
14  ms spent computing new branches
0  ms spent computing the branches radius
1  ms spent simplifying the branch structure
1  ms spent reducing branch angles
2  ms spent generating the triangle strip
Total generating time: 469

1698 attraction points were generated
150 iterations were performed
3924 generalized cylinders.

4305 vertices for the High Resolution Model.
5994 faces for the High Resolution Model.

1587 vertices for the Medium Resolution Model.
2766 faces for the Medium Resolution Model.

1020 vertices for the Low Resolution Model.
1800 faces for the Low Resolution Model.
##################################################
```

D The OpenGL Pipeline

OpenGL is a programming language specification that defines a multi-platform API for rendering 2D and 3D object graphics. The *pipeline* is the sequence of operations that are applied over the input data. Bases on what is explained on [35], we are giving a brief description of the OpenGL pipeline operations, up to version 4.1:

- Geometry if send to the GPU. Usually the information sent for each vertex is: position, normal, color and texture coordinates. The best way to carry out this process is using a *Vertex Buffer Object*, because in this manner the information is copied to the GPU internal memory.
- Each vertex is processed using the Vertex Shader. The default computations that a Vertex Shader performs are transforming the vertex coordinates from *world space* to *clipping space*, however many other functionalities can be programmed such as transforming the normal into *eye space*, modifying texture coordinates, etc
- During the *Primitive Assembly*, the vertices that came from the Vertex Shader are grouped into primitives, such as triangles, quads, lines, etc.
- For each of the generated primitives, a Geometry Shader is optionally executed. This shader is able to destroy, modify or generate new geometry, taking as input a single primitive.
- In the following phase, those primitives that are not inside the frustum of vision are discarded. This process is called *clipping* and also trims the primitives that are partially outside the frustum. Another process called *culling* is also applied, discarding those faces whose normal does not pass a certain test (for instance, if it points in the same direction as the view direction).
- *Rasterizations* is the process of calculating which pixels a certain primitive covers. For each detected pixel a fragment is generated. Also, some properties, such as color, are asigned to each fragment, interpolating the properties of the vertices of the primitive that generated it.
- A Fragment Shader is applied over each of the generated fragments. During this process the final color is computed taking into account, for instance, illumination effects.
- Finally, different test and operations can be applied over the processed fragments. The most common test is the *z-test*, that discards a fragment if its depth is bigger than the one in the *Frame buffer*, meaning that an object that was closer to the camera was painted before. An example of fragment operation could be *alpha blending*, that mixes the color of the fragment with the color that was in the same position in the *Frame buffer*.
- The final result is left in the *Frame buffer*. From there it can be painted on the screen or used as a texture for a future step.

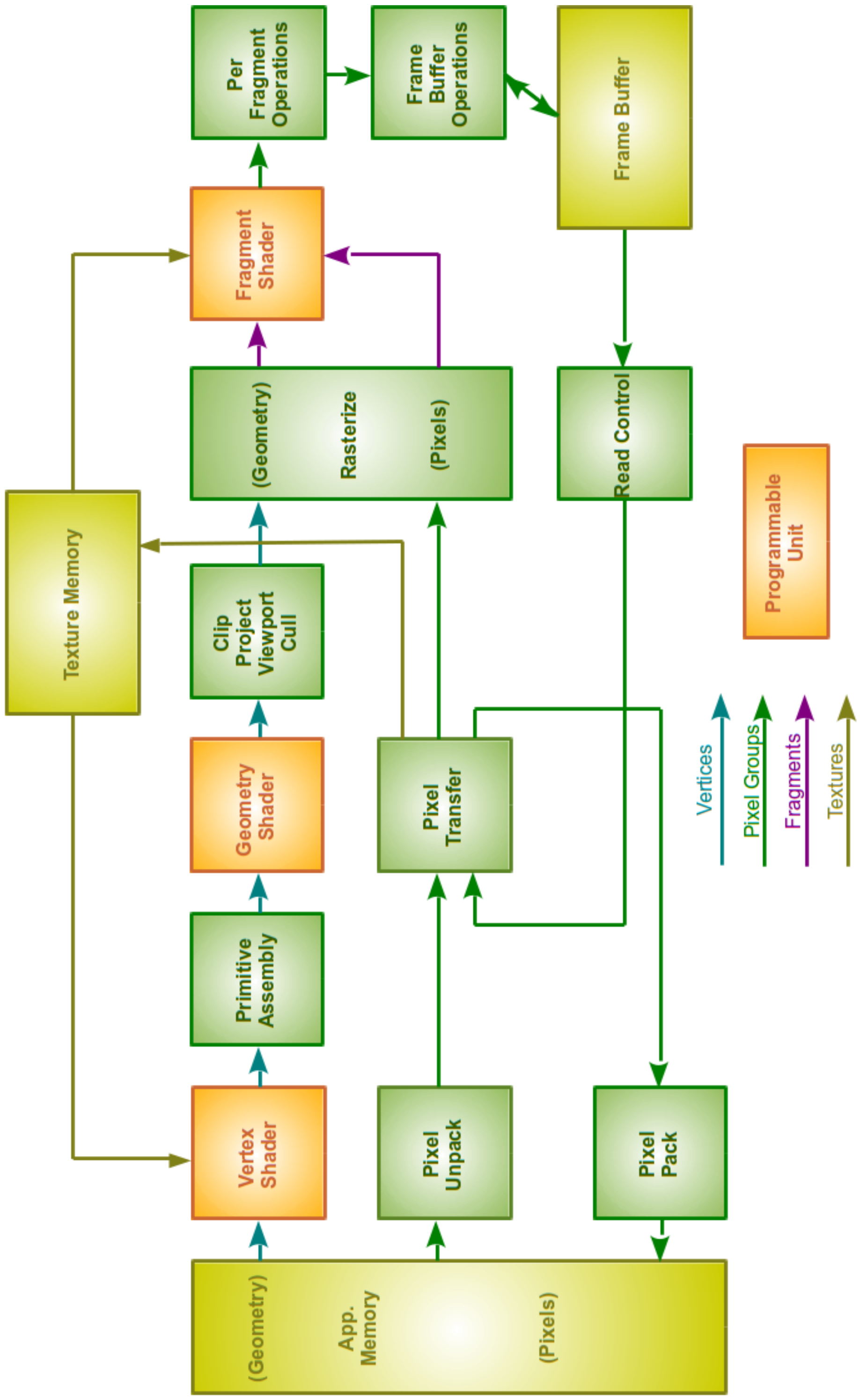

Figure D.1: OpenGL pipeline up to version 4.1.

E Vertex Buffer Objects

E.1 Vertex Arrays

Instead of individually specifying the vertex data in immediate mode (using glBegin() and glEnd() operations), one can store vertex data in a set of arrays. One array must be built for each property of the vertex, including vertex positions, normals, texture coordinates and color information. Finally, primitives can be defined dereferencing the array elements with array indexes.

The advantage in comparison with the immediate mode is that vertices are only processed once for each draw call. This means that if two primitives share a vertex, redundant computations are avoided. Also less calls to OpenGL functions are used.

E.2 Vertex Buffer Objects

When using Vertex Arrays, the geometry data stays in the main memory meaning that, at each call, all the geometry must be sent to the GPU. This might become the bottleneck of a system, if enough geometry is being rendered.

Vertex Buffer Objects store the geometry data on the GPU memory. This way, the data must be only sent once and, when drawing, only indexes must be sent through the bus. On the other hand, it might be a problem if the GPU memory is not able to store all the geometry data or if one wishes to modify the stored data.

F Glossary

- **ANN:** Approximate Nearest Neighbor library. A library that implements useful data structures and operations for efficient neighbor search.
- **Bézier Curve:** Parametric curve often used in computer graphics.
- **Billboard:** Oriented, textured polygon.
- **Clipping Space:** A geometry element is said to be in *clipping space coordinates*, when the Model View Matrix and the Projection Matrix have been applied to it.
- **Computer Vision:** The field of Computer Science that deal with acquiring, processing, analyzing and understanding images.
- **Eye Space:** A geometry element is said to be in *eye space coordinates*, when the Model View Matrix has been applied to it.
- **Feature:** Property extracted over an object. Makes explicit certain information that was implicit in the original object.
- **Fragment Shader:** A program that will process the fragments, produced during the rasterization, into a set of colors and single depth values.
- **Frame:** Each of the images that compose an animation or an interactive visualization.
- **Frames per second (fps):** Frequency at which an imaging device produces consecutive frames.
- **Frustum View Culling:** Test that prevents rendering the geometry that is not contained inside the vie frustum.
- **GLM:** OpenGL Mathematics, library useful for easily performing operations with 3 or 4-dimensional matrices and vectors.
- **GPU:** Graphics Processing Unit, a purpose specific processor for graphic and floating point operations.
- **Kd-tree:** Data structure that segments the space using parallel plains to the coordinate axis. It is used to perform efficient proximty or fixed-radius searches.
- **Machine Learning:** The field of Computer Sciene that deals with the construction and study of systems that can learn from data.
- **Parallel Transport Frame:**
- **OpenGL:** Programming language specification that defines a multi-platform API for rendering 2D and 3D object graphics.
- **Orthophoto:** Photography of a terrain on which the perspective deformation has been corrected, obtaining an orthogonal projection.

- **Render:** Expression the refers to the process of generating an image. It is also used to describe the image resulting from this process.
- **Supervised Learning:** Machine learning procedure that learns from labeled data.
- **Vertex Shader:** Program that is ran once for each vertex. Its usual function is to transform the vertex 3D position into a 2D position in the screen.
- **World Space:** A geometry element is said to be in *world space coordinates*, when neither the Model View Matrix nor the Projection Matrix have been applied to it.

G User Manual

G.1 Tree and Shrub Generator

Before starting the program, the user should specify the desired parameter setting on the file *parameters.txt*. Once this is done, simply open the executable to start the program. It will then begin the generation of the 3D model and, after it is finished, the resulting model will be shown in the window, as can be seen in Figure G.1.

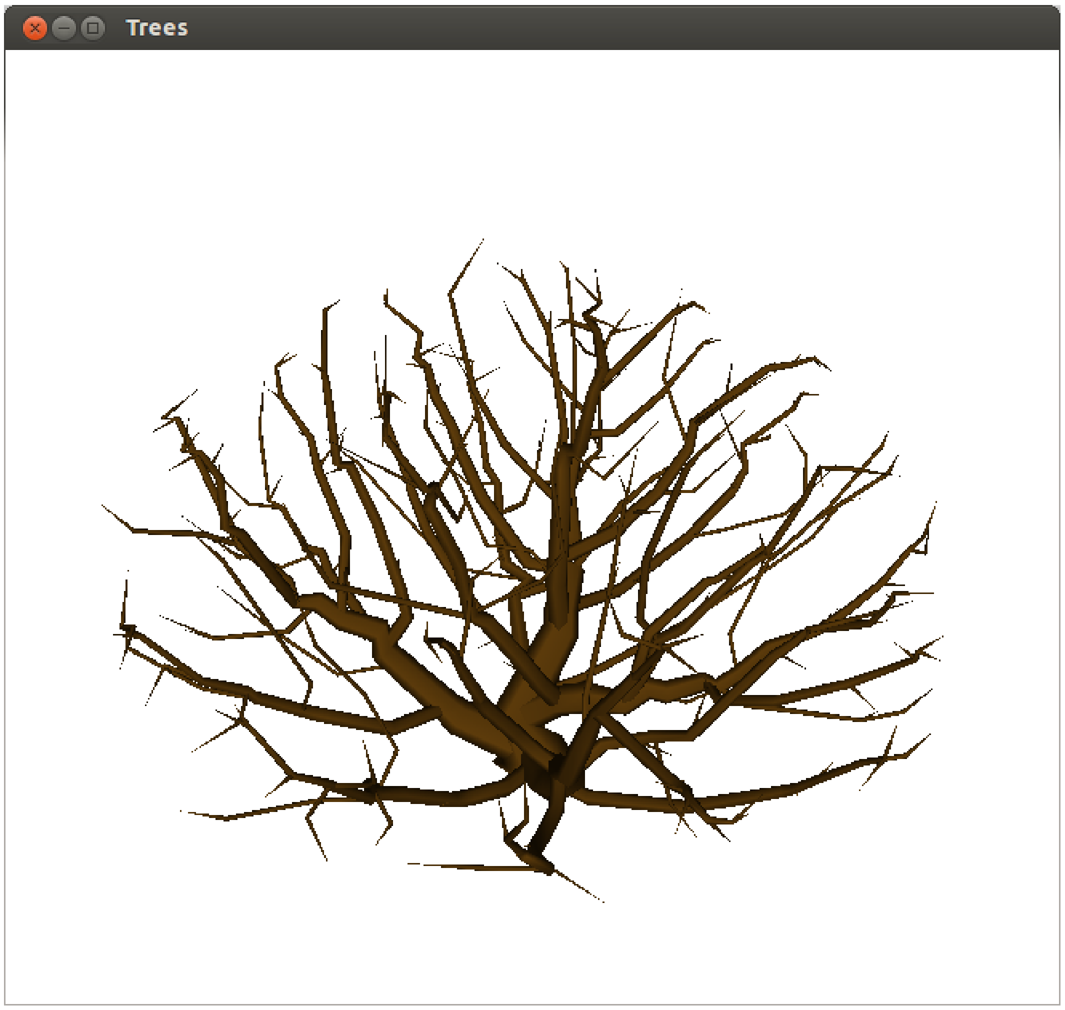

Figure G.1: Main window of the tree and shrub generator program.

If the window is focused, few actions can be performed to navigate through the scene:

- **Pushing the key H:** Show the high resolution model through the window.
- **Pushing the key M:** Show the medium resolution model through the window.

- **Pushing the key L:** Show the low resolution model through the window.
- **Pushing the key S:** Show the skeleton, of the high resolution model, through the window.
- **Pushing the key T:** Show the parallel transport frame axis, over the tree skeleton.
- **Pushing the key W:** Write the model to a file. If executed from a command line, the program will ask the user for the file name.
- **Pushing the mouse left button:** While keeping the button pushed, if the mouse is moved upward and downward, a rotation is performed about the X axis of the camera. If the mouse is moved from right to left, a rotation is performed about the Y axis of the camera.
- **Pushing the mouse right button:** While keeping the button pushed, if the mouse is moved upward and downward, a zoom towards or against the object is performed.

G.2 Visualizer

To start the program simply open the executable. It will then begin to load the 3D vegetation models, to compiled and link the shaders, to load the texture, to generate the billboards, etc. This process will take few seconds and, upon completion, the 3D landscape will begin to be shown through the window, as can be seen in Figure G.2.

If the window is focused, few actions can be performed to navigate through the scene:

- **Pushing the key A:** The camera will advance forward.
- **Pushing the key S:** The camera will advance backward.
- **Pushing the key D:** The camera will rotate to the right.
- **Pushing the key A:** The camera will rotate to the left.
- **Pushing the key Z:** The camera will advance upward.
- **Pushing the key W:** The camera will advance downward.
- **Pushing the key V:** Switches between the normal camera and the renderization of the view frustum.
- **Pushing the mouse left button:** While keeping the button pushed, if the mouse is moved upward and downward, a rotation is performed about the X axis of the camera. If the mouse is moved from right to left, a rotation is performed about the Y axis of the camera.

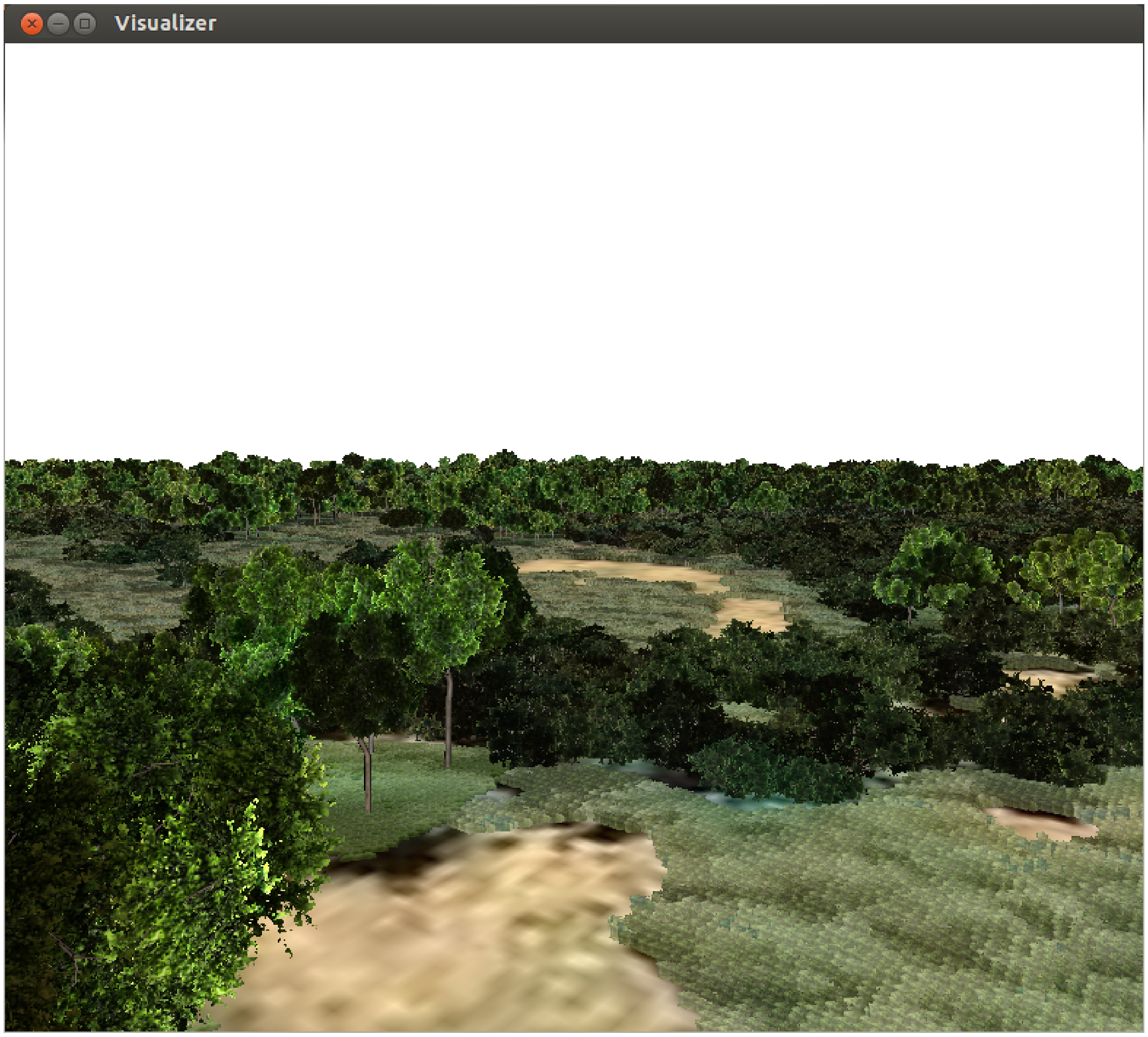

Figure G.2: Main window of the visualizer of the final application.