Realistic reconstruction and rendering of detailed 3D scenarios from multiple data sources

Doctoral Thesis

Óscar Argudo Medrano

Doctoral dissertation submitted for
the International Doctorate Mention

Universitat Politècnica de Catalunya

Department of Computer Science (CS)
PhD in Computing

Advisor: Carlos Andújar Gran
Co-Advisor: Antoni Chica Calaf

Barcelona, June 2018

The thing to be wished for is,
not that the mountains should become easier,
but than men should become wiser and stronger.

— EDWARD WHYMPER,
first climber of the Matterhorn

Abstract

During the last years, we have witnessed significant improvements in digital terrain modeling, mainly through photogrammetric techniques based on satellite and aerial photography, as well as laser scanning. These techniques allow the creation of Digital Elevation Models (DEM) and Digital Surface Models (DSM) that can be streamed over the network and explored through virtual globe applications like Google Earth or NASA WorldWind.

The resolution of these 3D scenes has improved noticeably in the last years, reaching in some urban areas resolutions up to 1m or less for DEM and buildings, and less than 10 cm per pixel in the associated aerial imagery. However, in rural, forest or mountainous areas, the typical resolution for elevation datasets ranges between 5 and 30 meters, and typical resolution of corresponding aerial photographs ranges between 25 cm to 1 m. This current level of detail is only sufficient for aerial points of view, but as the viewpoint approaches the surface the terrain loses its realistic appearance.

Another important aspect of current datasets is their cost of acquisition. Selected sites can be reconstructed at higher resolutions through multiple techniques, including terrestrial LIDAR – a technology that measures distances using the time of flight of a laser beam –, shape-from-motion and multi-view stereo. Acquiring the data usually involves a displacement to the site, mounting the sensors on some vehicle (like a car, or a plane for aerial views) and moving through the scene to be captured. In order to improve the level of detail, a new acquisition with more samples per surface area has to be carried out. This implies either using better equipment and/or performing a more in-depth scan of the area. The associated costs severely limit the scalability of this approach, preventing it to handle arbitrarily large areas, and still result in deeply incomplete models due to occlusions. Dense vegetation scenarios like forests are prone to numerous reconstruction artifacts due to their complex geometry, self-similar appearance and occlusions.

One approach to augment the detail on top of currently available datasets is adding synthetic details in a plausible manner, i.e. including elements that match the features perceived in the aerial view. By combining the real dataset with the instancing of models on the terrain and other procedural detail techniques, the effective resolution can potentially become arbitrary. There are several appli-

cations that do not need an exact reproduction of the real elements but would greatly benefit from plausibly enhanced terrain models: video games and entertainment applications, visual impact assessment (e.g. how a new ski resort would look), virtual tourism, simulations, etc.

In this thesis we propose new methods and tools to help the reconstruction and synthesis of high-resolution terrain scenes from currently available data sources, in order to achieve realistically looking ground-level views. In particular, we decided to focus on rural landscapes, mountains and forest areas.

Our main goal is the combination of plausible synthetic elements and procedural detail with publicly available real data to create detailed 3D scenes from existing locations. Our research has focused on the following contributions:

- An efficient pipeline for aerial imagery segmentation
- Plausible terrain enhancement from high-resolution examples
- Super-resolution of DEM by transferring details from the aerial photograph
- Synthesis of arbitrary tree picture variations from a reduced set of photographs
- Reconstruction of 3D tree models from a single image
- A compact and efficient tree representation for real-time rendering of forest landscapes

Acknowledgments

"It is always further than it looks, taller than it looks, and harder than it looks". Three principles of mountaineering not meant as discouragement, but as appreciation for any progress made and as motivation to keep pushing towards the summit. And like any expedition, this thesis would not have been possible without the many wonderful people I have met along the path.

First of all, I want to express my deepest gratitude to my advisors: Carlos and Toni. They are the best guides I could have hoped for. They engaged me on this fantastic trip, have kept me motivated, and only thanks to their knowledge and encouragement I have been able to succeed. Special thanks to Isabel as well, for her continuous support to everybody in the group. Six years ago I was feeling lost, but she oriented me with valuable advice: "relax, enjoy this summer, and only afterwards make a decision about enrolling in the master's degree". And now, I could not be happier about the path I chose!

I also want to thank all the members and professors of the ViRVIG group, from whom I have learned so much, and all the people working at CRV, current and past, for the countless hours of lunch talks, coffee, games, and laughs. Sometimes it helps to get lost for a while in the forest of procrastination.

My most sincere gratitude also to Eric Guérin and Eric Galin. For hosting my research stay and helping me become a better researcher. To all my other coauthors, thank you as well, as you have been part of the journey too. Without you, this work would not be the same.

Finally, infinite thanks to Sílvia for always being there, walking along with me, even when I do not follow the easiest route.

I would like to dedicate this work to all my friends, specially those from VGAFIB – with you I first discovered the world of Computer Graphics – and my mountaineering pals – hopefully our passion for nature has been reflected in the results. And to my family, because it is not always easy to understand what I do or why.

When you finally get to the summit, you take a deep breath and pause to enjoy the views. And by doing so, you immediately start dreaming of higher and more challenging peaks.

This thesis has been funded by the Spanish Ministry of Education, Culture and Sports PhD Grant FPU13/01079, and by the Spanish Ministry of Economy and Competitiveness and FEDER Grants TIN2014-52211-C2-1-R and TIN2017-88515-C2-1-R.

Contents

1 Introduction

During the last years, we have witnessed significant improvements in digital terrain modeling, mainly through photogrammetric techniques based on satellite and aerial photography, as well as laser scanning. These techniques allow the creation of Digital Elevation Models (DEM) and Digital Surface Models (DSM) that can be streamed over the network and explored through virtual globe applications like Google Earth [9] or NASA WorldWind [18].

The resolution of these models has improved noticeably in the last years, reaching in some places of urban areas resolutions up to 1 m for the DEM and less than 10 cm per pixel in the aerial imagery. However, in rural and forest areas, typical DEM resolutions range between 5 and 30 meters and from 25 cm to 1 m for the corresponding imagery. This current level of detail is only sufficient for aerial points of view; as the viewpoint approaches the surface the terrain loses its realistic appearance. On a global scale, the best available datasets are the ones obtained by NASA Shuttle Radar Topography Mission (SRTM) [24], which provide a resolution of 1 arc-second (30 m). However, this dataset contains voids at various desert and mountain regions - including all 14 peaks over 8000 m and most of the peaks over 7000 m of the Himalayas. Therefore, applications like Google Earth or WorldWind integrate data from various local sources to improve details.

Another important aspect of current datasets is their cost of acquisition. Selected sites can be reconstructed at higher resolutions through multiple techniques including terrestrial LIDAR – a technology that measures distances using the time of flight of a laser beam –, shape-from-motion and multi-view stereo. Acquiring the data usually involves a displacement to the site, mounting the sen-

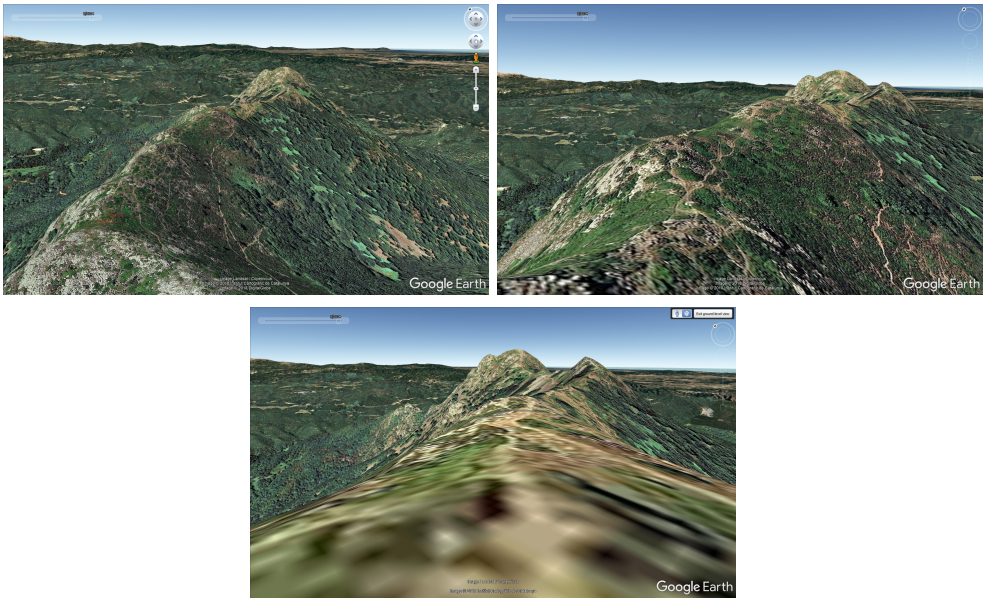


Figure 1.1: Far aerial, medium distance aerial and ground-level views of Les Agudes peak as seen in Google Earth, using the 25 cm orthophoto by ICGC.

sors on some vehicle (a car or a plane for aerial views) and moving through the scene to be captured. In order to improve the level of detail, a new acquisition with more samples per surface area has to be carried out. This implies either using better equipment and/or performing a more in-depth scan of the area. The associated costs severely limit the scalability of this approach, preventing it to handle arbitrarily large areas, and still result in deeply incomplete models due to occlusions. Dense vegetation scenarios like forests are prone to numerous reconstruction artifacts due to their extremely complex geometry, self-similar appearance and occlusions.

As a consequence, rural areas and mountainous regions usually lack enough resolution for close views. Even 25 cm per pixel is too low for ground-level views, as exemplified in Figure 1.1. Note how easily identifiable trees and rocks in aerial views become blurry spots as one zooms into the terrain, complicating the perception of the natural elements around. Even in the medium distance aerial view, the texture on the closest areas starts to become fuzzy. Figure 1.2 compares a real photograph with the rendered ground-level view on the same spot using a 5 m DEM and 25 cm/pixel aerial image.

An example of a well known technology that enables ground-level views is Google Street View [10]: it allows detailed inspection - in available areas - from surface points of views by using a fixed set of 360°panoramic photographs. These

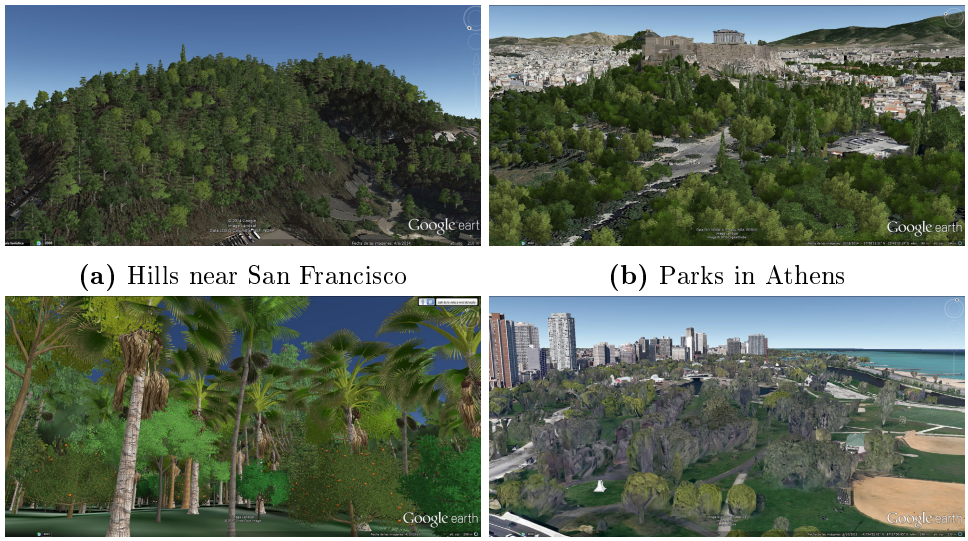
Figure 1.2: Pedraforca summit: comparison between real photograph and ground-level view as seen in Google Earth.

viewpoints are obtained using a camera set mounted on a vehicle, but some trekking routes and natural parks have also been mapped using a backpack with the camera. Recent popularization of head-mounted displays and smartphone-compatible 360-cameras is boosting the amount of available spherical panoramas. A remarkable example is Project360 [14] by the outdoor sports company Mammut, which hired experienced alpinists and created sets of panoramas in famous climbing routes like Mont Blanc, Everest or El Capitan (Yosemite). However, in this kind of applications the navigation is constrained to jumping between fixed positions and does not provide a smooth interpolation between viewpoints – although there is ongoing research addressing this issue. Moreover, each panorama is an independent image that has to be sent to the user, so the amount of redundant data to stream can be huge.

Another approach is to augment the detail on top of currently available datasets in a plausible manner, i.e. including synthetic elements that match the features perceived in the aerial view. By combining the real dataset with the instancing of models on the terrain and other procedural detail techniques, the effective resolution can potentially become arbitrary. There are several applications that do not need an exact reproduction of the real elements but would greatly benefit from plausibly enhanced terrain models: videogames and entertainment applications, visual impact assessment (e.g. how a new ski resort would look), virtual tourism, simulations, etc.

Some virtual globe applications are already applying this strategy. For example, Google Earth [9] includes a 3D tree view for certain locations. Figures 1.3a and 1.3b show trees rendered using a billboard for each species. The repetition and planar nature of the billboards is clearly evident. Moreover, for viewing angles more perpendicular to the terrain, the billboards are automatically disabled. Note also in 1.3b how some trees appear at locations where one would not expect them, as well as identifiable trees in the aerial image not appearing in

3D. For ground-level views, like in Figure 1.3c, 3D models improve the realism but there is still lack of shading and the terrain texture is still the same photograph used in aerial views, so it is basically showing the pictured canopy on the ground. Finally, Figure 1.3d shows polygonal models for trees (and buildings) which have been probably obtained from a laser-scanned point cloud or a set of oblique photographs.



(a) Hills near San Francisco **(b)** Parks in Athens

(c) Surui Forest, Brazilian Amazon **(d)** Lincoln Park, Chicago

Figure 1.3: Screenshots from Google Earth.

1.1 Contributions

The goal of this thesis is the reconstruction and synthesis of high-resolution terrain landscapes from existing locations, suitable for realistically looking ground-level navigation. In particular, we have focused on natural landscapes such as mountainous terrain and large areas of forest. We propose new methods and tools to leverage currently available public datasets and help the reconstruction and synthesis of detailed 3D models of these locations, by combining plausible synthetic elements and procedural details with the real data.

Here we summarize our contributions, which we will later see in depth in chapters 4, 5 and 6. Figure 1.4 shows the relationship between the different sections and how they combine towards the main goal.

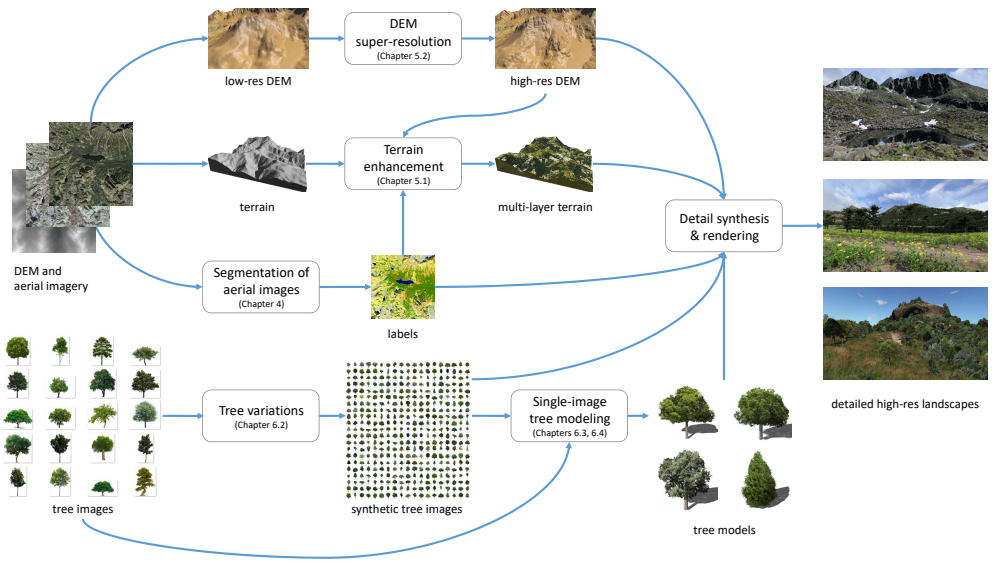


Figure 1.4: Overview and relationship between our contributions.

- **A complete and efficient pipeline for the segmentation of aerial imagery**

  Placing synthetic elements onto a terrain set in a plausible manner requires first identifying where each kind of element should be placed according to its high resolution aerial photograph. In our case, categories may not be known in advance – different terrains could contain different categories – or they may even depend on artistic or technical criteria – for example, a category for ground might be specialized into sand or rock and use different rendering techniques. Therefore, it is unfeasible to rely on predefined training sets. We compared several feature sets and classification algorithms in order to build a segmentation pipeline that can be trained fast and using only a few examples per label, while still yielding good accuracy rates. We also provided some insights on how to build this training set for best performance, as well as some post-processing tools that can be helpful to users such as filtering or reclassifying according to a perceptual metric. Finally, we validated the usefulness of our proposed per-pixel segmentation by comparing an enhanced terrain using either our segmentation or the current official land-cover map.

- **Plausible terrain enhancement from high-resolution examples**

  One strategy to increase detail on low-resolution elevation models is to replace parts of it with other patches that have similar features taken from

a high resolution dataset. We improved an existing technique based on this strategy in order to not only replace the high resolution elevation but also coherently transfer any other additional information layers the source dataset may have, like vegetation cover, population densities, or erosion sediments.

- **Super-resolution of DEM by transferring details from the aerial photograph**

  As we have seen in the previous section, current high resolution aerial images are in the order of 25 or 50 centimeters per pixel, while DEM are rarely found at 2 m grid resolution or less, being 5 to 15 meters the usual resolution range. We demonstrated it is possible to infer additional details on DEM from the aerial image through the use of a Convolutional Neural Network. Our results indicate that the neural network is able to double the effective resolution of a DEM, and the participants in our user study perceived the resulting terrain as being a faithful high resolution representation.

- **Synthesis of arbitrary tree picture variations from a reduced set of photographs**

  Billboards are a common technique for including vegetation on a terrain, and using photographs from local species can contribute to the plausibility of the resulting scene. Nevertheless, as soon as the users see many repetitions of the exact same picture, the scene looks very unnatural. We proposed a method that is able to generate new tree images by interpolating between samples from a small set of tree photographs and synthesize variations of them. This way, we make possible to populate an entire scene without the user noticing repetitions, thus improving the realism.

- **Reconstruction of 3D tree models from a single image**

  The principal limitations of using billboards for the trees are given by the static nature of a flat image: it will always look the same from any direction, it does not receive proper shadows, and it does not easily allow for animation of external effects like wind, among others. We proposed an algorithm that reconstructs a full 3D tree model from a single tree photograph at different levels of detail: a relief approximation of the tree crown, a set of branchlet billboards, and a highly detailed branching structure up to the leaves level.

- **A compact and efficient tree representation for real-time rendering**

  A forest terrain of moderate size can contain tens of thousands of trees, which poses a challenge for rendering this scene in real time for free navigation applications. We introduced a compact multi-resolution tree model representation using different Levels of Detail that allow efficient rendering from arbitrary directions as well as realistic lighting and shading effects.

1.2 Document outline

This thesis manuscript has been structured as follows. In this chapter we have introduced the problem, motivation and our goals. Chapter 2 serves as a brief introduction to the different types of terrain data and small survey of available datasets. Chapter 3 surveys related work on the various areas covered: aerial image segmentation, terrain modeling and vegetation synthesis. Then, the next chapters explain our work and contributions following the possible order of a terrain enhancement pipeline: we start with the segmentation of the aerial image (Chapter 4), then we improve the detail of the terrain and add additional information like vegetation density layers (Chapter 5), and finally we synthesize plausible vegetation that resembles local species (Chapter 6). Finally, Chapter 7 summarizes the different contributions and outlines ideas of future research.

2

GIS Preliminaries

The Encyclopedia Britannica defines a *Geographic Information System* (GIS) as a "computer system for performing geographical analysis", and includes in the definition all the components used to recollect, organize, manipulate, query, visualize, and interact with geographical data. In this thesis, we will tackle visualization and interaction from a Computer Graphics perspective. This chapter provides a brief introduction to the concepts from GIS and types of data that we will use in the text. Moreover, since the development and applicability of our project relies partially on the public availability of terrain data – mainly elevation and aerial imagery – we also briefly report some of the sources we are aware of or we have used during the development of our work.

2.1 Elevation

The elevation of a point corresponds to its height with respect to another reference point, for example the mean sea level. Typically, elevation is represented using topographical maps that show the contour lines alongside some color or patterns palette that depends on the type of terrain or intention of the map. However, in digital systems, elevation data is commonly provided as a uniform raster grid or heightmap: each point/cell contains an elevation value, and the horizontal spacing between neighboring cells represents the spatial or horizontal resolution of the dataset. This resolution should not be confused with the vertical accuracy: in Catalonia, the 2 m elevation grid provided by ICGC [12] has a vertical mean square error of 0.15 m.

Usually, the name Digital Elevation Model (DEM) is applied to any elevation raster grid. The term Digital Surface Model (DSM) is used when the elevation values represent the top-most layer of the terrain, including vegetation, buildings, bridges or roads. If the elevation data corresponds to the bare-ground terrain level, it is common to define the dataset as a Digital Terrain Model (DTM). In some places, the terms DEM and DTM are used as synonyms, while DSM is commonly reserved for those datasets that include vegetation and objects.

Elevation data is currently obtained mainly through remote sensing approaches like photogrammetry, synthetic-aperture radar (SAR) or LiDAR, although direct land surveyed data can also be used. For example, the 5 m elevation grid provided by ICGC has been generated through stereo pair photogrammetry of aerial images to obtain a dense set of 3D points, but also combines surveyed positions, level curves and other known profiles from a vector data topographic base. Then, all these points are combined and triangulated to generate a mesh, and finally the grid is obtained by sampling and interpolating the grid positions on this mesh surface. The more recent 2 m elevation grid dataset has been generated from a LiDAR point cloud dataset, keeping only the points labeled as ground, adding breaklines and contours to ill-sampled areas, and triangulating the points to produce the mesh.

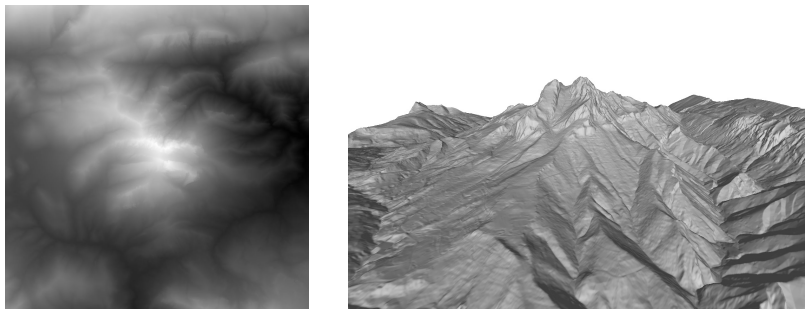


Figure 2.1: Elevation grid (normalized for visualization) and 3D render of Pedraforca mountain, using the 5 m resolution DTM from ICGC.

Datasets

In 1996, the U.S. Geological Survey (USGS) completed a topographic elevation model called GTOPO30 [11] that provides global coverage using a grid resolution of 30 arcseconds, approximately 1km at the equator. This set was obtained by combining information from several sources. Later, USGS and NGA developed an enhanced model to replace GTOPO30 named GMTED2010, Global Multi-resolution Terrain Elevation Data 2010 [8]. It provides elevations at 30, 15 and 7.5 arcseconds spatial resolution (1000, 500 and 250 m) and incorporates the best available global elevation data.

The ASTER Global Digital Elevation Model [1] was developed jointly by NASA and Japan's Ministry of Economy, Trade, and Industry. The first version was released in 2009, and an improved second version in 2011. The ASTER GDEM covers land surfaces between 83° N and 83° S - which encompass 99% of Earth's land - at 1 arcsecond resolution, but it is advised that the data may contain anomalies or artifacts that can reduce its usability for certain applications, and should be regarded as "'experimental or research grade"'.

One of the most widely used datasets comes from the Shuttle Radar Topography Mission (SRTM), which obtained digital elevation between latitudes 60° N and 54° S at a raw resolution of 1 arcsecond (30 m) [24]. Initially, SRTM Digital Terrain Elevation Data was publicly distributed at 3 arcsecond resolution (90 m) for the latitudes covered, and at 1 arcsecond in the United States. The original datasets contained many voids and single pixel errors in the form of pits and bumps. Therefore, some researchers have worked on filling the void either with interpolation or by combining sources. While the no-data areas only amount to 0.2% of the surveyed area, it is a problem in mountainous areas, gorges and canyons. For instance, all 8,000 m mountains and most of the highest summits in all mountain ranges were affected. SRTM version 3, released in 2013, filled all voids primarily from ASTER GDEM2 but also using GMTED2010 and other sources. Since 2014, the full dataset at 1 arcsecond is also publicly available.

A highly detailed, globally available and consistent dataset is still missing nowadays. However, it is possible to find middle and high resolution DEM at country or region level. These datasets are usually provided by the respective geographical or geological service of the area, and it may be difficult to find them as their webpages are often written in the official language of each region.

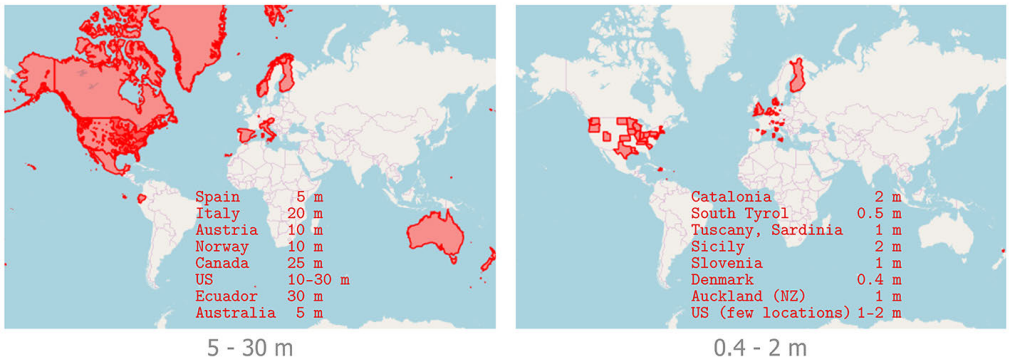


Figure 2.2: OpenDEM availability maps for middle resolution (left) and high resolution (right) elevation datasets as of June 2018, and non-exhaustive listing of the available resolution on sample regions.

OpenDEM Searcher [19] maintains a list of publicly available elevation sets at middle (2-30 m) and high ($\leq$2 m) resolution, shown in Figure 2.2. For example, as of June 2018, the USGS 3D Elevation Program Datasets provide 1/3 arcsecond (10 m) nationally in the US, but only a few areas have 1/9 arc-second (3 m) and even fewer at 1 m. In Europe, some high resolution datasets covering mountainous areas can be obtained from Spain (Basque Country and Catalonia), Italy (Tuscany, Sardinia, Sicily and South Tyrol), Switzerland, Slovenia, England or Wales.

In Spain, the *Instituto Geográfico Nacional* (IGN) [13] provides Web Map Services to download datasets. Currently, they provide 1000, 200, 25 and 5 m grid resolution. Similarly, in Catalonia, *Institut Cartogràfic i Geològic de Catalunya* (ICGC) [12] provides WMS to access Digital Elevation Models at 15, 5 and 2 m resolution. The first two were acquired using photogrammetry techniques, the more recent 2 m dataset has been derived from LIDAR point clouds.

2.2 Aerial imagery

Aerial imagery is usually provided as orthophotographs or orthorectified photographs. The image distortions introduced by the lenses, camera parameters and terrain morphology are corrected in order to simulate an orthographic projection and uniform scale. This rectification process needs the DEM or topographical representation of the underlying terrain. The resulting orthoimage has the expected properties of a map, and can be used to measure planar distances. Therefore, we can define their resolution as the ground distance between two neighboring pixels.

There are also different imagery products depending on the bands (light wavelengths) they represent. For satellite images, it is common to find multispectral images containing from 3 to 15 different bands, or panchromatic images that accumulate the response among several wavelengths and can potentially provide higher resolutions. On the other hand, aerial photography taken from planes is normally distributed in the visible spectrum as RGB bands, sometimes including the Near Infrared channel as well.

Datasets

It is easy to explore and download low resolution satellite imagery, even in nearly real time - within a few hours after the satellite took them. For example, NASA provides the Global Imagery Browse Service [6] with access to multiband imagery

from Terra, Aqua or Landsat satellites among others. The resolution, however, is very coarse at 250 m/pixel for the first two and about 30 m for Landsat.

Private enterprises do own and sometimes sell higher resolution datasets. For example, DigitalGlobe uses its own constellation of satellites and sells high resolution imagery from 30 to 60 cm per pixel in the region of interest. The company ESRI provides a layer for ArcGIS called *World Imagery* which combines the 15 m resolution images from TerraColor with high resolution datasets from several parts of the world where available, like 30 cm for the US, 60 cm for parts of Western Europe, or 1 m in some other regions of the world. This layer even contains examples of very high resolution sets from 10 cm pixel down to 3 cm pixel for very specific locations. Similarly, Google Earth shows aerial imagery from 15 m to 15 cm.

Regional services, again, can provide much higher resolution. For example, in the US there is full aerial imagery coverage at 1 m resolution through the National Agriculture Imagery Program (NAIP) dataset [17], which is also updated yearly. In Spain, IGN provides via Web Map Services aerial imagery at 50 and 25 cm per pixel, and it is planned to offer 10 cm in the future as proposed in the Plan Nacional de Ortofotografía Aérea [21]. In Catalonia, ICGC provides web map services to access current or historical aerial images at 50 and 25 cm per pixel,

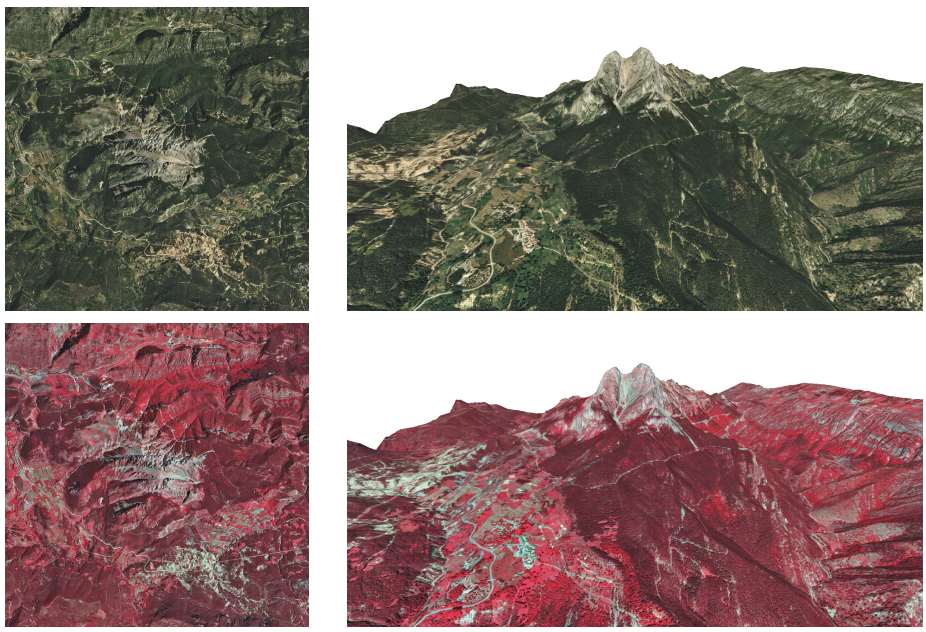


Figure 2.3: Aerial image and textured DEM of Pedraforca mountain. 25 cm RGB and NIR-R-G orthophotos from ICGC.

and they are gradually updating the 10 cm/pixel imagery service; as of January 2018, it covers the full coastline plus the metropolitan area of Barcelona and a handful of county capital cities.

2.3 LiDAR point clouds

LiDAR scanners use pulses of laser light to measure distances by measuring the changes in the return times and wavelength of the reflected pulse. One of the major advantages of LiDAR is that it is possible to record multiple returns (echoes) of the light pulses, so in a forest area it might be possible to obtain different bounces of a pulse penetrating the canopy cover before it finds the ground. This way, it is possible to filter vegetation and ground points and build a DTM or DSM from the same dataset. Currently, airborne LiDAR is considered as one of the best and most precise methods for obtaining elevation data.

LiDAR information is distributed as a point cloud, usually in LAS format. Alongside the 3D position, each point can have additional information fields like the intensity of the return pulse, the return number and total number of returns of the emitted pulse, scan direction, and even a standard classification into a handful of classes including ground, low/medium/high vegetation, building, water or noise. RGB color, when provided, is normally obtained from calibrated photographs taken during the same flight.

More recently, there has been a growing interest towards not only recording the discrete set of returns of the pulse but a continuous wave, called full-waveform

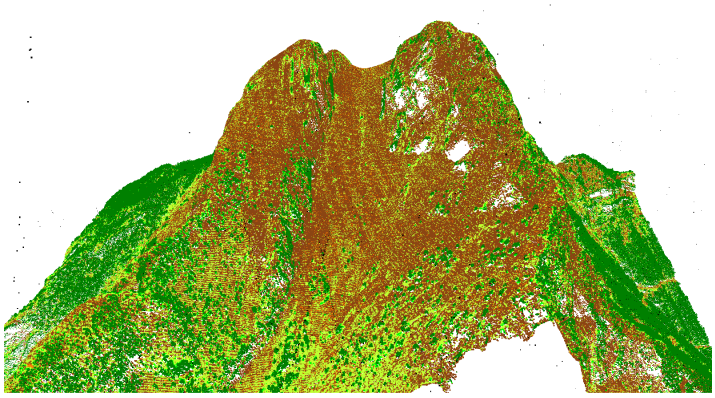


Figure 2.4: Class-colored LiDAR point cloud around Pedraforca Mountain, with an average density of 2.1 points/m^2.

LiDAR. This allows for studying not only geometric but also radiometric properties of the object, and statistical analysis of these features has been shown to be useful for point classification [RBJ+14].

Datasets

Due to the large size of point clouds, LiDAR data is usually found at local or regional scales. The website OpenTopography [20] maintains a list of point cloud datasets. In the US, there are local datasets with resolutions ranging between 10 and 20 points per square meter. In the Italian South Tyrol region, there is a dataset covering the bottom of the valleys with at least 4 points/m^2 [22].

A LiDAR dataset covering all Spain is being developed, with a first pass ensuring a density of 0.5 points/m^2 and a second pass at 1 point/m^2. In Catalonia, ICGC [12] currently provides full LiDAR coverage with varying resolutions between 0.5 to 4.4 points/m^2.

2.4 Land Cover

Land Cover maps provide a segmentation of the terrain into regions depending on the material or elements on the surface, i.e. vegetation, rocks, water, roads, etc. The defined classes usually depend on the dataset and region, but it is common to find several types of forest cover, agricultural areas, and urban constructions. It is also possible to define a label hierarchy, thus allowing the user to choose the desired level of detail. For example, from general to specific, a region could be labeled as: forest, dense trees, conifers, mountain pine (*pinus mugo*). These maps are a valuable tool in monitoring earth resources and human activity impact.

The main difficulty for creating land cover datasets is their cost of acquisition and maintenance. Usually, creating such a map is done via human interpretation of the aerial or satellite imagery: experts usually identify homogeneous regions in the images, delineate them using selection tools, and assign them a label that is often verified using other existing catalogs or even on-site survey.

While on a large scale significant changes on a terrain might take several years, on a local scale the cover is easily affected by human activities and natural events (landslides, fires), thus rendering a map outdated. There have been efforts in crowd-sourcing the creation of Land Cover maps, like Geo-Wiki [5] platform or the Landspotting project [SWP+13], which integrated aerial image labeling into different online multi-player game mechanics.

Datasets

Despite the difficulty on generating such maps, there are some world-wide cover maps usually derived automatically from satellite imagery. The Global Land Cover 2000 [7] provides a world-wide segmentation of the images from the SPOT 4 satellite of the year 2000 into 21 classes – which are later specialized on a region basis – and using a 1 km resolution. MODIS land cover [16] uses supervised segmentation on the MODIS satellite imagery at 500 m resolution from NASA's Terra satellite to provide 16-class land cover maps at annual time steps since 2001. GlobCover [3] uses the 2009 MERIS sensor images from ENVISAT satellite mission, and has a spatial resolution of 300 m.

Higher resolutions are available for more regional areas. The most recent edition (2012) of the European CORINE Land Cover inventory [2] maps all 39 member countries of the European Environment Agency with at least 100 m resolution. In Spain, the SIOSE portal [23] provides national coverage maps using polygonal regions with at least 15 m width. In Catalonia, the fourth version of the *Mapa de Cobertes de Sòl de Catalunya* (MCSC) [15] provides a hierarchical classification into up to 241 categories, using polygons with minimum width of 10 m, and distributed as a 5 m raster grid as well.

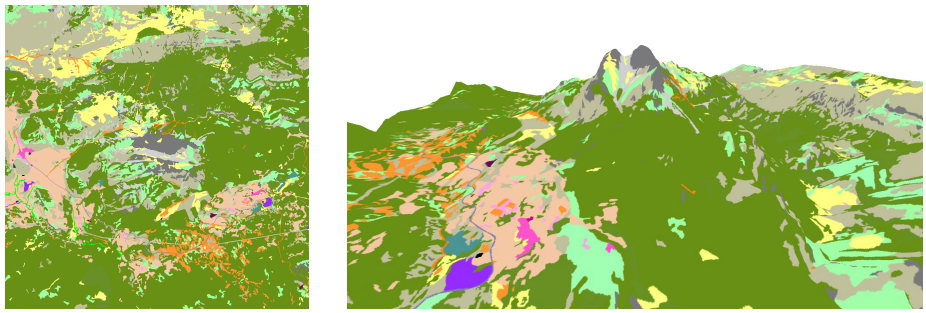


Figure 2.5: Land cover map of Pedraforca mountain, from MCSC.

2.5 Other information layers

There are many other types of geographical information layers available depending on the region and provider. From geological maps with information about the soil type, to biophysical variables in a forest – like biomass index, average tree diameter or estimated height –, or even historical and yearly registers of aerial imagery.

3

State of the art

In this chapter we will review the state of the art on the topics related to the goals of the thesis. We have identified three major areas of study that we will detail in the next sections. First, the segmentation of aerial images into a set of labels, which has been mainly addressed from machine learning and computer vision fields. Second, the enhancement of details on terrains, mainly through procedural modeling methods that take inspiration from geological phenomena. Finally, modeling and rendering vegetation combines computer graphics and computer vision with biology.

3.1 Segmentation of aerial images

Synthesizing plausible details onto terrains, and providing a realistic and smooth transition between the orthophoto and the added details, requires a segmentation of this aerial image in order to identify the different elements it contains. For some years now, segmentation tasks have been almost always undertaken with machine learning approaches. Typically, a set of feature descriptors is obtained for each pixel on the image, and then some algorithm learns the mapping from this feature space into the set of segmentation categories or classes. In this section, we will explain relevant works on both feature descriptors and learning algorithms, focusing on those aimed to classification or segmentation of aerial images.

3.1.1 Descriptors

Descriptors or features are n-dimensional vectors representing measurable characteristics about a pixel – and sometimes neighboring region – that will help the segmentation algorithm in discriminating between the different classes. There has been extensive research on proposing novel features and comparing their performance with existing ones. While recent deep learning end-to-end approaches have eliminated the need for the design of features, each layer of such networks can still be seen as a set of features computed over the ones produced by the previous layer, so the first layer is actually a set of image features. Moreover, extensive training sets are required in order to correctly train a deep network, while hand-crafted descriptors can potentially lead to similar results from much smaller training sets.

A survey about natural image descriptors and statistics is provided by Pouli et al. [TWE11], and they classify statistics into first, second and higher order. First order statistics are those computed over individual pixels, for example the first four moments of the intensities histogram: mean, variance, skewness (asymmetry) and kurtosis (sharpness of the distribution peak). While these statistics are easy to compute, they do not provide information about spatial relationships and structure between pixels.

Second order statistics are those computed over pairs of pixels, and the two main statistics of this type are gradients and power spectra. Gradients are a discrete approximation of the (oriented) derivative of the pixel intensity, and histograms of oriented edges have been used to discriminate between image categories. The power spectrum, computed through Fourier analysis, provides information about the different spatial frequencies on an image. It has been found that the average spectra over several images and orientations follows a power law: $A = 1/f^{\beta}$, where β is called the spectral slope and the amplitude A decreases with respect to frequency f. Torralba and Oliva [TO03] showed that the spectral signature of images is correlated with the category as well as scale (Figure 3.1): images of man-made environments tend to show more vertical and horizontal edges while natural images are more isotropic and denser in high frequencies, but on large-scale natural scene shots the horizon dominates and textures appear smoother.

Similarly, patterns and regularities for natural images have also been found on higher order statistics, i.e. over larger groups of pixels. Gradient statistics can be computed hierarchically using Wavelet transforms, thus providing analysis on local regions with different orientations and frequencies instead of computing them over the whole image as the Fourier transform does. Moreover, much of

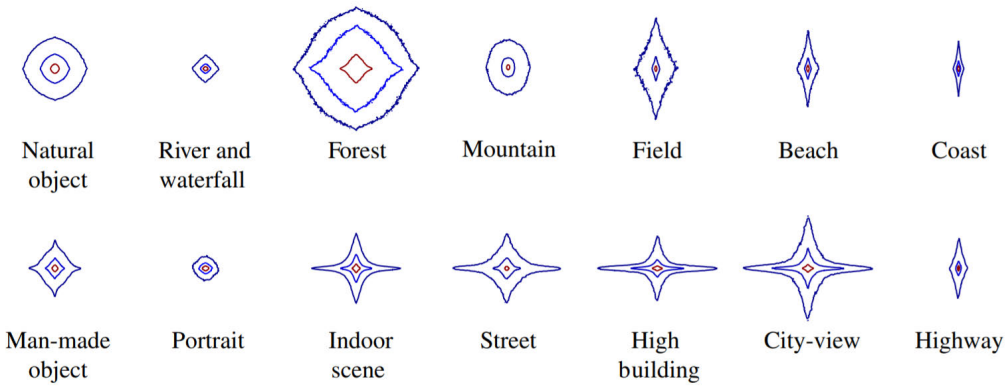


Figure 3.1: Spectral signatures of different scene categories obtained from the average of several hundred images in each case. Plot axes correspond to f_x and f_x, and the curves represent 60%, 80% and 90% of the energy of the spectral signature. Image from [TO03]

the structure of an image has been shown to be located in phase spectrum, rather than power spectrum, and second order statistics are insensitive to phase. Principal Component Analysis and Independent Component Analysis have also been used to study the sources of commonality and variations between images.

Ruiz et al. [RFR04] compared different texture and spectral feature descriptors sets for pixel classification of remote sensing images, in particular statistical features extracted from the Grey Level Concurrence Matrix (GLCM), energy filters, Gabor filters and wavelet transform features. They compared the accuracies obtained on three forest scenes and a urban area using a maximum likelihood classifier, and concluded that there is no universal criteria - the suitable set of features depends on the type of landscape units defined in each application.

Similarly, dos Santos et al. [SPd10] also compared the effectiveness of various color and texture descriptors for remote sensing image classification and retrieval. Although their task was not pixel based, the best descriptors were also dependent on the type of input dataset.

Tokarczyk et al. [TMS12] used Random Forests [Bre01] on high-resolution aerial images to evaluate different feature sets: raw intensities, pixel neighbourhood blocks, texture filters, PCA bases, and the first three layers of a trained deep network used as filter banks. Their results show that features based on patches dominate those based on individual pixels, i.e. texture holds important information in high-resolution images. However, complex feature extraction methods or even non-linear feature learning yield small or no improvement, while adding a significant computation cost.

Cheriyadat [Che14] proposed a minimization problem to automatically find the best reduced set of basis functions that sparsely encode the original feature vectors extracted from image patches raw pixel values, filter banks and texture descriptors. The feature vectors are then projected onto the basis function set and pooled to obtain the final patch representation fed to a linear SVM classifier.

Penatti and dos Santos [PNS15] studied whether Convolutional Neural Networks trained for classification of everyday objects would generalize to aerial and remote sensing images. They used the output of the last fully-connected layer of two networks: OverFeat [SEZ+13] and CaffeNet [JSD+14], and used it as input of a linear SVM for image classification. For comparison, they also trained the SVM using low-level feature descriptors and bags of visual words. On the aerial dataset, deep features achieved the best results, but for remote sensing images they were outperformed by the low-level descriptors.

3.1.2 Algorithms

The most basic image segmentation algorithms are based on unsupervised learning approaches like thresholding, clustering (e.g. k-means) or region growing algorithms, and produce a set of k clusters (on the feature space) for which a label is manually assigned afterwards. Therefore, semantic segmentation or classification is commonly treated as a supervised learning problem, with a given training set and known labels. Given the feature vectors for the pixels in the images, we can potentially train and use any of the several classification algorithms existing in the machine learning field: Naive Bayes, logistic regression, k-nearest neighbors, support vector machines, decision trees, neural networks, etc.

The performance of different algorithms depends on the data and problem. Therefore, we will describe in more detail and compare several classification algorithms in Chapter 4.2.3.

Multiple works also compare the accuracy of classification algorithms in the context of segmentation of aerial images, including Bayesian classifiers [Aks08], Random Forests [TMS12], SVM [Che14; PNS15] and Deep Learning methods [CPS+15].

Aksoy [Aks08] proposed a two-step algorithm for classification of hyperspectral images of urban areas. First, a Bayesian classifier is trained using per-pixel spectral and texture features to obtain the probability maps for each class, and each pixel is assigned its most probable class. Then, an iterative split-and-merge algorithm converts the pixel class map into contiguous regions. The final classification is obtained using again Bayesian classifiers, but this time trained with region-level statistics and features.

Frölich et al. presented Iterative Context Forests [FRD13], a classification system based on Random Forests that builds the trees level-wise and adds the class probability maps of one level as a new input channel for the next level (Figure 3.2). A huge randomized set of features is computed from the input channels by using diverse operators on pairs of pixels or rectangular regions. In [FBW+13], they apply this algorithm on remote sensing data.

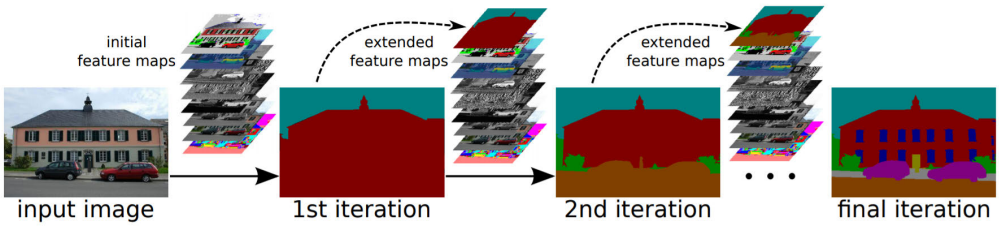


Figure 3.2: Iterative Context Forests, image from [FRD13]

A similar approach by Tokarczyk et al. [TWW+15] uses boosting instead of Random Forests. Instead of selecting features, they construct a vast and redundant set of simple features, and use them as input to a multiclass extension of AdaBoost [SS99]. Their best results are achieved with four-leaf decision trees as weak learners. Training on the set of Randomized Quasi Exhaustive (RQE) features outperforms predefined filter banks, but accuracies are very similar to the ones from raw pixel intensities in a 15×15 neighborhood in conjunction with the Normalized Difference Vegetation Index (NDVI) [KMN+69; Roe07], being the latter preferable to avoid the high memory and runtime requirements during the training with RQE.

Recent works on aerial scene classification and image retrieval have studied unsupervised feature learning approaches. Castelluccio et al. [CPS+15] compared three different options for classification with Convolutional Neural Networks: training the network from scratch, fine-tuning the last layers of the network using training images, or using the last fully-connected layer as a feature vector as in [PNS15]. They performed experiments with CaffeNet [JSD+14] and GoogLeNet [SLJ+14] networks. On the aerial images dataset, the best accuracies were achieved by fine-tuning, then using the feature vector, and finally by training from scratch. On the remote sensing dataset, training from scratch or fine-tuning provided the best results depending on the network used.

3.1.3 Filtering

Keeping the class with maximum a posteriori probability as the final pixel class can result in noisy labelings. The smoothness assumption states that nearby pixels tend to have similar labels, thus individual pixels are not independent variables but form a random field.

Conditional Random Fields [LMP01] are a kind of probabilistic graphical models that have been used, among other applications, in image segmentation either as a classifier or as a post-processing operator to improve the result of a segmentation. They usually include a unary energy term defined for a pixel or region, and pairwise terms defined for pairs of pixels or regions. When applying a CRF as a post-process, the unary term is usually the result of a pixel classifier, and pairwise terms try to enforce similar labels between neighboring pixels. Although the main issue with CRFs can be the huge size of the model and long inference times in the order of tens of hours, Krähenbühl and Koltun [KK11] managed to improve the results of a pixel-level classifier on 2 MPixel images in less than a second using a fully-connected CRF, i.e. with pairwise potentials defined for all pixel pairs on the image. Their model has also been used to improve fine-scale detail on recent deep network architectures [CPK+18].

Schindler [Sch12] compared different smoothing strategies that increase accuracy on high resolution aerial images: majority voting in a neighborhood, Gaussian and bilateral filtering, and approximate inference via global methods for random fields such as graph cuts and semiglobal labeling. His test results show that any smoothing method improves performance, and thus a smoothness assumption is necessary when segmenting high resolution images mainly due to fine texture details inducing more per-pixel errors. Among his compared algorithms, global methods outperform local methods, with contrast-sensitive graph cuts giving the best results for all datasets and metrics. However, the difference between the worst-performing smoothing method and the best is smaller than the

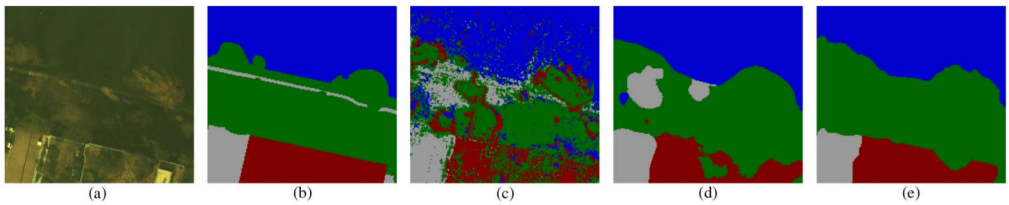


Figure 3.3: Impact of label smoothing on a low-contrast 40 cm aerial image: image (a), ground truth (b), raw segmentation (c), majority voting (d) and contrast-sensitive graph cut (e). Image from [Sch12].

difference between raw segmentation and the worst performing method. He also notes that including feature descriptors computed over the pixels neighborhood implicitly produces a smoother labeling.

3.1.4 Summary

Image segmentation is a vast and growing topic, mainly from Computer Vision and Machine Learning research. Most of the research focuses on designing new features or algorithms that lead to better accuracy results. Training times are often neglected, since it is common to assume that once a good trained model is produced, this model will be predefined and constant in the final application. In our case, we want segmentations that lead to plausible detail enhancement on arbitrary terrain landscapes, i.e. we mainly care about the perceived details from a user point of view, rather than the accuracy towards a specific dataset – which may not be representative of all the variety of terrains. Moreover, since classes will not be known in advance and may even be modified or specialized during modeling, we need a segmentation pipeline that allows for easy retraining of the model.

3.2 Terrain detail synthesis

There are two main reasons why we are interested in modeling details on terrain DEM. First, as we have already discussed, the current availability of high resolution datasets is not enough for large portions of the Earth, and specially on mountainous areas. Second, aerial photographs only represent the topmost layer of an area; in case we wanted to render a walkthrough at ground level in a forest area, for example, we would need to invent the missing details on the ground.

We can distinguish three main approaches for modeling terrains [STB+14; NLP+13]: procedural methods, which can empirically generate plausible terrains from a set of rules and parameters, simulations, based on modeling geological and physical effects for a time period onto a base terrain, and example-based approaches, which use a set of real terrains to generate new similar models.

3.2.1 Procedural modeling

Early approaches to synthesize procedural details on height fields made use of fractal noise generators such as Perlin noise [Per85], thus exploiting the observa-

tion that landform features repeat at different scales. For example, using subdivision methods like *midpoint displacement*, every time a new point is generated its height is set to the average of its neighbors plus a random offset that decreases iteratively [FFC82]. Other stochastic methods scale and sum several octaves of noise of increasing frequency to produce mountain-like structures [MKM89]. This type of noise is usually called *fractional Brownian motion* (*fBm*). Ebert et al.'s book *Texturing and Modeling: a procedural approach* [EMP+98] provides a good overview of these methods.

Rivers can also be modeled procedurally and incorporated into the landscape. Prusinkiewicz and Hammel [PH93] combined the midpoint displacement method with a squig-curve fractal model of a non-branching river using a context-sensitive rewriting mechanism. Kelley et al. [KMN88] presented a recursive algorithm that constructs a drainage network based on climate and soil parameters, and then fills the terrain height map from it using scattered data interpolation. Similarly, Genevaux et al. [GGG+13] also model terrains starting from the generation of a river network. They also use a classification of the water-courses into different hydrological categories (springs, junctions, deltas, ...) to build river blocks that will be combined to produce the final terrain using a modeling tree approach inspired by Constructive Solid Geometry.

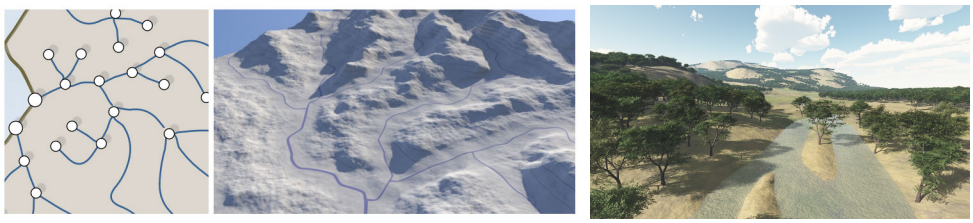


Figure 3.4: Genevaux et al. terrain generation from river networks. River network graph (left), produced terrain (center) and a rendered scene (right). Images from [GGG+13].

The main problem with these methods is that they provide little or no control over the position or appearance of generated features like ridges or valleys, the user can only adjust the initial parameters and regenerate the terrain. Specifying generative rules that preserve the overall coherence of the scene is a difficult task, mainly because of the indirect control over the generation processes. Therefore, several improvements have been proposed to provide user control.

Schneider et al. [SBW06] implement multifractals using GPU shaders to provide immediate visual feedback for a terrain editor interface, in which the user can interactively adjust parameters and paint the fractal's basis functions. Gain et al. [GMS09] present a sketching editor in which users can draw and edit the

vertical silhouette and shadow curves to generate mountain ridges, as well as defining noise patterns for closed regions. Hnaidi et al. [HGA+10] use sketched feature curves with elevation and slope parameters to define ridges, rivers, hills, or cliffs. These curves are then rasterized and a diffusion method fills the rest of the height map.

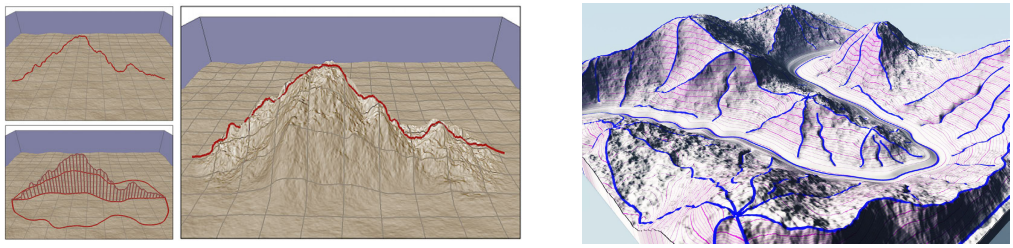


Figure 3.5: Sketch-based terrain editors. Left, sketching a mountain in [GMS09]. Right, sketched curves (in blue) act as constraints for the diffusion method of [HGA+10].

Genevaux et al. [GGP+15] introduce a hierarchical representation for terrains using a construction tree. Leaves are parametrized terrain features like mountains, rivers or lakes, and inner nodes are combination operators like carving, blending or warping. This model allows for intuitive control over the shape and distribution of both local and global landform features.

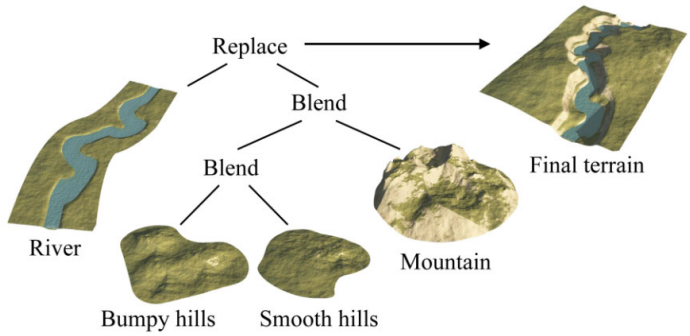


Figure 3.6: Hierarchical representation of terrains using feature primitives. Image from [GGP+15].

Dachsbacher et al. [DBS06] proposed a procedural texturing method on the GPU that accounts for elevation, slope, and geographic conditions like temperature, rainfall or solar distribution. Their model can be semi-automatically fitted to real data, such that new texture of arbitrary resolution is synthesized at runtime and the original orthophoto is not needed.

3.2.2 Simulations

Realistic landscapes with eroded mountains, sedimentary valleys and realistic plant distributions can be generated through physics-based simulations that model geological phenomena.

Erosion simulations

Erosion simulations are often used as a post processing step to add realism to procedurally generated terrains. In general, erosion processes transport rocks and sediments from one location to another. Thermal erosion is produced by the expansion and contraction of rocks caused by different temperatures. These rocks break and move material from high steep points to lower locations until the material stability angle is reached, thus smoothing sharp edges. Hydraulic erosion is caused by water flowing through the material, dislodging and carrying rock particles along with it and carving valleys. Wind can also be an important erosion effect in arid areas, wearing down exposed surfaces through abrasion.

The seminal work by Musgrave et al. [MKM89] introduced simple models for hydraulic and thermal erosion, moving sediments from a vertex to its neighbors. Hydraulic erosion techniques were further extended and refined in [Nag98; CMF98]. Corrosion simulation was introduced by Wojtan et al. [WCM+07].

The limitation of height fields is their inability to represent caves, overhangs or arches, since every position of the terrain grid can only contain one elevation value. Using a full 3D mesh or a voxel grid can solve this problem, but at a higher memory cost. Benes and Forsbach [BF01] proposed a layered data representation: instead of storing a single elevation, each 2D grid position contains a stack of terrain layers with an associated thickness and material properties. They demonstrate the use of this representation on thermal erosion simulations with different materials.

Stava et al. [ŠBB+08] use the layered representation [BF01] and integrate the simulation of force-based erosion caused by running water, dissolution-based erosion caused by slowly moving water that penetrates the soil, and sediment slippage produced by thermal weathering. They implemented the simulation entirely on the GPU, allowing interactive terrain modeling.

Peytavie et al. [PGG+09] also use a layered representation to build a framework that can represent arches and caves, as well as realistically simulating loose piles of rocks and providing editing and erosion tools. In Benes et al.[BTH+06],

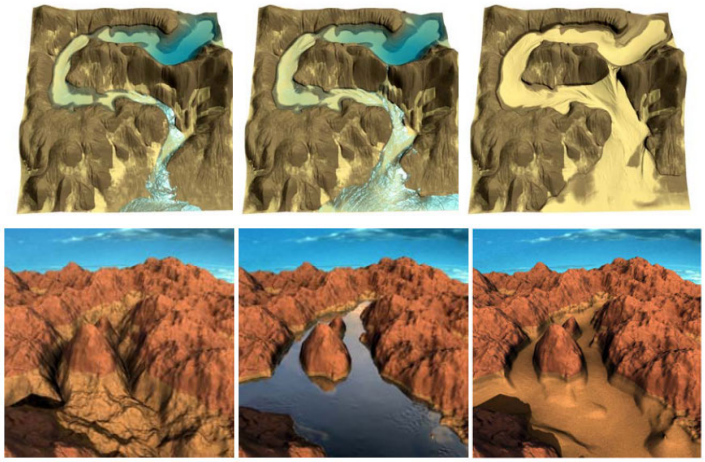


Figure 3.7: Stava et al. erosion simulations. Top, force-based erosion produces a meander break. Bottom, dissolution-based erosion softens the river bed and erodes the banks. Images from [ŠBB+08].

Figure 3.8: Stone arches, caves and piles of rocks modeled with by Peytavie et al. [PGG+09].

a 3D voxel grid representation is used to simulate hydraulic erosion through the Navier-Stokes equations of fluid dynamics.

Instead of using Eulerian approaches based on grids like the techniques before, Krištof et al. [KBK+09] introduced a Lagrangian approach for hydraulic erosion using Smoothed Particle Hydrodynamics (SPH) [GM77; Luc77]. SPH is able to simulate full 3D erosion on any terrain features and can be applied on large scale terrains, since particles act locally and only where the fluid is located. However, a considerable number of particles needs to be simulated for realistically looking results.

Simulation of erosion at the level of entire mountain ranges and for large temporal scales has been addressed in Cordonnier et al. [CBC+16]. They combine the development of mountains through tectonic uplift (the local speed at which terrain grows) as well as the fluvial hydraulic erosion caused by streams. Users can control the process using an uplift map, a gray-scale image representing the

growth speed on different parts of the terrain. In a subsequent paper [CCB+17], a set of gesture-based methods on multi-touch devices provides users with interactive tools to simulate the action of plate tectonics and compute uplift maps.

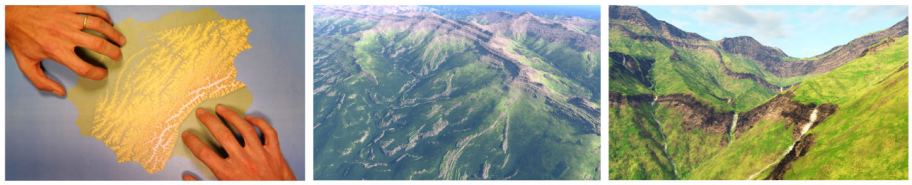


Figure 3.9: Gesture methods for interactive simulation of plate tectonics. Image from [CCB+17].

Ecosystem simulations

Ecosystem simulations aim at producing realistic plant distributions according to the characteristics of the environment. In general, they rely on particle-based simulations where plants compete for resources such as light, water and space [DHL+98]. Individual plants can be represented using circles that define the neighborhood area in which a the plant interacts with its neighbors. Biological rules are used to determine the outcome of interactions between circles: the competitive abilities of the individuals are based on their characteristics and the environment where they are located, and the fittest individual survives while the rest may die.

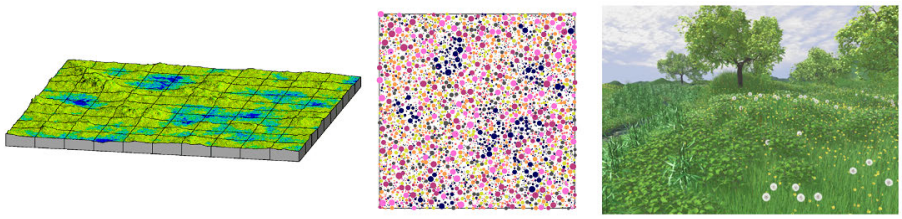


Figure 3.10: Deussen et al. ecosystem simulation. Left: the terrain water concentration ranging from high (blue) to low (yellow). Center: the final individuals distribution from eight plant species. Plants with a preference for wet areas are shown in blue. Right: rendered scene. Images from [DHL+98].

Several improvements have been proposed. Lane and Prusinkiewicz extended Deussen et al. method [DHL+98] to multi-level plant communities and groups of plants, simulating effects like self-thinning, succession and clustering, as a plant not only limits the development of other plants in its neighborhood but increases the probability in the vicinity of it. Alsweis and Deussen [AD05] included also

asymmetric plant competition in which only one individual in the overlapping area gains access to the resources, thus limiting the growth rate of the rest.

Ecosystem simulations are often applied on an already modeled terrain as a layer on top of it, but rarely modify it. Cordonnier et al. [CGG+17] propose a framework that models the interaction between vegetation development and terrain erosion, using a layered representation of the terrain and integrating stochastic events such as rock slides or fires. This framework has also been applied in [CEG+18] to simulate snow-covered terrains and related phenomena: melting due to radiation, wind transport effects (e.g. cornices on ridges), avalanches and skiers tracks.

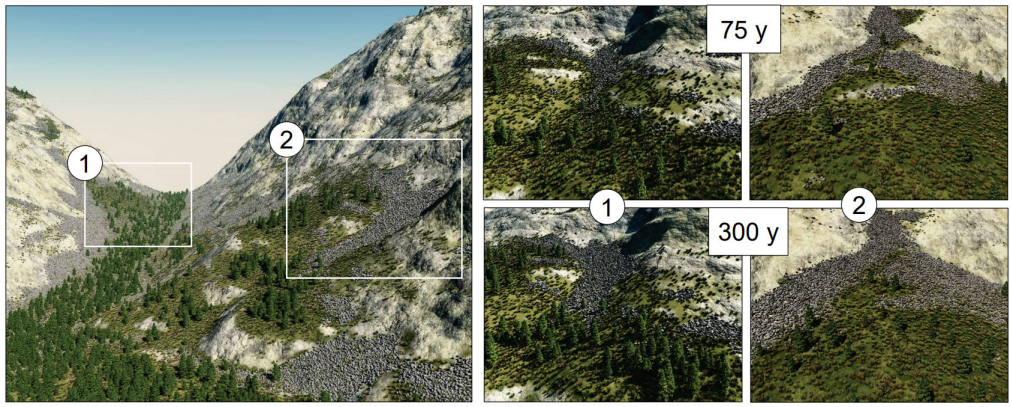


Figure 3.11: Combined ecosystem and erosion simulation by Cordonnier et al. [CGG+17]. In (1), falling rocks destroy vegetation. In (2), plants grow on accretion zones. Image from [CGG+17].

3.2.3 Example-based synthesis

Instead of modeling terrains using a set of rules or running physical simulations, these approaches borrow ideas from texture synthesis methods [WLK+09] and build terrains by combining information from available real terrains.

Brosz et al. [BSS07] use multi-resolution analysis to extract high resolution information from an existing DEM, and transfer these details onto another low resolution terrain. In Andujar et al. [ACV+14], a point cloud of a small part of the terrain is used to estimate the fractal parameters that will be used to add detail to the rest of the terrain DEM. They obtain this point cloud from using multi-view stereo on a set of roadside photographs.

Zhou et al. [ZST+07] generate terrains based on an example height map

and a user line sketch drawing that defines the main features such as mountain ridges or valleys. Their system, similarly to example-based textured approaches like [EF01; KSE+03], finds patches from the example terrain that match the sketched features, and blends these patches together using graph cuts.

Figure 3.12: Example-based terrain synthesis by Zhou et al. [ZST+07]. The grayscale image columns show, from top to bottom, the user sketch, the example terrain and the synthesis result. From left to right, the example DEM corresponds to: Mount Jackson, Mount Vernon and Grand Canyon. Images from [ZST+07].

Gain et al. [GMM15] specialize a hierarchical pixel-based texture synthesis method [LH05] for terrain generation, and also offer interactive user control through geometric modeling constraints, terrain type constraints using labeled exemplars, and copy and paste operations.

Guérin et al. [GDG+16] introduce a sparse representation for terrains. A height field is decomposed into overlapping patches, and each patch is represented as the weighted sum of several atoms found in an optimized dictionary extracted from (real) terrain exemplars. Through the use of multi-resolution dictionaries, they are able to increase details on coarse height fields or sketches.

Recently, Guérin et al. [GDG+17] make use of Conditional Generative Adversarial Networks (cGAN) [IZZ+16] to build an interactive sketch-based terrain modeling interface. Their system uses two cGAN: a first one that produces a heigh map from a sketch, and a second one that mimics erosion simulation to increase details on a height map. In order to train the first network, real DEM datasets are used and converted to sketches and features. To train the second network, a hydraulic and thermal erosion simulation [CGG+17] is run on the real DEM.

It is also possible to learn from exemplars the distribution of individual scene objects as well as structured content, and reuse it for synthesis. WorldBrush [EVC+15] presents an interactive editing system that uses a painting metaphor: learned statistical distributions and parameters are stored as palettes, which can be later used as input of a procedural content generator through painting tools like brushes or gradients. WorldBrush relies on the user to maintain

Figure 3.13: Terrain sketching using cGAN. Image from [GDG+17].

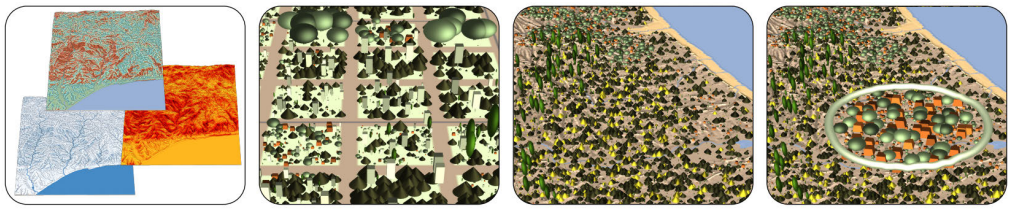


Figure 3.14: EcoBrush: terrain conditions (a) are used to locally index a database of plant distributions (b) and produce an initial ecosystem on the terrain (c), which can be modified by the user using brushes (d). Image from [GLC+17].

the realism between the terrain conditions and the selected distribution. EcoBrush [GLC+17], on the other hand, creates biome-specific databases indexed by abiotic terrain conditions such as temperature, rainfall, sunlight or slope. The input terrain is automatically populated with an initial ecosystem matching the abiotic conditions. Users can then apply brushes that adjust ecosystem age, plant density or variability, or even paint specific distributions.

3.2.4 Summary

We started this section introducing different procedural techniques capable of generating arbitrary detailed terrains, and how to control them towards specific outcomes (e.g. sketching interfaces). In order to enhance the detail on a terrain model, it is also possible to run simulations of varied effects like erosion from water or wind, ecosystems, snow, etc. Real data has mostly been used either as initialization on simulations or as a resource on example-based algorithms, but their goal has been generating plausible models.

We want to augment detail on existing DEM in a plausible manner as well, but focusing towards fidelity to real terrain by introducing additional information from other sources, for example the associated aerial imagery.

3.3 Vegetation modeling

Vegetation is an important component of natural scenes, having a huge impact on the realism and level of detail of them. Ever since the early days of computer graphics, effectively and realistically modeling plants has been an area of interest and, due to the complex nature of trees, it is still an ongoing research topic. In this section, we will discuss the state of the art on tree modeling from two different perspectives: generating synthetic trees and acquiring models for real trees. Although both approaches seem to solve opposite problems, they benefit from advances and novel ideas in either of them. Moreover, if our goal is not only to model a tree but also to efficiently render thousands of different trees – as is the case of forest terrains – we will probably have to take into account some of the existing rendering optimization techniques when modeling a tree, in order to choose appropriate data structures for representing it.

3.3.1 Synthetic generation of trees

For a comprehensive and detailed survey on procedural and rule-based methods of tree generation, the book "Digital Design of Nature: Computer Generated Plants and Organics" by Oliver Deussen and Bernd Lintermann [DL06] is a very useful reference. Here we will briefly summarize some of the methods described in chapters 4, 5 and 6 of the book, as well as more recently presented articles.

Procedural methods

Procedural methods, algorithms using a set of rules and its associated parameters, were the first approaches for reproducing certain types of plants or species. The first branching patterns were defined using cellular automata on regular grids by Stanislaw Ulam in 1966. One year later, Dan Cohen proposed the first continuous branching procedural model [Coh67]. The first branching structures in 3D were introduced by Honda [Hon71], using a very simple recursive algorithm characterized by branching angles and length ratios of consecutive branch segments. Aono and Kunii [AK84] extended Honda's work to a variety of branching patterns through statistical variations of angles, attraction and inhibition. Oppenheimer [Opp86] used a method inspired by fractals that recursively calls itself to simulate the branching along a trunk and to generate smaller branches onto large ones. Bloomenthal [Blo85] focused on the geometrical aspects of tree modeling, proposing techniques to represent the trunk, branches, and bark of a tree more faithfully.

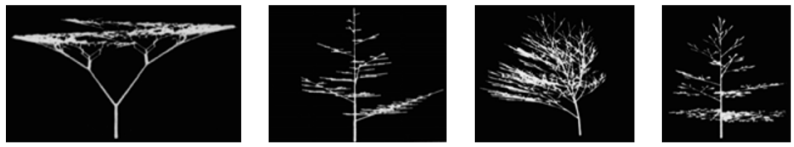


Figure 3.15: Branching structures by Aono and Kunii: (a) binary branching with angles 35° and -35°, (b) single branching with angle -70°, (c) use of an inhibitor to simulate wind, (d) branching angle depending on age. Image from [DL06].

De Reffye et al. [REF+88] proposed a procedural method inspired by botanical growth rules. The growth of the shoots is simulated in discrete time steps, and each bud carries the probability of dying (stop branching), resting or branching out. However, it takes a considerable knowledge of both botany and of their model to create images with great fidelity to nature. On the other hand, Weber and Penn [WP95] proposed a method that captured the approximate appearance of the trees instead of using botanical principles. The downside is the large amount of parameters of their method, about 80, including the enclosing shape of the tree, the height of the trunk before branching, the number of branching levels, branching angles and relative lengths, and phototropism.

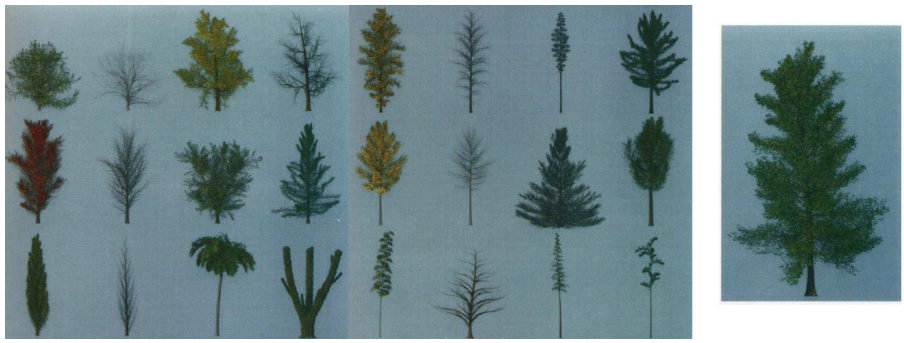


Figure 3.16: Trees generated by Weber and Penn, and a detailed view of a Black Tupelo tree. Images from [WP95].

L-systems

In parallel, another set of procedural techniques was introduced by Lindenmayer [Lin68]: the L-systems. Morphological forms of plants are described through string rewriting systems, thus exploiting the capabilities of formal languages. The main difference between rule-based systems and other procedural methods is that an initial state (a string) is transformed into a final state (another string)

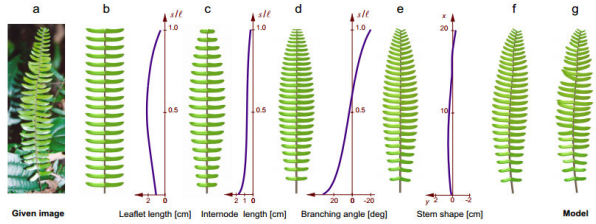
by applying a number of changes or substitutions defined by the set of rules (the grammar). In order to reproduce variability as seen in nature, stochastic rules and dynamically modified parameters were introduced. Furthermore, context sensitive grammars can allow signal exchanges between different parts of the plant. Prusinkiewicz and Lindenmayer's book "The algorithmic beauty of plants" [PL96] examines many aspects of plant modeling using L-systems.

L-systems have also been extended to use time information in timed L-systems and differential L-systems [PHM93], which permit the simulation of growth procedures through the use of differential equations for the parameter set. Environment-sensitive L-systems [PJM94] can simulate local aspects of the environment (for example, pruning) with an inquiry symbol based on the position. Open L-systems [MP96] also enable communication modules for sending and receiving parameters, so interaction between branches or trees such as light

(a) Animation with differential L-systems

(b) Tree interaction with open L-systems

(c) Pruning with environment-sensitive L-systems



(d) Modeling a fern leaf using positional functions

Figure 3.17: Examples of various extensions of L-systems. Images from [DL06].

competition can be simulated. Finally, user-defined functions and modeling tools can also be inserted [PMK+01], facilitating the reproduction of a particular plant.

Although rule-based procedures such as L-systems are a very powerful method that allows the generation of a large variety of plants, they are not very intuitive. The production of a precise plant is a difficult process even for skilled users. Small changes can cause a complete modification of the total shape, and after a rule or parameter change the whole expansion process must be worked through. On the other hand, procedural methods have the opposite characteristics: only a very limited number of plants can be modeled with a method, but the parametrization is usually straightforward and intuitive.

Therefore, some tools like the Xfrog modeling system by Lintermann and Deussen [LD99] offer a combination of the two approaches in the so-called rule-based object production approach. Here, a plant is represented by connecting components. Each component generates parts of the plant geometry such as leaves, stems or simple geometric primitives by using procedural methods. These components are then connected in a directional graph with multiplication nodes, which represents the rule system (Figure 3.18).

Figure 3.18: Examples of plants generated with the rule-based Xfrog system. Images from [DL06].

Competition for resources

One recent strategy is based on the observation of the factors that influence the final shape of a tree. In particular, competition for resources (sunlight, space) by different branches of a tree seems critical for the general shape of temperate-climate trees. The Space Colonization algorithm of Runions et al. [RLP07] uses this fact and generates a set of attractors inside the volume defined by the tree foliage. These attractors are then iteratively conquered by the branches as they occupy the available space. Figure 3.19 provides more details about the branching algorithm.

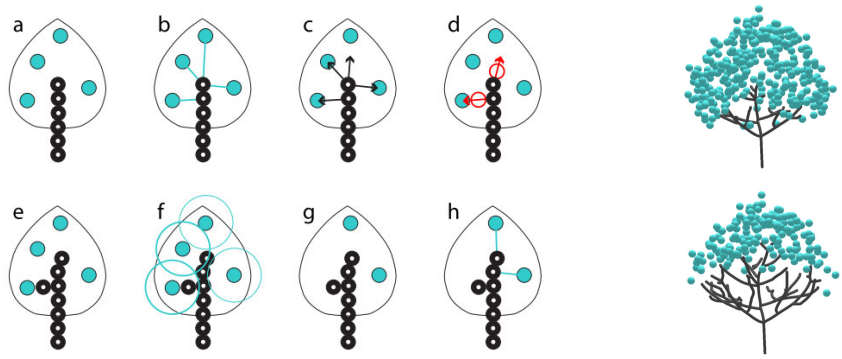


Figure 3.19: Runions et al. Space Colonization algorithm. (a) The tree consists now of six nodes (black) and four attraction points (red). (b) Each attraction point is associated with its closest node inside a radius of influence. (c) Influence vectors from a node to its associated attraction points. (d) Normalized sum of influence vectors (red arrows) provides new node locations (e). The neighborhoods of the two leftmost attraction points have been penetrated (f), so they are removed (g). A new iteration begins (h). Images from [RLP07].

Palubicki et al. [PHL+09] also propose a self-organizing model for plant development that extends the Space Colonization method by using a signaling mechanism to mimic different types of growth. In addition to the space colonization, they also model the competition for light using a shadow propagation algorithm, proposed also by Palubicki in [Pal07]. The results of space and light availability are used to control the fate of the buds: produce a new branch, produce a flower, remain dormant or abort. In [XM15] Xu and Mould improved the performance of tree generation algorithms based on space competition by using pregenerated guiding vector fields and Yao graphs.

Kohek and Strnad [KS15], on the other hand, used competition for incoming light as the main competition resource. They computed on the GPU how the shadows projected by the growing branches propagated through a regular grid.

Then branches could grow on directions that maximized the amount of received light while the shadow grid was being updated.

Other factors such as wind and surrounding space also influence a tree's growth. Pirk et al. [PSK+12] presented a technique that made it possible for content creators to change a tree's position inside a scene and see its shape adapt to changes in light distribution and the occupation of surrounding space. In [PNH+14] it was extended to include the effect of wind. The wind field was physically simulated to consider both high-frequency changes and long-term wind stress resulting in broken branches and bud abrasion. Recently, in [PJM+17], Pirk et al. have also modeled the combustion of trees, its propagation and the effects on the branching structure and texture.

Figure 3.20: Top: plastic trees from [PSK+12]. The original tree model adapts to growing close to another tree and obstacles casting shadows. Middle: scene with trees that react to a wind simulation in [PNH+14]. Bottom: combustion on a group of trees [PJM+17].

Inverse procedural modeling

Given the difficulty of predicting the outcome produced by procedural and rule-based models, some authors have proposed methods that guide the process towards a desired goal.

One possibility is to let the user explore the space of all possible models while guiding this search [TGY+09]. In order to help users explore the space of possible models it is interesting to provide a relatively small set of parameters that control

the generation. These may be provided by the procedural algorithm designer but they may also be obtained semi-automatically. In [YAM+15] Yumer et al. use autoencoder networks to find a lower dimensional space that can be used as the set of parameters for users to play with. A weak point of these systems is that the resulting trees follow the intention of the artist more closely but the cost of generating a great quantity of similar models is very high.

Consequently, to automate the process Talton et al. [TLL+11] presented an approach using Markov chain Montecarlo. Given a grammar and a high-level specification of the desired production, they optimize over the space of possible productions from the grammar in order to compute the one that better conforms to the specification. However, its convergence is affected by the fact that the algorithm receives its feedback from completely-generated models only. Ritchie et al [RMG+15] improved upon it by developing a new sequential Montecarlo algorithm capable of using incremental feedback provided by incomplete models.

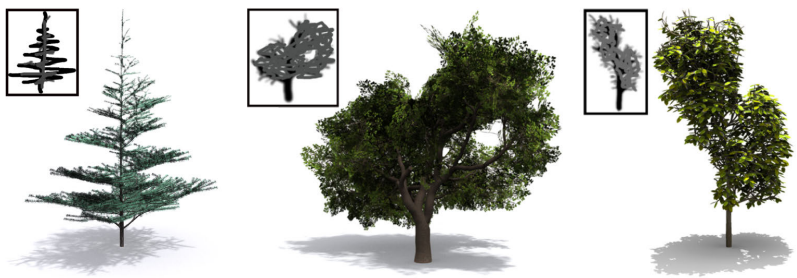


Figure 3.21: Talton et al. Metropolis Procedural Modeling: conifer, old oak, and poplar grammars targeted to sketches. Image from [TLL+11].

Stava et al. [SPK+14] developed an inverse procedural modeling algorithm specifically tailored for trees. Given an input tree model, their method estimates the values of the parameters for an ad-hoc procedural model capable of reproducing a wide variety of tree species. Then, they can generate more trees similar to the input exemplar, simulate environmental responses or different developmental stages (see figure 3.22).

Recently, Wang et al. [WLX+18] generate tree model variations by blending between two given models. They define a tree-shape space with a proper metric that quantifies deformations, such that the shortest path between two trees corresponds to the optimal deformations to morph from one to the other, allowing a continuous blend between them.

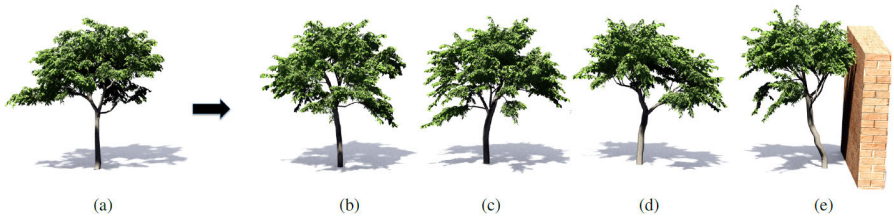


Figure 3.22: Stava et al. inverse procedural modeling approach. From an input model (a) the parameters for the procedural model are estimated. Similar trees (b-d) can be generated, as well as environmentally sensitive models (e). Image from [SPK+14].

Figure 3.23: Optimal blending path between the trees on the left and the trees on the right. Image from [WLX+18].

Sketch-based modeling

Some authors have also addressed the possibility of allowing artists to directly draw the desired shape and appearance of the tree. In Okabe et al. [OOI05] sketch interface, the user draws the branch structure in 2D. Sketched branches are then positioned in 3D using a greedy strategy under the following constraints: the branches projection onto the 2D draw plane must fit the sketch, branches must be located inside the 3D convex hull obtained from the sketch 2D convex hull by sweeping a circle along it, and the distance between sibling branches must be as large as possible.

Wither et al. [WBC+09] inspired on the methodologies used by botanists and artists when they create 2D drawings of trees incrementally: instead of drawing the full tree with details, they usually draw the shape and specify details on certain areas using zoom rectangles. Thus, the authors design a system that makes use of successive silhouettes sketched at different zoom levels and smaller areas; the details of a refined area are then propagated to the rest of the tree.

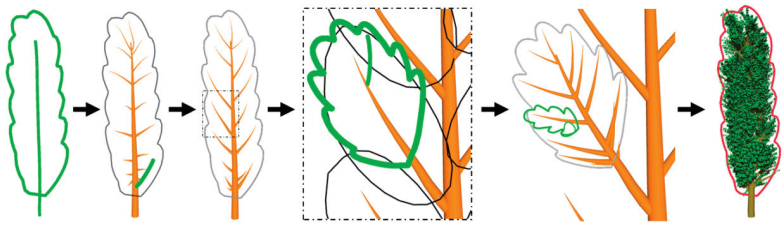


Figure 3.24: Wither et al. sketching system. User sketches at each stage are marked in green. Changes on small areas or single branches are transferred to the rest of the tree. Image from [WBC+09].

Longay et al. [LRB+12] developed a gesture based sketching system that integrates the procedural generation of trees based on self-organization [PHL+09] with a multi-touch interface to control the output, thus reducing modeling time while maintaining the artist's ability to obtain the desired result. The modelers can specify the overall shape of the crown with a lasso tool, or use a brush with different thicknesses to create attractors that will direct the growth of the tree. The final model can also be edited interactively.

In the recent modeling system described by Xie et al. [XYS+16], a repository of detailed branching structures obtained from cuts of real tree scans is provided to the user, who selects and arranges them spatially to specify the tree shape that will result of connecting them using botanical constraints. User-sketched lobes guide the growth of detailed branches and foliage using a space colonization approach.

Figure 3.25: Xie et al. modeling from real tree parts by arranging shapes (in pink) and drawing foliage lobes. Image from [WBC+09].

3.3.2 Model acquisition from real trees

Recently, improvements in imaging and scanned point clouds acquisition have led to methods that reconstruct a tree model from a real exemplar. These methods have the advantage of producing realistically looking results and often do not

need to spend some trial and error time tuning parameters of an algorithm nor expertise in the rule definition. The obvious disadvantage is that they need input data which may not always be available.

From photographs

Shlyakhter et al. [SRD+01] recovered the trunk and major branches of a tree from a set of instrumented photographs. Starting from a manual segmentation of the tree crown on each picture, they reconstruct the visual hull as the intersection of all the back-projected volumes of the silhouettes. The main trunk and first levels of branching are obtained as the medial axis of this visual hull, and the rest of the volume is filled with small branches and details using an L-system.

Reche et al. [RMD04] also start from a set of calibrated photographs. They extract an alpha mask for each picture using an image matting algorithm, and combine the results to generate an opacity volume grid. This volume is then rendered view-dependently by attaching onto opaque cells small billboards extracted from the original photographs.

Neubert et al. [NFD07] also compute an opacity grid from the matting of the input pictures, and a 2D skeleton for each matting that they call attractor graph. The volume is then used to initialize the positions of a 3D flow simulation using particles that will define the branches. These particles are attracted to other nearby particles as well as to the direction field computed from the set of 2D attractor graphs. Once the branching structure has been computed, the model of the branches is generated and populated with leaves extracted from photographs according to the density grid.

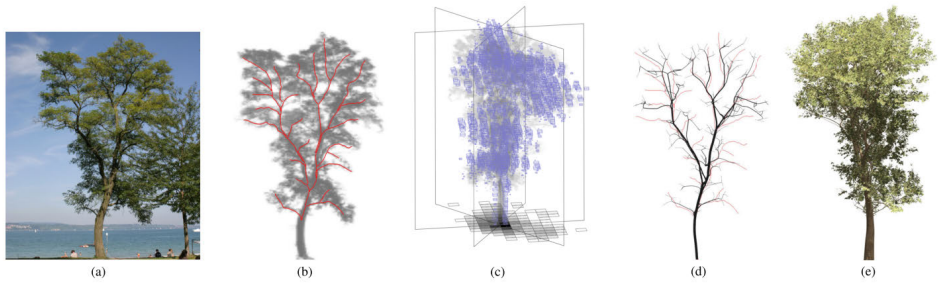


Figure 3.26: Neubert et al. method overview: one of the input images (a), its density estimation and attractor graph in red (b), the density grid (c), branching structure obtained from the particle simulation (d) and final rendering (e). Images from [NFD07].

Some methods transform the input collection of photographs into a set of points using a structure-from-motion algorithm. Quan et al. [QTZ+06] used the generated point cloud to extract and segment individual leaves using a graph-based optimization, and asking the user to model the branch structure. On the other hand, Tan et al. [TZW+07] classified the points as branch/leaf and synthesized hidden branches from visible ones with texture-based synthesis algorithms. Then, they added the crown instantiating leaf clusters identified in the point cloud. Also starting from a point cloud – in addition to an example mesh of a leaf – the technique by Bradley et al. [BNB13] is capable of generating plausible foliage configurations. In order to do this, it fits the exemplar leaf to the point cloud, extracting a statistical model of the shape, appearance, and transformations between neighbors.

Most reconstruction methods that take photographs as input need either several images to work properly or require significant user interaction. However, in Tan et al. [TFX+08], they propose a reconstruction from a single image and little user input. The user draws one stroke in the photograph to identify the crown, and another one (or sometimes more) for the visible branches. The crown is segmented and the visible branches are converted to 3D using the approach proposed in [OOI05]. This initial skeleton is extended into the crown by iteratively substituting an existing branch by a subtree from a database. Finally, the leaves are added using rectangles textured with the input image foliage.

Figure 3.27: Tan et al. single image method overview: input image (a), user-drawn strokes (b) for crown (red) and branches (blue), generated branch structure (c) and final tree model (d). Image from [TFX+08].

Guenard et al. [GMB+13], also extract the foliage from one input image and compute a vector field from this segmented shape that is used to obtain a tree skeleton. This base skeleton is populated with leaves and branchlets via an L-system. This method, however, needs knowledge of the tree species to feed correct parameters to the L-system and has to generate a series of candidate models to select the one that better fits the input image.

From scanned point clouds

Some of the approaches we have already seen in the previous section ran a structure from motion algorithm to the input photographs and obtained a 3D point cloud [QTZ+06; TZW+07; BNB13]. However, others start directly from a laser-scanned point cloud. In recent years, these techniques have become more popular due to the increasing number of scanned datasets produced, specially from city streets.

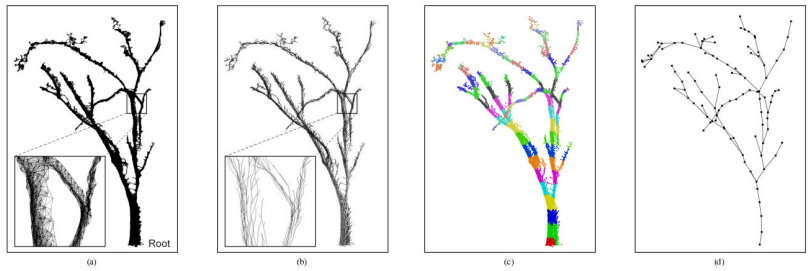


Figure 3.28: Xu et al. main skeleton production: neighbors graph (a), shortest distance paths to root (b), clustered distances to root (c) and skeleton formed by connecting neighboring bins (d). Image from [XGC07].

Xu et al. [XGC07] method starts building a graph connecting each point to its neighbors, and computes the shortest distance path from the root point to each other point. This set of lengths is then quantized, and connecting the centroids of neighboring bins produces the skeleton of the tree. As several connected components may appear in the initial graph due to scanning resolution and occlusions, an appropriate position and angle is found to attach each component skeleton to the main trunk. The points that are not connected to the main skeleton are considered to be leaves, and clustered according to some species-specific parameters to define leaf locations. Fine branches are synthesized to reach these positions.

Livny et al. [LYO+10] also reconstruct a tree skeleton from a neighboring graph between points with distance-weighted edges. After a first skeleton has been computed as the minimum-weight spanning tree of the graph, an orientation field is derived and used to optimize the spatial embedding of the skeleton through a least squares problem. In an follow-up work, they also propose a lobe-based representation to reconstruct and lightly store or transmit trees [LPC+11]. Using three parameters that depend on the tree species, they follow the edges in the points graph and identify points in which the average edge length is bigger than the expected diameter of the branches. These points are unlikely to be connected to the branching structure, so they cluster them in lobes. Each lobe is

represented as the triangulated surface produced from its α-shape, an extension of convex hulls that allows non-convex envelopes. Afterwards, during the tree reconstruction step, each species has a library of textured branch patches and each lobe volume is iteratively filled by fitting and anchoring patches until the desired level of detail is reached.

Figure 3.29: Two examples of Livny et al. iterative lobe reconstruction using branch patches. Images from [LPC+11].

While the previous methods were developed for terrestrial laser scanning, Hu et al. [HLZ+17] recently focused on airborne LiDAR point clouds, in which branching structures are often difficult to capture. They first segment individual tree point clouds from the input point cloud using a normalized cut method. Then, for each individual tree point cloud, the tree skeleton is obtained as a spanning tree generated of the point cloud neighbors graph, using directional fields and bending angle constraints to restrict the growth direction of the branches.

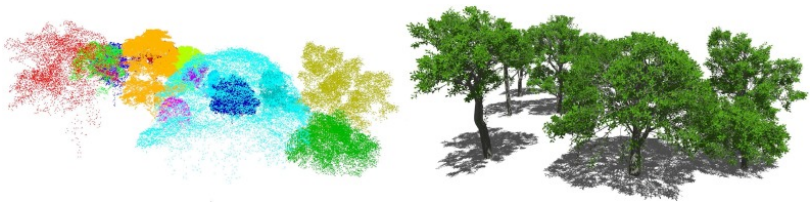


Figure 3.30: Hu et al. tree reconstruction from airborne LiDAR point clouds. Image from [HLZ+17].

3.3.3 Summary

We have seen in this section several methods to generate synthetic trees. While procedural methods have the advantage of being capable to produce as many different models as desired, one of their principal drawbacks is the non-intuitiveness and difficult control towards obtaining a particular tree type or shape. Some methods like the Space Colonization algorithm guide the synthesis towards a specific volume or shape, and thus integrate well with modeling tools via user sketches. There has also been extensive research on reconstructing specific real

trees. However, most of the approaches require access to the physical location in order to capture several photographs or a point cloud before the reconstruction, and their output is usually limited to just that tree.

We will present a method capable of generating plausible tree models from just a single picture, since it is quite easy to obtain different tree images. This picture will serve as shape initialization of a procedural algorithm which will be capable of generating more than one model resembling the desired individual, while eliminating the need for user interaction.

3.4 Forest rendering

Trees are complex objects and their representation usually requires a large number of polygons. Some characteristics that influence the rendering realism of a plant are: the number and variety of organ shapes, the number of organs and the spatial organization of these organs depending on the species [BMG06]. Also, a plant aspect drastically varies with observation distance: at close scale we see a branching system with detailed organs, but at large distances the aggregation of individual details yields an overall impression of a fuzzy volume. This is why most classical rendering optimization and simplifications techniques that are successfully applied on regulars objects usually produce non-adequate results for plants, and a wide range of techniques tailored for plants has been developed.

3.4.1 Image-based representations

The most common tool to achieve a real-time rendering scenes with a large number of trees are billboards. Classical billboards use a single quad oriented towards the camera, but there is no parallax when the camera moves and a tree slightly behind another may pop in front depending on the angle. Cross billboards use a small set of fixed quads crossing each other, but artifacts appear when a quad is viewed from a grazing angle.

Jakulin [Jak00] extended the cross billboard representation for the crown using slicing, and used traditional geometry rendering for the trunk and limbs. In a preprocessing step, several sets of parallel slices are created from various viewpoints, and for each set the crown leaves are assigned to the closest slice. The leaves of each slice are then rendered to a 2D texture. During rendering, the two slice sets closest to the viewing direction are selected, and the slices rendered using the correct transparency and blending (figure 3.31e).

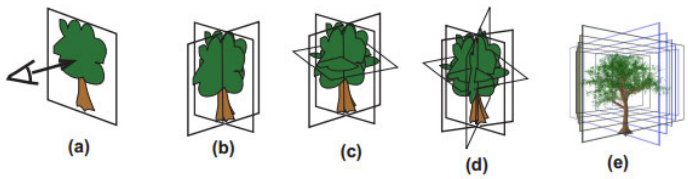


Figure 3.31: Tree representations using billboards: (a) classical, (b) cross billboard, (c) 3-cross billboard, (d) 4-cross billboard, (e) billboard slices. Image from [BMG06].

Another extension of billboards was proposed by Qin et al. [QNT+03] in a representation they called quasi-3D trees. This representation stores a set of 2D buffers containing the vertical tree billboards with information of color, normal vectors, relative depths and shadowing. Two or three horizontally sliced masks are used to cast shadows. Self-shadowing due to skylight is stored in a small voxelization.

Image-based representations are also widely used to build a hierarchy of billboards from a single quad representing the whole tree to hundreds of quads for the branches or even to the leaves level. Behrendt et al. [BCF+05] explain why approaches that work for smooth objects, like the *billboard clouds* by Decoret et al. [DDS+03], do not work well with plants. They propose a new clustering algorithm that uses information about the tree hierarchical structure to build the set of billboards. Realistic lighting is approximated using spherical harmonics. For distant trees, they define square tiles in the background consisting of many plant models which are represented by shell textures - sets of parallel semi-transparent billboards. Other proposed adaptations of *billboard clouds* for trees are adding penalty terms to plane selection and tweaking their position [FUM05], stochastic selection of the billboard planes [LES+06], or indirect texturing to reduce the resolution of the impostor textures [GSS05].

Some authors also leveraged volumetric textures for representing the complex geometry of trees. Decaudin and Neyret [DN04] proposed a volumetric texture rendering technique for forest flyovers, using a sliced triangular prism shape that covers the ground using aperiodic tiling. They combined two different types of slicing: a simple one parallel to the ground for most views except grazing angles, and a more complex one parallel to the screen used near the landscape silhouette. This technique, however, did not allow individual placement of trees. They later generalized their representation [DN09] and proposed creating 6 sets of axis-aligned slices (one for each direction), thus allowing for arbitrary view directions of complex and transparent individual objects.

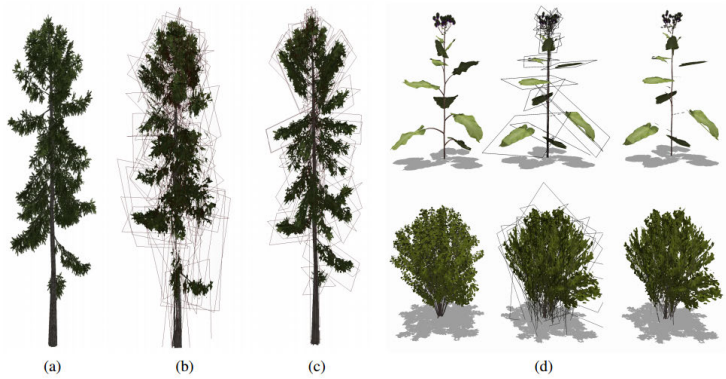


Figure 3.32: Original tree (a), billboard cloud using *k-means* clustering (b), improved clustering by Behrendt et al. [BCF+05] with hierarchical information (c). Other examples (d). Image from [BCF+05].

3.4.2 Points and lines

Point-based rendering was already used in the early work of Reeves and Blau [RB85], as particle systems can represent irregular 3D volumetric structures such as trees with ease, and the level of detail can be smoothly adapted by adding or removing points. In Weber and Penn [WP95] and Deussen et al. [DHL+02], the tree geometry is progressively reinterpreted as branches become lines and leaves become points.

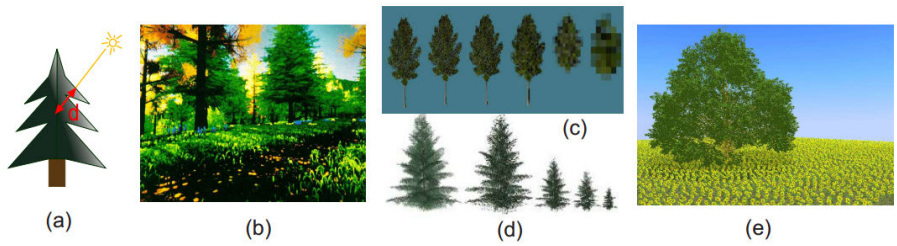


Figure 3.33: Point-based representations of trees: (a)(b) stochastic shadowing model for particle systems proposed by Reeves and Blau [RB85], (c) results by Weber and Penn [WP95], (d)(e) results from Deussen et al. [DHL+02]. Image from [BMG06].

Gilet et al. [GMN05] proposed a technique to visualize trees based in the *sequential point trees* by Dachsbacher et al. [DVS03]. A tree or group of trees is organized into a regular grid, and inside each cell a hierarchical clustering computes a binary tree such that the finest level contains the original triangles and coarser levels are point-based representations defining an average of their

subtrees. Clasen and Prohaska [CP10] also build a sequential model that approximates branches with lines and foliage with ellipsoids, and select the Level of Detail using an image error metric.

3.4.3 Stochastic simplification

Vegetation rendering has been also approached from a simplification viewpoint. However, most mesh simplification schemes do not work on vegetation scenes because the individual elements in it are already simple. The complexity of such scenes is due to the *aggregate detail*: having a lot of elements even if they are very simple. Cook et al. [CHP+07] addressed this problem and proposed to render a scene using a randomly selected subset of elements statistically altered to preserve the overall appearance. Each selected element is modified independently of its neighbors. In particular, the element area is scaled with the inverse of the fraction of selected elements, and its color shifted to the mean element color to preserve contrast.

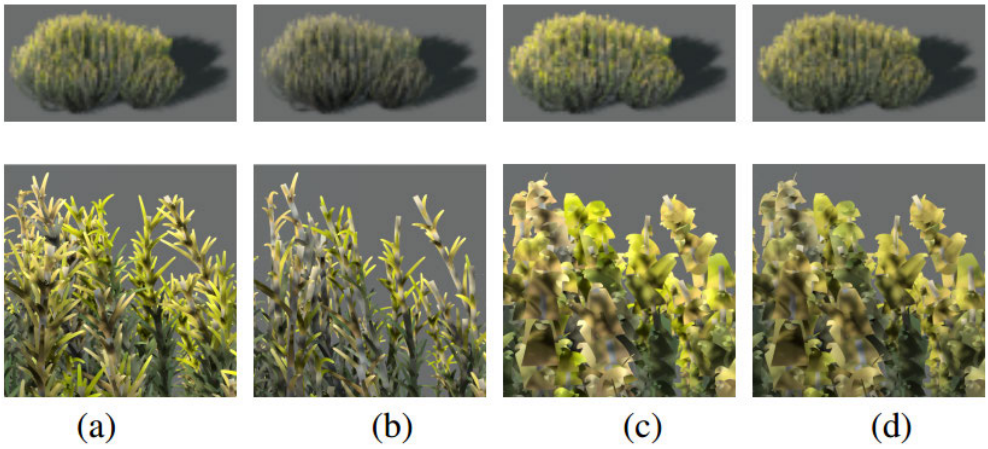


Figure 3.34: (a) Original model, (b) model with only 10% of leaves, (c) with area correction, (d) with area and contrast correction. Image from [CHP+07].

Neubert et al. [NPD+11] improve the stochastic prunning of Cook et al. by introducing a view-dependent Precision and Recall measure and heuristic: an unset pixel that should have been covered is as bad as a set pixel that should not have been covered. They use this measure to find the optimal scaling depending on the fraction of selected elements. Moreover, instead of a pure stochastic selection of the elements, they define a priority based on the degree of silhouette preservation and the orientation of the elements.

3.4.4 Mixed techniques

Recent works propose a combination of different representations and rendering techniques depending on the distance to the camera. The approach by Bruneton and Neyret [BN12] proposes a representation for medium-distance views called *z-field*. The tree is rendered from 181 views on the upper hemisphere, and they store the minimal and maximal depths, the ambient occlusion and the opacity. During rendering, the three closest precomputed views are used to reconstruct the tree shape and compute the shading. For very distant trees that account for less than a pixel, they also propose an illumination model mapped on the terrain that reproduces the photometric behavior of a forest. Although the modeled forests display a variety of realistic lighting effects such as view-dependent reflectance, slope-dependent reflectance, hotspots and silverlining (the silhouettes of backlit trees appear brighter), each different tree model requires about 50 MB.

Kohek et al. [KS17] generate tree geometry on the fly using a particle flow algorithm similar to [RCS+03], starting from a particle distribution inside the crown shape defined using spherical coordinates functions. Coarser levels of detail are obtained by lowering the frequency of trajectory sampling. Distant trees are rendered using an icosphere perturbed in the vertex shader according to the radial functions, and ray casting the probability function to instantiate the leaves.

3.4.5 Summary

As we have seen, in order to achieve efficient real-time rendering of large-scale tree scenes, it is common practice to use impostors or other LOD techniques that alleviate the rendering cost for far instances by simplifying their representations. One of the main issues with these techniques are artifacts due to transition between models. We will propose a novel representation of tree crowns that enables continuous adjustment of the render complexity and smooth transitions between different representations.

4

Aerial image segmentation

In this chapter, we introduce a pipeline for the automatic segmentation of aerial imagery, based on machine learning techniques. First, we will discuss the specific issues that we face when compared against other image segmentation problems. Next, we will describe the experiments and results that guided the design of the segmentation pipeline. Finally, we will evaluate and demonstrate the usefulness of this pipeline.

4.1 Introduction

Aerial images provide many visual cues about the type of terrain and elements above it such as rocks, vegetation, lakes, etc. Therefore, the required step before enriching a Digital Elevation Model with plausible detail is the identification of where each kind of element should be placed according to its high resolution aerial photograph, i.e. the segmentation of this image. Each pixel needs to be labeled according to a set of categories, either by selecting a label from this set or by assigning a probability of the pixel belonging to each of the categories, thus creating a set of probability maps.

There exist publicly available Land Cover maps, which are in fact segmentations of the terrain. We have already listed some datasets in Chapter 2.4. In summary, datasets covering wide areas also have very coarse pixel resolutions in the order of tens or hundreds of meters per pixel. While it is possible to find regional maps at higher resolution, like the Catalonia Land Cover Map [15] at 5 m per cell, this is still not enough for our intended plausible synthesis. Moreover,

the process of creating such maps involves a tremendous amount of human effort: experts usually identify homogeneous regions in the aerial images, delineate them using selection tools, and assign them a label that is often verified using other existing catalogs or even field work. Therefore, creating a cover map at very high resolution – for example, 25 cm pixel like the aerial photographs – is a prohibitively laborious task. That is why machine learning approaches, which can be trained to segment aerial images, are already used to some extent.

We define the *segmentation for synthesis* problem (S4S from now on) as the segmentation of a high resolution aerial photograph for synthesis purposes, for enhancing a terrain model with plausible details perceived in this aerial picture (Figure 4.1). This problem has some unique issues when compared to other segmentation approaches.

First, categories in S4S are usually defined according to different detail synthesis techniques artists might want to apply. For example, in a particular desert scene for a Dakar rally game, one artist might want to distinguish rock, sand, cacti and palm trees, whereas in a tropical forest scene the artist could be interested in segmenting vegetation and rivers. This prevents or complicates the compilation of a predefined training set that will be later used for any type of terrain.

Second, we want to give artists the possibility to add new classes dynamically. This way, they can progressively refine the appearance of different materials, which due to their variety are hard to know in advance (e.g. forest, shrub, grass, crops, sand, bare rock, scree, water courses, inland marshes, snow...) and can decide to distinguish non-anticipated categories (e.g. deciduous from coniferous forests).

Consequently, this flexibility means that a S4S pixel classifier should be able

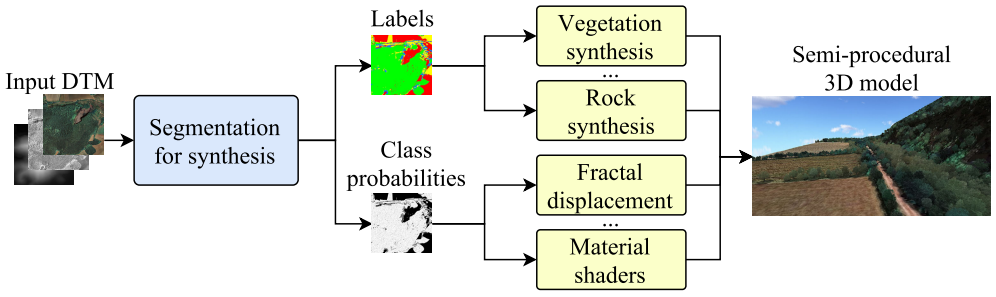


Figure 4.1: Segmentation of aerial images for synthesis. Each class can then be rendered using specific shaders and procedural content (yellow boxes).

to work with relatively small training sets (containing examples from a varying set of classes) and that both training and segmentation times should be within the range of a few minutes, so artists can easily iterate on building the training set and obtaining the desired results. Notice also that we cannot assume balanced classes in the segmented exemplars, neither the exemplar class distributions to be representative of the true class distributions.

As we have seen in Section 3.1, although there is an extensive literature on image segmentation in remote sensing applications, previous approaches either require extensive training sets (unfeasible in the context of dynamic categories), use descriptors tuned for very specific categories (e.g. crops), assume balanced and representative datasets, or rely on expensive-to-train classification algorithms.

Our goal is to explore what are the optimal components of a standard image segmentation pipeline specifically tailored for the segmentation-for-synthesis case, and provide the necessary tools to apply it effectively.

4.2 Segmentation pipeline

We will describe now the design of our proposed segmentation pipeline, shown in Figure 4.2 below.

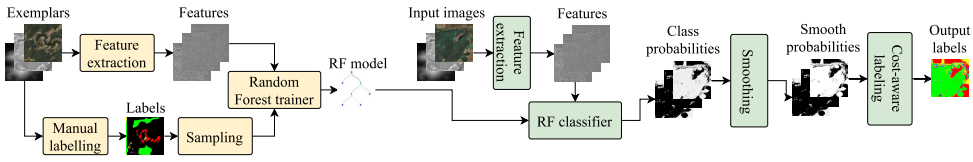


Figure 4.2: Overview of the segmentation pipeline.

4.2.1 Input data

The input data we used during the design and test of our proposed pipeline consists in aerial imagery (visible light as RGB, plus an additional Near Infrared channel) and the elevation of the underlying terrain as a heightfield.

We downloaded our datasets from the Web Map Services of the Institut Cartogràfic i Geològic de Catalunya [12], which provides 25 cm/pixel aerial imagery (RGB and Near Infrared) and elevation as a 5 m/pixel grid.

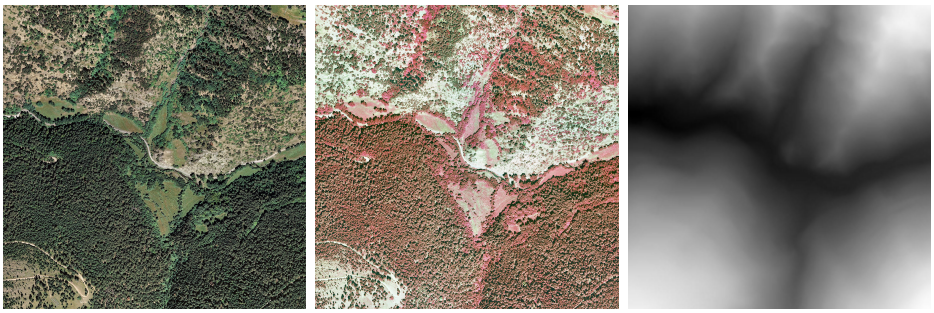


Figure 4.3: Input of the segmentation pipeline, dataset Andorra: aerial RGB image, aerial image combining NIR-R-G channels, terrain elevation normalized for visualization.

In Spain, the datasets from the Plan Nacional de Ortofotografía Aerea [21] also provide RGB and NIR imagery at similar resolution, as well as the 5 m DEM grid. As an example from another country, the italian South Tyrol province catalog [22] contains 20 cm/pixel aerial imagery (also RGB and Near Infrared) and 2.5 m/pixel elevation grids.

4.2.2 Feature set

Previous studies on the segmentation of aerial images have found that the optimal feature set depends on the specific set of categories [RFR04; SPd10]. Recent deep learning approaches obviate feature engineering, but for the specific case of pixel-wise aerial image segmentation they have shown little or no improvement with respect to low-level features [TMS12; PNS15]. We also performed a test using end-to-end learning on a deep neural network, which we will discuss later in Section 4.2.11.

Since we want to let artists choose the desired classes, we decided to include a large number of well-known color and texture features used previously in the context of segmentation of remote sensing images, see e.g. [RFR04], and test their contribution on the classification of typical classes (Section 4.2.7). Features were computed based on the red, green, blue, near infrared and elevation of the pixel itself and its neighborhood, at the original resolution of the input images (25 cm/pixel).

Our 91 chosen features can be divided into seven groups: Height, Color, NDVI, GLCM, Spectral, HOG and LBP.

Height features [4 values]: we considered the elevation at the pixel loca-

tion (interpolated from the lower resolution available elevation map), as well as its gradient (orientation and magnitude). Orientation is represented with two sin/cos values to avoid discontinuities. As we shall see, we propose to use gradient magnitude but discard elevation and gradient orientation as they penalize generalization (see Section 4.2.7).

Color features [10 values]: we use the RGB color components of the pixel, as well as its components in HSL and CIELab color spaces. Hue is represented with sin/cos values. For the CIELab space we use a generic conversion since we do not need exact chromatic coordinates but just a perceptual-based descriptor.

NDVI [1 value]: the Normalized Difference Vegetation Index (NDVI) is a well-known ratio used to predict whether a region contains live green vegetation. Using the Red (R) and Near Infrared (I) channels, NDVI is computed as NDVI = (I−R)/(I+R). Note that this index is not helpful for the detection of vegetation when it is dry, deciduous or shadowed.

GLCM [20 values]: the Grey Level Co-occurrence Matrix is a 2D array C^d in which each element $C^d(i,j)$ indicates how many times the grey tone i co-occurs with the grey tone j in the direction $d = (dr, dc)$, being dr the displacement in rows and dc the displacement in columns. For a discretized grey-scale image I, C^d is defined as:

$$C^d(i,j) = |\{(r,c) \mid I(r,c) = i \text{ and } I(r+dr, c+dc) = j\}| \tag{4.1}$$

Normalizing the matrix C^d such that all entries sum up to 1, we obtain the matrix N^d. As in [SS01], we use the following set of features based on N^d to characterize texture: energy, entropy, contrast, homogeneity and correlation.

Energy	$\sum_{i=0}^{n}\sum_{j=0}^{n}(N^d(i,j))^2$
Entropy	$-\sum_{i=0}^{n}\sum_{j=0}^{n}N^d(i,j)\log N(i,j)$
Contrast	$\sum_{i=0}^{n}\sum_{j=0}^{n}(i-j)^2 N^d(i,j)$
Homogeneity	$\sum_{i=0}^{n}\sum_{j=0}^{n}\frac{N^d(i,j)}{1+\lvert i-j\rvert}$
Correlation	$\frac{\sum_{i=0}^{n}\sum_{j=0}^{n}(i-\mu_i)(j-\mu_j)N^d(i,j)}{\sigma_i\sigma_j}$

We extract these five features over four different normalized symmetric grey-level co-occurrence matrices computed on the pixel lightness discretized into 8 values. We used displacement vectors $d_1 = (0,1)$, $d_2 = (1,0)$, $d_3 = (1,1)$ and $d_4 = (-1,1)$. The matrices are computed using a sliding window of 15 pixels.

Spectral features [6 values]: we compute the Discrete Fourier Transform $F(u,v)$ of the image lightness and compute the power spectrum $P(u,v) = \sqrt{Re(F(u,v))^2 + Im(F(u,v))^2}$. Since, the power spectrum is symmetric about the origin $u = 0$, $v = 0$, it can be sampled in rings to extract features that can be useful for classification purposes [SS01]. According to [TWE11], 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 spectral slope. That means that, on a log-log scale, the power as a function of frequency lies approximately on a straight line. We extract features of the power spectra calculated over a sliding window of 16 pixels. To compute the total power spectrum over rings, the frequency domain is partitioned into 4 ring-shaped regions around the origin. There will be one feature as the result of adding the power values over each region. Linear regression is used on the logarithm of these values as a function of the logarithm of the mean frequency they represent, obtaining information about the spectral slope and the intersect value. These 2 values will also be used as features.

HOG features [8 values]: the Histogram of Oriented Gradients counts occurrences of gradient orientations over a sliding window around a pixel. Image lightness must be first processed using an edge detecting filter (we chose a Gaussian derivative filter, since it is considered relatively invariant to rotations). For each pixel, we obtain the gradient magnitude and orientation from its response to both horizontal and vertical filters. Afterwards, we just count over a sliding window how many magnitude responses turned out to be bigger than a certain threshold and add them to a specific bin according to the orientation associated to it. We used a sliding window of 15 pixels and discretized our histogram into 8 bins.

LBP [42 values]: the Local Binary Pattern descriptor [OPM02] encompasses two complementary measures: local spatial patterns and gray scale contrasts. The most basic version consists in, for each pixel in an image, comparing its value to its neighbours and then building a binary number with the results of these comparisons. Once this is done, compute an histogram of the decimal numbers encoded, for all the pixels inside a sliding window. A rotation-invariant descriptor based on this one can be extracted by only keeping the so called *uniform patterns*, which are those that at most contain two bitwise transitions from 0 to 1 (or vice versa) when the pattern is traversed circularly. In this case, all the *uniform patterns* are assigned a label independent of the starting traversing point, achieving rotation invariance. It has been shown that for 8-neighborhoods with radius 1, 90% of the patterns are *uniform* ones. We compute a rotationally invariant version of LBP over the channels HSL and RGB using 8 points and radii 1, 2, 4 and 8, and using 16 points and radii 16, 32 and 64.

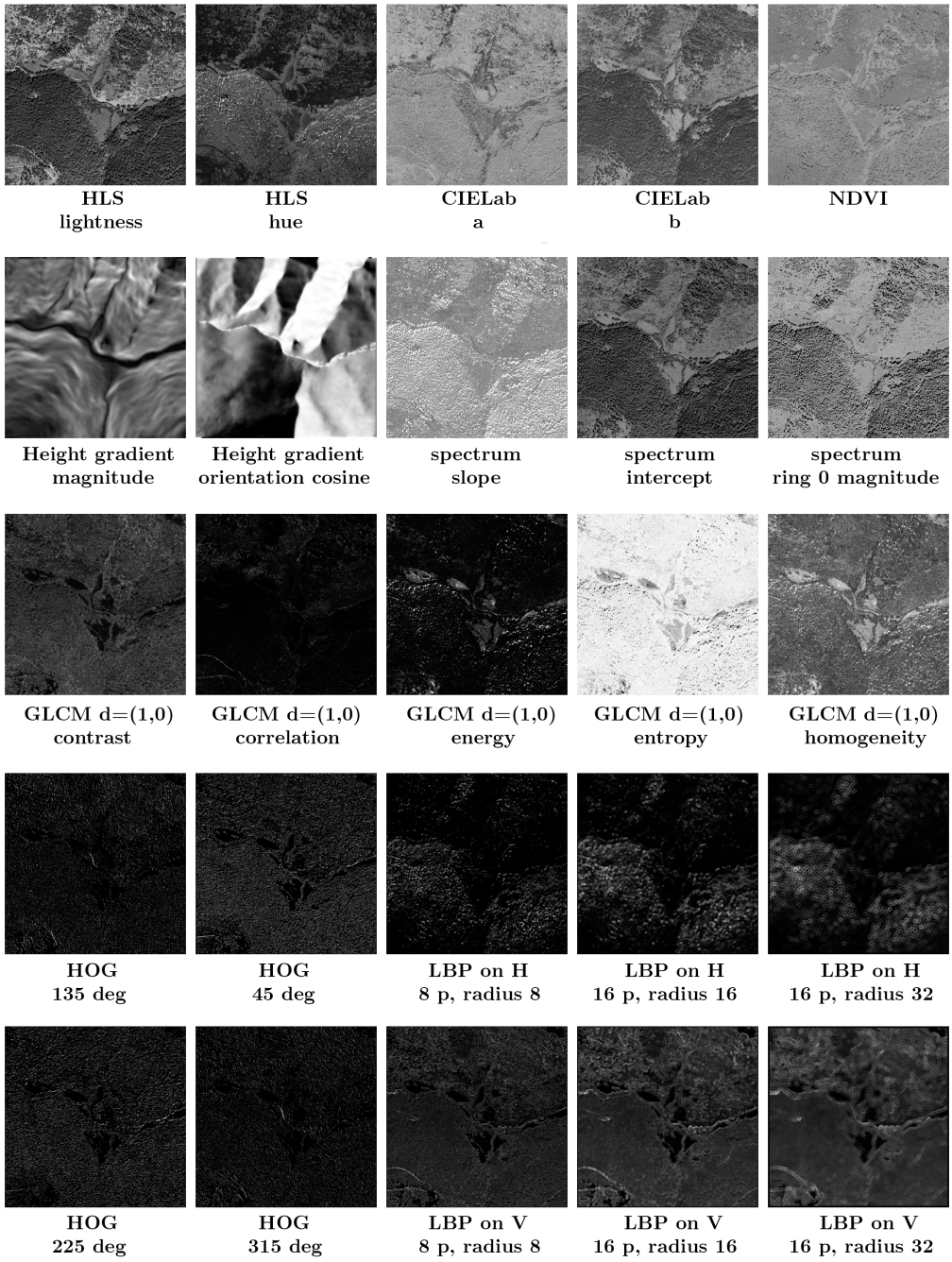


Figure 4.4: Visualization of some features computed on dataset Andorra.

4.2.3 Algorithms for multi-class segmentation

We now describe multiple traditional classification algorithms usually available on many machine learning toolkits such as `KNIME` software [BCD+08] or the `scikit-learn` library for Python.

Ideally, we aim to obtain the best possible accuracy while achieving interactive training and inference times. Let $X = \{x_i\}$ be a set of feature vectors (observations) with associated labels l_i, and let $C = \{C_k\}$ be the set of classes. We want our algorithm to estimate the probability $P(C_k|x_i)$ of x_i being an observation from the class C_k.

K Nearest Neighbors estimate $P(C_k|x_i)$ given a distance metric d and a user-defined number of neighbors K as:

$$P(C_k|x_i) = \frac{K_k}{K} \tag{4.2}$$

where $K_k = |\{x_j \in X_i \mid l_j = k\}|$ and X_i is the set of K observations that are closest to x_i according to the metric d.

Multinomial Logistic Regression estimates the posterior probabilities $P(C_k|x_i)$ using the softmax function:

$$P(C_k|x_i) = y_k(x_i) = \frac{\exp(w_k^T x_i)}{\sum_j \exp(w_j^T x_i)} \tag{4.3}$$

The maximum likelihood function given parameters w_j is defined as:

$$P(T|w_1, ..., w_k) = \prod_{i=1}^{|X|} \prod_{k=1}^{|C|} P(C_k|x_i)^{t_{ik}} = \prod_{i=1}^{|X|} \prod_{k=1}^{|C|} y_k(x_i)^{t_{ik}} \tag{4.4}$$

where t_i for an observation x_i with label $l_i = k$ is a vector of zeros that contains a 1 at position k, and T is the $|X| \times |C|$ matrix of target variables with elements t_{ik}. The parameters w_j can be estimated by minimizing the minus logarithm of this function, also known as the *cross-entropy error*:

$$E(w_1, ..., w_k) = -\ln\left(P(T|w_1, ..., w_k)\right) = \sum_{i=1}^{|X|} \sum_{k=1}^{|C|} t_{ik} \ln\left(y_k(x_i)\right) \tag{4.5}$$

The Hessian matrix of E is positive definite and so the error function has a unique minimum. The IRLS (Iteratively Reweighted Least Squares) algorithm can be used in order to find the w_j parameters that minimize it.

Support Vector Machines find the optimal separating hyperplanes, i.e. the planes that maximize the margin for linearly separable data. For the two class problem, let $t_i = -1$ when $l_i = 0$ and $t_i = 1$ when $l_i = 1$. Given the plane $\pi : g(x) = \langle w, x \rangle + b = 0$ we want to find w, b such that:

$$\max_{w,b} \left\{ \min_{1 \leq i \leq |X|} d(x_i, \pi) = \frac{\langle w, x_i \rangle + b}{\|w\|} \right\} \quad \text{subject to} \\ t_i(\langle w, x_i \rangle + b) > 0, \quad 1 \leq i \leq |X| \tag{4.6}$$

Rescaling w, b such that $|\langle w, x \rangle + b| = 1$ for the points closest to the hyperplane (support vectors), the problem can be reduced to minimizing $\frac{1}{2}\|w\|^2$ and can be solved using quadratic programming techniques.

Using Lagrange multipliers we can find the dual form which expresses w as a linear combination of the input features:

$$w = \sum_{i=1}^{|X|} \alpha_i t_i x_i \quad \text{subject to} \quad \sum_{i=1}^{|X|} \alpha_i t_i = 0 \tag{4.7}$$

Kernel functions $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ can be introduced to efficiently work in higher dimensional spaces and find more complex boundaries. For prediction we use the function $y(x) = \text{sgn}(\langle w, \phi(x) \rangle + b)$ which can also be rewritten as:

$$y(x) = \text{sgn}(\sum_{i=1}^{|X|} \alpha_i t_i k(x_i, x) + b) \tag{4.8}$$

To support multiple classes one can train a separate SVM $y_k(x)$ for each class k and make predictions for new inputs as:

$$y(x) = \max_k y_k(x) = \max_k (\sum_{i=1}^{|X|} \alpha_{ik} t_{ik} K(x_i, x) + b_k) \tag{4.9}$$

Binary Decision Trees partition the input space into cuboid regions. The division is built as a branching structure: at each intermediate node the set of observations is split into two subsets according to a certain criterion, and leaf nodes contain the examples that fulfilled all the conditions along the path that leads to them. One feature and one threshold for it are chosen at each node, usually by trying to minimize the entropy on the resulting subsets. Large tree depths often lead to over-fitting, therefore trees are pruned before all the labels of the elements on the leaves are homogeneous.

In order to classify a new example, it is pushed down the tree following the conditions at each node until the corresponding leaf is found. Then, the

probability for a given class is reported as the ratio between the examples of that class in the leaf and the overall number of examples in the leaf.

Random Forests consist on building multiple Decision Trees with randomly selected subsets of features. Once the model is trained, class probabilities can be obtained by averaging the values reported on the different trees. This process usually leads to better performance because it reduces the variance of the model while maintaining the bias. Namely, whereas a single tree might be sensitive to the noise of the data, their average is much more robust.

4.2.4 Dataset used during design tests

In order to evaluate the algorithms, features and other designs of the pipeline, we needed segmented aerial images. However, land cover datasets, as we mentioned in the introduction of this chapter, lack enough detail for our Segmentation for Synthesis goals. Therefore, we manually classified the terrain shown in Figure 4.3, which we will refer to as Dataset Andorra from now on.

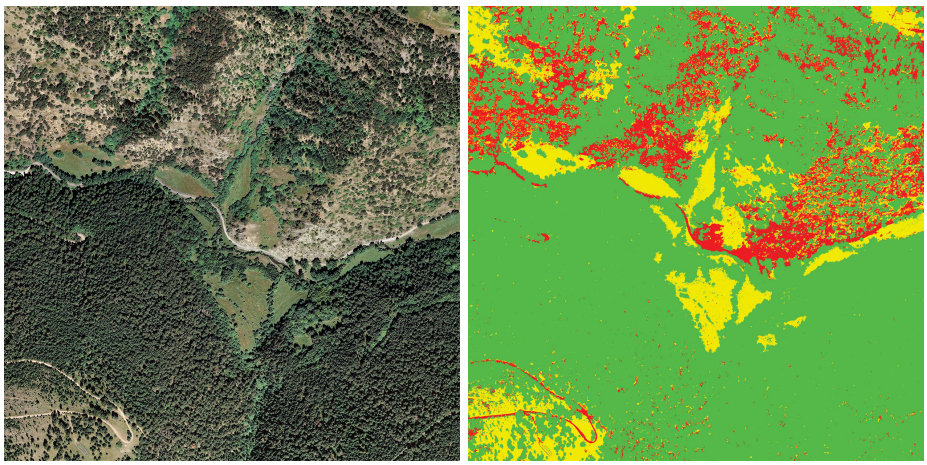


Figure 4.5: Dataset Andorra: aerial image and manual segmentation (red: ground, yellow: grass, green: tree).

The aerial image has 3600×3600 pixels, i.e. the terrain is a square of 900 meters per side. We used image processing software to assist the segmentation into three categories: ground, grass and trees. Figure 4.5 shows the aerial image and the manual segmentation. Note that the classes are not distributed homogeneously, there are much more tree pixels than grass or ground. However, this is usually the case on S4S both during training and inference steps, and we will address it on Section 4.2.8.

4.2.5 Segmentation algorithms comparison

Given the variety of learning algorithms, and each one with its own parameters to test and adjust, we decided to first select the one that better suits our Segmentation for Synthesis needs – the best possible accuracy with low inference and training times –, before adjusting the rest of the pipeline. All of the algorithms described in Section 4.2.3 can output the estimated per-class probability $\hat{p}_k(x)$, not only the best label k,

We performed this comparison using Dataset Andorra, with the feature set discussed in Section 4.2.2. We randomly sampled 250,000 pixels for training, and another 250k pixels for validation.

The algorithms were executed on KNIME [BCD+08], which contains implementations for all of them, so we just needed to input the data and adjust the parameters. We will discuss below the parameter setup of the Random Forest model. For the SVM, we tried two kernels: a linear kernel with gamma and bias set to 1, and a radial basis function with three different values of σ: 0.1, 1.0 and 10. On logistic regression tests, we did a maximum of 500 boosting iterations with a heuristic stop after 50 iterations of no improvement on a cross-validation set. Lastly, the only parameter to set up on the kNN model is the number of neighbors, which we report on their corresponding rows, and we did not assign them distance-based weights.

Algorithm	Accuracy	Training	Inference
Random Forest (N=50, D≤15)	**94.80%**	6 min 9 s	58 s
SVM Polynomial (linear)	93.80%	13 h 26 min	3 h 7 min
SVM RBF ($\sigma = 0.1$)	74.51%	33 h 9 min	20 h 40 min
SVM RBF ($\sigma = 1.0$)	**94.68%**	5 h 35 min	2 h 59 min
SVM RBF ($\sigma = 10.0$)	92.90%	45 h 30 min	4 h 46 min
Logistic Regression	93.78%	1 h 40 min	1 min 42 s
Logistic Regression (norm.)	93.76%	1 h 36 min	45 s
kNN (k = 20, norm.)	92.46%	n/a	3 h 52 min
kNN (k = 10, norm.)	92.51%	n/a	3 h 36 min
kNN (k = 1, norm.)	90.44%	n/a	2 h 59 min
kNN (k = 20)	75.25%	n/a	2 h 54 min
kNN (k = 10)	74.85%	n/a	2 h 18 min
kNN (k = 1)	64.47%	n/a	1 h 35 min

Table 4.1: Average times and accuracies of the different machine learning algorithms we tried. In some cases, features were normalized (indicated as *norm*).

Table 4.1 compares validation set accuracies and running times averaged over five executions (except for SVM, which we only executed once due to its high training time cost). Random Forest, SVM and Logistic Regression all provided similar accuracies. However, Random Forest were much faster to train than the other two, also offering the fastest classification times. We thus claim that *Random Forest is the best option for our S4S pipeline*.

The two parameters of Random Forests are the depth (D) of the trees, and the number (N) of trees. For each combination of $D \in \{5, 10, 15, 25\}$ and $N \in \{10, 25, 50, 75, 100\}$, we measured the average accuracy obtained on 10 different pairs of training and validation sets from Andorra. Figure 4.6 shows the accuracies obtained. Unless explicitly stated, all experiments in this article use N=50 and D=15 since it gives the best trade-off between computation efficiency and accuracy. Our results are similar to the ones obtained by [TMS12], who also chose N=50, D=15. Regarding the number of features each tree can randomly select, all tests were set up to use $\lceil \log_2 91 \rceil = 7$ features.

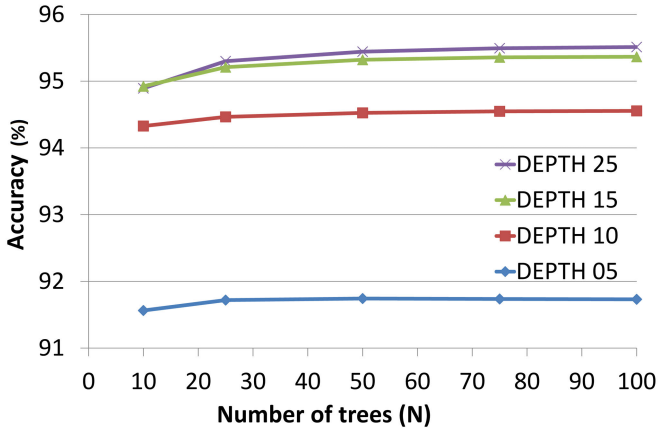


Figure 4.6: Accuracy of Random Forests depending on number of trees (N) and maximum depth (D).

4.2.6 Features on multiple resolutions

Some features in the chosen feature set are computed using sliding windows of up to 16 pixels (except LBP, which uses larger radii). Since our input data has a resolution of 25 cm/pixel, our feature set takes into account neighborhoods up to 4 m wide. We could increase this span either by increasing the sliding window size – which would also significantly increase feature extraction costs –, or using an image of the same area at a coarser resolution.

We performed an experiment using again Dataset Andorra to assess whether providing multiresolution features improved accuracy. We used a training set and a validation set, each one containing about 2% of the input image pixels (250 k samples), and measured the accuracy of a RF classifier (D=15, N=50) with features computed at varying resolution levels. We averaged the results over 10 executions.

Our baseline performance was given by a minimum feature set which included the four height features H (elevation, gradient magnitude and orientation), as well as the LBP features. LBP was included in the baseline because it is computed using radii up to 64 pixels. We then added the rest of the features (F) computed at varying resolutions: 25 cm/pixel, 50 cm/pixel, 1 m/pixel and 2 m/pixel.

Table 4.2 shows the resulting accuracies. Since the best accuracies were given by the highest resolution image (25 cm/pixel), we also tried combining the features of this one with one of the downsampled images. Results slightly improved in all three combinations; the best case was adding 1 m/pixel features, yielding a 0.4% gain in accuracy. However, the additional memory and computation time needed to obtain such a small accuracy improvement made us discard the multiresolution approach for S4S purposes.

Feature sets	Accuracy	Training	Inference
LBP, H	78.34%	93 s	22 s
LBP, H, F(0.25 m)	**94.85%**	91 s	28 s
LBP, H, F(0.50 m)	93.94%	94 s	28 s
LBP, H, F(1 m)	93.09%	97 s	28 s
LBP, H, F(2 m)	91.71%	94 s	28 s
LBP, H, F(0.25 m), F(0.5 m)	95.14%	134 s	37 s
LBP, H, F(0.25 m), F(1 m)	95.26%	106 s	36 s
LBP, H, F(0.25 m), F(2 m)	95.14%	103 s	36 s

Table 4.2: Average accuracies using different sets of feature resolutions.

4.2.7 Feature analysis

Besides excellent classifiers, Random Forests have been shown to be able to estimate the importance of the features and thus have been used both for feature quality estimation and feature selection [ČB10]. Quality estimates for feature j can be based on the difference between classification accuracy on the original data set and on a modified data set where the observed values for that feature x_i^j have been randomly permuted (thus preserving the original distribution) between examples [ČB10].

Figure 4.7 shows the relevance (as % of accuracy loss, following [ČB10]) of the tested features on randomly selected samples of a varied training set containing 8 different classes (we will introduce this set later, see Figure 4.17).

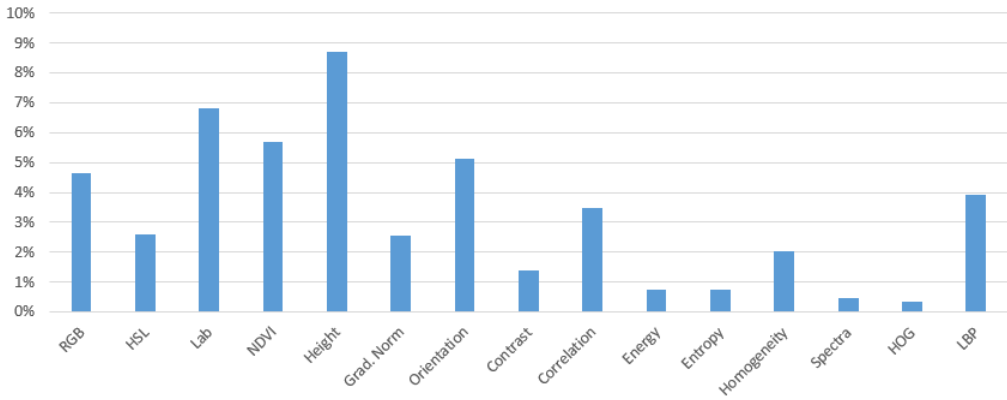


Figure 4.7: Accuracy loss after randomly permuting the samples of a certain feature group. Original accuracy was 91.6%.

The height feature group appears to play an important role in accuracy. However, it does not depend on the appearance of the patch containing the pixel but on its absolute location, and so the causal relation learned by RF (e.g. snow and water confined to specific height ranges) might be very specific to the chosen exemplars, and could generalize poorly on different scenes. Slope orientation can also introduce bias in the results. In our selected training set, we were not aware that bushes and grass regions had been selected only from a few orientations, and thus when classifying full terrains we obtained incorrect results (see Figure 4.8). Therefore, we decided to remove height and slope orientation (sin/cos pair) in our final pipeline, resulting in 88 features.

Notice as well the contribution of the NDVI feature, which is computed from the near-infrared channel. For datasets lacking this channel, we could still use our pipeline expecting an accuracy around 6% lower. Similarly, discarding some of the features (energy, entropy, spectra, HOG) leads to small accuracy losses (below 1%), and thus are potential candidates to be omitted to simplify the implementation. However, it is important to observe that after removing a feature, the losses shown in Figure 4.7 would no longer be valid, since the new Random Forest could then rely more strongly on previously less used features. In case we wanted to reduce the size of the feature set, this analysis should be repeated each time a feature is discarded, in order to select the next least useful feature.

Figure 4.8: Orientation features can introduce undesirable bias in the segmentation. For example, using the training from Figure 4.17 – in which inadvertently all bush and grass examples have been taken from similarly oriented slopes – produces an incorrect segmentation (left). Just removing the orientation features eliminates this bias and yields a better segmentation (right), with bushes and grass distributed more evenly as expected.

4.2.8 Training set sampling

Unlike typical classification problems, with fixed classes and large training sets, we deal with dynamic classes and user-provided exemplars. Given an exemplar image, one easy way to construct a training set is to ask the user to select manually a few homogeneous regions for each class. The system will then properly sample a subset of the labeled pixels and use them as training data. This option puts much less effort on the user than asking for a completely-segmented exemplar, a task that could take several hours for a reasonable sized dataset.

In this setup, we have to face three different problems:

- Non-representative neighborhoods: since training instances will be selected from the manually-identified homogeneous regions, the corresponding pixels are likely to have coherent neighborhoods that are not necessarily representative of arbitrary pixels (in contrast to selecting arbitrary pixels from a completely-segmented exemplar).
- Imbalanced data: in some exemplars, the ratio of the majority class (e.g. tree) to the minority class (e.g. scree) instances can be very large. This can be due to the predominance of some classes in the chosen exemplars, or to the fact that some objects (e.g. pathways) can be more difficult to select manually than others (e.g. forest). Therefore some degree of class imbalance is expected.
- Different class distributions: the class distribution largely depends on the scenario the exemplars have been chosen from. Therefore, the class distribution used for training (e.g. 80% trees) might be quite different from the true class distribution at inference.

Non-representative neighborhoods

We will first address the non-representative neighborhood problem. We conducted an experiment to evaluate how large was the effect of training with data coming from homogeneous regions. We used again the well-separated Dataset Andorra, for which a complete segmentation was available. Each pixel was considered to be *homogeneous* if all pixels within its $r \times r$ neighborhood had the same class (see Figure 4.9). Otherwise, the pixel was considered to be *heterogeneous*.

Then, we built three training sets: one with only homogeneous pixels, one with only heterogeneous pixels and one with 50% of each type. Each training set was randomly undersampled to about 260 K training examples. We trained a RF (D=15, N=50) for each training set and measured the resulting accuracies on a validation set including pixels from all across the image. Figure 4.10 compares the resulting segmentations on a small part of the image. Averaged accuracies over 10 executions are shown in Table 4.3.

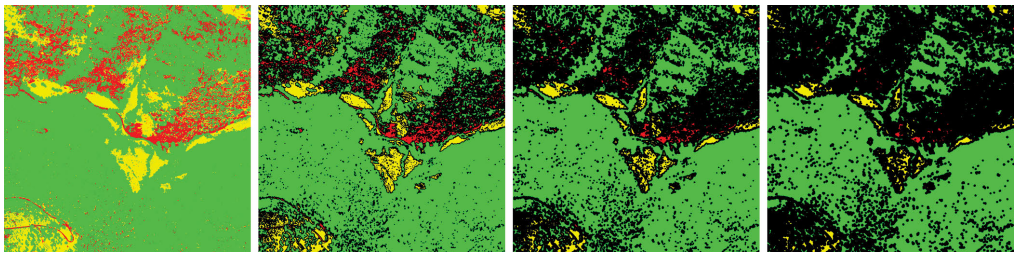


Figure 4.9: From left to right: segmentation of Andorra, and pixels with homogeneous neighborhood for $r = 5, 10, 15$ respectively. Pixels with an heterogeneous neighborhood are those in the black regions, although they have a class.

Window size	Homogeneous	Half and half	Heterogeneous
$r = 2$	92.9%	**95.2%**	93.7%
$r = 5$	89.7%	**95.1%**	94.9%
$r = 10$	86.8%	94.9%	**95.1%**
$r = 15$	84.6%	94.9%	**95.1%**

Table 4.3: Average accuracies depending on the training set selection.

The first column in Table 4.3 shows that as we increase the window size r, which means that samples with homogeneous neighborhoods are taken farther from the image-space boundaries between classes, the accuracy of the classifier trained only with homogeneous neighborhood pixels decreases. For small values of r, the window used by some of the features is still able to incorporate information of bordering areas, but for large values of r all the windows are always

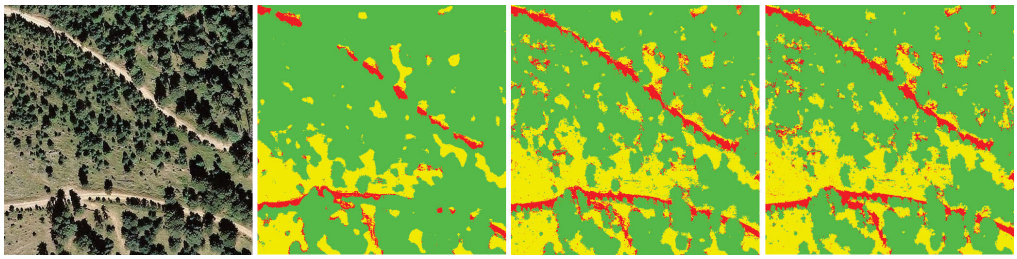


Figure 4.10: From left to right: part of the photograph, classification using homogeneous neighborhoods, classification using heterogeneous neighborhoods, and manual classification.

describing the content of a single class. This makes the trained model more error-prone in the pixels close to class boundaries, as shown in Figure 4.10.

On the other hand, if we train the classifier on heterogeneous neighborhoods only, increasing r leads to better performance of the classifier. This also makes sense, as for small r the features computed over a window always take into account other nearby classes, and the trained model has never been shown observations from same-class regions. However, in this case the difference in accuracy between small and large values of r is much smaller, probably due to the fact that some of the features are computed using only the information in the pixel and it is easier to generalize to homogeneous regions after training with heterogeneous samples, than the opposite case.

Finally, if we train using half the samples from each distribution, the accuracy remains pretty much constant. Note that this is not exactly the same as allowing the samples to come from anywhere in the image, as for different window size r the proportion of pixels considered homogeneous or heterogeneous varies and does not necessarily represent 50 and 50% of all the image.

These results show the positive impact of including some examples near the (image-space) class boundaries. Both the heterogeneous and the 50%-50% training sets resulted in high accuracy, since both include examples with representative neighborhoods. However, if training examples are taken from homogeneous regions, we cannot guarantee representative neighborhoods as long as user-selected region boundaries might not match class boundaries. Fortunately, reducing the window size used for the homogeneity criterion (i.e. allowing homogeneous pixels to get closer to class boundaries) leaded to reasonable accuracy (above 90% in the experiment, see Table 4.3).

So far we have shown that heterogeneous neighborhoods are important, but we have been sampling randomly from the pixels that met the homogeneity criteria defined by the window size r. We want to test whether there is a better sampling strategy. We will compute a distance field D_k for each class k in the segmentation, storing the image-space distance $D_k(x, y)$ from the pixel (x, y) to the closest pixel labeled as another class $j \neq k$. We normalize D_k to 0..1 range, and add a random value as follows:

$$S_k(x, y) = \lambda \cdot D_k(x, y) + (1 - \lambda) \cdot \text{rand}(0,1)$$

Then, when we construct our training set, we will order all the pixel samples of each class k according to S_k and choose them accordingly. For $\lambda = 0$, they are ordered completely randomly. For $\lambda = 1$, they are ordered according to the distance field. So if we choose the N samples with highest S_k value, we are selecting the N samples that are farther from another class, i.e. we are going towards the center of the segmented regions. Conversely, if we choose the N samples with lowest S_k, we are selecting the N samples that are closest to a different class, i.e. we are going towards the boundaries of the segmented regions. Varying λ yields different degrees of randomness/distance in the sampling strategy, as shown in Figure 4.11.

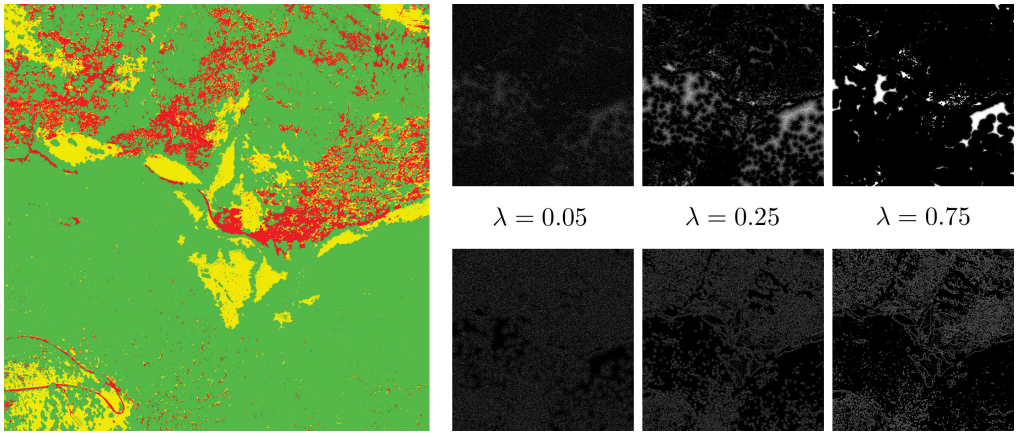


Figure 4.11: Sampling the training set examples using S_k for different values of λ and towards the segmented groups center (top) or towards the borders (bottom). Pixels in white represent selected samples for the training set. Contrast and point size has been adjusted for better visualization.

After trying different values of λ, the results shown in Table 4.4 indicate that best accuracy is always obtained for $\lambda = 0$, i.e. for a completely random sampling in the image.

λ	min S_k (to border)	max S_k (to center)
0	**94.6**	**94.6**
0.05	94.6	94.5
0.10	94.5	94.1
0.25	94.4	91.4
0.50	93.9	89.2
0.75	93.1	88.1
0.90	91.5	85.9
1	88.9	87.5

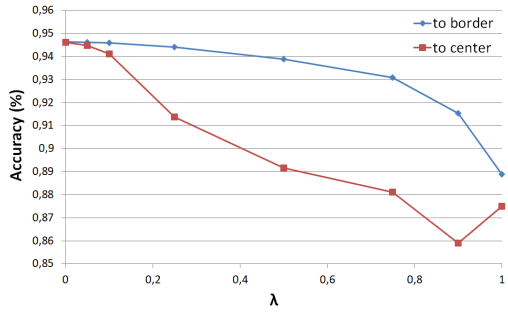


Table 4.4: Average accuracies when gradually sampling on forcing training samples towards border or towards center of the segmented regions.

Moreover, these results also agree with those obtained in the previous experiment. Sampling with preference towards the border (i.e. heterogeneous neighborhoods) does not penalize accuracy as much as sampling towards the center (homogeneous neighborhoods). Moreover, lower values of λ (more randomness, so sampled distribution is not only homogeneous or heterogeneous) achieve better results, like we obtained with the 50/50 distribution in the previous test.

In conclusion, taking into account the results from both experiments, we can now define the best strategy for obtaining a training set. Since a complete pixel-wise segmentation of the exemplars is a very time-consuming task, a good trade-off is achieved by asking users to **select *nearly-maximal* homogeneous regions**, with its boundary near examples from other classes. Then, the training set can be randomly sampled within those regions. Following this guideline, exemplars are still easy to label (with respect to a complete pixel-wise segmentation) while still resulting in reasonably good classification accuracy. Figure 4.17 shows a set of exemplars with nearly-maximal homogeneous regions manually segmented.

Imbalanced data and dissimilar class distributions

We now address the problems of imbalanced data (one class having much more exemplars than other classes) and dissimilar class distributions (training distribution different from target distribution), two problems than can easily appear in a S4S project.

First, we analyzed the impact of different distributions on Andorra. The real distribution on this set is 74.5% tree, 13.7% grass and 11.8% ground. Using a

training set consisting of 74% tree, 14% grass and 12% ground gave an accuracy of 94.72%. Using the same amount of training samples distributed equally among each class (33.3%), gave 93.39% accuracy, and inverting the proportions (i.e. 13% tree, 43% grass and 44% ground) resulted in 90.30% accuracy. Therefore, even if training and target class distributions are very different, it is still possible to achieve good accuracy segmentation.

Next, we executed a more detailed test varying the percentage of the class tree samples in the training set, and distributing the rest of the samples equally among grass and ground classes. Figure 4.12 shows the curve obtained. As expected, the best segmentation results are achieved when the training distribution roughly matches the target distribution (vertical dashed red line). However, we only observe small changes in the overall accuracy even with completely erroneous distributions.

Thus, we decided to give the user the opportunity to provide the expected class distribution on the type of images to be classified as rough estimates of the per-class percentages. We sample the training set according to the user-provided distribution, train the RF, and use it to segment the input image. This way the user has some control on the desired result, accuracy is less sensible to the class distribution on the selected exemplars, and potential model overfitting, if any, is biased towards user expectations.

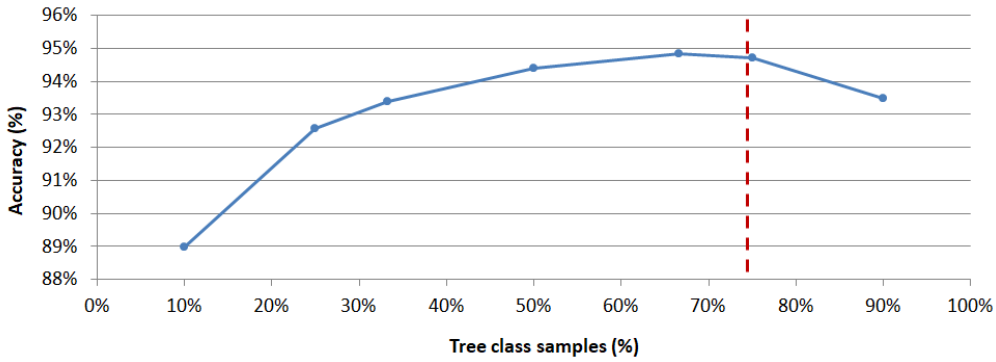


Figure 4.12: Accuracy obtained for different distributions of the training set samples.

4.2.9 Smoothing class probabilities

Since the output of our classifier is a set of per-class pixel probabilities, rather than just the identifier of a class, we could apply post-classification filtering techniques. Although including feature descriptors computed on the pixel neighbourhood implicitly produces smoother labellings, we explored different fast options working on the probability maps. Therefore, users can apply post-classification filtering techniques to correct noise or small errors in the final labeling.

Schindler [Sch12] compares different smoothing methods that can be used to produce a final labeling from class probabilities. Due to their speed and simplicity, we decided to implement the three filtering methods he presents: majority voting, Gaussian filter and bilateral filter. Additionally, the user can provide, for each label, the minimum expected footprint σ of the objects belonging to this class. For example, a single pixel (0.25 m in our case) labeled as tree surrounded by non-tree pixels is very likely to be erroneous, while a single grass pixel may be plausible.

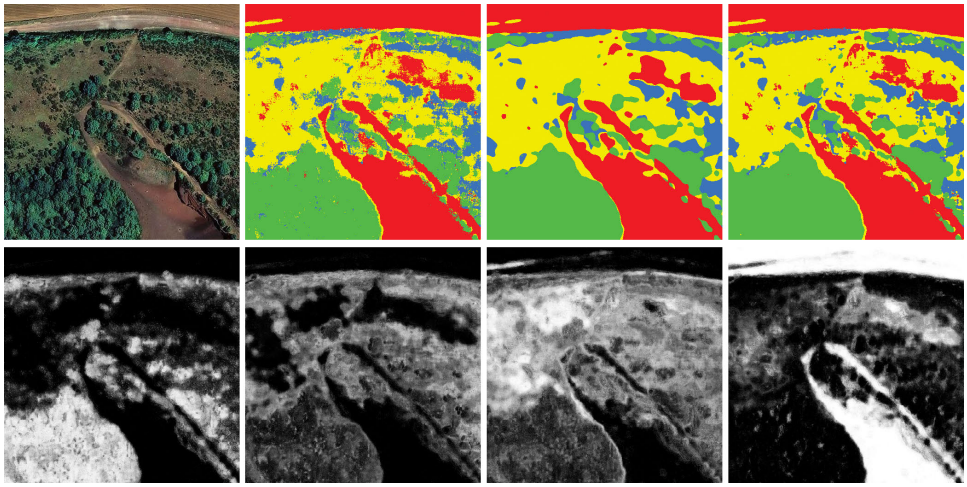


Figure 4.13: Top row: input image, classification output without filtering (MAP), bilateral filter with $\sigma = 4$ pixels, bilateral filter with $\sigma_{\text{tree}} = 4$, $\sigma_{\text{bush}} = 2$, $\sigma_{\text{grass}} = 1$ and $\sigma_{\text{ground}} = 0.5$ pixels. Bottom row: probability maps of classes tree, bush, grass and ground.

Figure 4.13 shows an example from Croscat dataset (4 classes: tree, bush, grass and ground) in which no filtering results in a very noisy classification, while a bilateral filter over-smooths some grass or bush regions. However, filtering with a different σ on each probability map eliminates undesired single-pixel trees while keeping small vegetation clusters.

4.2.10 Cost-sensitive classification

Recall that we aim at replacing pixels by synthetic detail in the context of free-camera applications, supporting large zoom levels and close-up views. Misclassified pixels are likely to have a varying perceptual impact, depending on the predicted/true labels and on the specific detail synthesis techniques that will be applied to the output labeled images. For example, misclassifying terrain with any of the other vegetation classes (tree, shrub, grass) is likely to have a higher visual impact than e.g. confusing shrubs and trees.

Given the estimated class probabilities $\hat{p}_k(x)$ for a test pixel x, its final class $\hat{y}(x)$ is computed as the one minimizing the expected misclassification costs,

$$\hat{y}(x) = \arg \min_{i=1,\dots K} \sum_{j=1}^{K} c_{i\,|j}\, \hat{p}_j(x) \tag{4.10}$$

where $c_{i\,|j}$ is the user-defined cost of misclassifying as class i a pixel whose true label is j.

Without loss of generality [OGG08], elements on the diagonal of the cost matrix $C = c_{i\,|j}$ can be assumed to be 0, and elements off-diagonal can be assumed to have a positive cost. Scaling C by a positive constant does not affect the optimal decision, so the minimum non-zero cost can be assumed to be 1. For $K = 4$ classes, the cost matrix requires only 11 values to be provided by the user.

Table 4.5 shows an empirical cost matrix example where misclassification of terrain pixels as tree incurs the highest penalty, since this would cause synthetic trees to pop up unexpectedly when zooming into a terrain region. Figure 4.14 shows the effect of applying this cost matrix.

Predicted / Truth	Tree	Bush	Grass	Ground
Tree	0	1	3	7
Bush	2	0	3	6
Grass	3	2	0	4
Ground	4	4	4	0

Table 4.5: A reasonable cost matrix for vegetation and ground classes.

Costs should be based on a perceptual study considering the particular detail amplification techniques. In Section 4.3.2 we provide some insights on how to derive well-founded costs based on perceptual differences between pairs of classes.

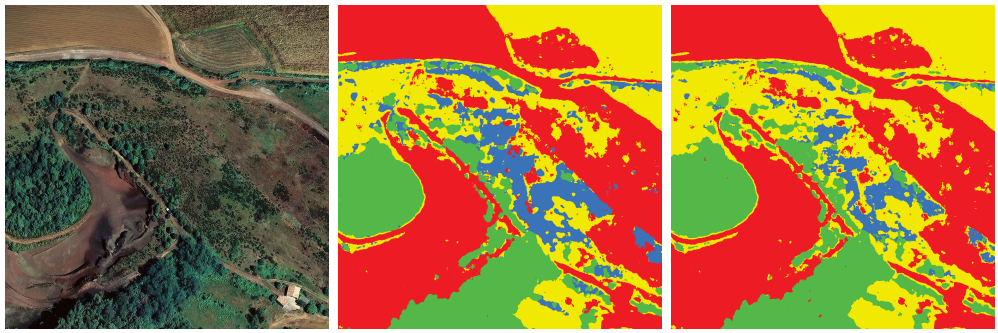


Figure 4.14: Segmentation of Croscat terrain into four classes: tree, bush, grass and ground. Aerial image (left), segmentation after filtering (middle), and afterwards applying the misclassification cost matrix from Table 4.5 (right).

4.2.11 End-to-end segmentation using Convolutional Networks

As mentioned in Section 4.2.2, in the past years there has been a growing interest into applying Deep Networks to a variety of problems, as modern GPU hardware and novel techniques have enabled their efficient training and they have proven to outperform previous methods in many domains. One of the advantages of Deep Nets is that they can provide an end-to-end learning framework, i.e. there is no need for feature engineering as the input of the network is directly the image.

Recently, Deep Learning has been successfully applied to image segmentation using Fully Convolutional Neural Networks (FCN). In [LSD15], Long et al. convert image classification networks like VGG-16[SZ14] into per-pixel segmentation networks by turning the fully-connected layers into convolutions, thus being able to output classification maps from any input size. Since typical classification architectures apply several pooling layers that reduce the size of the original image, the output classification map has lower resolution. The FCN proposed by Long et al. performs upsampling as well as fusing information from previous layers in order to output a per-pixel segmentation. They are able to reduce the maximum upsampling required from 32× to 8×, but explain that fusing further layers to reduce the upsampling only yields slight improvements at a larger cost.

In order to evaluate the use of FCN in our pipeline, we used the implementation and trained weights of the network `FCN-8s-atonce` using Caffe [JSD+14], as provided by the authors. This network performs an 8x upsampling and reported their best performance in the PASCAL VOC segmentation dataset. Figure 4.15 shows the architecture of this network.

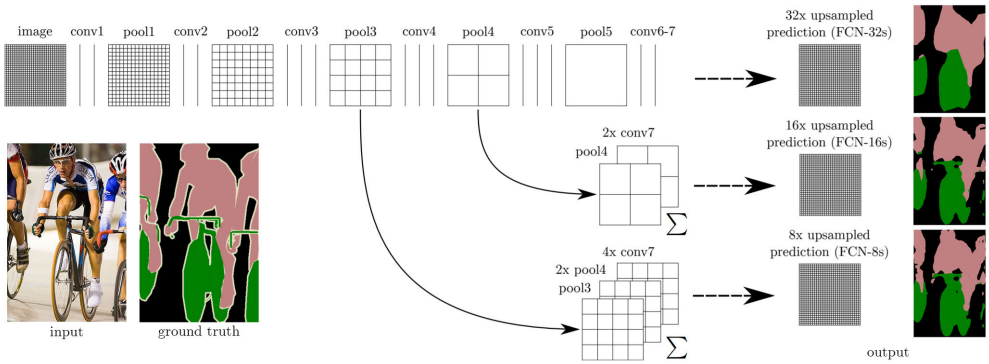


Figure 4.15: Architecture of the FCN proposed by Long et al. [LSD15]. Each time a *Pooling* layer is applied, the image is reduced to half the size. Therefore, the prediction map from *conv7* layer has to be upsampled 32x. However, better results are obtained if the prediction by *conv7* is upsampled only twice, merged with the prediction from *pool4*, and upsampled 16x. Taking it yet one step further, the prediction map only needs to be upsampled 8x. The authors show that further repeating this process yields diminishing gains, so they stop at 8x. Figure adapted from their article [LSD15].

First, we modified the last layer of the net to match our classes, and retrained using our fully-segmented Dataset Andorra split as 75% training and 25% validation. After 10 training epochs (11 min 38 s), the obtained accuracy was 89.26%. Classifying a 1000×1000 image took 15.42 s in CPU, 0.84 s in GPU. Figure 4.16 shows the output of this network. Note that the output is smooth, as the network is in fact learning an $8\times$ upsampling of the per-pixel labeling in its last layers. Modifying the network to avoid this upsampling was out of the scope of our work.

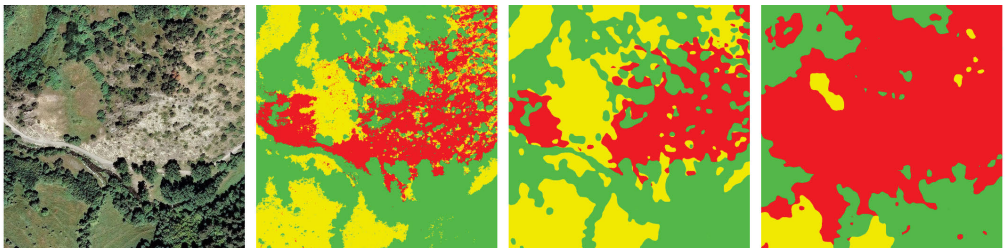


Figure 4.16: Part of Andorra terrain set. Left: aerial image, center-left: ground truth, center-right: classification using FCN-8s-atonce [LSD15] with fully-classified training set, right: classification using FCN-8s-atonce [LSD15] with nearly-maximal homogeneous regions training set.

However, in this first test we trained the FCN using a complete per-pixel segmented dataset. Recall that our goal is to allow the user to train the classifier using a few homogeneous training regions, as in our experiments. If we train the net in this way – by letting the loss function ignore pixels outside training regions, using a special label –, results are much worse as shown in Figure 4.16 rightmost image, which shows the classification after 100 training epochs (4 h 40 min). This can be caused by the net not being allowed to learn the shapes of boundaries between classes. Even after a huge training time, results are still far from an acceptable segmentation, effectively rendering FCNs unusable for our S4S pipeline.

4.3 Results

In this section, we will analyze the results of our segmentation pipeline. As mentioned before, obtaining a fully per-pixel segmented dataset is nearly impossible, as it would take an immense effort. Therefore, we will compare our results on a per-region basis. Moreover, since our goal for the segmentation is to add plausible detail on terrains, we will evaluate the quality of our segmentation with a user study.

4.3.1 Training set

We downloaded several small terrain regions from Catalonia, using the web map services (WMS) provided by ICGC [12]. For each terrain, we downloaded the aerial RGB and Near Infrared images, and the 5 m DEM. We manually segmented nearly-maximal homogeneous regions using standard image editing software, computed features for these pixels, and stored the list of examples for each class. Figure 4.17 shows this training set.

4.3.2 Classification accuracy of human labelers

We conducted a first experiment to analyze how humans classify *regions* in aerial images. The purpose was twofold: to obtain ground truth labels to test our algorithm with, and to analyze the difficulty and resulting accuracy of human labelers. Since per-pixel classification is a long and tedious task, we selected a total of 56 uniform regions (not necessarily continuous) from 8 datasets (Figure 4.19-top) representing a variety of forest and rural areas around Catalonia.

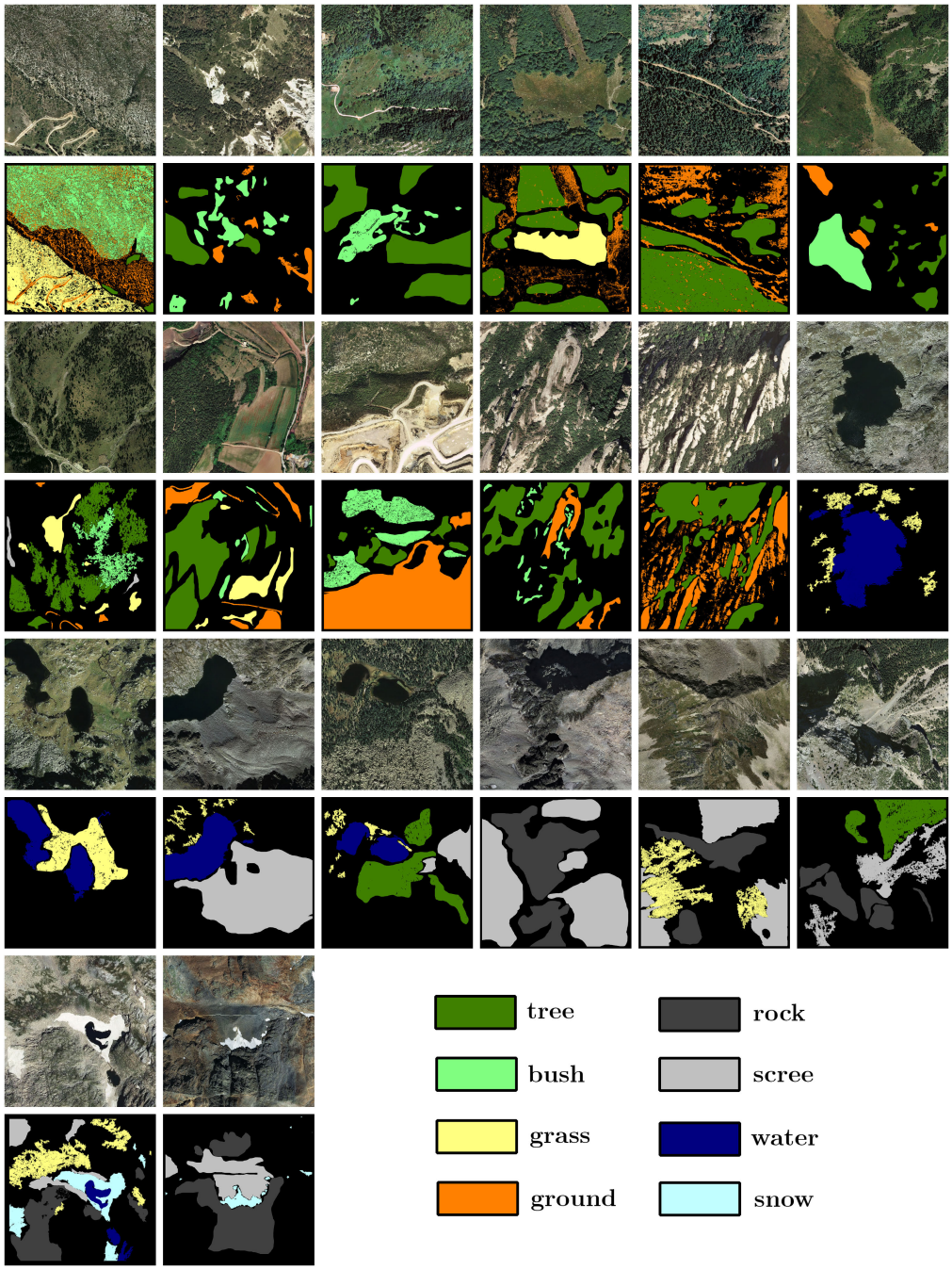


Figure 4.17: Terrains on the training set and manually segmented regions.

For each region, we asked users to choose between four labels: tree, bush, grass and ground, as well as three additional labels expressing doubts between two classes: tree or bush, bush or grass, grass or ground. During the test, one region at a time was presented to the user as a blinking, semi-transparent overlay on top of the current aerial photograph (Figure 4.18). Users were able to zoom in and out using the mouse wheel. Twenty-three users (19 male, 4 female, ages 22-45, normal-sighted, most of them familiar with computer games) participated in the experiment.

Figure 4.18: Interface of the first user study, users were asked to provide their perceived label for the region highlighted using a red blinking overlay.

Ground-truth labels were not directly available, so we computed and compared against the crowdsourced labels. One way to obtain these labels is through majority voting, i.e. taking the most voted label at each region. However, this strategy does not account for how good each individual labeler is, nor the intrinsic difficulty of labeling each particular region. Following Whitehill et al. [WWB+09], we inferred the label of each region using a probabilistic graphical model in which observed labels depend on three causal factors: difficulty of the image, expertise of the labeler, and the true label of the image. Using an Expectation-Maximization approach, all three causal factors can be inferred from the observed labels. For our special case in which a user expressed doubt between two classes, we treated this as two different observed labels for that region given by the same user.

Figure 4.19-middle shows the resulting ground truth labels using the implementation provided in [26]. These labels will be used in Section 4.3.3 to test our classification pipeline.

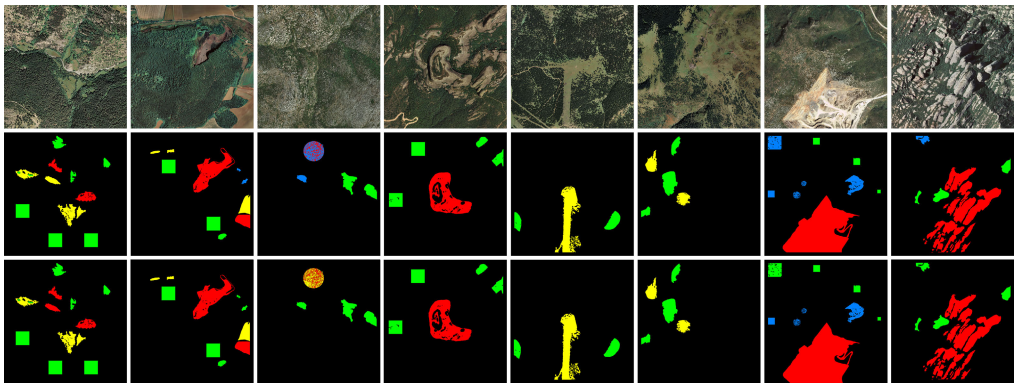


Figure 4.19: Datasets used in the first user study (top), *ground-truth* labels computed following [WWB+09] (center), majority class given by our per-pixel classification (bottom). The datasets shown are (from left to right): Andorra, Croscat, Garraf, Rocacorb, Peguera, Setcases, Garraf-Quarry, Montserrat. For better visualization, we have used a different color scheme: red (ground), green (tree), blue (bush), yellow (grass).

Alongside ground truth labels, we can also read from the probabilistic model the region difficulty parameter β. Whitehill et al. [WWB+09] model image difficulty as $1/\beta \in [0, \infty)$. When β is close to 0, the image difficulty is very high and even the most skilled users become just pure random labelers. Figure 4.20 shows the β parameters averaged by input image (left) and by ground truth label (right). Notice that difficulty estimates exhibit large variance across classes and datasets. In particular, bush regions are hard to classify, and consequently images that contain regions similar to bushes are more prone to misclassification errors.

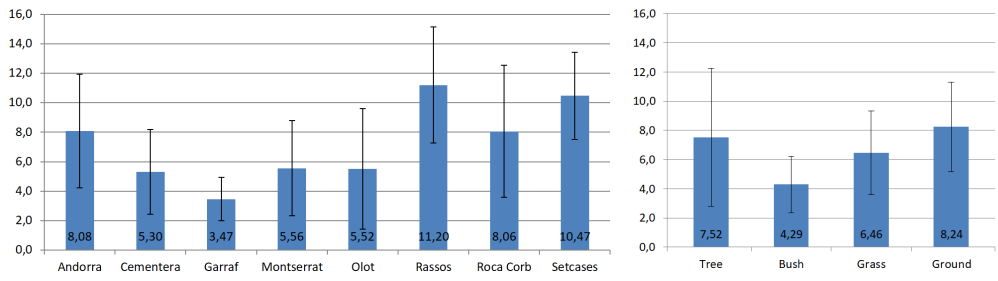


Figure 4.20: Average classification easiness for each dataset (left) and for each class (right). The lower the value, the higher the difficulty. Intervals represent the standard deviation.

Table 4.6 shows the resulting users' confusion matrix. Participants were relatively good at distinguish between high and low vegetation (tree/grass), but bushes were easily confused with trees or grass. These confusion matrices provide a perceptual basis to measure how easy a label can be misclassified as another label, and thus provide a solid foundation to build perceptual cost matrices: in the context of S4S, the higher the confusion rate between two classes, the lower should be the corresponding misclassification cost. The matrix shown in Table 4.5 (Section 4.2.10) is an example derived from this confusion matrix.

Predicted / Truth	Tree	Bush	Grass	Ground
Tree	85.4%	14.6%	0.0%	0.0%
Bush	11.1%	63.9%	23.7%	1.3%
Grass	0.0%	0.4%	95.4%	4.0%
Ground	0.0%	0.2%	8.3%	91.5%

Table 4.6: Confusion matrix as percentages per label.

4.3.3 Classifier accuracy vs human accuracy

The user study above shows that identifying tree, shrub, grass and terrain classes on 25 cm/pixel aerial images is not an easy task, and that users often express doubt or misclassify well-delimited regions. This suggests that a manual segmentation from a single user is just a biased estimate of the *ground truth*. Therefore, we decided to conduct a follow-up study. We asked eight new users (7 male, 1 female, ages 22-32, normal-sighted, all of them familiar with computer games) to classify the regions in Figure 4.19, but this time only the four classes were available to them, as they were not allowed to express doubt.

The number of regions each user marked differently from the 23-subject ground truth ranged between 5 and 10, concretely: 5, 5, 5, 8, 8, 9, 10, 10. The average is 7.5 incorrectly classified regions out of 56 (87% accuracy). This shows again the inherent difficulty in classifying some regions in the aerial images.

We compared this user accuracy with that of our classifier, keeping the majority class among all the pixel labels inside each of the regions to produce a unique label of the whole region. The classifier output showed 7 misclassified regions (see Figure 4.19 bottom row), similar to the expected error from human labelers. Figure 4.21 shows examples of these missclassified regions. All but one of these error regions correspond to what users perceived as *ground truth* bush and were confused with trees or grass, a distinction that is also usually hard for humans as we explained in the previous section. During this second study, most

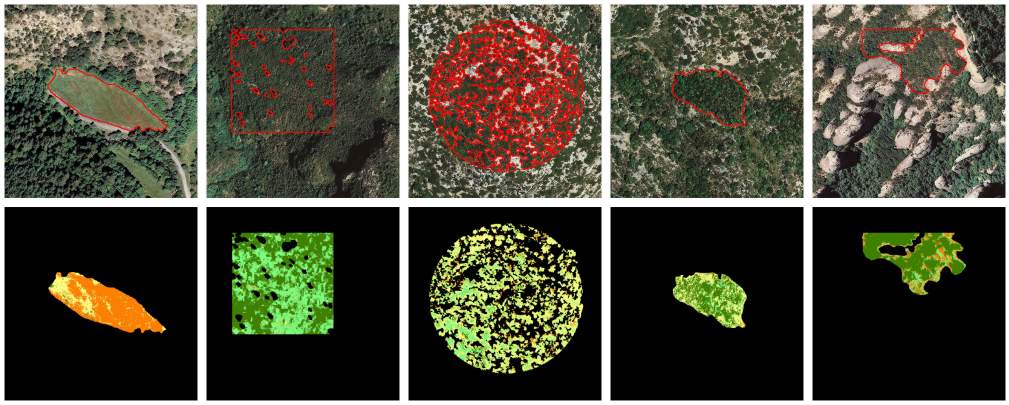


Figure 4.21: Some misclassified regions and their corresponding per-pixel classifications. The first region was perceived by the users as grass, and the rest as bush. The majority class from the classifier results inside each region is, from left to right: ground, tree, grass, tree, tree.

users expressed difficulty on determining whether a highlighted region belonged to the bush class. Also, they had to zoom in and out of the image and look at other areas to compare vegetation appearance in the rest of the image before choosing.

4.3.4 Visual validation of complete segmentations

We used our classifier to perform full per-pixel classifications of different datasets. The first eight datasets consisted of a 3600×3600 aerial image (rgb and infrared), and the corresponding digital elevation model. These datasets were segmented by our algorithm into four classes: tree, bush, grass, and ground. The last three datasets were bigger (16000×16000 pixels) and included four additional classes: scree, rock, water, and snow.

The test hardware was a single PC equipped with an Intel Core i7 at 3.40 GHz and 16 GB of RAM.

We used OpenCV and Matlab to extract the 88 features of each pixel. For a km^2 of terrain (4000×4000 pixels), computing the height, color and NDVI features took less than a second, spectral features 2 min and 40 s, HOG 12 s, GLCM 3 min, and LBP 2 min and 45 s. On average, the feature extraction performance is 32 s/Mpixel, or 8.75 min/km^2. Note that features only need to be extracted once per terrain.

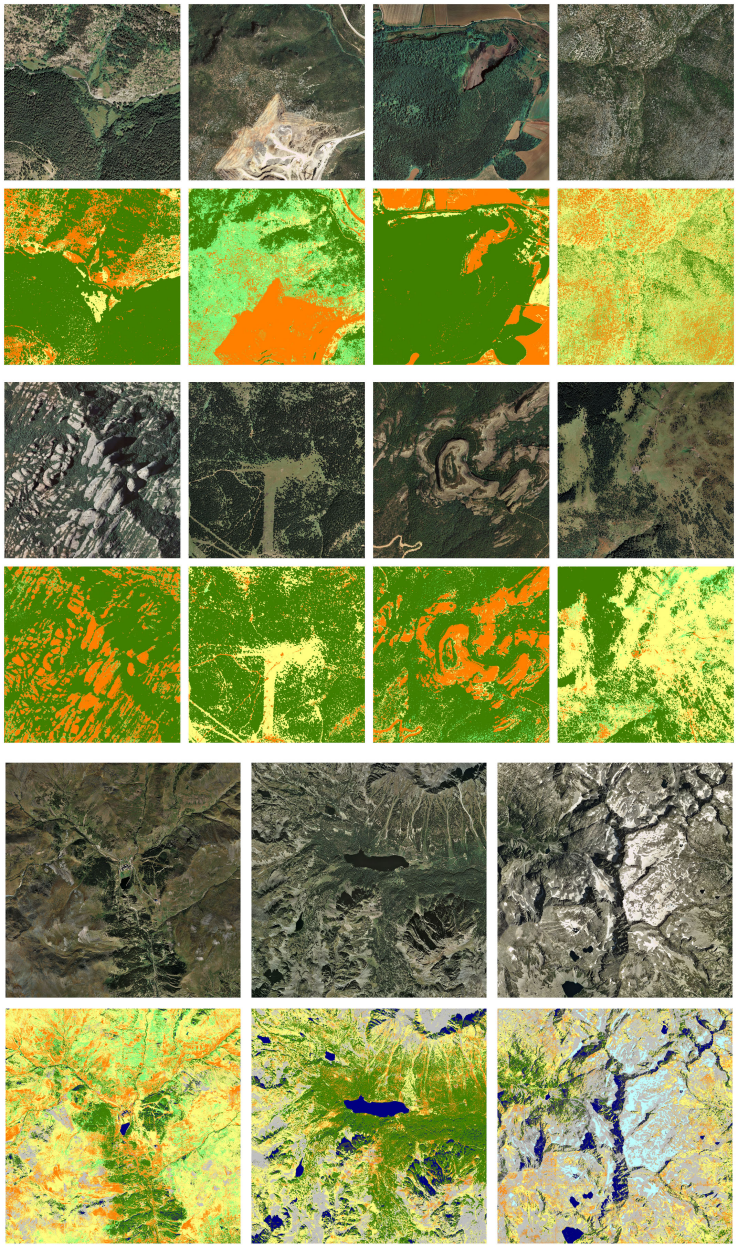


Figure 4.22: Top and middle: Andorra, Garraf-Quarry, Croscat, Garraf, Montserrat, Peguera, Rocacorb, and Setcases, segmented using 4 classes (tree, bush, grass, ground). Bottom: Besiberris, Nuria, Sant Maurici, segmented into 8 classes. Color legend as in Figure 4.17.

The classifier, as well as the various tests presented in previous sections, was implemented using KNIME [BCD+08]. The training set was randomly sampled from the gallery of nearly-maximal homogeneous regions that was easily segmented by an expert (Figure 4.17), the proportion of samples of each class being user-adjustable and reported in Table 4.7. These training regions are not contained in any of the 11 sets to classify. Training a random forest using 400K samples took about 2 min. Classification took around 21.5 s/Mpixel: on average 5 min 45 s per square kilometer or 4000×4000 pixels patch.

Dataset	Tree	Bush	Grass	Ground	Rock	Scree	Water	Snow
Andorra	75	-	15	10	-	-	-	-
Garraf	20	10	40	30	-	-	-	-
Croscat	65	5	10	20	-	-	-	-
Rocacorb	60	5	10	25	-	-	-	-
Garraf-Quarry	30	30	10	30	-	-	-	-
Montserrat	40	15	5	40	-	-	-	-
Setcases	35	20	35	10	-	-	-	-
Rassos	40	5	40	15	-	-	-	-
Nuria	15	10	30	30	4	10	1	-
Sant Maurici	30	5	20	10	15	15	5	-
Besiberris	4	1	12	8	40	25	5	5

Table 4.7: Percentage of samples from each class per dataset.

Figure 4.22 shows the 11 datasets as well as the output of our classifier – as the most probable class – before applying smoothing or cost matrix. These segmentations can be assessed only visually since no ground-truth is available.

4.3.5 Detail synthesis application

The most relevant application of the proposed pipeline is to synthesize terrain and vegetation detail depending on the per-pixel labels. Here we show some sample images obtained by a straightforward approach.

Regions labeled as trees or shrubs were covered by a blue noise pattern, each point representing the location of a synthetic tree/shrub. The minimum distance between generated points was determined by the size of the synthetic vegetation models. In our case the radii of tree and shrub billboards were respectively 20 and 8 pixels (which given the 25 cm resolution of the aerial images correspond to 5 and 2 meters). Grass regions were enhanced by instantiating 3D models of grass and flowers.

Terrain regions were detailed on-the-fly through displacement mapping shaders

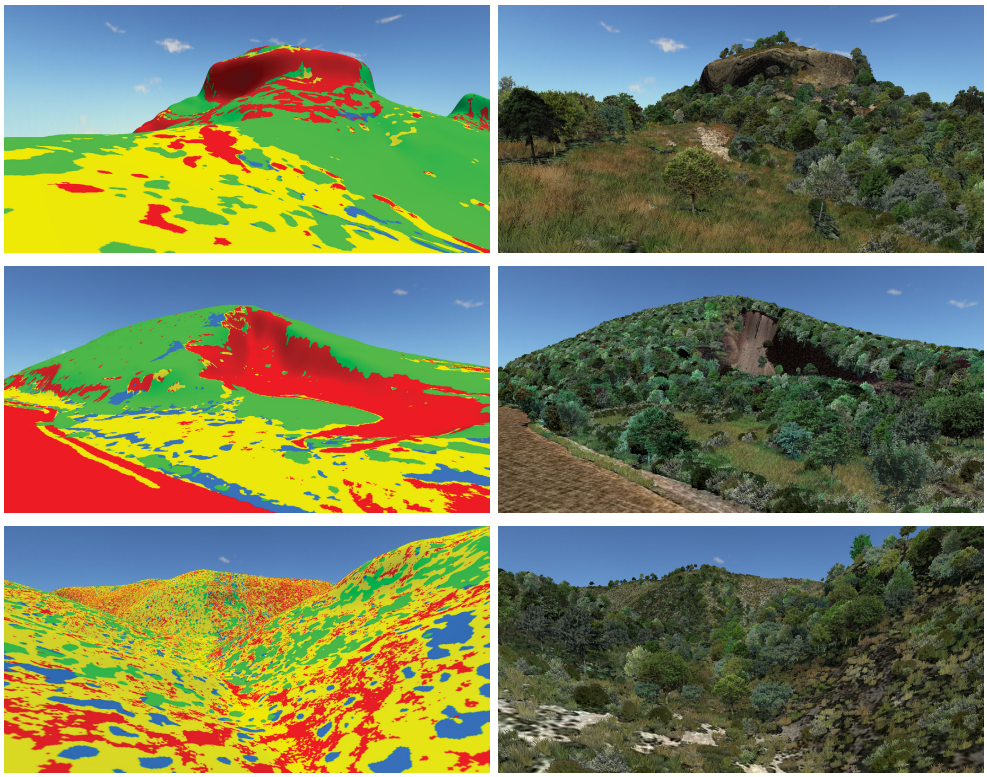


Figure 4.23: Example application: vegetation synthesis on terrain. Rocacorb, Croscat and Garraf shown with the result of the classifier (left) and the rendered images with vegetation on the terrain (right).

perturbing the original DTM according to Fractional Brownian Motion noise. Fragments covered by water and snow were rendered using specific shaders. Figures 4.23 and 4.28 show some resulting images.

4.3.6 Adding new classes and training examples

Adding new classes or specializing an existing class is a use case that we had in mind when designing this S4S pipeline. Here, we will show how it can be easily done using the provided workflow. Suppose a user wants to classify the terrain shown in Figure 4.24 into the following set of classes: tree, bush, ground, rock, beach, sea and road.

First, it is possible to just apply the classifier using the previously built training set (Figure 4.17) to see the results, as in Figure 4.24 top-right. Then,

for the three vegetation classes, the user decides that the training set already contains a good collection of samples. Thus, it is only needed to provide extra training samples for the new beach and road classes, and for the sea class, as it has a very different appearance than the water class examples segmented from lakes. Similarly, a small region for rock class has also been added because these rocks are also very distinct from the ones in the training set. Figure 4.24-middle shows the regions marked by the user, in less than 5 minutes using standard image processing software.

Figure 4.24 bottom-right is the output of our classifier after including the new samples and classes on a training set, training a new Random Forest model and running the classification for this terrain.

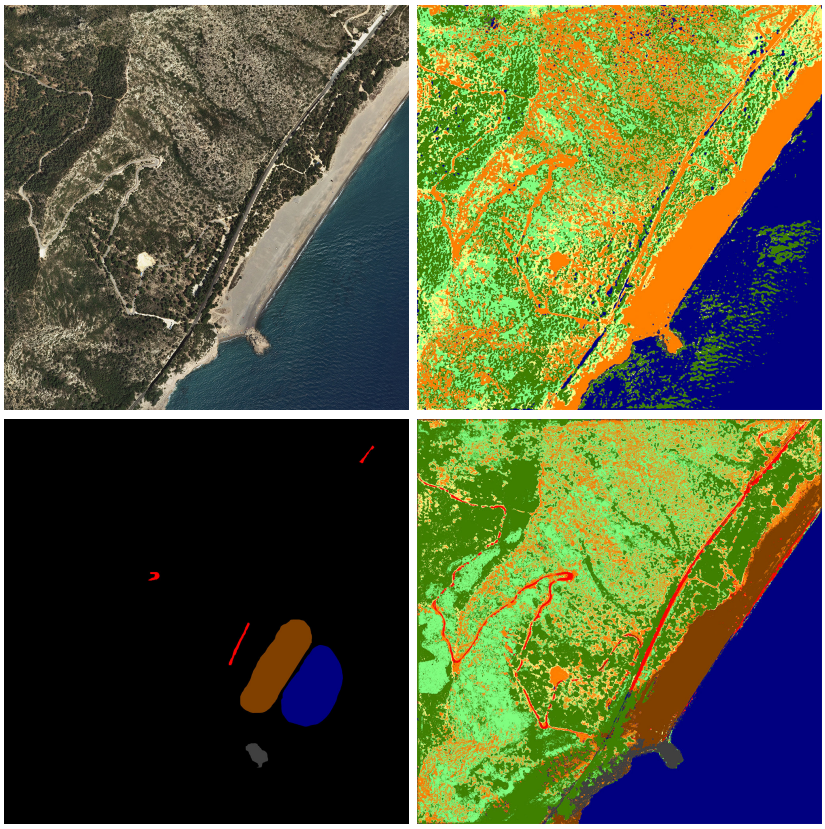


Figure 4.24: Segmentation of Torn dataset, in which new classes have been combined with the existing training set. The user provided new classes: road (red), beach (brown), and sea (blue), as well as additional samples for rock class. On the right column, we show the result using the previous training set (top) and after adding the additional segmentations from the user (down).

4.3.7 Detail synthesis validation

We conducted a final user study to evaluate the quality of our segmentation pipeline in the context of detail synthesis. We selected a set of 30 viewpoints on a large DEM (Figure 4.22, Nuria terrain), and rendered the scene with procedurally-generated detail (vegetation, rocks, water) placed according to either our segmented image (OURS) and an official 5 m/pixel land cover map (LCM). The set of images used in the study can be found in the supplementary material published in [ACC+18].

Our study consisted of a *selection task*, in which participants were presented both images (OURS/LCM) and had to choose which one showed a better placement of the synthetic elements, and a *scoring task*, in which participants were shown one particular image at a time (either OURS or LCM) and had to indicate the plausibility of detail placement using a 4-point Likert scale. Both tasks started with a tutorial video and displayed instructions throughout the trials.

For the selection task (Figure 4.25-left), participants were shown 12 random image tuples (from the set of 30), where each tuple consisted of rendered images from the same viewpoint, using either our segmented image or the LCM. Users were requested (forced choice) to select the best image in terms of the placement of the added elements. A reference image showing just the DEM textured with the orthophoto was also shown. For all participants, the order of the tuples was randomly chosen, along with the order of the images within the tuple.

For the scoring task (Figure 4.25-right), we randomly selected 12 images as in the first task, but now each image was shown separately rather than in a tuple. One half of the images were detailed using our segmentation, the other half through LCM. Participants were asked to assign a score {1,4} based on

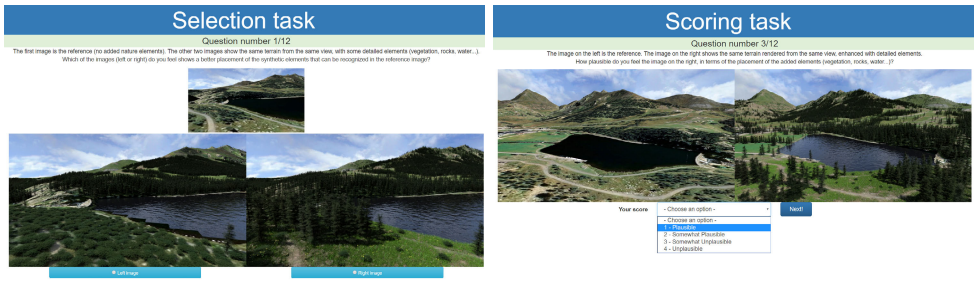


Figure 4.25: Design and interface of the detail synthesis user study. Left, plausibility task; right, scoring task.

how plausible they thought the detailed elements were placed, again according to a reference (no synthetic detail) image. Low values corresponded to successful cases, whereas high values indicated the presence of synthetic elements not matching the reference image. The order of the two tasks (selection then scoring) was fixed for all participants.

The study was deployed on a website, and participants (21, contacted through email) could complete the task remotely using the device of their choice. We applied Bayesian data analysis [Kru14] using an experiment design in the same spirit of [GTB15]. Reported results represent the posterior mean, and the confidence interval (CI) represents the range including 95% of the posterior probability.

Concerning the selection task, we modeled the posterior probability of each method (OURS, LCM) being selected as the best as a Bernoulli random variable

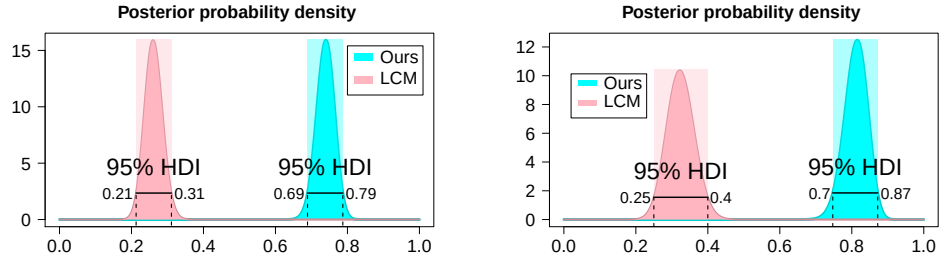


Figure 4.26: *Left*: Posterior probability $p(\theta|D)$ of each method being chosen as the most plausible in a selection round. LCM: 26% (21%-31% HDI); Ours: 74% (69%-79% HDI). *Right*: Posterior probability of each method getting the highest score in the scoring task. LCM: 32% (25%-40% HDI); Ours: 81% (75%-87% HDI). Shaded rectangles indicate the 95% Highest Density Interval (HDI).

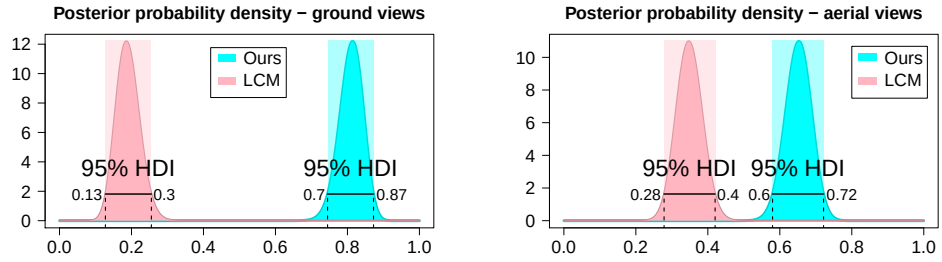


Figure 4.27: Posterior probability of winning a selection round, conditioned on the camera elevation: ground views (1 m to 30 m), and aerial views (90 m to 180 m).

with a uniform Beta prior. As shown in Figure 4.26-left, our method was significantly more likely to be selected as the best in terms of detail placement, when comparing against LCM-based placement. Figure 4.27 shows also the posterior probability, but this time conditioned to the viewpoint type (aerial/ground); as expected, the advantage of OURS over LCM is higher for ground-level views where element misplacement is more apparent and the lower resolution of the LCM plays an important role. Nevertheless, our segmentation outperformed LCM also for aerial views, presumably because LCM involves manual user input and lacks the fine-grained contours of our fully-automatic per-pixel classification.

Regarding the scoring task, we computed how likely each segmentation was to get the highest plausibility score in the 4-Likert scale. Figure 4.26-right shows that our method was significantly more likely to get the highest score. Summarizing, these results show that our segmentation pipeline is suitable for plausible detail synthesis and that it outperforms an official (expert-assisted) land cover map.

4.4 Summary

To summarize, the key contributions of our aerial image segmentation work are:

- A complete pipeline for segmenting aerial images, which can then be used by external procedural synthesis algorithms to generate plausible details on top of publicly-available DTMs, enabling detailed close-up views of real scenarios in video games and entertainment applications.
- A performance comparison of state-of-the-art machine learning algorithms for S4S. Our experiments show that Random Forests provide an excellent option considering accuracy and performance both at training and inference times. RF training times in the order of a few minutes allow artists to experiment with different sets of classes (e.g. adding water or sand) and to add training examples from new segmented regions as soon as they become available. Fast inference times enable fast segmentation and data amplification.
- An analysis on the contribution of different color and texture pixel descriptors (at varying resolution levels) in the classifier accuracy. The analysis of feature contributions revealed the key role of NDVI, an excellent discriminator of non-dry vegetation. Adding features from downsampled images only yielded very slight accuracy improvements. We did expect a higher

impact, considering the fact that human perception of vegetation types (at least on 25 cm/pixel aerial images) seems to be affected by surrounding regions.

- A discussion on different strategies for sampling the training set from partially-segmented exemplars. We have shown that training examples from homogeneous regions (the fastest manual labeling approach) lead to suboptimal accuracies, but getting closer to the actual region border is always beneficial.

- A user study analyzing human accuracy when manually labeling uniform regions in aerial images. We estimate the difficulty of the regions, the expertise of the labelers, and the "true" label of the regions. We also show how a perceptual cost matrix to apply as a post-classification filter can be easily derived from this user study using the confusion matrix of the participants. Our Random Forest classifier is able to achieve accuracies similar to those obtained by the participants in the uniform regions of the labeling task.

- A second user study demonstrating the effectiveness of our approach. We asked users to compare renders built from images segmented using either our approach or official land cover maps. The classification from our classifier pipeline are significantly more likely to be selected as better when

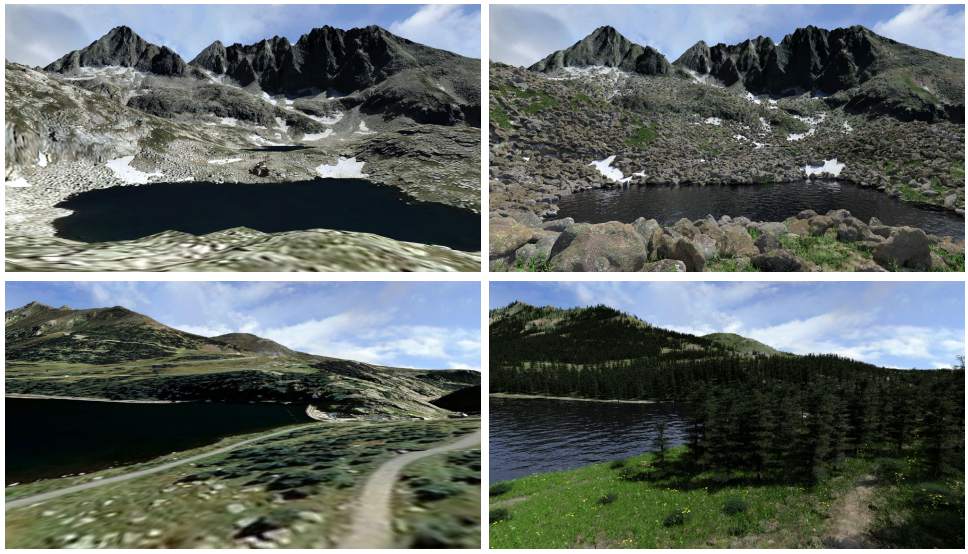


Figure 4.28: Examples of detail synthesis on big terrains (4×4 km), enhanced using the result from our segmentation pipeline.

choosing between both images side to side, and significantly more likely to be given the best plausibility score when showing a render of the enhanced terrain as in Figure 4.28.

4.5 Publications

Our contributions on aerial image segmentation led to the following publication:

- O. Argudo et al. "Segmentation of Aerial Images for Plausible Detail Synthesis". In: *Computers & Graphics* 71 (2018), pp. 23–34

5

Terrain enhancement

We will discuss in this chapter two strategies we have developed for improving the resolution and adding detail on terrain models, through the use of exemplars from other high resolution datasets. First, we present a dictionary-based approach that can be used to improve resolution on an elevation model, but also to synthesize new information layers available on the exemplars like vegetation density, for example. Next, we will see how some of the perceived details of aerial imagery can be transferred to the elevation model in order to increase its effective resolution.

5.1 Coherent terrain synthesis from exemplars

Current resolution for publicly available DEM can vary considerably from one region to another, as we have seen in the introduction (see Section 2.1). While it is true that some regions have a unique geomorphology not to be found anywhere else in the world, in most cases it is possible to find another area similar enough and more detailed.

Therefore, a possible strategy to increase the resolution of a low resolution DEM is to replace patches of it with patches from a higher resolution dataset that have matching characteristics, like in the example in Figure 5.1. This replacement method was indeed one of the proposed applications of the dictionary-based sparse representation by Guérin et al. [GDG+16], which we summarize in the following section.

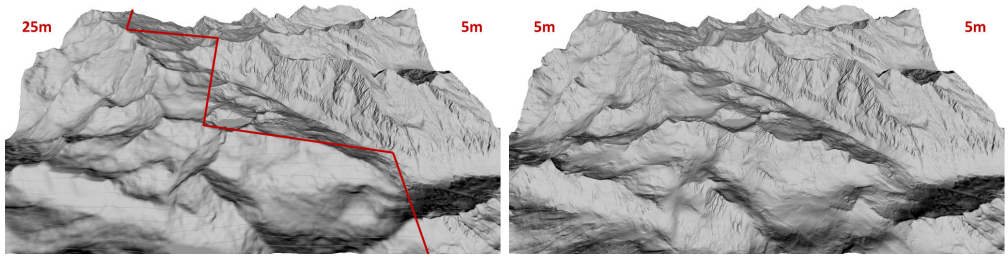


Figure 5.1: Left, a terrain set extracted from the central Pyrenees, combining elevation from two service providers at different resolutions. On the right, the low resolution part has been augmented using similar patches extracted from the high resolution part.

5.1.1 Sparse representation of terrains

We now introduce the sparse terrain representation by Guérin et al. [GDG+16], which we will later extend. Given a terrain $\mathcal{T}$, we can decompose its domain into a set of overlapping patches $\mathcal{P}$, such that they cover it completely. In their implementation, they use disc-shaped primitives distributed over a regular grid with step size the radius r of the disk. A dictionary $\mathcal{D}$ is a collection of atoms, terrain patches centered to zero mean and normalized. Then, given a dictionary $\mathcal{D}$, each patch $\mathcal{P}_i$ can be represented as a vector $\mathbf{x}_i$ of coefficients in the dictionary such that:

$$\mathcal{P}_i = \mathbf{x}_i \cdot \mathcal{D} \tag{5.1}$$

and $||\mathbf{x}_i||_0$, i.e. the number of non-zero elements in $\mathbf{x}_i$, is called the *sparsity* of the representation. Logically, the larger this value is, the better the terrain $\mathcal{T}$ can be represented using $\mathcal{D}$.

Finding the decomposition $\mathbf{x}$ of a patch $\mathcal{P}_i$ on the dictionary with a sparsity constraint of s, i.e. expressing $\mathcal{P}_i$ as a linear combination of s dictionary atoms, is an NP-hard problem. However, an approximate solution can be found using the *Orthogonal Matching Pursuit* algorithm[MZ93; TG07]. Briefly, this method finds the atom that maximizes the projection of $\mathcal{P}_i$ onto it, removes this projection from $\mathcal{P}_i$, and iterates s times.

5.1.2 Multi-resolution and multi-layer dictionary

One of the applications of the sparse terrain representation, already proposed by Guérin et al. [GDG+16], is terrain amplification. A multi-resolution dictionary is a set of two dictionaries $(\mathcal{D}, \widetilde{\mathcal{D}})$, low and high resolution, with a one-to-one

correspondence between their atoms. Then, if we find the set of coefficients $\mathbf{x}$ that lead to a representation of a terrain $\mathcal{T}$ using $\mathcal{D}$, by replacing $\mathcal{D}$ with $\widetilde{\mathcal{D}}$ and keeping the decomposition we can reconstruct a more detailed version of the terrain: $\widetilde{\mathcal{T}} = \mathbf{X} \cdot \widetilde{\mathcal{D}}$. We used this method to obtain Figure 5.1 right. Note also that atom radius can be different in $\mathcal{D}$ and $\widetilde{\mathcal{D}}$, usually $r_{\widetilde{\mathcal{D}}} = k \cdot r_{\mathcal{D}}$, with k being the amplification factor.

The dictionary is created on a preprocessing step, and can be reused for many synthesis executions. To obtain the low resolution dictionary, we just down-sampled the exemplars provided, although it would also be possible to use existing multi-resolution data if available.

We propose to augment the information contained in the dictionary with additional layers. These layers can represent any kind of data, and be provided or computed from other layers. For example, elevation, vegetation density for different species or aerial imagery can come from public web map services, while terrain slope, flow accumulation (upstream drainage area) or average solar irradiance can be derived from the elevation layer.

The motivation behind a multi-layer dictionary is that different information layers on the same terrain tend to be correlated. The shape of the terrain is a consequence of the geological properties of that area; for example, the rain does not sculpt the terrain in the same way on granite than it does on sandstone. In the same way, the type of soil – related to geology of the terrain – as well as terrain slope, orientation or elevation will determine the vegetation that can grow in that area. Therefore, we want to exploit this coherence between the various information layers to improve the terrain augmentation.

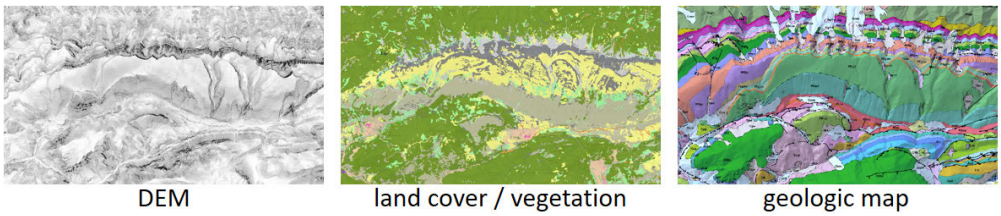


Figure 5.2: Example of different layers provided by ICGC [12] on the Cadí mountain range. It is easy to observe certain coherence between them.

We will use superscripts to refer the different layers, and subscripts for the individual atoms. Thus, $\mathcal{D}_i^j$ refers to the j-th layer of the i-th atom in the dictionary, as exemplified in Figure 5.3. For a better readability, instead of the layer indices $j \in \{0, \ldots, l-1\}$, we will use letters representing the type of layer. Table 5.1 lists the different layers and their indices.

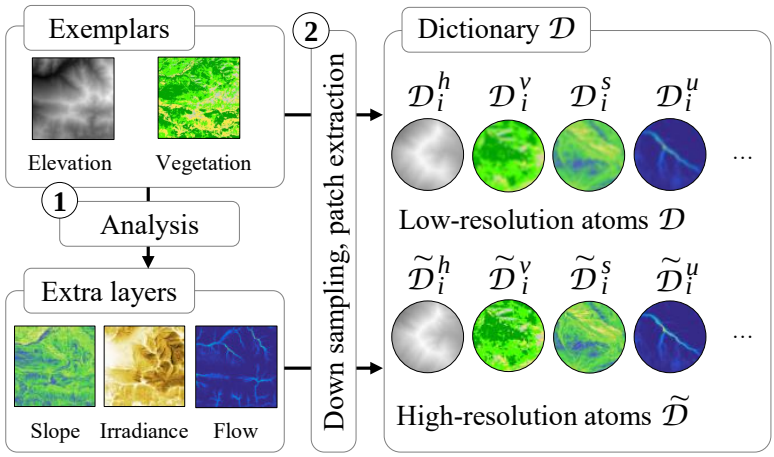


Figure 5.3: Multi-resolution multi-layer dictionary.

	information	type
$\mathcal{D}^e$	elevation	vector
$\mathcal{D}^h$	normalized elevation	vector
$\mathcal{D}^a$	elevation mean (altitude)	scalar
$\mathcal{D}^s$	elevation deviation (steepness)	scalar
$\mathcal{D}^c$	class	vector
$\mathcal{D}^v$	vegetation density	vector/scalar
$\mathcal{D}^u$	upstream area	vector/scalar
$\mathcal{D}^l$	light (irradiance)	vector/scalar

Table 5.1: Summary of superscripts of the different layers presented in this section.

Layers can store vector or scalar data per patch: elevation or vegetation density map, for example, are stored as the array of values inside the patch, while the mean elevation or mean vegetation density of a patch are stored as a scalar layers. We will see applications of different types of layers in the following sections.

While any kind of spatial domain shape could be used for the dictionary atoms, we use disc-shaped primitives as in [GDG+16]. If the radius of the disc is r, each atom is actually a $2r \times 2r$ square patch and its values are multiplied by the following radial fall-off function:

$$\left[\left(1 - d^2\left(1 - \frac{1}{2r+1}\right)\right)^+\right]^2 \tag{5.2}$$

where d is the spatial distance between the patch center and the pixel coordi-

nates. Figure 5.4 shows an example of how the disc-shaped atoms are internally constructed and represented.

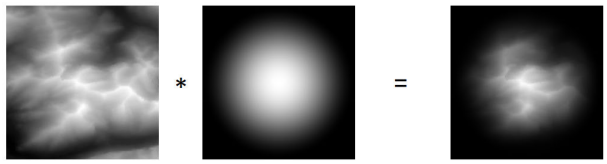


Figure 5.4: Construction of disc-shaped patches.

This masking ensures that we give more importance to the values on the center of the patch, and less on the extremes, which will overlap with neighboring patches during the reconstruction. The distribution of the discs over the exemplars used to build the dictionary can be arbitrary, although we usually distributed them on a regular grid with r or $2r$ spacing, depending on the size of the exemplar. However, when distributing discs on the input terrain $\mathcal{T}$, we want to ensure full coverage of it. Therefore, we always used a regular grid with r spacing, blend the overlapping patches according to the mask weights at each position, and normalize them afterwards as they do not necessarily add up to 1 at each position. Larger spacing leads to empty gaps not covered by any disc. Smaller spacing could be used, but then there is too much overlap and the results appear over-smoothed.

5.1.3 Multi-layer terrain synthesis

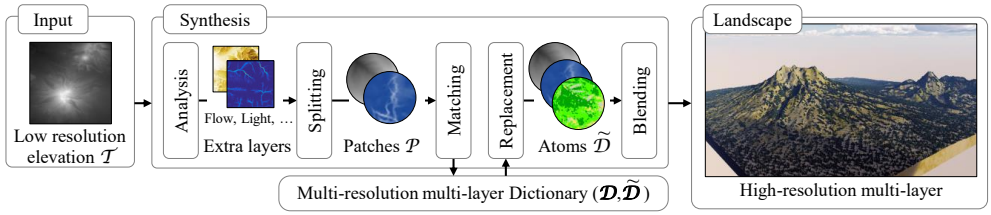


Figure 5.5: Multi-layer terrain synthesis pipeline.

Figure 5.5 shows the overview of the coherent synthesis pipeline. The input is a low resolution terrain $\mathcal{T}$, and optionally additional information layers or control layers for the matching process (not shown in the figure). The synthesis process first computes the required extra layers, which are automatically derived from the DEM like a flow map, the mean elevation or average sun irradiance. The terrain (elevation along with all additional layers) is then split into patches. For each patch, the matching function finds the best low resolution atom from the

multi-resolution dictionary to replace the patch with the high resolution elevation as well as all additional layers contained in it. All patches are blended together to produce the final terrain model.

In order to be matched with a given atom $\mathcal{D}_i$ of the dictionary, an input terrain patch $\mathcal{P}$ comes with a subset Γ of the set of layer indices $\Omega = \{0, \ldots, l-1\}$. Let $\mathcal{P}^j$ be the j-th layer of the input terrain patch. Let g_j denote the matching function of the j^{th} layer. We define the matching function $g : \mathbb{R}^n \times \mathbb{R}^n \to [0, 1]$ as:

$$g(\mathcal{P}, \mathcal{D}_i) = \sum_{j \in \Gamma} \omega_j \, g_j(\mathcal{P}^j, \mathcal{D}_i^j) \qquad \sum_{j \in \Gamma} \omega_j = 1 \tag{5.3}$$

The coefficients ω_j form a partition of unity and weight the relative impact of the different layers in the matching. The matching functions g_j detailed in the next subsections evaluate the similarity between patches and atoms for the different types of layers. The higher the value $g(\mathcal{P}, \mathcal{D}_i)$, the better the correspondence between the input patch and the i-th atom. Let k denote the atom index that minimizes the matching function:

$$k = \underset{i}{\operatorname{argmax}}\, g(\mathcal{P}, \mathcal{D}_i)$$

The reconstructed patch $\widetilde{\mathcal{P}} = \{\widetilde{\mathcal{P}}^j\}$ contains the layers of the high-resolution matched atom k. The reconstructed elevation will be computed as:

$$\widetilde{\mathcal{P}^e} = (\mathcal{P}^h \cdot \mathcal{D}_k^h)\, \mathcal{P}^s\, \widetilde{\mathcal{D}}_k^h + \mathcal{P}^a$$

All the other layers will be directly reconstructed from the dictionary atoms, therefore

$$\forall j \in \Gamma - \{e\},\ \mathcal{P}^j = \widetilde{\mathcal{D}}_k^j$$

In the particular setting of $l = 1$, $\Gamma = \{h\}$ and $g_h(\mathcal{P}^h, \mathcal{D}_i^h) = |\mathcal{P}^h \cdot \mathcal{D}_i^h|$ where the layer h contains vector data and atoms in the dictionary are normalized, we obtain the regular Matching Pursuit algorithm [TG07] with a sparsity of $s = 1$. Our framework generalizes this approach by introducing additional layers in the matching step and allowing complex matching layouts.

The landscape reconstruction from the patches $\widetilde{\mathcal{P}}$ is performed by blending the overlapping patches with the fall-off function of the distance to the patch center (Equation 5.2). In the remainder of this section, we rely on this general framework and show how the different types of layers are processed.

Orientation

Consider the layer h that represents the elevation of a terrain. Recall that dictionary atoms $\mathcal{D}_i^h$ and terrain patches $\mathcal{P}^h$ are centered at zero in order to avoid a constant component term in the projection which would make the matching less meaningful. Consequently, if we use the matching function $g_h(\mathcal{P}^h, \mathcal{D}_i^h) = |\mathcal{P}^h \cdot \mathcal{D}_i^h|$, a good matching can be achieved with $\mathcal{P}^h \cdot \mathcal{D}_i^h < 0$, *i.e.* by inverting the dictionary atom, which produces inaccurate results for our landscape generation: North faces may become South faces, ridges may become valleys, with dramatic consequences on additional layers, such as riverside vegetation on top of mountains as illustrated in Figure 5.6.

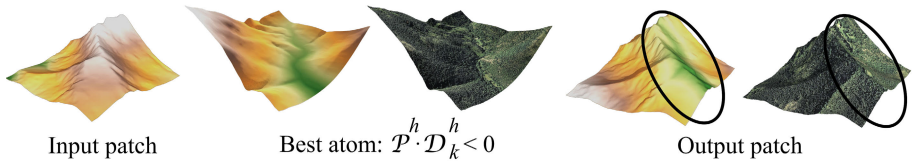


Figure 5.6: By using standard sparse synthesis, the atom that best matches the input ridge elevation data has a negative coefficient and corresponds to a valley. The vegetation density of the valley atoms are incorrectly placed on top of the terrain patch. In contrast, our approach avoids inversions.

In our framework, we are synthesizing layers that store other important properties, therefore we need to preserve the overall coherence between layers. To solve this problem, we use the following matching function that maps onto $[0, 1]$:

$$g_h(\mathcal{P}^h, \mathcal{D}_i^h) = (1 + \mathcal{P}^h \cdot \mathcal{D}_i^h)/2 \tag{5.4}$$

By avoiding the absolute value computation, matching with inverted atoms no longer occurs.

Figure 5.7 demonstrates using a toy example the importance of preserving the terrain orientation. The dictionary was created from multi-layer data containing the elevation (used for patch matching) and the normal map (used only as synthesis layer). The generated terrain is a high resolution elevation map augmented with a coherent normal map. In contrast, when allowing for atom inversion, the result is completely inconsistent. Note that the obtained normals are not exactly the same as in the computed ground truth, as that would obviously require the dictionary terrain to contain all possible shapes and normals found in the input. Still, the distribution of orientations is well preserved.

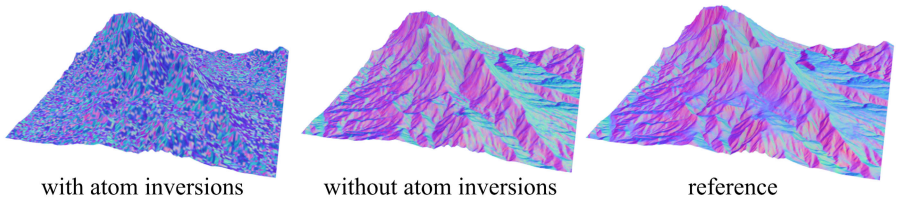


Figure 5.7: Sparse synthesis generates patches with arbitrarily-oriented atoms, resulting in inconsistent orientations that can affect the generation of other layers. Our method preserves the orientation and generates coherent patches.

Elevation and slope layers

The matching algorithm can also benefit from information about the altitude of the atoms and patches, as well as their mean elevation deviation, which can be used as an approximation of the slope.

Consider for example a perfectly shaped conical mountain, like a volcano, that has vegetation on the lower parts, then a transition to grass on the middle, and only rocks and volcanic scree on the upper parts of the mountain. The centered and normalized atoms of the upper parts can be identical to those of the lower parts. Similarly, a steep slope will probably have sparse bush-like vegetation and many rocks or cliffs, while a nearly flat slope is likely to have a forest or a meadow.

Since both patches and atoms have their constant term (mean) removed, and the dictionary atoms are also normalized, we can introduce in the matching function scalar layers $\mathcal{P}^a$ for the mean altitude, and $\mathcal{P}^s$ for the slope. Since these layers are scalars, they only need to store one value per atom/patch. Then, we define the altitude layer matching as:

$$g_a(\mathcal{P}^a, \mathcal{D}_i^a) = k(|\mathcal{P}^a - \mathcal{D}_i^a|) \qquad k(x) = e^{-x^2/\sigma^2}$$

Analogously, we use the same matching function for the slope function g_s.

The standard deviation coefficient σ serves as a user-control parameter. In our implementation, we define σ automatically by setting that the Gaussian should be equal to 0.5 at the medium difference between the elevation of patches and atoms in the dictionary.

$$\sigma = (2\sqrt{\ln 2})^{-1} \max_{i,j}(|\mathcal{P}_j^a - \mathcal{D}_i^a|)$$

Note that this definition compares every patch j mean with every atom i mean (or slope). In practice, we can reuse a fixed σ value computed from a variety of terrains, or compare a small subset of atoms and patches in order to approximate it.

When we introduce these functions onto the general matching equation (Equation 5.3), the weight ω_a allows users to control altitude-dependent content such as snow on high peaks, whereas ω_s controls slope-dependent content such as sediments or trees.

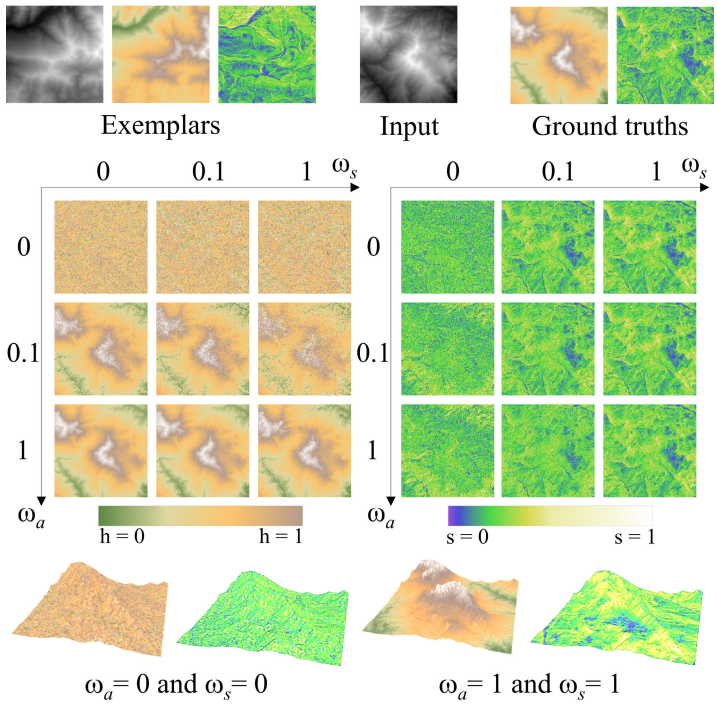


Figure 5.8: Two texture layers computed on the input DEM with different combinations of weights ω_a and ω_s.

Figure 5.9 illustrates the contribution of each ω. The dictionary is constructed from a single exemplar with two additional layers: a texture with the color-coded elevation, and another texture with the color-coded slope. The input terrain is then matched against this dictionary, and augmented with two synthetic textures. Note that these textures are obtained by simply blending the texture patches from the best matching atoms, so they are implicitly encoding information about the mean elevation and the slope of these atom on each position of the terrain. The matching will use the elevation, mean altitude and slope functions with

weights $\omega_h = 1$, and varying ω_a and ω_s, normalized in order that $\omega_h + \omega_a + \omega_s = 1$. We can see how even relatively small values of ω_a or ω_s already produces better matches with atoms of similar altitude/slope.

Figure 5.9 illustrates the impact of the coefficients ω_a and ω_s on a vegetation synthesis example when using a dictionary containing a mean altitude layer and a slope layer that are derived directly from the elevation data. The synthesis was performed on a input elevation map $\mathcal{T}$ with steeper slopes than the exemplar, which explains the scattered vegetation compared to large forests in the exemplar.

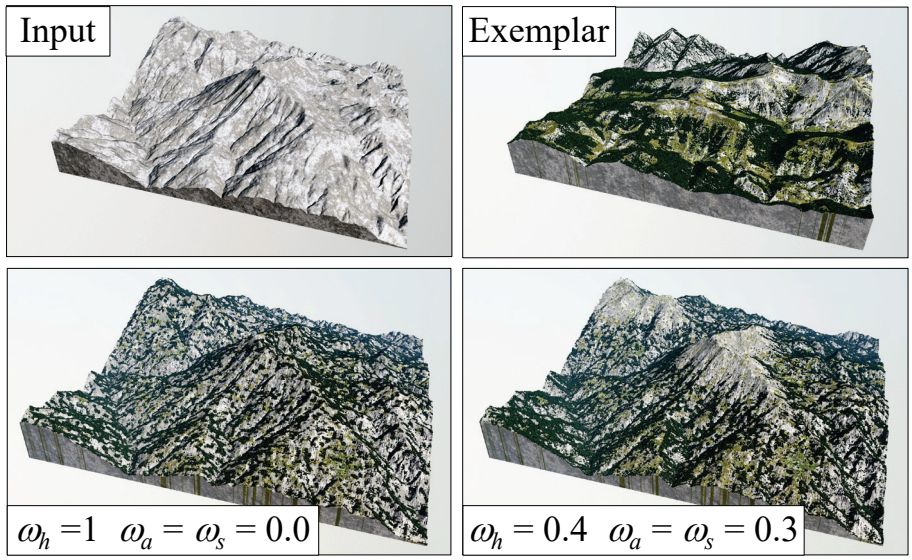


Figure 5.9: Activating the altitude and slope matching functions gives a vegetation distribution that better fits the exemplar.

Context layers

Some layers often exhibit features or properties that are not local but range beyond the domain of a patch, so taking into account the context can improve the overall matching process. We introduce context layers, which can be used to increment the spatial extent of a patch in an efficient manner.

In general, using small patch/atom sizes lead to better shape matching between the input patches and the dictionary atoms, but these matches can be less meaningful as they do not provide much information about the surrounding shape or overall structure. On the other hand, if the patches are big, more

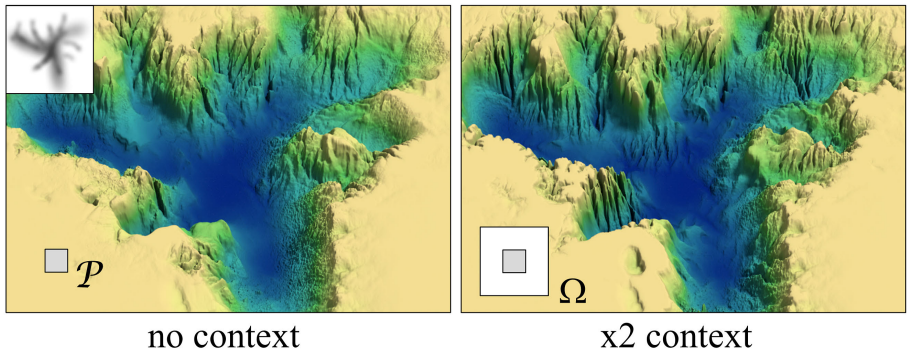


Figure 5.10: Elevation synthesis using context layers. The input sketch is shown as an inset on the top left. The grey square denotes the patch size, and the white one indicates the approximate extent of the neighborhood Ω considered for computing the context layers.

shape and structural information will be used on the matching function, but the chances of modifying the overall shape are bigger, as it will be more difficult to find an atom with the same shape.

Therefore, we propose to use two different spatial domains for matching and for replacement. During the matching, we want the spatial extent to be big in order to account for the context of the patch, but only the smaller central part of this domain will be included in the synthesis step. Formally, the context data for a given patch $\mathcal{P}^j$ is computed over a spatial domain Ω_j embedding $\mathcal{P}^j$. In our experiments, the radius of the domain ranged from twice to eight times the radius of the patch.

A naive implementation of this new matching domain can produce a significant increment on the computation time: since the number of elements in vector layers increases quadratically with the radius of a patch, matching functions that apply a dot product increase their computation time and memory as well. Thus, instead of extending the size of the patch, we derive additional layers that represent the vector data (e.g. elevation) using a hierarchical down-sampling pyramid. For example, if we down-sample the elevation layer h to half the size, we obtain the layer $h_{1/2}$ in which the same patch size represents a spatial extent twice as large. We can then define matching functions for each of these new layers, although in practice we just use Equation 5.4 on each of them with a different weight.

Note that this is a different concept from the low and high resolution dictionaries $(\mathcal{D}, \widetilde{\mathcal{D}})$. Although context layers include data from a larger domain

$\Omega_j \supset \mathcal{P}^j$, atoms $\mathcal{D}_j$ and $\widetilde{\mathcal{D}_j}$ have identical spatial extents. Atoms in $\widetilde{\mathcal{D}}$ are used exclusively in the last step of the terrain synthesis when replacing patches with their high resolution counterparts.

Figure 5.10 compares the results obtained by increasing the size of the neighborhood when using context layers for capturing the landform features. Context layers allow for a more realistic reconstruction of the cliffs, valleys and flat terrain landforms such as plateaus, since the best matching atom can be determined according to the features in the neighborhood of the patch.

Environmental layers

Layers representing *global* environmental information can be used to further improve the overall landscape, i.e. terrain and vegetation, synthesis process. Contrary to elevation or density layers that only provide *local* information at a given point, or context layers which represent a larger domain but still only contain information around the point, global layers store parameters that are derived from a *global* analysis of the terrain. Such layers are particularly useful for implicitly representing correlations between neighboring patches and introducing coherence terms in the equation that evaluates the matching between patches and atoms.

In our system, we experimented with two types of global information layers: the upstream drainage area that approximates the average flow of water passing through a patch, and the average solar irradiance that represents the average amount of sunlight received by a patch. If not provided, those layers can be computed from the elevation data of the exemplars and the input $\mathcal{T}$ to generate the corresponding layers for patches $\mathcal{P}^u$ and $\mathcal{P}^l$ and dictionary atoms $\mathcal{D}^u$ and $\mathcal{D}^l$. The matching function can be simply defined like in the previous vector layers:

$$g_u(\mathcal{P}^j, \mathcal{D}_i^j) = (1 + \mathcal{P}^j \cdot \mathcal{D}_i^j)/2 \qquad j \in \{u, l\}$$

The upstream drainage area is computed by simulating the flow of a large number of particles randomly distributed over the surface of the terrain and measuring the number of particles passing through every patch. Figure 5.11 shows that taking the drainage area into account allows for a better matching reconstruction of ravines and gullies.

Solar irradiance (layer $\mathcal{P}^l$) is calculated based on latitude and longitude by intersecting rays from the sun position along its trajectory with the terrain. This captures terrain self-shadowing and provides average direct illumination from the sun.

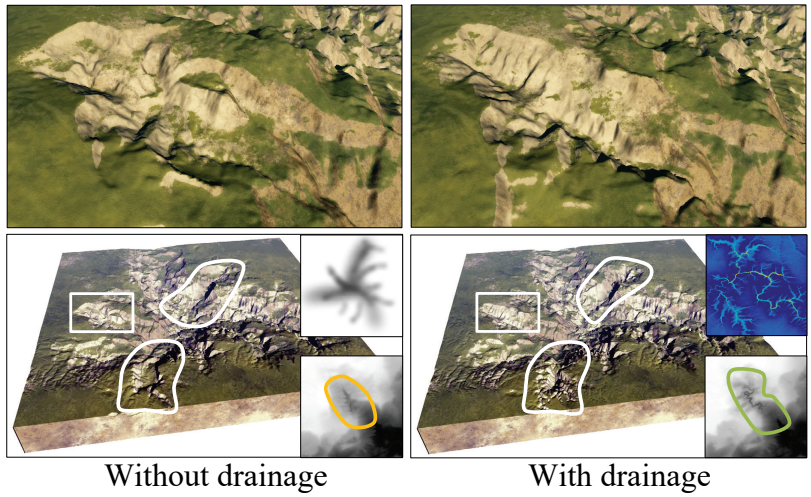


Figure 5.11: Influence of the upstream drainage area layer: gullies and ravines are better reconstructed using it.

Class density layers

We introduce class layers as a powerful tool to control the style of the synthesized multi-layer terrain model, e.g. rocky mountains, snow peaks, hills with forests. Classes are defined by using an abstract layer that defines the class of patch. However, instead of specifying a single class, we allow for a density value between 0 and 1, thus allowing several classes to be present at the same location.

Dictionary atoms are first labeled to identify multiple differentiated regions from distinct classes (Figure 5.12). A classification layer stores vectors whose components define the relative probability (terms range in $[0, 1]$ and sum to 1) that the patch should belong to the corresponding class. An exemplar with 3 different classes leads to 3-component vectors for the class layer (Figure 5.12). The matching method consists in choosing atoms that have preferably the same class distribution as the input patches. We define the matching function g_c as:

$$g_c(\mathcal{P}^c, \mathcal{D}_i^c) = \frac{1}{n} \sum_{k=1}^{n} \widehat{\mathcal{P}}^c(k) \cdot \widehat{\mathcal{D}}_i^c(k)$$

where n represents the number of samples; and $\widehat{\mathcal{P}}^c(k)$ and $\widehat{\mathcal{D}}_i^c(k)$ represents the normalized class vector k^{th} sample for the input patch and dictionary atom respectively. While it is possible to take into account all the class vectors in the patch/atom class layer, class densities normally present smooth variations

and using only a few samples leads to the same or very similar results while using much less computation time and memory. In our implementation, we used a Poisson disc-based distribution of samples inside the patch area, and $n = 8$ provided a good tradeoff between efficiency and similarity to full computation.

Normalization of the class vectors is important so that patches with smoothly varying classes do not match with atoms of a single class instead of atoms with slightly different class distributions. For example, image we have density maps with 2 classes and the current patch sample is $\mathcal{P}^c = (0.7, 0.3)$, so it represents a mix between the two classes with larger density of the first class. Suppose now that our dictionary has two atoms a and b: $\mathcal{D}_a^c = (1, 0)$, $\mathcal{D}_b^c = (0.6, 0.4)$. If both atoms are equally valid candidates as the best match, we would like to choose b since the distribution of classes is more similar to the distribution on the patch. However, if we apply the dot product without normalizing these vectors, $\mathcal{D}_a^c \cdot \mathcal{P}^c = 0.7 \ > \ \mathcal{D}_b^c \cdot \mathcal{P}^c = 0.54$, resulting in a better score for atom a. If we normalize the vectors, $\widehat{\mathcal{D}}_a^c \cdot \widehat{\mathcal{P}}^c = 0.92 \ < \ \widehat{\mathcal{D}}_b^c \cdot \widehat{\mathcal{P}}^c = 0.98$, and now b has a better score.

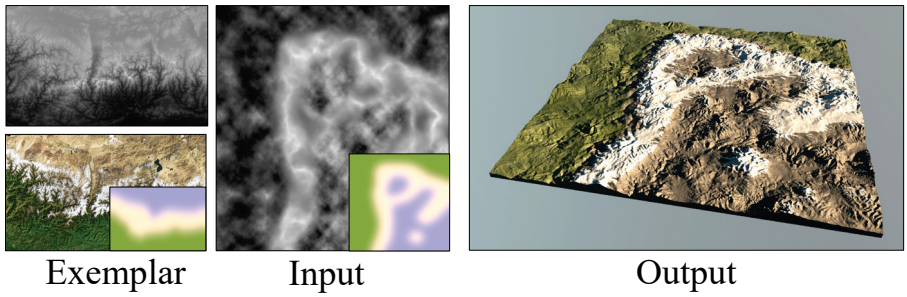


Figure 5.12: Terrain authoring from a sketch defining 3 classes: hills with forests, desert mountains, snowed peaks. The dictionary was built from an exemplar of the central part of the Himalayas, using the aerial photograph (bottom) as a guide to produce the segmentation into different types of terrains (bottom inset), which would be later represented as different textures on the output terrain.

Matching multiple dictionaries

Our method allows to use dictionaries created from different exemplars and featuring various types of landscapes. While a dictionary can be built from as many exemplars as we like, in some cases it may also be useful to build different dictionaries for different types of terrains or biomes, and let the user specify where to use each one and how to blend them, as in Figure 5.13.

Without loss of generality, we explain how to perform matching on multiple dictionaries with 2 biomes. In this particular setting, we consider two dictionaries $\mathcal{A}$ and $\mathcal{B}$ that represent the two biomes. The user has to paint two additional input layers α and β that describe the relative influence of the respective biomes on top of the input terrain. Note that the sum of α and β should be 1 everywhere. For the matching and reconstruction, we use the following algorithm for all patches $\mathcal{P}$ in the input terrain:

1. Find the atom $\mathcal{A}_i \in \mathcal{A}$ that maximizes $g(\mathcal{P}^\alpha \odot \mathcal{P}, \mathcal{P}^\alpha \odot \mathcal{A})$.
2. Find the atom $\mathcal{B}_j \in \mathcal{B}$ that maximizes $g(\mathcal{P}^\beta \odot \mathcal{P}, \mathcal{P}^\beta \odot \mathcal{B})$.
3. Blend the two high resolution atoms and generate $\mathcal{P} = \widetilde{\mathcal{P}}^\alpha \odot \widetilde{\mathcal{A}}_i + \widetilde{\mathcal{P}}^\beta \odot \widetilde{\mathcal{B}}_i$ where $\widetilde{\mathcal{P}}^\alpha$ and $\widetilde{\mathcal{P}}^\beta$ are the upsampled versions of $\mathcal{P}^\alpha$ and $\mathcal{P}^\beta$.

where $\odot$ denotes Hadamard element-wise vector multiplication.

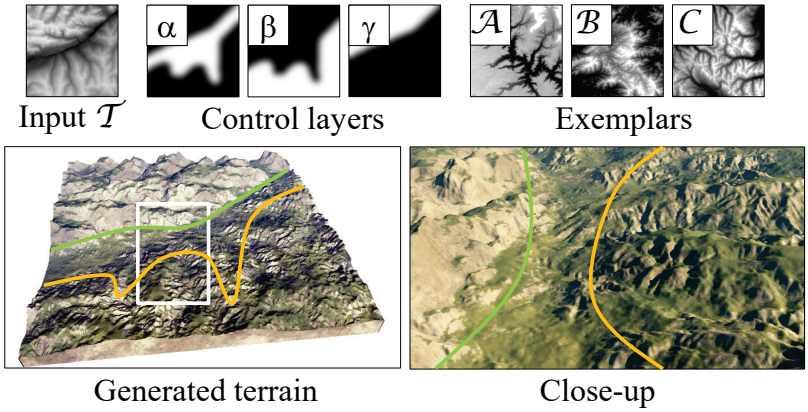


Figure 5.13: High-resolution landscape generated from a low resolution elevation map, three exemplars and control layers indicating the preferred exemplar. The dictionaries were created from the Rocky Mountains, the Grand Canyon and Smokies National Park elevation maps.

Note that this algorithm is similar to applying the matching algorithm independently on each dictionary, and then merging the chosen atoms from each one according to the weights of the biome distribution layers α and β. Additionally, before computing the matching function g on a dictionary, all the vector layers on the patch $\mathcal{P}$ are masked by the influence that dictionary will have on this patch, which is encoded by the corresponding biome layer, and each atom of $\mathcal{A}$ is also masked accordingly. The motivation behind this masking is that

we only want to match between the parts of the atoms and patch that will be represented on the blended terrain. Atoms that match the patch but that do not lie in the area of influence on the patch will obtain a low matching score. This yields more coherent results than simply blending the different layers synthesized independently.

5.1.4 Applications and results

Our system automatically synthesizes *coherent* high-resolution multi-layer landscapes, i.e., terrains with detailed elevation, soil type, sediment layers covered with a realistic distribution of different types of vegetation from a few input low-resolution layers (usually, elevation and control layers). Instead of relying on complex procedural ecosystem simulations, our method reproduces the patterns and characteristics of the dictionary exemplars and preserves the overall coherence of the different layers, as shown in Figure 5.14.

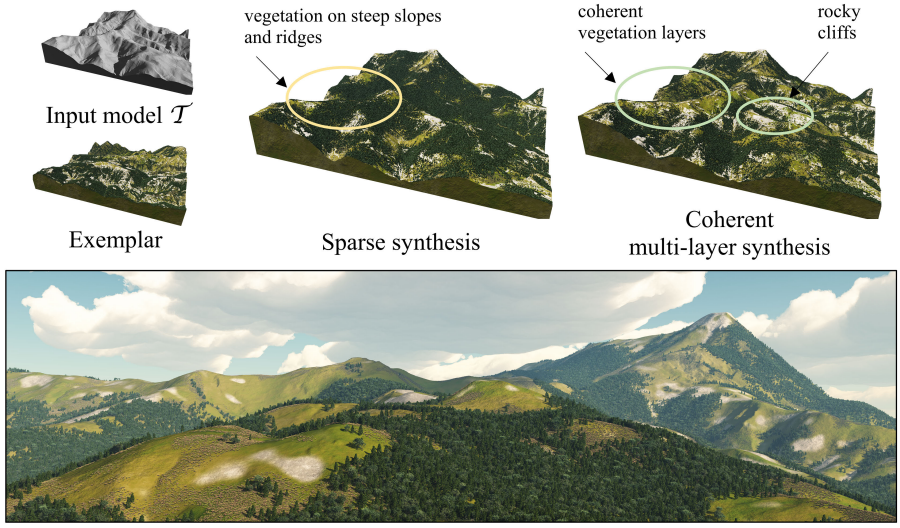


Figure 5.14: Comparison between sparse synthesis and our method. Left image shows the vegetation distribution without taking into account the orientation, mean altitude and slope. Right image shows the coherent distribution, which better preserves the appearance of the exemplar terrain.

An important feature of our framework is its *versatility*: it can be used to generate an arbitrary number of layers of different types. Moreover, our dictionary-based system allows the user to enhance the database with as many atoms as needed, completing it with atoms featuring sediments or different kinds of plants.

User control

The user *controls* the multi-layer terrain generation process by adjusting the weighting coefficients for the input and synthesized layers (Figure 5.9). Artists can freely provide, besides the elevation data, arbitrary layers as inputs, choose the layers and define their purpose: soil type, humidity, vegetation density or biome as long as the dictionary atoms encode these layers. The algorithm produces high-resolution terrains or additional layers consistent with the user-provided input and exemplars, as shown in Figure 5.14.

Figures 5.12 and 5.13 show examples of sketch-based authoring. The sketch contains a coarse description of the desired classes (forest, desert, peaks), with the purpose of synthesizing consistent biomes. The influence of the sketches in the matching process can be controlled by modifying the weighting coefficients.

Similarly, Figure 5.15 shows an example where control is achieved by sketching as class density layers the expected distribution of three different vegetation types (tree, shrub/grass, sand soil), represented together using RGB for visualization purposes.

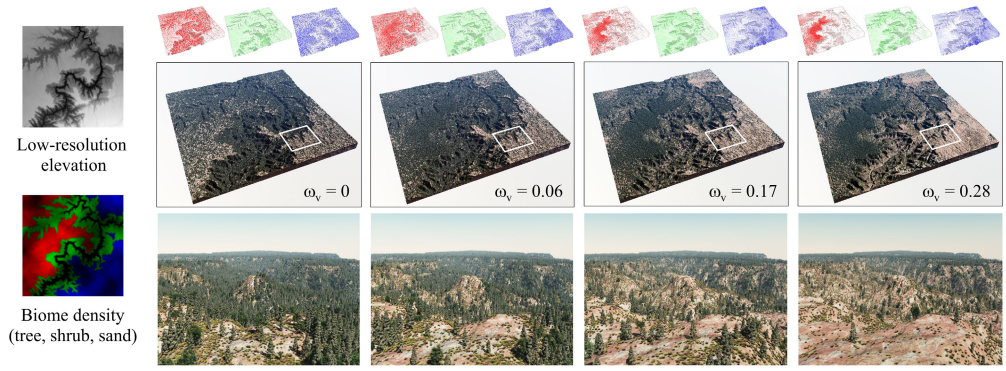


Figure 5.15: Vegetation control over a large terrain: the vegetation distribution dictionary was created from the eastern Pyrenees exemplars (Figure 5.14), and the different species were prescribed by the user by defining three different classes (trees in red, bushes in green and sand in blue). Individual synthesized layers and rendered results are shown for different values of the control parameter ω_v.

Our matching framework can further simplify user control tasks. Instead of defining a vegetation distribution for each vegetation type, the user may simply provide a vegetation density layer (single scalar), and use the dictionary to convert the overall density into vegetation distributions for the different types of vegetation. Figure 5.16 illustrates this case. Since the user only wanted to

specify an overall density map, a procedure can be specified to create on the dictionary another layer that encodes the weighted sum of the specific vegetation layers (e.g. tree density contributes much more than grass density to the overall sum). This derived layer is then used to compare the average density between atoms and patches during the matching step.

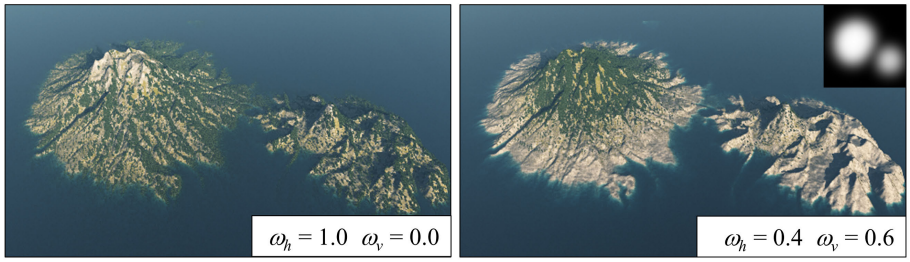


Figure 5.16: Vegetation control: left image shows a terrain without vegetation density control, i.e., following the distribution of the dictionary exemplar, whereas right image shows a smooth disc-shaped constraint. The algorithm automatically selects atoms with no vegetation under water.

Iterative synthesis

Our method allows to execute the synthesis process iteratively, using some of the output layers of one iteration as input layers for the next iteration. Therefore, we can use several dictionaries with different layer subsets in a coherent way.

Figure 5.17 shows an example of a two-step workflow. Given the input elevation data, the first iteration applied a multi-resolution dictionary extracted from a real dataset of Catalonia to synthesize coherent population and vegetation densities on top of this terrain by taking into account the DEM, mean elevation and slopes. Then, the second iteration used a dictionary obtained from a small part of Catalonia ($10 \times 10\,\mathrm{km}^2$) with information on per-class vegetation distributions. During the matching step, the vegetation density layer computed in the first iteration was introduced as a density control layer to guide the matching and placement of vegetation. It can be seen how the vegetation has been distributed according to the previously generated density, whereas if we apply this dictionary directly on the input terrain without the density map it produces a very different result.

This example demonstrates that output layers (here, vegetation density) can be in turn used to guide the synthesis in a coherent feedback loop.

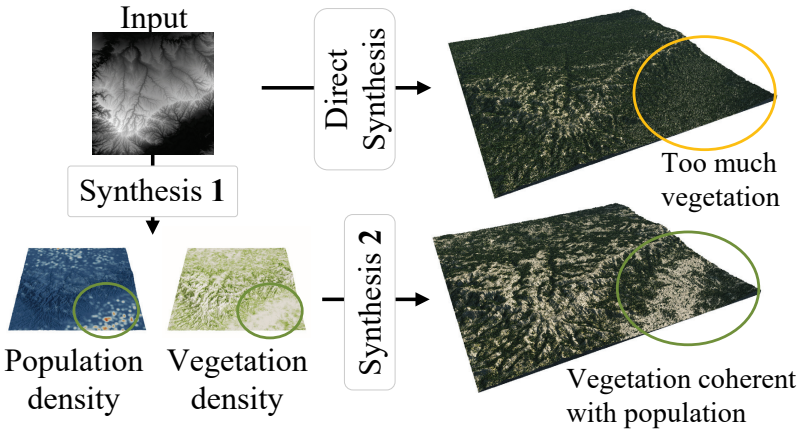


Figure 5.17: Example of a two-step vegetation synthesis. First, we produce a vegetation layer coherent with a population density. Then, this layer is used to produce the distribution of three vegetation classes. Using directly the second dictionary on the input terrain cannot account for population density and thus places forests on areas with a high population density.

Terrain authoring

As a demonstration on the versatility and effectiveness of our pipeline for editing terrains, Figure 5.18 shows an example of an authoring session in which the user incrementally edits a low-resolution elevation map – using any preferred image processing software – and this map is augmented to higher resolution and additional coherent vegetation layers.

In Figure 5.18 step 1, the input is the initial terrain map, which shows a cone-shaped mountain that is enhanced and populated with vegetation following the distribution exemplar from the Pyrenees (Figure 5.14). Then, in step 2, the user edits the elevation map by copying from another DEM smooth volcanic hill on the top right corner. The render shows how the vegetation is redistributed on this region after running again the pipeline, while it is maintained as before on the rest of the terrain. Step 3 adds a canyon to the lower part of the terrain. Finally, in step 4, the designer re-used the synthesized vegetation layers from the previous, down-sampled them, modified the density in the canyon to leave it empty and the density on the volcanic hills to have only grass, and then relaunched the vegetation synthesis process on the same elevation map as in step 3 but now with these modified vegetation density maps as matching layers. The output vegetation distribution follows the user constraints. This authoring session took less than 10 minutes, including user editing and terrain synthesis.

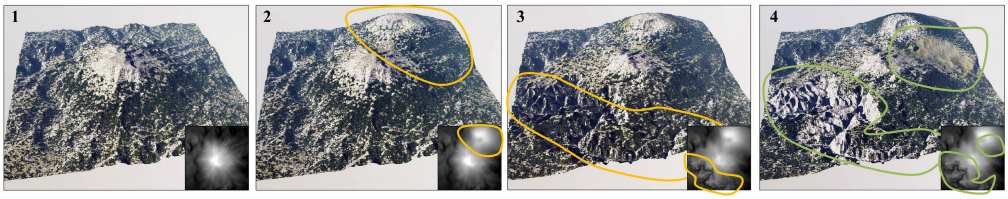


Figure 5.18: Example of an incremental authoring session. Orange shapes highlight DEM editing steps, and green the vegetation density edits.

Terrain texturing

Up to some extent, our pipeline can also be used to synthesize textures, like aerial photographs, on a terrain. For example, in Figure 5.19 we used a high resolution dataset from Mount Rainier (Washington, US) as exemplar to augment a scaled DEM of Olympus Mons, the largest volcano in Mars (and known Solar System). In this case, we did not only transfer the high resolution elevation data but also the texture, which the exemplar had as aerial imagery. However, as there is no guarantee on the continuity of texture between neighboring patches, the more structured this texture is the more apparent becomes the discretization of the terrain into patches and their blending. However, for natural landscape scenes with moderate degree of variation on the aerial imagery (i.e. not covering an area with very different biomes) results are convincing.

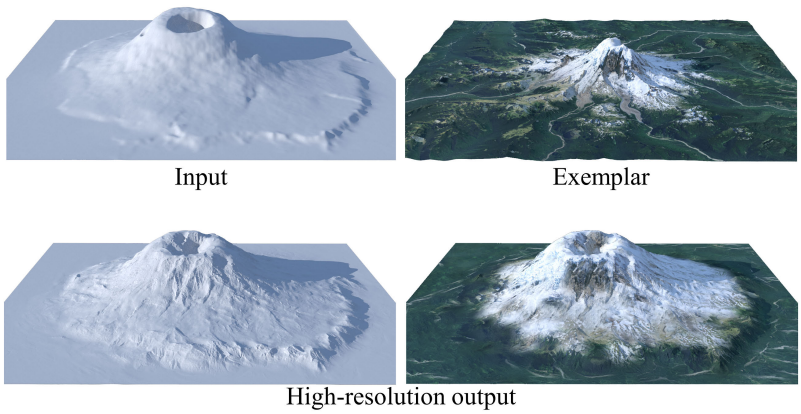


Figure 5.19: Example of a high resolution multi-layer dictionary extracted from Mount Rainier and applied to a low-resolution model of Olympus Mons.

Similarly, another application is terrain-aware aerial image repairing as we show in Figure 5.20, in which we perform shadow removal on the aerial image

of Montserrat mountain range, which has very steep rock faces. In this case, the input terrain contained a mask indicating where to repair the texture. The dictionary is built using atoms the unmasked parts. Like in the previous example, using this dictionary a texture layer is synthesized for the whole input terrain, and texture on the masked parts are used to replace the original texture. As we can see in Figure 5.20, our texture synthesis provides better results than context-aware image inpainting techniques available on image processing packages. These methods do not take into account the terrain, so the steep rock faces appear textured with trees instead, and even some parts of the road are copied on it.

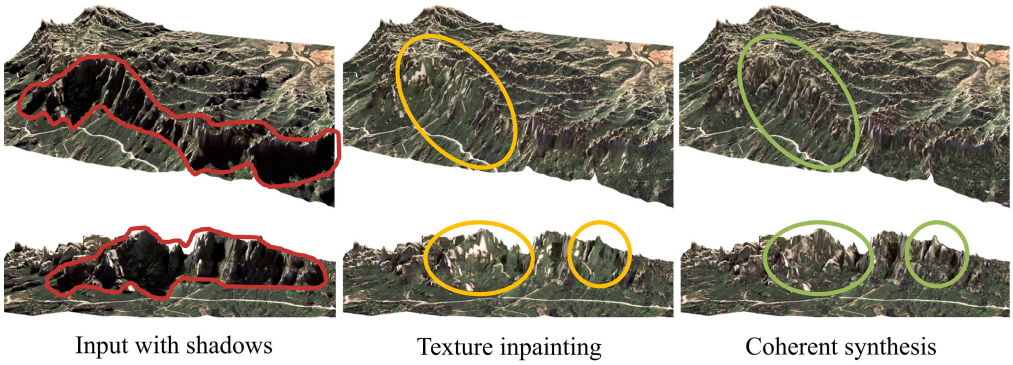


Figure 5.20: Shadow removal on aerial imagery using coherent synthesis on the DEM, compared with context-aware image inpainting.

5.1.5 Performance

Our method was implemented and tested on an Intel Core i7 with 16 GB of RAM. Table 5.2 presents an overview of the different cases and reports the corresponding statistics. Timings demonstrate that our approach is efficient and can be used in practical terrain authoring applications. Although our current framework has been coded into a single-threaded CPU implementation, our method lends itself for a parallel implementation on the GPU.

The *dictionary generation step* can be performed once as a pre-processing step and does not count in the synthesis generation time. The dictionaries can be extended easily by adding new layers to existing atoms, or by adding new atoms from other exemplars. Increasing the size of the dictionary allows for more variety and yields better results, at the cost of a more computationally demanding matching step. *Matching* performs in a few seconds and even less than a second for average-size dictionaries. The matching step of Figure 5.13 required one pass

Layers		Multi-layer terrain synthesis							Time (s)		
Input	Output	Figure	Patch	Atoms	Patches	Input	Output	Scale	Dict.	Match.	Synth.
h, c	h, t	5.12	24^2	3920	289	185^2	740^2	$\times 4$	0.20	0.07	0.43
h, c	h, t	5.13	24^2	5401	1521	450^2	1350^2	$\times 3$	0.35	3.19	9.95
h	h, v	5.14	32^2	2116	324	280^2	1400^2	$\times 5$	0.08	0.03	0.47
h, v, c	h, v	5.15	10^2	3969	10816	513^2	5130^2	$\times 10$	0.07	1.91	7.45
h, v	h, v	5.16	24^2	3481	29584	2048^2	2048^2	$\times 1$	0.11	3.07	7.47
h	h,t	5.19	12^2	2304	729	154^2	2464^2	$\times 16$	0.03	0.04	1.50

Table 5.2: Statistics: patch size, number of atoms in the dictionary, number of patches, size of the input and output terrains, amplification factor. We also report timings (in s) for the dictionary construction, matching process, and patch replacement.

per class; timings take into account the multiple passes. *Synthesis and blending* performs in less than a few seconds on most examples. Notable exceptions are reported for the largest synthesized model (Figure 5.15), with a large terrain (5130×5130) involving more than 10 k patches, and Figure 5.16 which contains almost 30 k patches.

The time needed to evaluate $g_j(\mathcal{P}^j, \mathcal{D}_i^j)$ for complex data becomes the more expensive as the number of layers increases. In our implementation, we optimized the computation by using a Poisson disc-based distribution of samples inside the patch area and evaluating g_j only over the reduced set of center points.

5.1.6 Discussion

Our dictionary-based framework has several applications. The input can be real digital elevation models, rough sketches drawn by hand, or a combination of both, containing a single layer (e.g. elevation), or multiple layers. The output terrain contains always as many layers as available in the dictionary, thus providing coherent data amplification.

A key feature of our approach is to provide a unified, easy-to-control, and flexible model for multi-layer landscape synthesis generating plausible and predictable results. Although alternative methods exist for some specific problems, none of them cover simultaneously all the applications supported by our framework. Our method provides control to the user and allows him to create any arbitrary layer in a coherent way. These two features are key for dictionary reusability and, ultimately, for effective terrain creation, editing and synthesis. *Context* layers combined with *global environmental layers* allow us to generate spatially coherent patches that more faithfully reproduce landform features such as gullies, erosion lines or plant clusters. Finally, our approach can be extended

easily by considering other types of layers and defining the corresponding appropriate matching function.

As for all example-based approaches, our method may require a large input dataset to synthesize terrains. The dictionary extraction pre-processing step is very efficient. Although it may be difficult to find real world exemplars with appropriate layers, the set of exemplars can be completed with results obtained by computer simulations. Our framework offers many possibilities for reusing dictionaries, since it is built independently of the synthesis step.

Although the coherence between the different layers of an output patch is guaranteed by construction, the coherence between neighboring patches in the output is affected by the variety of atoms in the dictionary. This limitation has two consequences. First, mixing exemplars from radically different biomes (e.g. rain forest and desert) into the same dictionary may result in poor spatial coherence or sharp transitions. That limitation may be alleviated by providing a sketch of the desired distribution, as described in Section 5.1.3.

Another limitation of our method is that it does not properly handle structured layouts such as road-networks, villages or cities: the synthesis process does not guarantee that the structures would seamlessly link between two neighboring patches. Our method can nevertheless synthesize statistic information such as population density (Figure 5.17), which in turn may be used as input to generate villages or cities [EBP+12].

5.2 DEM super-resolution through aerial imagery

Current resolution of aerial imagery is about an order of magnitude higher than DEM, so virtual globe applications show the terrain mesh textured with these pictures. If we look at as these textured terrains, it is easy to infer some intermediate scale terrain detail, in the range between the resolutions of the image and the elevation mesh. We decided to study whether this type of perceived details could be transferred as geometric detail on the elevation mesh.

We saw in Section 2.1 that current DEM resolutions on country-level datasets range between 5 and 30 meters, while regional datasets can be found at greater detail like 2 m, 1 m. While highly detailed datasets will probably become more commonly available in the future, there is still a considerable cost on their acquisition (they are usually obtained through LIDAR instead of photogrammetry), processing and data storage. Therefore, it is reasonable to assume that we can leverage the currently available high resolution datasets to improve other

neighboring regions or areas that have similar characteristics and for which only medium resolution elevation maps exist.

In the previous section, we introduced an algorithm for synthesizing detail and additional coherent layers on top of (low resolution) elevation maps. However, dictionary-based methods focus on generating plausible detailed terrains. On this part we emphasize on the fidelity of the reconstruction, we want to generate a terrain that is as close as possible to the real one. As we show in Figure 5.21, using large terrain patches during replacement causes large distortions on the terrain shape. If we reduce the patch size, then the terrain is closer to the original shape but the matching is less meaningful, as there is less information in the patch shape, and more repetitions appear. Moreover, these methods do not take into account the information contained in the aerial photograph.

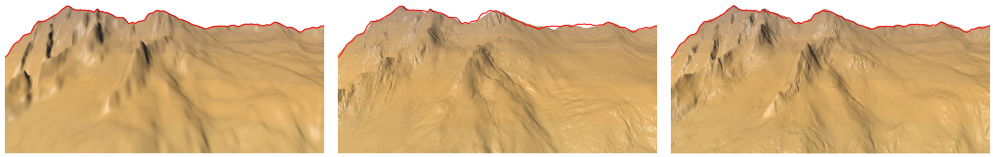


Figure 5.21: Terrain super-resolution using dictionaries. Left: 15 m DEM with the silhouette highlighted. Center: enhanced terrain using 16×16 patches and the 15 m silhouette overlaid. Right: enhanced terrain using 8×8 patches and the 15 m silhouette overlaid.

Inferring geometric details from a photograph is a problem that has also been approached using shape-from-shading methods. However, these methods usually make assumptions or rely on the lighting conditions in order to estimate the geometry from the occlusions and shadows in the picture. In our case, we cannot rely on two aerial image datasets having similar lighting conditions; flight times may be different and, consequently, shadow orientations and lengths. Furthermore, we have observed that for mountainous terrains as the ones we are interested in reconstructing, shadows are usually too dark in order to being able to infer any details from them, as we exemplify in Figure 5.22. Small rocks and scree on the image texture appears as very contrasted and high frequency details, which leads to a lot of noise in the inferred geometry. Finally, we would not like to infer detail from the photograph on tree and vegetation areas, as we are only modifying the DEM which represents the elevation of the terrain. A shape-from-shading approach is not aware of the elements shown in the aerial image unless we provide a segmentation along with it.

Recently, deep learning approaches have proven to outperform other methods on many machine learning and image processing tasks, like image segmentation [LSD15] or image super-resolution [DLH+16] by using Convolutional Neural

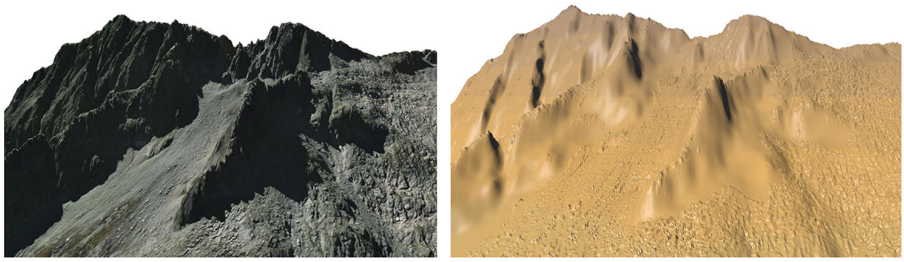


Figure 5.22: Terrain super-resolution using shape from shading. Left, terrain textured with the aerial image. Right, terrain with added geometric details.

Networks (CNN). Given the difficult nature of our problem, we decided to investigate whether these convolutional networks would help us to infer geometric details on a DEM from the aerial image. First, in Section 5.2.1 we will see how we obtained and constructed the dataset to train the network, and Section 5.2.2 will discuss the different architectures we experimented with and the intuitions behind them. Finally, in the rest of sections we will present the results from a numerical, visual and users point of view.

5.2.1 Dataset

We chose mountainous areas with alpine-like features because in this kind of terrain we can find many geological formations that are hard to preserve in mid-resolution DEM but appear clearly visible on aerial photographs, such as sharp ridges, boulders, couloirs, glacier crevasses, etc. Although in areas covered with vegetation it is not possible to infer details on the DEM from the aerial photograph, it would still be possible to use our method in case we had the appropriate DSM or Lidar point cloud obtained at the same time than the aerial imagery.

We downloaded several terrain regions from two publicly available geographical services that provided very high resolution aerial imagery (1 m or less per pixel) and high resolution DEMs (around 2 m) in mountainous regions. From Institut Cartogràfic i Geològic de Catalunya [12] we downloaded 10 terrain areas from the Pyrenees, summing up a total area of 643 km^2. From Südtiroler Bürgernetz GeoKatalog [22] we downloaded 12 regions of the South Tyrol Alps, a total area of 304 km^2.

Figure 5.23 shows all the downloaded terrains as textured meshes. Note the different aspect that aerial imagery shows on the Pyrenees and the Tyrol. Apart

from snow and glaciers on the Tyrol set, the colors, saturation and shadows also have distinct appearance. This will be useful in order to evaluate whether our proposed network can generalize well from the training examples to other regions.

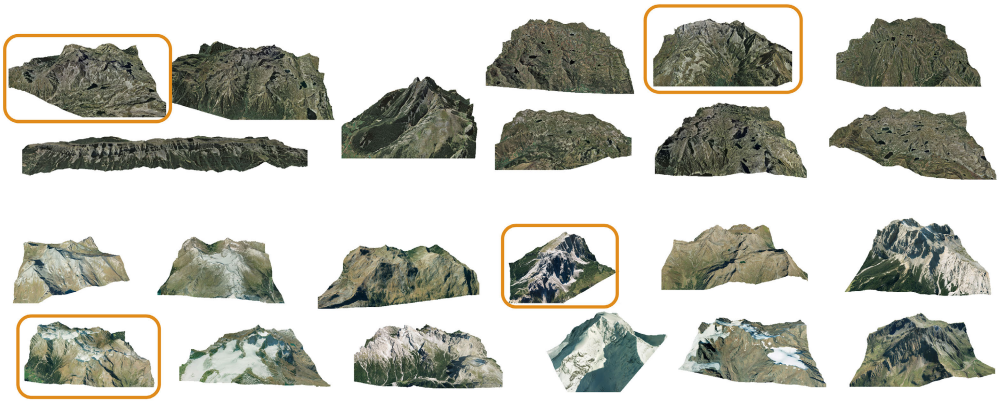


Figure 5.23: Terrains dataset. On the top half, the 10 terrains from the Pyrenees. On the bottom half, the 12 terrains from Tyrol. The four terrains highlighted in orange will be used as test and evaluation of the results.

In order to build a uniform set, we scaled all aerial imagery to 1 m resolution and all DEMs to 2 m resolution. The low resolution version of the DEM was automatically derived from the 2 m resolution by downsampling. We used 15 m as low resolution, since middle resolution DEM providers usually return resolutions multiple of 5 m between 5 and 25 m, and 15 m is quite common.

Four of the downloaded regions (highlighted in Figure 5.23) were completely excluded from training and validation splits, in order to be later used as evaluation (see Section 5.2.5): Bassiero (48.7 km^2) and Forcanada (25.7 km^2) from Pyrenees, Dürrenstein (9.9 km^2) and Monte Magro (64.7 km^2) from Tyrol.

The remaining regions were split into 400×400 m tiles, yielding a total of 4741 tiles. As data augmentation, each tile was also rotated at 90, 180 and 270 degrees, transposed, and flipped vertically and horizontally. We obtained 33,187 total tiles, 23744 from Pyrenees and 9443 from Tyrol. These tiles were then distributed into three sets:

- **Pyrenees**: 15000 tiles training, 8000 tiles validation.
- **Tyrol**: 6400 tiles training, 3000 tiles validation.
- **Both**: 22000 tiles training, 11000 tiles validation.

5.2.2 Network architecture design

Our network design takes inspiration from recent Convolutional Neural Networks (CNN) architectures that have proven to be successful in image applications like segmentation and super-resolution. We will briefly introduce in this section those that we consider more relevant.

Regarding single-image super-resolution, Dong et al. [DLH+16] construct a three layer network of convolutional filters. The intuition behind this structure is that the first layer of convolutions acts as a bank of filters that extracts patches and features from them, then the second layer encodes the patches onto some space using a non-linear mapping, and the third and final layer reconstructs the high-resolution patches from this representation. In their experiments, they show that good results are obtained with kernel sizes 9×9, 3×3 and 5×5 for each convolution layer, respectively, and using 64 filters (channels) on the first layer and 32 on the second. They also observed that increasing the number of non-linear mapping layers (depth of the network) did not yield better results, probably due to difficulty during training.

Kim et al. [KKM16] showed that learning residuals instead of a direct low-to-high mapping – i.e. learning the difference between the low and high resolution image and adding it to the upscaled low resolution– allows for deeper networks with smaller kernel sizes, simplifies training, and yields better results. They use 20 layers of 3×3 convolutions with 64 channels. The input of the network is the low-resolution image interpolated to the desired high resolution size.

Finally, Long et al. [LSD15] showed how to convert an image classification network like VGG-16 [SZ14] onto a per-pixel segmentation network using a Fully Convolutional architecture (FCN). In VGG-16, the spatial resolution of the image is downsampled after every two or three convolutional layers through a max pooling layer, an operation that outputs the maximum value inside the kernel support. After some pooling steps, the image feature maps are encoded as a vector and fully-connected layers output the classification prediction for the whole image. Instead, Long et al. propose to compute the class per pixel without using fully-connected layers but applying 1×1 convolutions. Note that, given a set of n filters on a $w \times h$ image, a 1×1 convolution will compute for each position of that image a value based on these n filters, and the result will be a $w \times h$ map of values (in their case, the different classes).

One last step is required on the FCN in order to produce a segmentation map the same size as the input image. Since this 1×1 convolution is applied after some pooling operations have taken place, the spatial extent of the image

has been downsampled several times. Therefore, an upsampling step using a deconvolution – or a convolution with fractional stride – is required in order to recover the original size. Moreover, Long et al. [LSD15] showed that better results are obtained by upsampling and combining the filter maps from different levels of the network.

With these ideas in mind, we designed our network architecture which is depicted in Figure 5.24. It has two main components: an orthophoto analysis network (top row of the diagram), and a DEM analysis network (bottom row of the diagram). These two separate components are concatenated at different spatial resolutions and a height offset is computed from the joint information provided by both components.

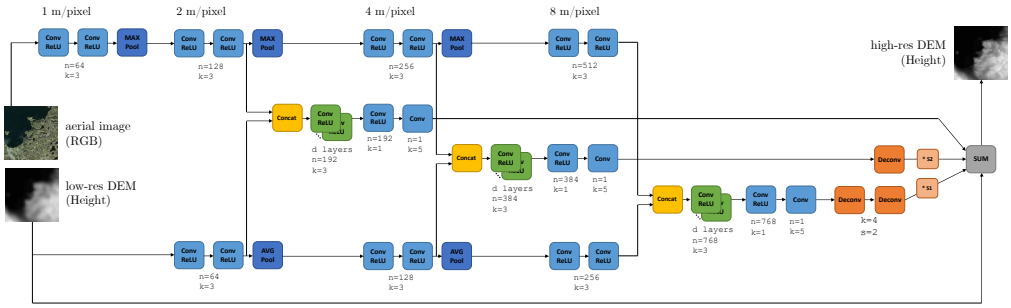


Figure 5.24: Architecture of our FCN.

The orthophoto analysis network will produce features from the aerial image in order to guide the offset computation depending on elements detected on the photo, so it can be thought of as a network that is inferring a segmentation of this orthophoto although we never end up computing the classification map. It has been adapted from the first layers of the image segmentation network `FCN-8s-atonce` by Long et al. [LSD15]. This allowed us to initialize this part with the weights of their network, which has been already trained with a huge set of images, and will improve the convergence of our training process. This part starts with an RGB aerial image at 1 m resolution, and applies two 3×3 convolutions followed by a ReLU activation layer. The resolution is then halved through a max pooling layer and two additional convolution-ReLU layers are applied, now with twice the number of channels.

The DEM analysis network follows the same structure, but starting at 2 m resolution instead of 1 m. The input is a low resolution heightfield upsampled to 2 m resolution (we used bicubic upsampling in our datasets) and with its mean substracted. Note that, in this part of the network, we changed the max pooling operation for an average pooling. The intuition behind this is that we are more

interested in the average shape of the low resolution terrain, rather than finding a signal in any of the positions inside the extent of the convolutions. In practice, we did not observe significant differences on the error metrics with either max or average pooling on the DEM analysis network.

Orthophoto network and DEM network are merged together by concatenating their features into a joint network. We can then apply d additional 3×3 Convolution-ReLU layers (green boxes in Figure 5.24) to the merged feature set before the last two convolutions that will produce the height offset. This offset is upsampled again to the original DEM resolution (2 m in our case) through as many deconvolutions as pooling layers had been applied. We fuse the information of different resolutions to produce the final height offset map as a weighted sum of the individual height offsets at each resolution. The weights of this sum are also learned parameters.

Network hyperparameters

Our network architecture has two main hyperparameters. In order to experiment with several network architectures and find the best hyperparameters, we used a subset of Pyrenees set distribution, with 5000 tiles for training and 2000 tiles for validation. After every training epoch, validation error was measured on the same randomly selected set of 100 tiles out of the 2000 in order to speed the computation, but a complete measure on the whole validation set was performed every 10 epochs.

The first hyperparameter is the set of resolutions that we use to compute the height offset. Theoretically, we could continue downsampling and merging information from orthophoto and DEM analysis networks to produce offsets as long as the spatial size of the image is still large enough. We compared four architectures varying the maximum resolution to which we downsample: up to 2 m, 4 m, 8 m and 16 m. In each case, the final height map is obtained as the weighted sum of all the offsets computed at each resolution and upsampled to 2 m (see Figure 5.24).

Figure 5.25 shows the evolution of the validation set RMSE with two different learning rates. Adding downsampled resolutions to the net contributes to lower error rates. However, the contribution of an additional downsample diminishes every time: using up to 16 m is only marginally better than using up to 8 m, but we need to include and learn 5 additional convolution layers, 2 deconvolutions and the scaling parameter. We can also see how this affects the learning process when using small learning rates (Figure 5.25 left): the network that uses reso-

lutions up to 8 m converges faster. Therefore, in our final architecture, we used downsampling up to 8 m.

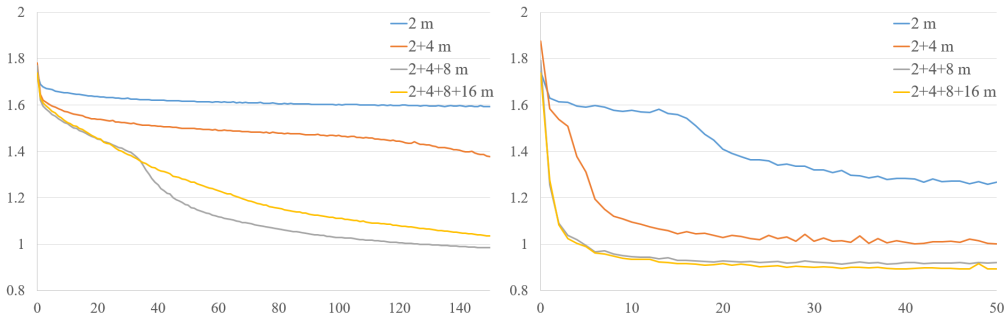


Figure 5.25: Validation RMSE on different network architectures with varying numbers of downsampling stages. Training using Adam and learning rate $\alpha = 10^{-6}$ (left) and $\alpha = 10^{-4}$ (right). Note the different scale of the horizontal axes, corresponding to the number of trained epochs.

Then, we tried $d = 1, 2, 3$ layers of 3×3 convolution-ReLU after merging the features at 2, 4 and 8 m. Results showed a slight improvement when adding one and two additional layers, but only marginal improvements using a third layer at the cost of slower convergence (Figure 5.26). In our final architecture, we kept $d = 2$ which produces the best results according to the validation set.

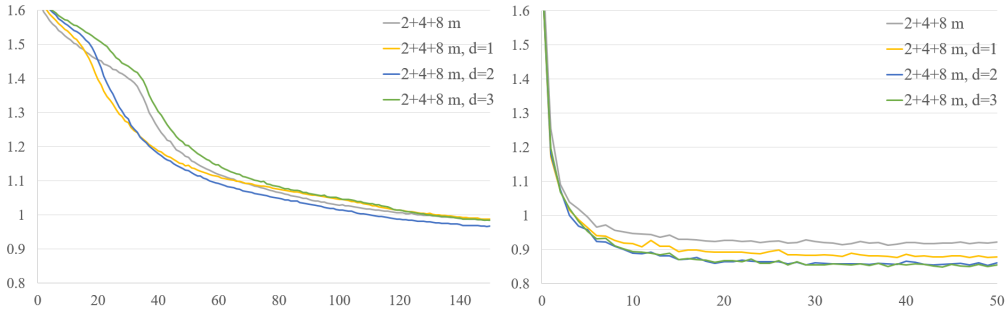


Figure 5.26: Validation RMSE on different network architectures varying the number d of convolution+ReLU layers applied after concatenating image and DEM features. Training using Adam and learning rate $\alpha = 10^{-6}$ (left) and $\alpha = 10^{-4}$ (right). Note the different scale of the horizontal axes, corresponding to the number of trained epochs.

Table 5.3 shows the training time of each network as ms/iteration, where each iteration corresponds to a forward and backward pass of a mini-batch of 4 terrain tiles. Note that the purpose of this table is comparative as in our

current implementation the selected terrain tiles and photos are read from disk at every iteration; keeping them in memory or in a database with an appropriate format would speed up these timings. Reconstruction times per tile do not show significant differences when using either small batches or individual tiles.

Network	Training	Inference
2 m	374 ms/iter	13 ms/tile
2,4 m	546 ms/iter	20 ms/tile
2,4,8 m	612 ms/iter	27 ms/tile
2,4,8,16 m	661 ms/iter	31 ms/tile
2,4,8 m, d=1	803 ms/iter	38 ms/tile
2,4,8 m, d=2	998 ms/iter	50 ms/tile
2,4,8 m, d=3	1130 ms/iter	62 ms/tile

Table 5.3: Training and inference times of the different networks.

Summing up, if we take into account the accuracy obtained and the training/inference times of the various networks we presented, it is reasonable to keep $2, 4, 8\,m, d = 2$ as our final network choice. In the following sections, we will present the results obtained on this specific architecture.

5.2.3 Training and validation

We have implemented our network using Caffe [JSD+14] and Python 3.5. Training and execution times have been measured on a computer equipped with Intel Core i7 3.40GHz and NVIDIA GTX 1070 hardware.

The orthophoto analysis part was initialized using the already trained weights of the image segmentation network FCN-8s-atonce [LSD15] and provided by the authors. Upsampling layers were initialized as a 2D bilinear kernel. The weights on the rest of layers were initialized using a Gaussian distribution with a standard deviation of 0.001 and biases were set to 0. We did not freeze the values on any weights or biases during training, all network parameters can be modified during the backpropagation step.

The model optimization has been performed using Adam [KB14], a gradient-based solver with adaptive moment estimation. We used the solver parameters recommended by the authors: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. Each training iteration was executed using a mini-batch of 4 randomly selected tiles, which corresponds to the maximum size we could run on our GPU. Figures 5.25 and 5.26 show the effect of learning rates $\alpha = 10^{-4}$ and $\alpha = 10^{-6}$. Using $\alpha = 10^{-4}$ makes the model converge very quickly, which can produce overfitting on some models.

In fact, using $\alpha = 10^{-3}$ caused the optimization to diverge. On the other side, setting $\alpha = 10^{-6}$ causes some models to take very long to converge, and smaller α can be impractical. Therefore, unless stated otherwise, we trained models using $\alpha = 10^{-5}$. Finally, L2 regularization was added with weight decay constant 10^{-4}.

Figure 5.27 shows the training error and validation error (RMSE) for each of the tree distributions we constructed (see Section 5.2.1). Due to the large size of the validation set, we only evaluate the validation error on a subset of 500 tiles after every 1000 training iterations, and evaluate the error on the full validation set after each epoch (full pass on the training set). As can be observed, with only a few training epochs the validation error converges. However, as training progresses on Pyrenees or Tyrol, the variance increases: the validation error becomes higher than the training error. This is a sign of overfitting the model to the training data. Luckily, when training on the joint distribution, the validation curve does not seem to suffer of overfitting. In fact, as we will confirm later in Table 5.5, it is beneficial to train on data from multiple sources.

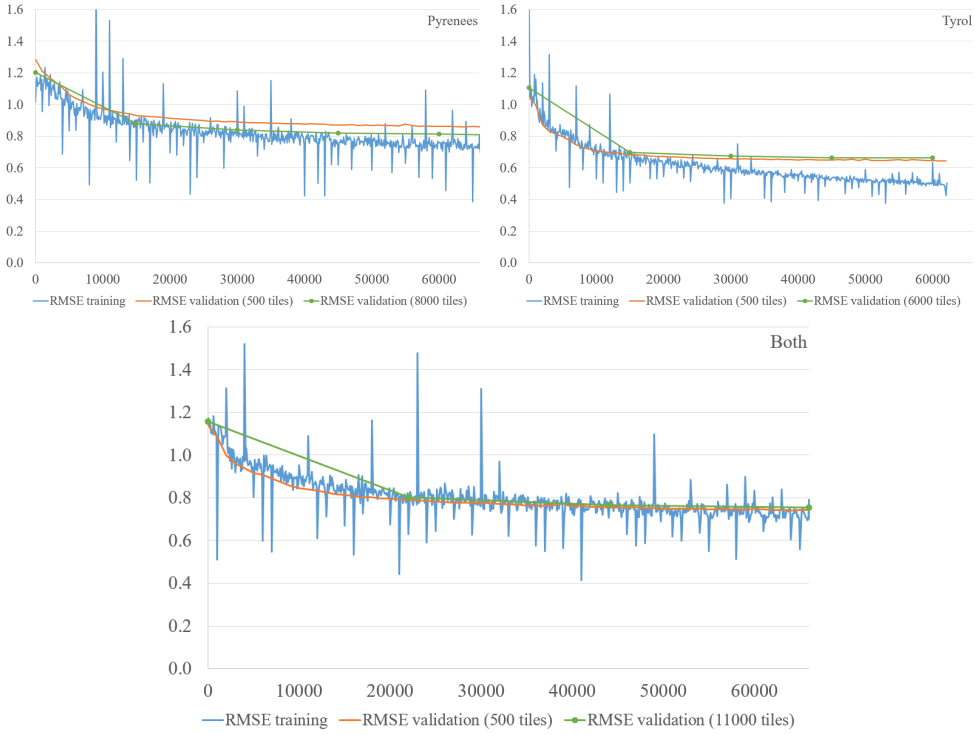


Figure 5.27: Training RMSE (blue) and validation RMSE on the full validation partition set (green with dots) and on a randomly selected subset of 500 tiles from it (orange). Training on Pyrenees, Tyrol, and both distributions.

Our network consistently reduces the error with respect to the low resolution

15 m DEM, as shown in Tables 5.4 and 5.5. For Pyrenees this error reduction is equivalent to using an 8 m DEM (assuming bilinear interpolation), while for Tyrol it is equivalent to a 7 m DEM (with the same assumption). This means that our output produces new elevation data faithful enough to resemble one of double the resolution. The additional detail contributes to its high resolution appearance and is visually plausible as verified by the user study described in Section 5.2.6. Training on Both converges to a RMSE on the validation set around 0.75 m. Table 5.4 summarizes the equivalent resolutions of the validation data errors.

set	**RMSE**			**equivalent res.**	
	bilinear	bicubic	validation	bilinear	bicubic
Pyrenees	1.45m	1.23m	0.80m	8m	10m
Tyrol	1.93m	1.50m	0.66m	7m	9m

Table 5.4: Accuracy improvement on validation data. The first two columns show the error between the 2m DEM (our ground truth) and the upscaled versions of the 15m DEM using bilinear and bicubic interpolation. The third column shows the error achieved by our network when densifying 15m DEMs on the validation set. Finally, the last two columns contain the resolution at which each of the interpolation filters (bilinear and bicubic, resp.) result in a RMSE equivalent to our method.

We also trained our network using 30 m (instead of 15 m) as low resolution DEM and keeping 2 m as high resolution. Initial validation errors were about 2.45 m and converged at around 1.56 m, in the order of the errors of the 15 m dataset. However, if we then apply the network trained from 15 m to increase the detail again, the output has larger RMSE, probably because the intermediate terrain (output from 30 m) does no longer contain the correlations between aerial image and DEM at 15 m. It may be observed that using either 15 or 30 meters as low resolution data, the trained network is capable to reduce the RMSE to about a 60% of the initial error, which is similar to the error of the data with twice the resolution.

It is also possible to evaluate the individual contribution of the two analysis networks. In order to do so, we just avoid concatenating the corresponding features into the joint analysis part of the network, and proceed with the training as usual. Figure 5.28 shows the equivalent networks that would be obtained.

If we train using only the DEM subnetwork, an approach similar to single image multi-resolution, the validation RMSE on Both converges to 0.89 m. On the contrary, if we train using only the aerial image network and sum the offset to the low resolution DEM, like in a shape-from-shading approach, the same distribution RMSE converges to 0.98 m. Recall that the combined network converges

to 0.75 m on this set, so these results confirm that both networks are contributing and complementing each other in our proposed architecture.

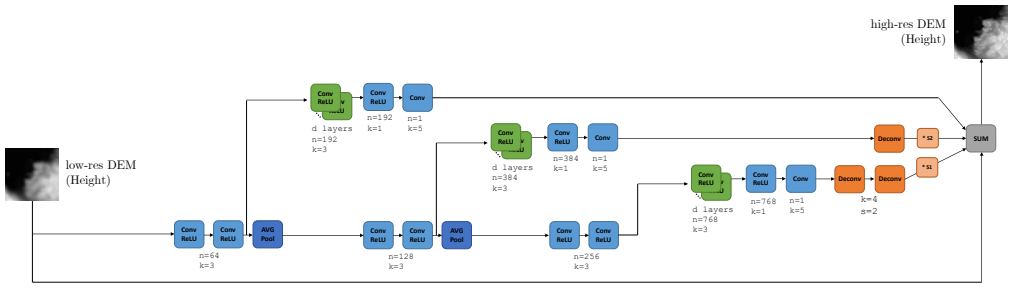


Figure 5.28: Super-resolution network using only the DEM.

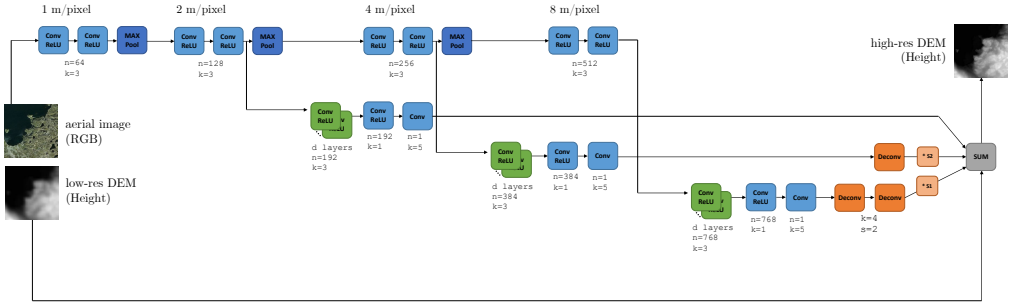


Figure 5.29: Shape-from-shading network using only the orthophoto.

5.2.4 Qualitative evaluation

We now apply the trained network using Both distribution (tiles from Pyrenees and Tyrol) to our test terrains that we omitted from training and validation splits. Figure 5.30 shows a render of the enhanced DEM on Forcanada (from Pyrenees), with specular lighting for better perception of the details. Figure 5.31 shows close up views on this terrain. For this figure, we have selected a normal example (top), an area in which the DEM did not contain much information (middle), and an area in which the aerial image contained large and dark shadows (bottom). The amplified results show that our network is able to leverage both the DEM and orthophoto information, and also compensates for the lack of good features from either of them.

Figure 5.32 shows one view on each of the four test terrains, with different amplification methods applied. The starting point for all tested amplification

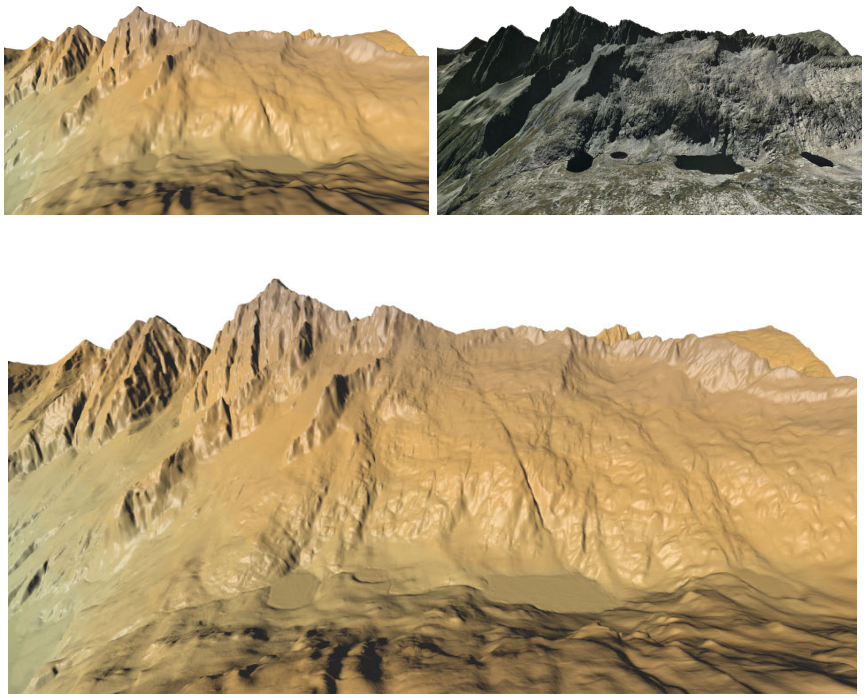


Figure 5.30: Visual evaluation of terrain super-resolution using our CNN. Top row: the low elevation dataset, and the corresponding aerial image used as texture. Bottom: the output terrain produced by the CNN.

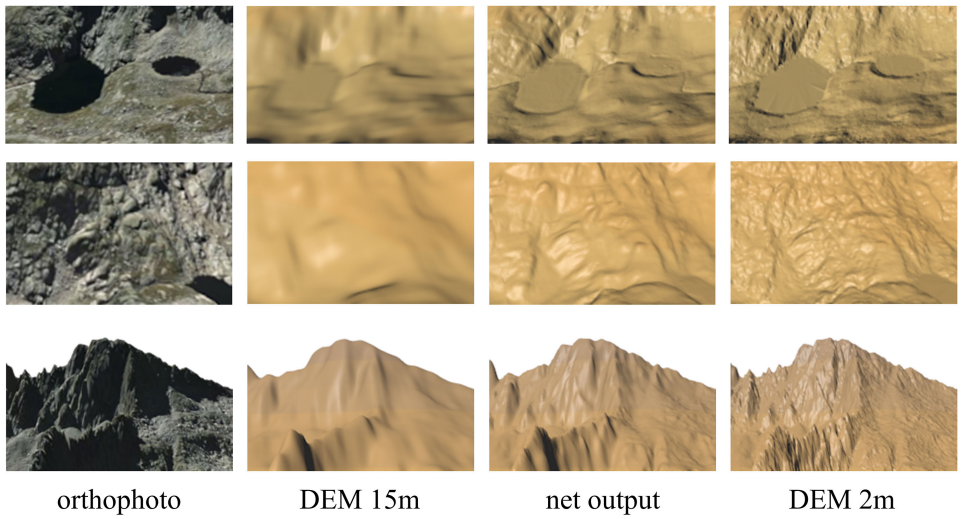


Figure 5.31: Visual evaluation of close-up areas.

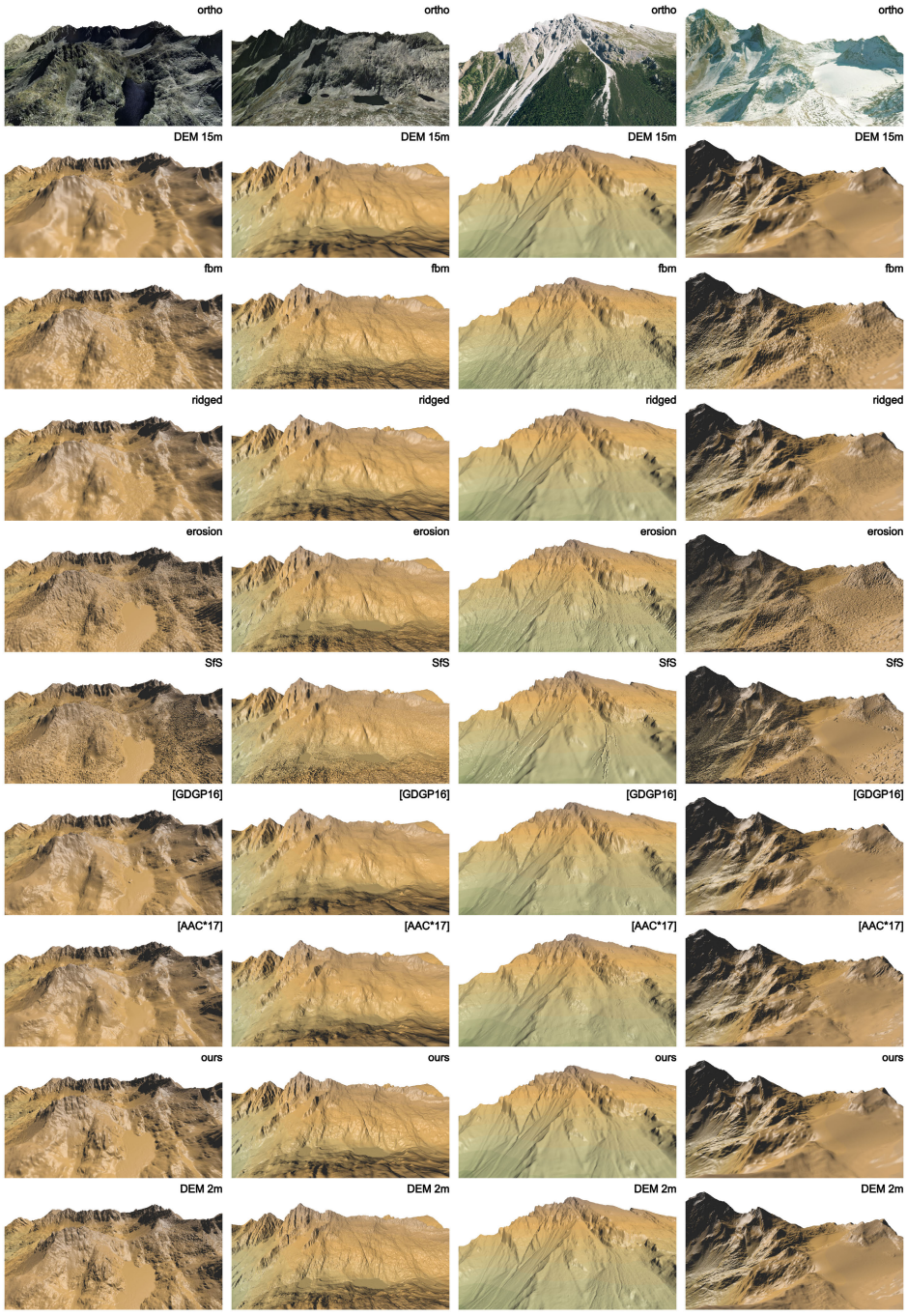


Figure 5.32: Comparison of different DEM amplification methods on the four test terrains: Bassiero, Forcanada, Dürrenstein and Monte Magro.

methods was the 2 m DEM generated by bicubic upsampling the original 15 m DEM and encoding it as a 16-bit PNG image. We did parameter tuning for all methods until we got visually pleasant results, as described below. Parameters using non-default values are shown in italics.

Noise-based amplification methods, such as Fractional Brownian Motion (fbm) and Ridged noise, can potentially provide as much detail as wanted. However, they place detail everywhere unaware of the morphology or terrain type. Therefore, we can clearly see bumps appearing on the flat lakes of the first two terrains, or on the smooth glacier of the last terrain. Noise-based versions in Figure 5.32 were generated using the *Fractal Noise* filter from Wilbur 1.86 (a freely distributable height field editor by Joseph Slayton). For fBm noise, we used *H=0.2*, lacunarity=1.9, *octaves=10*, amplitude=1. For ridged multifractal noise, we used *H=0.5*, lacunarity=1.9, *octaves=10*, amplitude=1, offset=1, gain=2. In both cases, the noise image was scaled by *0.02* and then added to the DEM. Notice that Wilbur's noise filters have been designed for terrain authoring rather than terrain super-resolution. We could not test more advanced options such as modulating noise strength with slope or altitude, which are found in other terrain authoring packages.

Erosion simulations add detail on terrains by transporting particles from upper to lower altitudes, creating gullies and accumulating sediments on flat regions. These simulations produce realistically-looking terrains and increase detail on low resolution terrains. In our particular case, however, we found two main issues. First, since our low-resolution terrains are already 15 m, much of the geological features that erosion causes are already present at this scale. Therefore, the simulation rapidly encounters many local minima and generates a large number of small rivers and pits. While this problem can be alleviated through parameter tuning, the method is still unaware of the underlying aerial image and can create inconsistent features (e.g. river-like flows or cavities appearing on top of a snowfield). Erosion images shown in Figure 5.32 were generated using Wilbur 1.86. We first added Gaussian random noise (with σ=0.5%) to prevent locally flat areas from producing large straight-segment channels. We applied the *Incise Flow* filter, which computes water flow across the surface and erodes the terrain accordingly. We used default values (amount=1, flow exponent=2/3, effect blend=1) , except for the *pre-blurring filter=0.5*, which controls the width of the water channels. We then applied the *Fill basins* filter to fill little depressions at the places where flows join, and we finally blurred the DEM image using a 3x3 box filter to avoid jagged edges.

Shape from shading approaches aim to produce offsets on a base model by analyzing the luminance of the corresponding texture. Its main problems, visible

in our images, are sensitivity to orthophoto noise and high frequency details such as small rocks, and shadowed areas in which relative luminance variations are tiny and therefore little or no offset is applied on the terrain (for example, right side of the first terrain, or above the central lake of the second). Displacement maps were created from the aerial RGB images using the GIMP plugin by Omar Emad (with default parameters). This map was scaled by *0.01* and then added to the DEM.

Then, we compare with the terrain augmentation techniques using dictionaries: the sparse synthesis by Guérin et al. [GDG+16], and the coherent multi-layer synthesis [AAC+17] presented in this chapter (Section 5.1), using the mean elevation and slope layers in the matching function. The dictionary was built from the terrains in Pyrenees and Tyrol as well as their rotated, flipped and transposed versions – as in Both distribution – with 8×8 pixel patches. These methods produce realistically looking terrains since they replace entire parts of the low resolution DEM by other high resolution DEM parts from the dictionary. By construction, the level of detail we can see in the result is the same as the high resolution detail provided in the exemplars (2 m in our case). However, relief shapes represented in the low resolution terrain are often substituted by different formations. For example, on the second terrain, we can see noticeable changes on the small rounded lake or the long diagonal gully in the rock walls above the big lake.

Finally, our network is capable of combining low resolution DEM and aerial image information and yields a consistently detailed terrain. While the resulting images look smoother than other algorithms, and lack very fine scale details, the relief features are more visually faithful to the 2 m DEM ground truth. The smooth look of our network-generated terrains is consistent with the discussion on the previous section: our RMSE is equivalent to that of an 8 m DEM, so our output terrain is a consistent upsample of the low-resolution input.

5.2.5 Quantitative evaluation

Our goal is to produce faithful amplification of terrains, and we have seen that the validation error is reduced to about a 60% when we train the network. Now, we analyze numerically how well our network performs on the four test terrains. Table 5.5 summarizes the RMSE and PSNR values obtained using our networks as well as those methods that could potentially augment the resolution of the DEM.

One enhancement method in the line of shape-from-shading approaches is

RMSE (m)	Bicubic	[YYD+07]	[GDG+16]	[AAC+17]	Net (Pyrenees)	Net (Tyrol)	Net (Both)
Bassiero(Pyrenees)	1.406	2.681	2.571	3.184	1.013	1.125	**1.005**
Forcanada(Pyrenees)	1.632	3.017	2.890	3.571	1.101	1.266	**1.097**
Dürrenstein(Tyrol)	1.445	3.650	3.508	4.637	1.122	0.941	**0.901**
Monte Magro(Tyrol)	0.917	2.293	2.205	2.993	0.708	0.600	**0.587**

PSNR (dB)	Bicubic	[YYD+07]	[GDG+16]	[AAC+17]	Net (Pyrenees)	Net (Tyrol)	Net (Both)
Bassiero(Pyrenees)	60.5	54.9	55.3	53.4	63.3	62.4	**63.4**
Forcanada(Pyrenees)	58.6	53.2	53.6	51.8	62.0	60.8	**62.0**
Dürrenstein(Tyrol)	59.5	51.5	51.8	49.4	61.7	63.2	**63.6**
Monte Magro(Tyrol)	67.2	59.3	59.6	57.0	69.5	70.9	**71.1**

Table 5.5: RMSE (in meters) and PSNR (in dB) of different algorithms applied to 15m DEMs with respect to 2m DEMs. The network has been trained for 60,000 iterations on Pyrenees, Tyrol, and both datasets.

Times (s)	**[YYD+07]**	**[GDG+16]**	**[AAC+17]**	**Ours**
Bassiero	199	6.1	20.9	14.9
Forcanada	104	3.2	12.9	8.0
Dürrenstein	40	1.4	4.2	3.3
Monte Magro	264	8.4	27.7	18.6

Table 5.6: Running times (in seconds) of three of the implemented enhancement algorithms.

to apply some iterations (3 in our tests) of a bilateral filtering on a bicubic upsampled terrain and using the aerial image as the source, following the spatial-depth super resolution method by [YYD+07]. The error increases with respect to just upsampling, mainly due to discontinuities introduced by shadows in the aerial image that are not present in the DEM as well as failing to add detail on dark areas.

The dictionary-based amplification methods of [GDG+16] and [AAC+17], described in Section 5.1, also increase the RMSE with respect to the bilinear or bicubic upsampled versions of the 15 m DEM. Note, however, these approaches are concerned with the plausibility of the synthetic results rather than fidelity to ground truth, and while RMSE increases the appearance of the terrain is that of a 2 m terrain, as shown in Figure 5.32. In the next section, we will evaluate how users perceive these details in terms of fidelity and plausibility.

Finally, the last three columns in Table 5.5 show the results for our network trained on the three different datasets for 60,000 iterations (about 16.5 hours, of which 5.5 are spent on disk reads). In all cases, the RMSE is reduced with respect to the upsampled versions, even when we apply to a terrain the network trained using only tiles from the other data source (e.g. applying the net trained using Tyrol to Forcanada from Pyrenees). This implies that our network is able to generalize well to other areas and data sources, even when the aerial images have different appearance.

Moreover, the best results are achieved when we apply the network trained using both data sources. On one hand, this is probably due to the variance reduction and less overfitting we saw in Figure 5.27 when training on the distribution that uses both data sources. On the other hand, this also means that the more data variety our network is trained on, the better it will generalize and perform. Consequently, as more regions release high-resolution datasets, our network can be trained further, incorporating these datasets, and it will keep improving it performance on other areas.

Table 5.6 compares the running times of our networks and our implementation of the method we compare against. Note that our network runs on GPU using Caffe [JSD+14], our implementation of [GDG+16] and [AAC+17] uses Matlab built-in functions, and the bilateral filtering runs on multi-threaded CPU.

5.2.6 User study

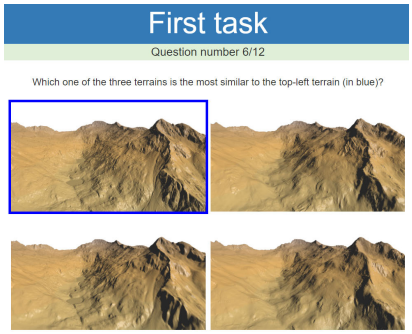
We conducted a user study to compare our method with competing approaches in terms of visual fidelity (perceived closeness to ground-truth) and visual plausibility (perceived consistency with the aerial image). We selected a set of 6 viewpoints on our four test datasets and rendered the scenes produced by the different methods.

Our study consisted of two parts: fidelity and plausibility. In the *fidelity task* (Figure 5.33), participants were presented three images ([GDG+16], [AAC+17] and ours) and had to choose the most similar to a reference image, which was a render of the *high-resolution DEM*. Since this task clearly penalizes deviation from ground truth, we excluded fractal methods that clearly increase such a deviation.

In the *plausibility task* (Figure 5.34), participants were presented two images (a pair randomly selected from all the methods we compared against to), and had to choose the most plausible according to a reference image, which included a render of the *low-resolution DEM* with and without aerial imagery.

The study was deployed on a website, and participants (22, contacted through email, aged 18-55) could complete the task remotely using the device of their choice. In both tasks, the order of images was randomized between methods and between trials (12 and 10 trials for each task, respectively).

We applied Bayesian data analysis [Kru14] and modeled the posterior probability of each method being selected as a Bernoulli random variable with a



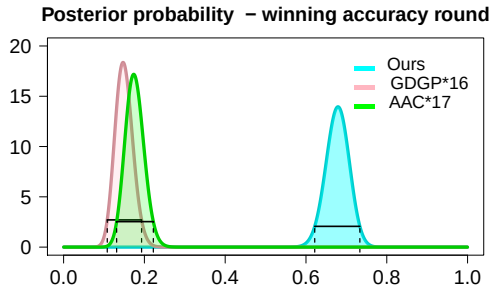
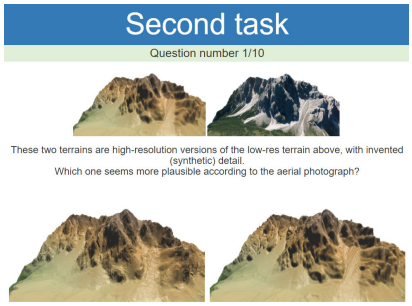


Figure 5.33: User study first task: fidelity. Left, layout of the web page. Right, posterior probability $p(\theta|D)$ of winning a fidelity round.



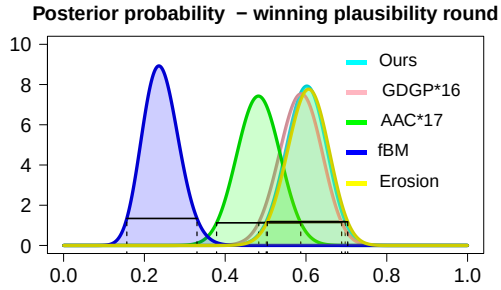


Figure 5.34: User study second task: plausibility. Left, layout of the web page. Right, posterior probability $p(\theta|D)$ of winning a plausibility round.

uniform Beta prior. Reported results represent the posterior mean, and the confidence interval (CI) represents the range including 95% of the posterior probability.

Fidelity to ground truth. As shown in Figure 5.33, our method was significantly more likely to be selected as the best in terms of accuracy with respect to the high-resolution ground truth, when comparing against the dictionary-based approaches. Mean probabilities were: 15% (11%, 19% CI) for [GDG+16], 18% (13%, 22% CI) for [AAC+17] and 68% (62%, 73% CI) for our method. These results clearly demonstrate that the elevation detail created by our approach is more faithful to actual DEM data than competing example-based approaches.

Plausibility of added detail. We computed how likely each method was to be selected as most plausible in a round with a random competitor. Figure 5.34 shows no clear differences among methods, with the exception of fBm which was

significantly less likely to be selected as the best (notice that glossy reflections could have over-emphasized noise features giving unnatural effects). None of the other differences were statistically significant. Our method performed reasonably well, considering that all other methods can provide higher resolution detail.

5.3 Summary

Our contributions on terrain enhancement can be summarized as:

- **A novel multi-layer example-based approach to synthesize realistic landscapes**

  We have presented a multi-layer example-based approach to synthesize realistic landscapes, i.e. terrains containing heterogeneous information layers such as elevation, vegetation density or soil type. The cost function for matching dictionary atoms with terrain patches allows joint synthesis of coherent information layers. The input can be real digital elevation models, rough sketches drawn by hand, or a combination of both, containing a single layer (e.g. elevation), or multiple layers. The output terrain contains always as many layers as available in the dictionary, thus providing coherent data amplification.

  A key feature of our approach is to provide a unified, easy-to-control, and flexible model for multi-layer landscape synthesis generating plausible and predictable results. Our method provides control to the user and allows him to create any arbitrary layer in a coherent way. As the implementation is fast and can be easily parallelized, it could be possible to develop interactive terrain editors based on our strategy.

- **A new Convolutional Neural Network architecture for terrain super-resolution**

  We have proposed an architecture of a CNN capable of inferring high resolution detail from a low resolution DEM and the corresponding orthophoto.

  Numerical evaluation of our network showed that it effectively doubles the resolution of the elevation data. When we input a 15 m resolution DEM, the resulting DEM has a RMSE with the 2 m ground truth similar to the RMSE of this ground truth downsampled to around 8 m and then upsampled again using bicubic interpolation. The obtained RMSE is also better than other approaches like dictionary-based methods or depth-from-shading filters.

  Qualitative evaluation from the user study showed that this error decrease is perceived as higher fidelity to the original terrain when compared with

other approaches. Moreover, our generated DEM is comparable in terms of plausibility with alternative DEM amplification methods, some of which can potentially provide details at an arbitrary scale.

5.4 Publications

Our work on terrain and DEM enhancement led to the following two articles. The first one contains the dictionary-based method for enhancing DEM from exemplars, and the second one the DEM super-resolution using the aerial image.

- O. Argudo et al. "Coherent multi-layer landscape synthesis". In: *The Visual Computer* 33.6 (2017), pp. 1005–1015
- O. Argudo, A. Chica, and C. Andújar. "Terrain Super-resolution through Aerial Imagery and Fully Convolutional Networks". In: *Computer Graphics Forum* 37.2 (2018), pp. 101–110

We also provided an online repository with the source code, datasets, and trained weights of our convolutional neural network, available at:
`http://gitrepos.virvig.eu/oargudo/fcn-terrains`

6

Plausible vegetation synthesis

The vegetation layer plays a key role for the realism of natural scenes like mountains or forests. Typically, artists create a set of tree models resembling the species of the represented terrain area. However, time and budget constraints may limit the size of the modeled set. Procedural approaches can help in generating as many as needed varied trees, but adjusting the parameters and controlling these methods to produce a particular type of tree is not always intuitive and requires expertise in the used algorithms and tools. We envision a fully automatic tree generation pipeline for populating terrain models with plausible native vegetation, starting from photographs of local species. This chapter will describe this pipeline alongside the contributions we have made.

6.1 Pipeline for vegetation modeling from pictures

The first step for populating a scene with local vegetation is obtaining a representative set of trees found in the region. Some tools like Flickr map search [4] offer a combined search by semantic tags and location, so we can look for tree photographs taken in or nearby our terrain area. Another possibility is to obtain a list of the species from an area – for example, from land cover maps like [15] – and then search images using these names. In either case, the obtained images still need to be analyzed to check whether they contain a tree and segment it – ideally, with alpha matting.

While recent Deep Neural Networks have proven to achieve remarkable results on image segmentation [GOO+17], they are still not directly usable in our fully

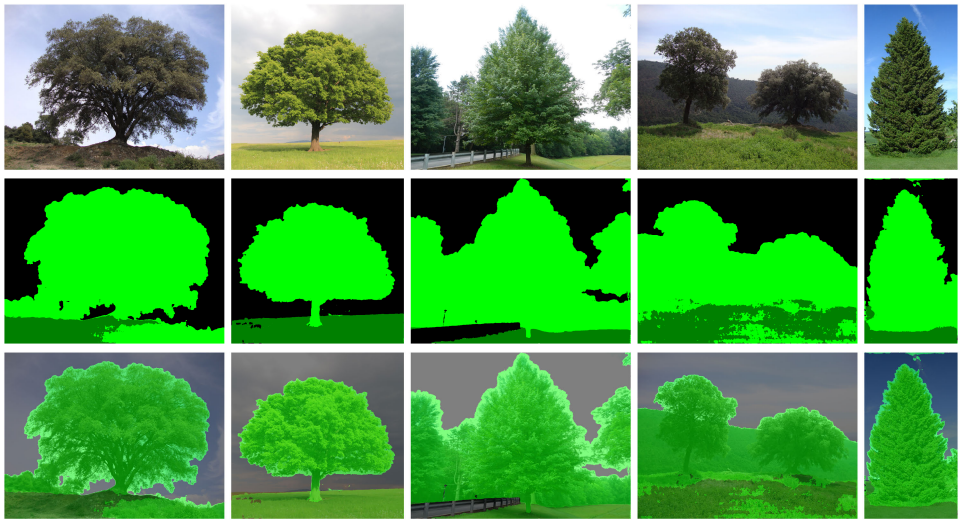


Figure 6.1: Vegetation segmentation using a state-of-the-art deep network [CZP+18]. Input photographs, segmented vegetation (light green) and terrain (dark green), and segmentation overlaid on the picture.

automatic pipeline. For example, Figure 6.1 illustrates vegetation segmentation using the state-of-the-art network DeepLab-v3+ [CZP+18] with the Xception-65 backbone [Cho16] and trained on both ImageNet and Cityscapes [COR+16] datasets. Note that Cityscapes contains tree and terrain (grass/soil/sand) labels, but as the name suggests this collection is made of urban street scenes. This test shows that vegetation cover is effectively recognized, but more training or specific datasets would be required in order to correctly identify the individual trees.

Regarding the matting step, Figure 6.2 shows results using the trained CNN by Cho et al. [CTK16]. In this case, the quality usually depends on the input trimaps provided by the user indicating opaque, fully transparent and intermediate regions (Figure 6.2 top). As image segmentation and matting was out of the scope of this thesis goals, we have been providing tree photographs, segmentation and mattes manually. Yet, we believe this step could be fully automated in the near future.

The next step in our proposed pipeline is creating tree models from the set of pictures. While static images can be directly used as billboards in distant views, they lack the volumetric detail needed for closer views or shading. Therefore, we propose to automatically generate tree models that resemble the trees in the photographs. The result will be a 3D model of the branching structure and textured leaves that will look as similar as possible to the provided picture.

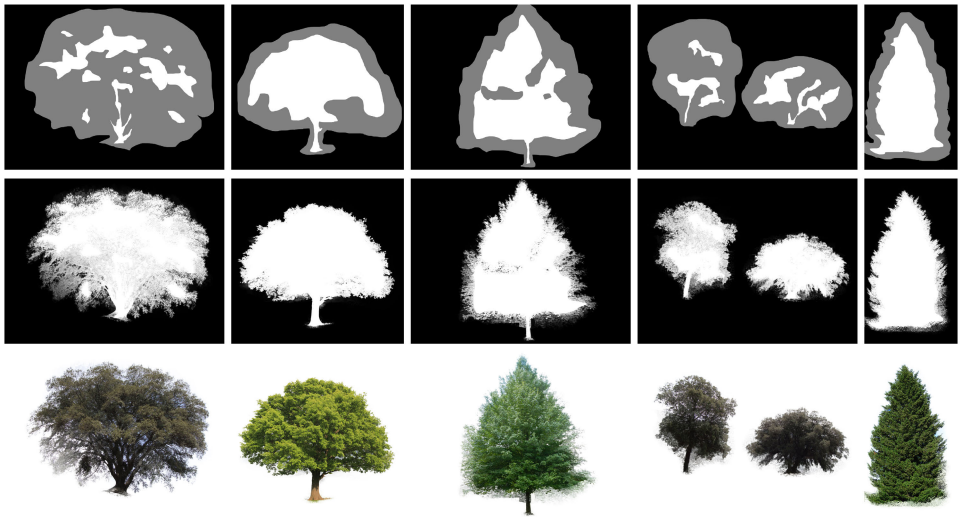


Figure 6.2: Tree matting using a state-of-the-art deep network [CTK16] on the same photographs from Figure 6.1 top. From top to bottom: user-painted trimaps, output matte, result using the obtained matte as alpha channel

The last step is including the models in the scene and being able to efficiently render thousands of them. Consequently, the data structure used for the tree model must be properly designed to be rendered as fast as possible on a GPU, and allow for Level of Detail techniques without noticeable transitions between representations.

Additionally, a realistically looking forest scene does not contain repetitions of the same tree or, at least, appears not to have them. Tree variety can be added at different stages of our proposed pipeline: before the modeling step by obtaining or synthesizing more tree pictures; during the modeling step by creating more than a tree model per photograph; and during rendering by adding color variations, transformations and small deformations on each instance of a tree model. Furthermore, if we independently animate the leaves or branches of each tree during navigation through the scene, it will not only increase variety but realism as well.

6.2 Tree picture variations

Using photographs of trees as billboards is a common technique for populating a scene with vegetation. However, in order to produce realistic and convincing

renders using them, a good alpha mate is also needed. There exist some billboard libraries that contain tree pictures with matting, but they are usually too few to cover a scene without repetitions, and usually only a scarce set of variations per species is included. Moreover, since these billboards have been edited by some artist in order to create a good alpha channel, they usually ask for a small fee per picture. Therefore, in this section, we will describe our approach for generating a huge variety of pictures of trees from a reduced set of exemplars.

We assume a *library* $\mathcal{L}$ of tree images (RGBA side views of trees) is available. Tree images from the library (or a user-defined subset of it) will be referred to as *exemplars*. Our approach has two main stages: exemplar processing and tree synthesis.

An overview of the exemplar processing stage is shown in Figure 6.3. We start by resizing all exemplars to a fixed resolution. A transparent border is added if necessary to preserve the aspect ratio. Each non-transparent exemplar pixel is then segmented into crown and trunk classes using a trained Convolutional Neural Network. We then extract all contours of the alpha channel using a border following algorithm, and take the largest external contour as the overall tree contour C.

Then, we resample C to create a new contour C_r with a fixed number N of 2D points. The resampling uses three pinpoints (tree bottom, trunk top-left, trunk top-right) that are computed from the intersection of the tree contour with the segmented crown/trunk components. We then resample again the contour

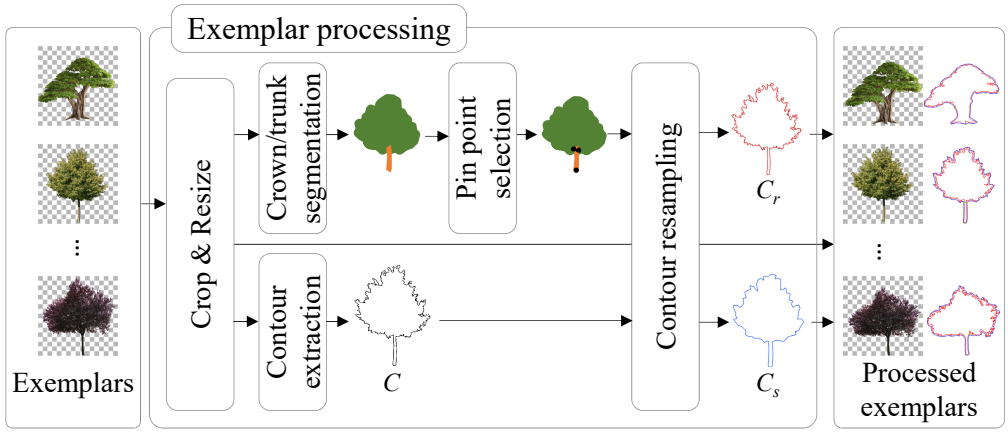


Figure 6.3: Overview of the exemplar processing algorithm. Given a collection of RGBA images of trees, we generate fixed-length contours representing the overall shape of the trees.

C_r to obtain a simplified contour C_s that will be used later to encode interior points of the tree as Mean Value Coordinates. The output of the preprocessing step is, for each tree exemplar, a couple of contours C_r and C_s, along with a binary mask representing the crown/trunk segmentation.

Regarding tree image synthesis, we propose two variants. The first one creates tree variations from scratch, using exemplars from the library. A new overall contour is created by computing a random convex combination of the exemplar contour vectors C_r. This contour already defines a preliminary segmentation of the output alpha channel. Then, we transfer the RGBA color from some exemplars to the target tree, using Mean Value Coordinates inside C_s. This allows us to associate each point of the target image to a matching point on the source(s) images from which we will compute the final RGBA color. Since we also transfer alpha values, the final shape of the tree is richer than the original contour.

The second synthesis variant requires the user to specify a tree image T. This image will be combined with features from the library to create variations of T. In this case, we apply to T all the processing steps we apply to exemplars, and then apply a similar synthesis procedure but giving higher weights to T features.

6.2.1 Exemplar processing

We now describe the preprocessing steps to be performed for each exemplar RGBA image from the library. The main outcome of these steps is a suitable encoding of the overall tree shape through a fixed-length contour.

Image normalization

We start by normalizing the images so that all exemplars have the same size. This normalization is beneficial for subsequent steps, specially for the segmentation through FCN. In particular, we perform the following normalization steps. We first compute a binary version of the alpha channel through alpha thresholding, and set all transparent pixels to black to simplify the classifier task. We then crop the image to the minimum axis-aligned rectangle that encloses all opaque pixels, and add a padding black (and transparent) border to make the image square without distorting the tree. Finally, we resize the image to a fixed size (we used 1024×1024 pixel images for the experiments). Figure 6.4 shows the normalized versions of a collection of exemplars from different sources.

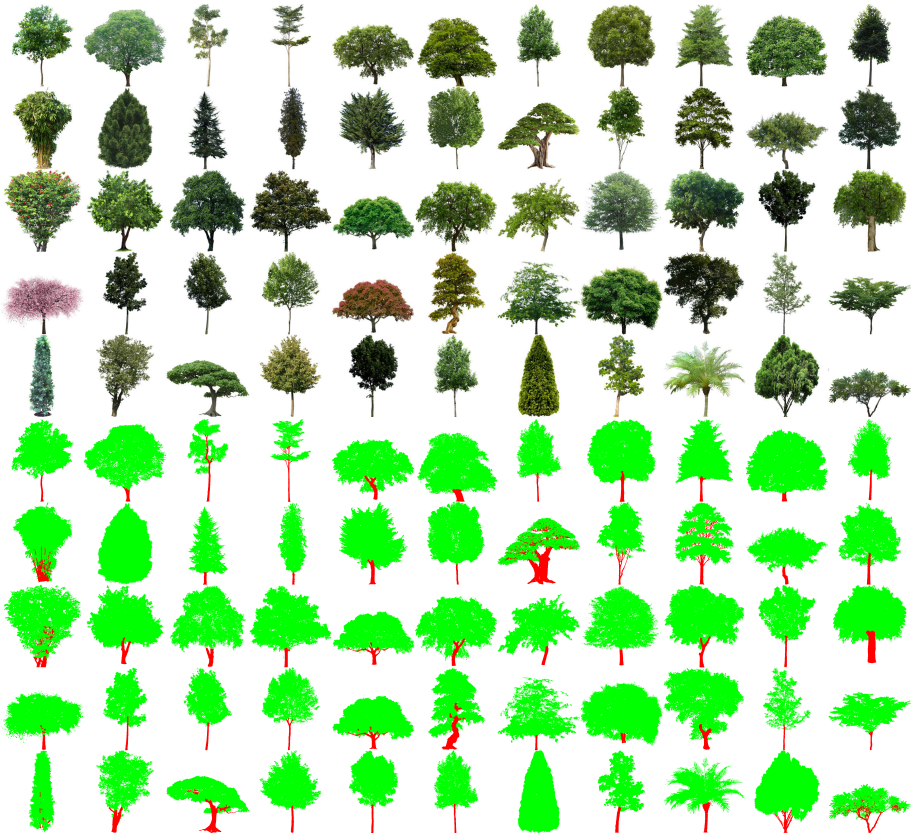


Figure 6.4: Tree images used as exemplars and our manual segmentation into crown/trunk.

Contour extraction

We extract the overall (external) tree contour from the alpha channel A. We first threshold the alpha channel (we used 0.5 as threshold, assuming normalized values) to get a binary alpha mask A_t. Then we apply a border-following algorithm [Suz+85] to A_t to extract all the contours separating opaque regions from transparent ones. Each contour is represented as a collection of 2D point coordinates.

Figure 6.5 shows the contours extracted from some exemplars. Typical tree images include multiple contours; trees with sparse foliage have multiple see-through parts (holes in A_t) and even multiple connected components (due e.g. to thin branches not appearing in A_t). In our exemplar set, the number of contours varied from 1 (trees with opaque dense foliage) up to 5,481 (a very

sparse tree).

We classify the extracted contours as exterior (not inside any other contour) and interior (inside another enclosing contour). We take as the overall contour the longest exterior contour C (Figure 6.5). Notice that $C = \{(x_i, y_i)\}$ will have a variable number of points depending on, among other factors, the fractal nature of the tree silhouette. So far, we only guarantee that contour C vertices are given in counter-clockwise order.

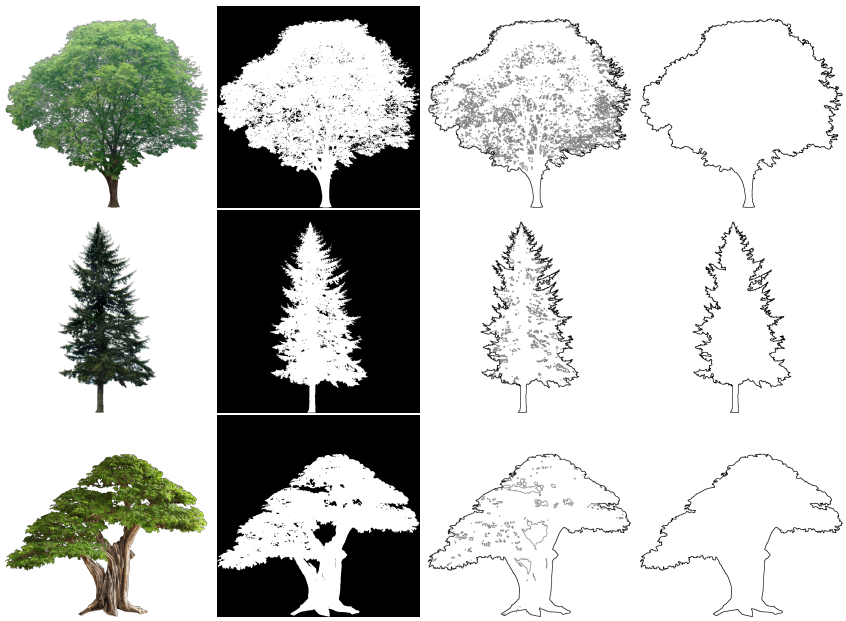


Figure 6.5: Some RGBA images (left), thresholded (0.5) alpha channel (center left), all contours extracted from this mask (center right), and longest external contour (right). Internal contours are shown in light gray, and external contours in black.

Crown/Trunk segmentation

The contour contains both the tree crown as well as the trunk. For better results, we would like to have approximately the same number of contour points belonging to the trunk independently of how big each trunk is. Therefore, we need to segment the crown from the trunk.

While several image segmentation approaches exist, state-of-the-art approaches mainly rely on deep Convolutional Neural Networks. Long et al. [LSD15] explain

how classification networks can be converted into fully convolutional networks (FCN) such that a per-pixel segmentation can be learned end-to-end. They extend various networks into their respective fully convolutional form. We decided to use the network they refer to as FCN-8s-atonce, which they obtain by extending into a FCN the VGG16 classification net [SZ14] - which, in turn, had been trained using the ImageNet database. The authors provide their network implementation and trained weights for the Deep Learning framework Caffe [JSD+14]. We downloaded it, modified the output layer to produce three classes - background, crown and trunk - and fine-tuned the net.

From our tree dataset, we segmented manually 55 exemplars (Figure 6.4), trained the net for 200 epochs (21 s/epoch), and obtained around 95% accuracy and 86% mean IoU (Intersection over Union) on the same training set. Since our number of exemplars was limited, we did not split them into train and validation sets. Moreover, we expect new exemplars to be very similar to those already provided, and obtaining a rough segmentation suffices for our needs as we shall see in the next section. Figure 6.6 shows the segmentation output of our net on new inputs not seen during the training phase. Segmenting each of these 1024×1024 images takes on average 15.5 s on a GTX 1070 GPU.

Figure 6.6: Trees segmented automatically using our FCN.

Pinpoint selection

The creation of new contours through linear combinations of existing contours requires the definition of a minimum set of matching points across all contours, so that e.g. the first contour point always refers to the point at the tree bottom.

For each exemplar, we select three pinpoints on the extracted contour C: tree bottom (B), trunk top-left (L) and trunk top-right (R), see Figure 6.7. The tree bottom is set to the index of the point with minimum y. If multiple points share such a minimum, we set B to the index of the median point. For the trunk L and R points, we traverse the points in C, starting from B, in both forward and backward directions. We stop the traversal as soon as the current contour

segment intersects the crown baseline, i.e. the lowest height on the segmented crown. In the rare case that the segmented image contains no trunk pixels, we set $L{=}R{=}B$.

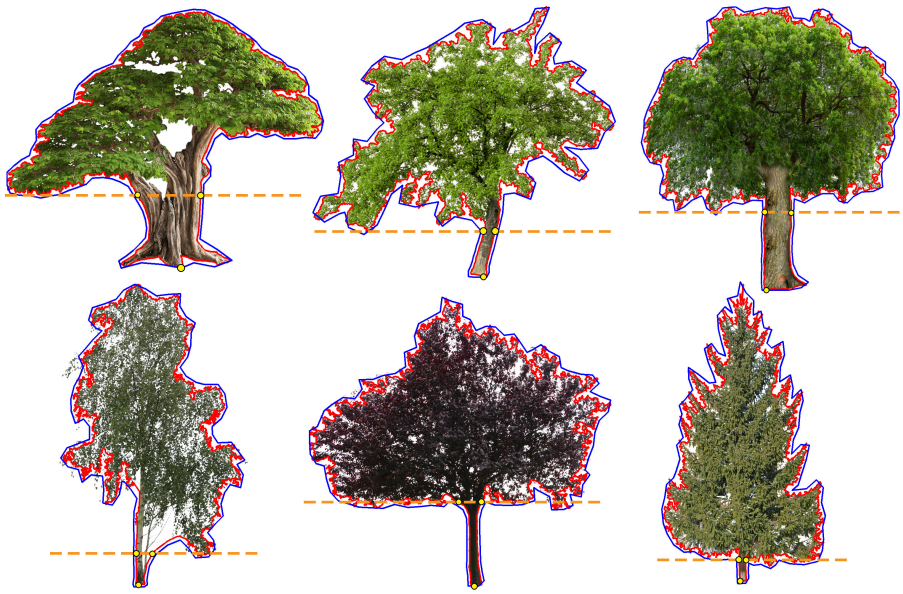


Figure 6.7: Resampled contour C_r (in red), simplified contour C_s (in blue), crown baseline (in orange) and pinpoints (in yellow). Top row: manually segmented trees, bottom row: automatically segmented trees.

Resampled contour C_r

Extracted contours C are variable-length and thus not suitable for generating variations through linear combinations of exemplars. Therefore, we resample the overall contour C of each exemplar to include exactly N points. Our resampling strategy considers three different segments on C: the crown segment (R to L), the left truck segment (L to B) and the right trunk segment (B to R). Each segment is assigned a fixed number of samples in the resampled contour C_r.

Resampling within each segment is performed as in chord-length parameterization, i.e. attempting to generate uniform chord lengths between samples. The output contour C_r has thus N points, all of them uniformly distributed (in a chord-length sense) within each segment. We then concatenate the mean-subtracted (x, y) coordinates of the N contour points onto a $2N$ vector $\boldsymbol{u}$ that represents the overall tree shape in $\mathbb{R}^{2N}$ space as $\boldsymbol{u} = [\overline{BR}, \overline{RL}, \overline{LB}]$.

With this parameterization, we are trying to enforce similar morphological elements of the tree (e.g. branches on the left side of the crown) to be located in nearby indices of the $2N$-dimensional vector among different exemplars. The reason of dividing the tree trunk in two segments using the bottom B is to prevent rotations of the trunk shape, and allowing the bottom of the tree to remain always the lowest point when morphing between shapes.

In our experiments, we used $N = 2000$ points, allocating 1600 points for the crown segment, and 200 points for trunk segments. In the special case $L{=}R{=}B$, we directly take all 2000 points from the contour C starting at B. Figure 6.7 shows the resampled contours (in cyan) for a few exemplars.

Simplified contour C_s

The resampled contour C_r is detailed enough to be used for contour synthesis, but too complex for the generation of Mean Value Coordinates. We thus further resample C_r to $M = 100$ points to generate a simpler contour C_s.

We observed that the distortions on the color image produced by the Mean Value Coordinates after warping the contour are more acute for those pixels near the contour or outside of it. Therefore, we actually generate C_s by first rendering the mask of the interior of C_r, dilating this mask for some iterations (8 in our tests) and resampling to M points the contour of this mask. Figure 6.7 shows the simplified contours (in red) for a few exemplars.

6.2.2 Tree picture synthesis

We now discuss different strategies to generate new tree images through random combinations of exemplars. We first define the overall tree shape by synthesizing a new contour through linear combinations of contours. Then, the contour is filled by transferring RGBA color from the exemplars.

Contour synthesis

Let $\boldsymbol{u_j} \in \mathbb{R}^{2N}$ be the vector that results from flattening the coordinates of the resampled contour C_r from the j-th exemplar.

We can linearly interpolate two contours $\boldsymbol{u'} = (1-t)\boldsymbol{u_0} + t\boldsymbol{u_1}$ with $t \in [0,1]$ to produce a continuous morphing between them. Figure 6.8 shows several snap-

shots of the interpolation between two contours. The quality of the interpolated contours is highly dependent on the matching contour points; the use of the B, L, R pinpoints prevents excessive rotations during morphing.

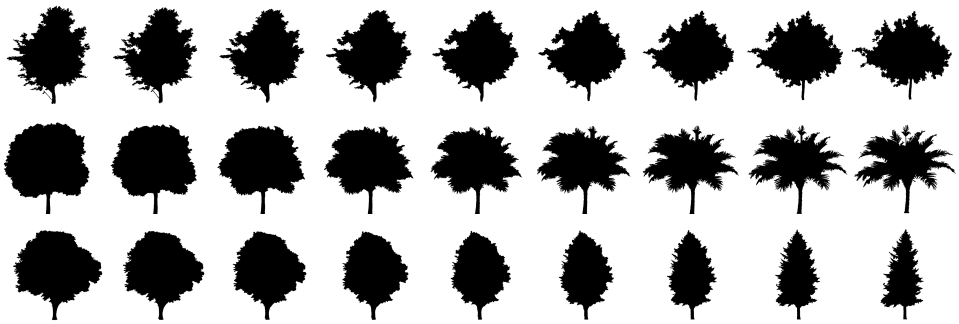


Figure 6.8: Morphing between two contours through convex combinations. From left to right, $w_2 = 0, 1/8, 2/8, \ldots, 1$

We can extend this idea to incorporate additional contours through convex combinations of existing contours, i.e., $\boldsymbol{u'} = \sum w_i \boldsymbol{u_i}$, with $w_i \geq 0$ and $\sum w_i = 1$. We avoid negative weights to prevent contours from being reflected (e.g. $-\boldsymbol{u_i}$ would result in an upside-down contour).

The generated contour C'_r as a combination of resampled contours C_r provides a preliminary version of the output alpha channel (see Figure 6.8), which will be refined later as we explain in the following section. We obtain this alpha mask by simply drawing the contour onto a blank alpha channel (with alpha set to 1.0 for all pixels), and then using a region fill algorithm from any seed outside the contour to clear the alpha values of the pixels outside this contour. This method is robust against potential self-intersections of the combined contours.

Color transfer

So far, we have generated new contours C'_r and their associated alpha masks. Analogously, we also obtain the new contours C'_s as a convex combination of the simplified contours C_s applying the same weights. We now explain how to fill the non-transparent pixels of the preliminary alpha channel with color.

We address this problem by transferring color –and transparency– from one or more exemplars (the *source* images) to the image being synthesized (the *target*). We pose this color transfer problem as an *image warping* problem. Let $w \times h$ be the resolution of the (processed) exemplar images, and let Ω be their $w \times h$

rectangular domain. Given a source contour C_s and a target contour C'_s, both with vertices in Ω, we aim at defining a smooth warp function $f : \Omega \mapsto \Omega$ mapping each vertex $(x_i, y_i) \in C_s$ to the corresponding vertex $(x'_i, y'_i) \in C'_s$. Such a warping function can be used to deform any source image E defined on Ω to a target image E' by simply letting $E' = E \circ f^{-1}$.

We define the mapping above through barycentric coordinates with respect to (a simplified version of) the source and target contours. In particular, we use mean value coordinates [HF06], which are well-defined for arbitrary planar polygons.

When transferring RGBA color from a single source exemplar E, the algorithm proceeds as follows. For each non-transparent pixel $p' = (x', y')$ of the target image E', we first compute the mean value coordinates λ'_i of p' with respect to the target contour C'_s. We then find the corresponding point on the source image, $p = f^{-1}(p')$, by simply using the resulting coordinates λ'_i with the vertices $\{v_i\}$ of the chosen source contour C_s, i.e. $p = \sum \lambda'_i v_i$. The final RGBA color for pixel p' is just $E(p)$. As in [HF06], color sampling can be improved through bilinear interpolation on the 2×2 grid of pixels surrounding each source pixel.

The color transfer approach above can be extended to take colors from multiple exemplars. Let (c_j, a_j) be the RGB and A components of the color extracted (through mean value coordinates) from j-th exemplar. We compute the output alpha value as $a' = \max a_j$, i.e. the final pixel will take the highest opacity from the source pixels. We do this to avoid an excessive amount of transparent pixels to be transferred to the target image. The color is computed as a random convex combination of the exemplar colors with non-null opacity values. Fig-

Figure 6.9: Examples of morphing between two trees through RGBA color transfer and contour deformation.

ure 6.9 shows examples of morphing between two exemplars using the RGBA color transfer to fill the interpolated contours.

Histogram transfer

As a result of the previous operations we have been able to generate new trees. We can add more variation by changing the color histogram of the generated image. Since we want the result to be plausible, we use a histogram transfer algorithm [GW07] so that the image we have generated has the same color distribution as another image provided as a reference.

In order to do this we compute the cumulative histograms for both images (source image and template image). We then interpolate linearly to find the unique pixel values in the template image that most closely match the quantiles of the unique pixel values in the source image. This process is performed for each RGB color channel separately and it always ignores transparent pixels.

Histogram matching is also useful both to improve the matching between the combined images, since they can be taken in varying lighting conditions, as well as to simulate the change of vegetation coloration during different seasons.

Figure 6.10 shows an RGB transition using histogram matching. Note how the histogram matching produces much more plausible results and smoother color transition.

Figure 6.10: Morphing two trees through RGBA color transfer; with histogram transfer (top) and without (bottom).

6.2.3 Examples and results

Figure 6.11 shows some convex combinations of multiple contour pairs with weights $w_1 = 1/3, w_2 = 2/3$ (upper triangle of the table) and $w_1 = 2/3, w_2 = 1/3$ (lower triangular part). Figure 6.12 shows the same combinations of multiple tree pairs, now distorting the image of each row towards the shape of the image in each column. Despite trying to combine radically-different tree species, most combinations look plausible.

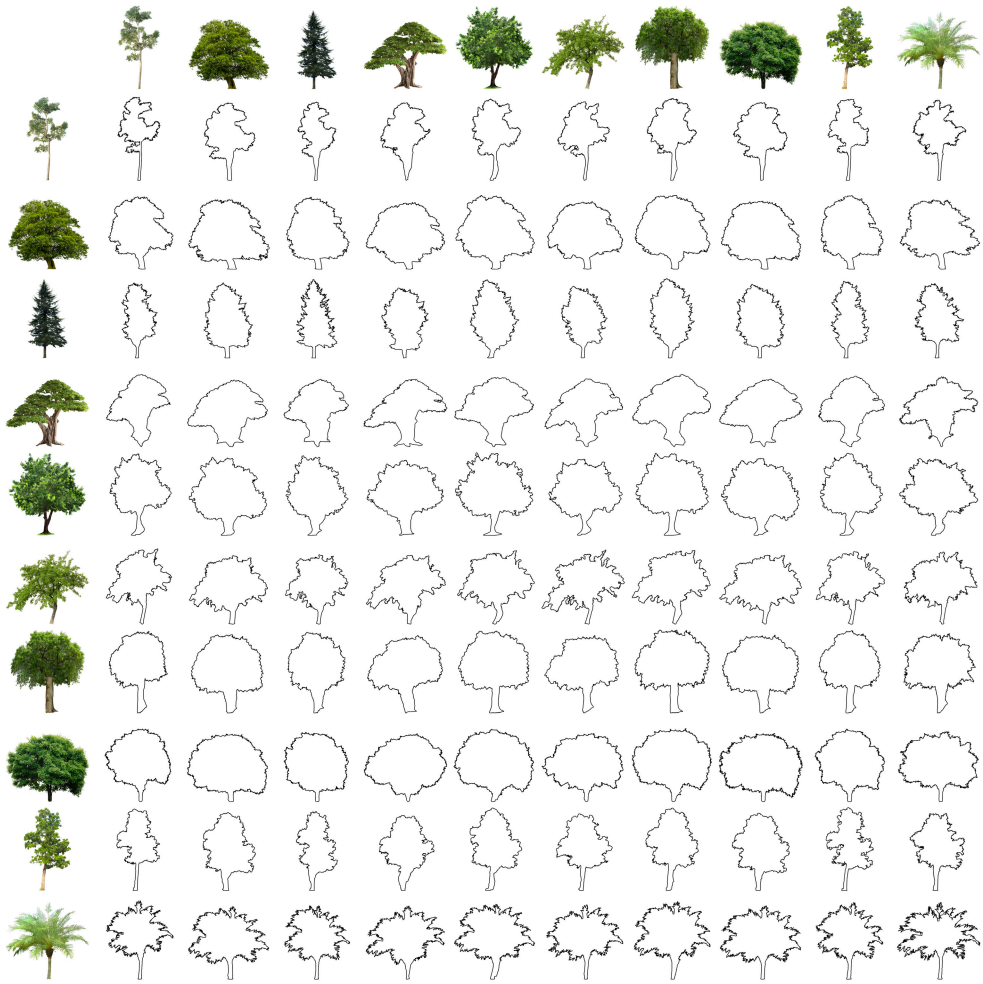


Figure 6.11: Contour combination table. For each cell, the contour has been obtained as a combination of the row and column exemplars contours, with corresponding weights 2/3 and 1/3, respectively.

Figure 6.13 illustrates the usage of tree the variations using small sets exemplars from the same species, by morphing between each pair with $w_1 = 0.25, 0.50, 0.75$. When the shapes of the exemplars are very similar between them, the resulting variations look quite similar between them. Still, all pictures are unique and can be used to populate a forest scene with less repetitions than using only the the ones in the exemplar set. While obtaining similar variations is reasonable within the same species, more variety could be introduced by using trees from other species to distort only the shape contour and preserving the color texture as in Figure 6.12.

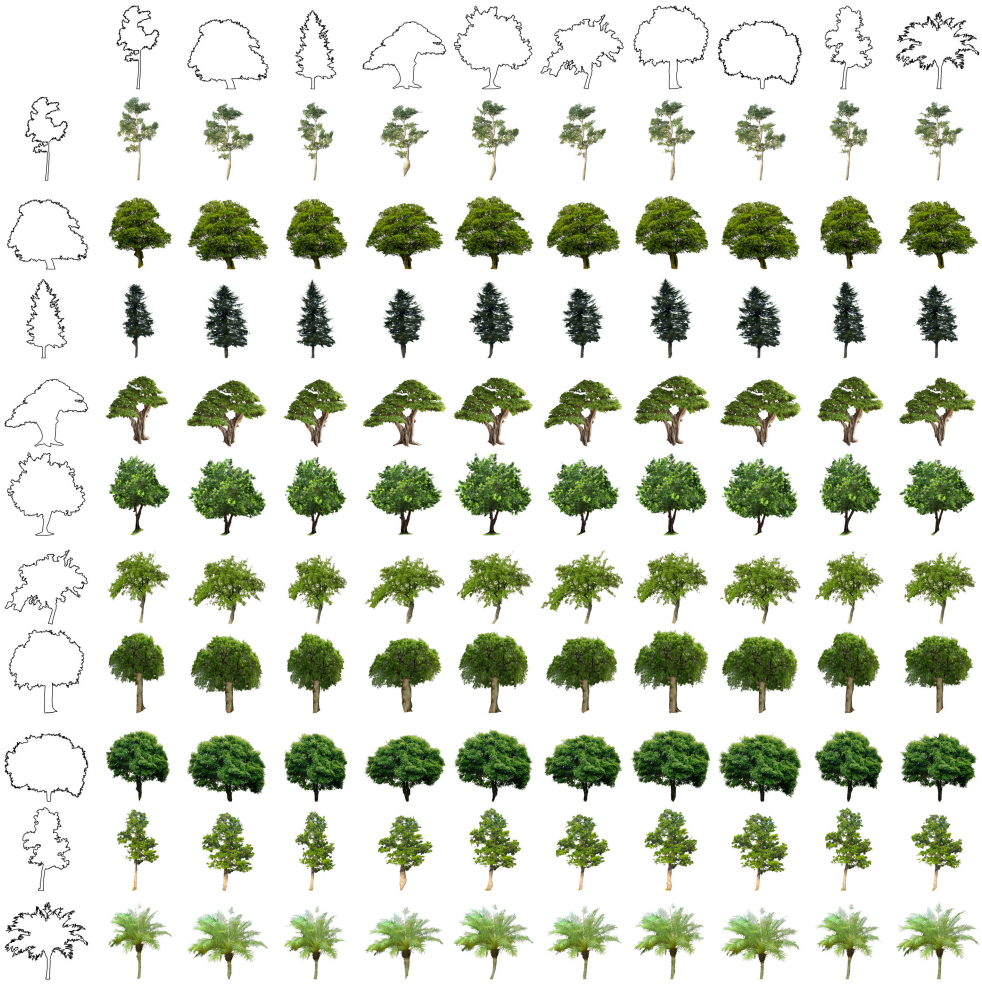


Figure 6.12: Tree deformations table. For each cell, the tree image from the row exemplar has been distorted to match the shape of the combined contours of Figure 6.11.

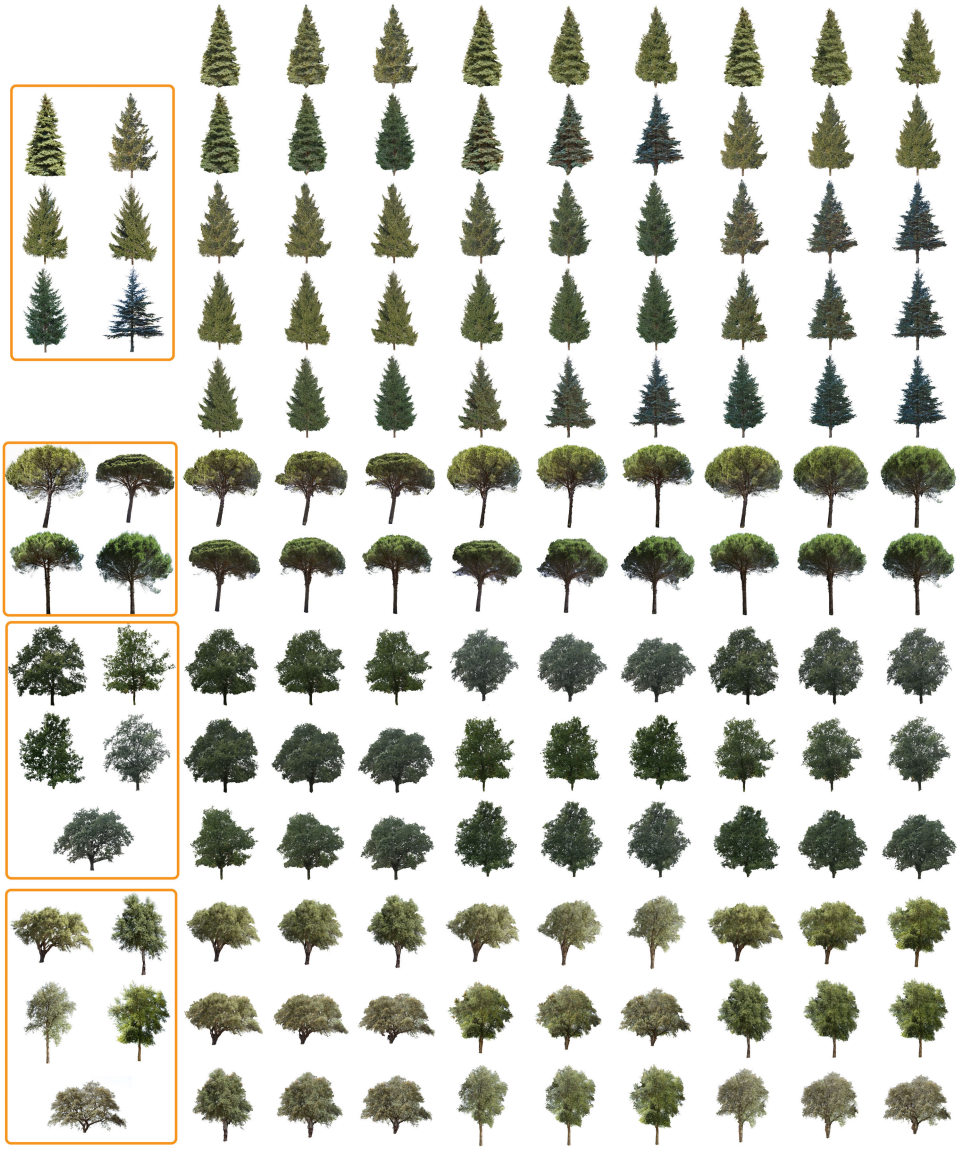


Figure 6.13: Tree variations within the same species: fir pines, stone pines, oaks, and cork oaks. The exemplars on each group have been highlighted.

Figure 6.14 shows more variations obtained by combining two randomly selected exemplars among the 55 trees dataset shown in Figure 6.4 plus 17 additional and not manually segmented pictures (some of them shown in Figure 6.6). The contour variations were computed using a convex combination of the contours with $w_1 \in [0.3, 0.7]$ and $w_2 = 1 - w_1$. For the color, it was randomly

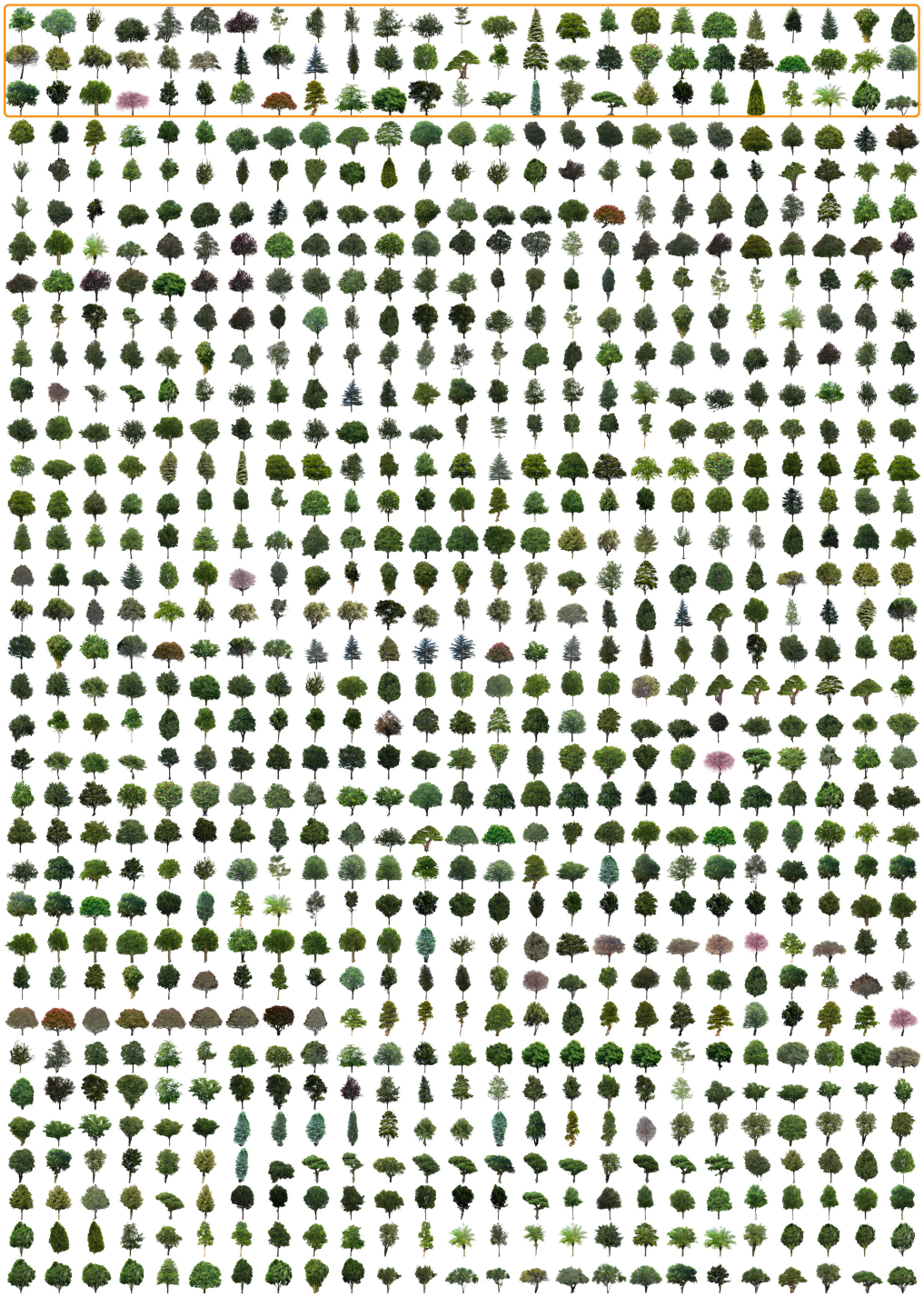


Figure 6.14: 800 randomly generated tree variations from 75 exemplars (top).

set as either the convex combination with the same weights as the contour, or one of the two warped images directly. Generating each of these new trees at 1024×1024 resolution takes between 8 and 12 seconds on an Intel Core i7 CPU at 3.40 GHz, mainly depending on the ratio of non-transparent pixels.

Although the synthetic tree images we create are not necessarily plausible from a botanical point-of-view (specially when combining exemplars from radically different tree species), these images are still suitable for mainstream applications such as video games, where indeed artists often look for fictional trees.

The color transfer approach works best when the two contours are not radically different. Otherwise, the source image needs to be severely distorted to fit the target contour, as shown in Figure 6.15.

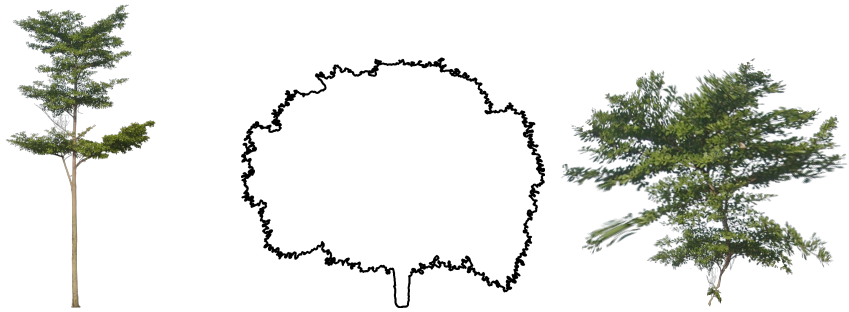


Figure 6.15: Failure example of the tree variations algorithm. The thin, sparse tree on the left has been used to fill the thick contour on the middle, obviously resulting in a large distortion.

6.3 Single-picture tree crown reconstruction

Directly rendering alpha matted pictures of trees as billboards is an inexpensive technique which can be used for large scenes with distant trees. However, their flat nature becomes evident as we approach them: they always look the same from any viewing direction, they lack volume and parallax produced by the branches, it is complex to make them cast and receive proper shadows, etc.

We will now present different approaches we have developed to generate a complete 3D model of the tree crown from a single picture. Currently, we have limited our scope to dense trees, i.e. those trees for which the foliage can be approximated by a single crown volume in which the branches are hardly visible.

However, for the case of trees consisting of various separated foliage nuclei, our proposed methods could be applied to each foliage region independently.

Moreover, since we will generate the complete tree from just a frontal photograph, an implicit assumption of our method is that the pictured foliage has to be representative of the whole tree, i.e. the tree should not exhibit strong, large-scale directional patterns that would completely change its appearance from another viewpoint (e.g. a palm tree). Small scale directional patterns (e.g. preferred leaf orientations due to phototropism and high-frequency features alike) have little impact on our synthesized textures and are thus well supported.

6.3.1 Base crown envelope

The first step of the 3D reconstruction consists in estimating the volume occupied by the foliage crown of the tree from the 2D contour of its silhouette. We will refer to this volume as the *crown envelope*.

Sketch-based systems [IMT99; Rep05] provide algorithms to *inflate* a mesh from a simple boundary curve. They either use a mapping function applied to a distance transform of the provided boundary or apply diffusion-based techniques. Results with typical tree silhouettes are rather poor (see side views in Figure 6.19) as elevation values on spine vertices directly depend on the distance to the silhouette (to make wide areas fat, and narrow areas thin).

The input crown's silhouette can be either obtained by automatic or manual segmentation of the crown image. Since we cannot make assumptions about the level of detail or quality of this segmentation, we first apply some Laplacian smoothing iterations to the contour. The main reason is that we will be inferring a smooth approximation of the crown volume, and it does not make sense to have a highly detailed silhouette but a smooth mesh elsewhere.

First, we will try to obtain an envelope mesh by generating a heightmap over the segmented image (see Figure 6.16). This heightmap would provide the frontal half of the envelope mesh, and its inverted version would provide the other half.

Due to the ill-defined nature of this problem, we fix the silhouette points of the heightmap on the $z = 0$ plane (z represents height) and we ask their tangents to be parallel to the z axis, while requiring that all points inside the crown reduce their bending energy. We can intuitively think of this process as extruding or inflating the segmented shape upwards.

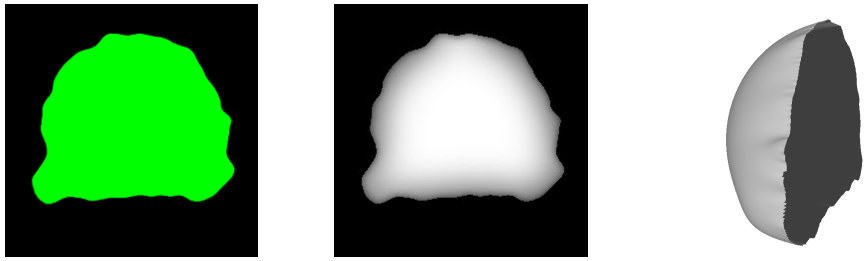


Figure 6.16: From the tree segmentation, we can compute a heightmap that represents the front half of the envelope mesh using the bilaplacian filter.

Formally, we achieve this by minimizing the thin-plate energy defined by a biharmonic equation inside the crown C, while restricting the silhouette S to have $z = 0$. Written as a continuous optimization problem, we propose to apply the biharmonic equation:

$$\nabla^4 z = 0 \tag{6.1}$$

with boundary conditions:

$$\begin{array}{ll} z(x,y) = 0 & \text{for } (x,y) \in \partial C \\ \frac{\partial z}{\partial n} = \nabla z \cdot \hat{n} = -f(x,y) & \text{for } (x,y) \in \partial C \end{array}$$

where z is a function from $\mathbb{R}^2$ to $\mathbb{R}$ that represents the heightmap, ∂C represents the boundary of C, $\hat{n}$ is the outwards-pointing unit normal of ∂C, and f is a positive real function controlling the gradient of the distance field at the silhouette points.

The optimization problem above can be written as a linear system for the z values using the bilaplacian filter [KCV+98]. We discretize the domain C using our input segmentation image as a 2D grid discretization, where a pixel subset S of C will represent ∂C. We will use this subset S to apply the Dirichlet boundary condition. We also define the outer silhouette S_o of the image as the set of pixels outside C but adjacent to a pixel in S. S_o will be useful to apply the Neumann boundary condition. In this setting, both boundary conditions are translated into equations:

$$\begin{array}{ll} z(i,j) = 0 & \text{if } (i,j) \in S \\ z(i,j) = -f(x,y) & \text{if } (i,j) \in S_o \end{array}$$

Then, for each unknown z value in the heightmap we add an equation obtained from the convolution of its corresponding point with its neighbors, repre-

senting the thin-plate energy computation:

$$\sum_{\substack{-2\leq d_i\leq 2\\ -2\leq d_j\leq 2}} M_{d_i+2,d_j+2}\cdot z(i+d_i, j+d_j) = 0 \tag{6.2}$$

$$M = \frac{1}{16}\begin{bmatrix} 0 & 0 & 1 & 0 & 0\\ 0 & 2 & -8 & 2 & 0\\ 1 & -8 & 20 & -8 & 1\\ 0 & 2 & -8 & 2 & 0\\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

This convolution is the result of the application of two successive laplacian filters, a well known edge detector image filter, as the bilaplacian itself is the result of computing the laplacian of a laplacian.

Solving the system yields promising results, but there are still some issues (as shown in Figure 6.17-left). As we have been restricting the problem on a heightmap, and due to the way we express the tangent constraints using the outer silhouette on this grid, tangents cannot be defined as vectors pointing in the z direction because there will always be a small displacement in x and y between S_o and S. As a consequence, the resulting envelope will never look as smooth as we want. Using larger values of f for the pixels in S_o alleviates this problem, but tangents will never match with the mirrored ones at the silhouette because they will always have non-null x and y components. What we really need is to define tangents at the same x, y position where we specify $z = 0$.

One possible way to solve the problem would be to compute a volumetric distance field instead of a heightmap, generalizing Equation 6.1 to 3D and extracting an isosurface from the resulting volumetric distance field. This would be straightforward but the resulting system would grow considerably, making it less efficient.

Instead, we found that it is easier to just free vertices in C and S_o from their grid positions. Inner vertices ($\in C\backslash S$) will no longer be restricted to a grid arrangement, so we extend the previous linear system with unknowns for the x and y components as well as the z as before. In this new setup, grid vertices have coordinates:

$$\begin{array}{ll} (x_x, x_y, x_z) & \text{if } (i,j)\in C\backslash S\\ (i,j,0) & \text{if } (i,j)\in S\\ (i',j',f(i',j')) & \text{if } (i,j)\in S_o \end{array} \tag{6.3}$$

where x_x, x_y and x_z are the unknowns, and (i', j') is just the projection of $(i, j) \in S_o$ onto the closest silhouette pixel in S. We use Equation 6.2 independently on each unknown to solve the problem. Notice that this is still a discretization of

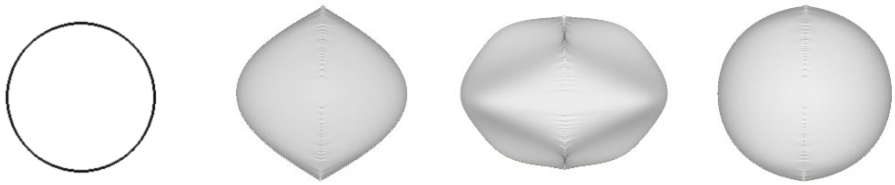


Figure 6.17: Inflating the envelope of a circle (left) while keeping the x and y components fixed does not produce the desired results (center left). However, expressing the minimization problem in 3D helps solve this problem (right) but only if a 8-connected silhouette is used. A 4-connected silhouette produces unacceptable artifacts (center right).

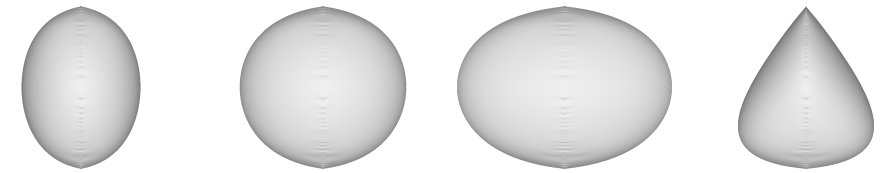


Figure 6.18: Side views of a circle inflated using different contour functions.

the continuous problem proposed in Equation 6.1, because the resulting mesh is still going to be a function of (x, y). We can think of this new version as creating a heightfield mesh initialized using the grid positions, and then allowing its vertices to move while staying close to their neighbors.

Lastly, it is important to mention that pixel connectivity plays a big role in the mesh generation process using these Equations 6.3. The set of pixels in S contains pixels classified as inside the crown but adjacent to pixels outside. If we assume 4-connectivity for the silhouette pixels S, results are not sufficiently smooth (e.g. we cannot reconstruct a sphere from a circle boundary, as shown in Figure 6.17). Using 8-connectivity, however, solves these artifacts.

As for function $f(x, y)$, which could be arbitrary, we provided artists with two choices: either a constant function $f(x, y) = k$ for roughly spherical crowns, or a quadratic function with the form $f(x, y) = k(1 - y)^2$ for approximately conical trees. In both cases we assume Y is the vertical axis pointing upwards, and (x, y) are normalized pixel coordinates in $[0, 1]$ range. Note that both functions are intuitively controlled through a single parameter (i.e. a slider): larger k values

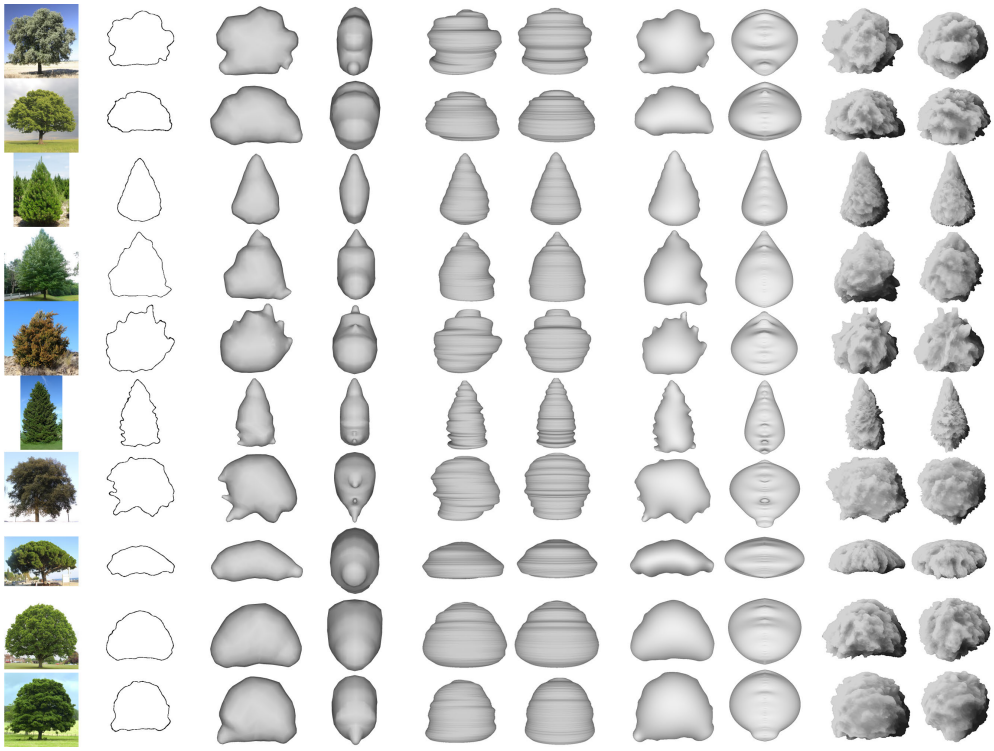


Figure 6.19: From left to right: input photograph, segmented silhouette, front and side views from a sketch-based system [IMT99], front and side views using a function-based crown model from [ACV+14], front and side views of our envelope crown, and front and side views of our detailed crowns.

result in more extruded crowns. Figure 6.18 shows some example functions.

Figure 6.19 shows some tree silhouettes and the resulting envelope. We implemented the solver for Equations 6.3 using the LDL decomposition in Eigen [25]. The generation times on an Intel i7 3.40 GHz CPU for crown shapes discretized as 256×256 pixel grids range between 1 and 2.5 seconds, depending on the interior and silhouette pixels (i.e., the number of equations and constraints).

6.3.2 Crown envelope with relief perturbation

The envelope reconstruction we have extracted using the bilaplacian equations has a smooth appearance. In fact, the silhouette is the only detailed part of the tree. We want to add more details to the crown shape as perceived in the photograph. However, a single frontal picture of a tree contains less than a half

of the tree's outer surface. We assume that the occluded parts will have a similar appearance as those which are visible, and thus use a texture synthesis approach to infer the missing parts.

First, the segmented photograph is scaled to have unit aspect ratio, and placed at the center of a 4:3 canvas such that it occupies 5/12 of its width and 5/9 of its height. We do this under the assumption that the original photograph must cover a face of a cubemap plus about a third of the neighboring faces. We then fill the rest of the canvas using the texture synthesis algorithm from [Har01]. The color cubemap is constructed by cropping the six faces of this synthetic texture, as shown in Figure 6.20, and applying a local synthesis again on the bands around the borders of the faces that were not originally in contact. This additional step guarantees continuity across all cubemap edges. The texture synthesis step for a 1024×768 image takes between 1 and 2 minutes on an Intel i7 3.40GHz CPU using the Resynthesizer plugin for GIMP. This step could be further optimized: for example, Photoshop CS6 content-aware fill executes in only 5-10 seconds.

This synthesized cubemap can be used to texture all the surface of the crown envelope. Recall that our goal was to add details onto this smoothed envelope. Our relief estimation method is inspired by shape-from-shading (SFS) algorithms like [GWM+08]. Unfortunately, we can assume very little about the lighting conditions of the input photographs, and typical assumptions in SFS approaches (known reflectance model, known direction of light sources) do not hold in our case. Furthermore, tree leaves form complex structures with high-frequency variation across the image – a worst-case scenario for SFS approaches, which describe the surface shape either in terms of the surface normal, or in terms of the heightfield gradient.

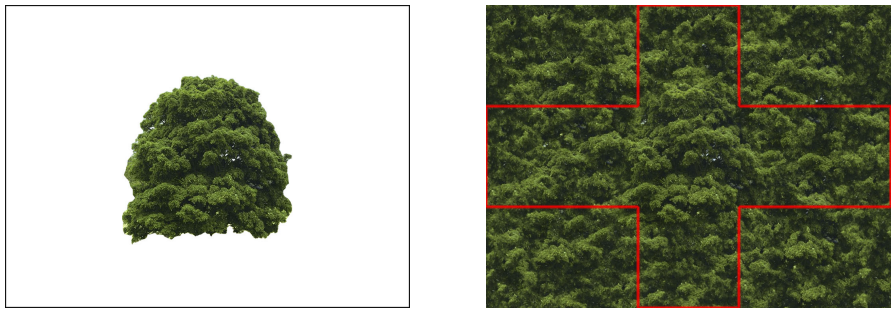


Figure 6.20: Left: placement of the segmented photograph inside the synthesized cubemap. Right: synthesis result and cubemap outline. Notice that the synthesized result includes the segmented photograph.

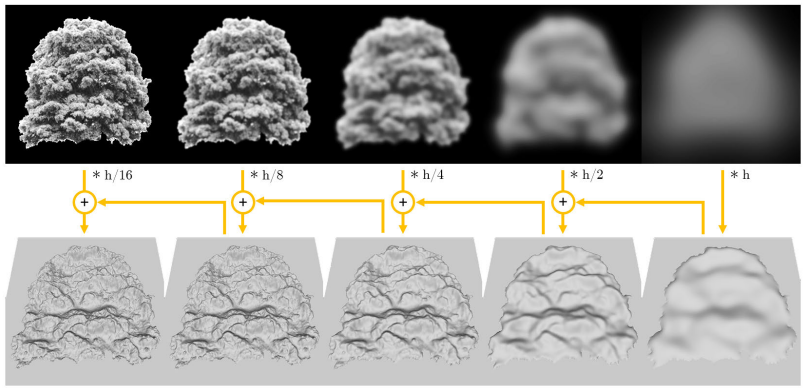


Figure 6.21: Gaussian decomposition pyramid (top) and multiscale relief generation example (bottom). Each heightfield is the result of adding its corresponding Gaussian pyramid level to the previous one, and decreasing the weight at each step.

Since we assume the envelope mesh already captures the rough and low frequency shape of the crown, we directly estimate depth from luminance. Despite the fact that a tree has a huge number of self-occlusions from the different branches and leaves, a fraction of incoming light is likely to reach occluded portions due to light scattering and non-opaque leaves. The deeper we are inside the crown, the darker we expect it to be. Again, we assume a dense foliage crown, preventing deep areas close to the crown center from receiving direct light.

We start by computing the luminance of the input image and normalizing it to be in the $[0, 1]$ range. Then, we compute a blur pyramid of this luminance image, similar to the *Gaussian pyramid* of Burt and Adelson [BA83]. We also tried a Laplacian decomposition with very similar results; the relief obtained from the Gaussian is more prominent due to lower frequencies being counted multiple times. Once we have N levels of the pyramid (we tested $N = 4$ and $N = 5$), we combine them using an exponentially-decreasing set of weights. Lower levels (blurrier) will have larger weights because they represent large-scale (and potentially deeper) structures, while higher levels are usually noisier and just add small fine details to the relief.

Figure 6.21 shows an example of relief perturbation map, computed on the plane of the input image for clarity. In practice, we compute a relief perturbation cubemap from the color texture cubemap we synthesized before, and use it to extrude or sink the surface of the crown envelope (Figure 6.22). The last two columns in Figure 6.19 show examples of detailed tree crowns obtained with the cubemap approach.

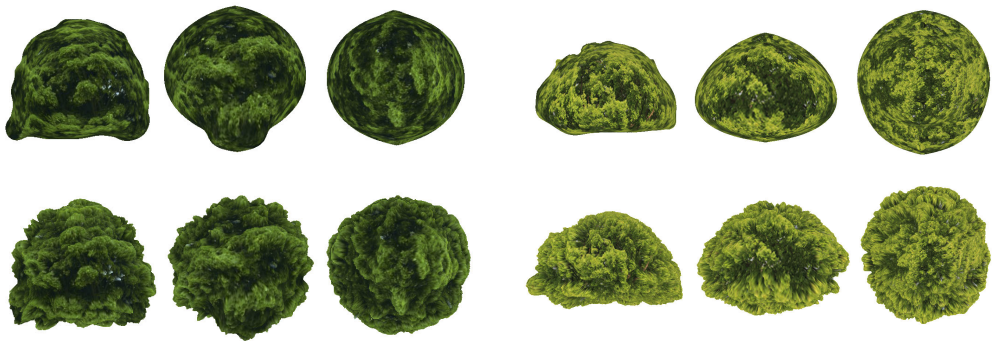


Figure 6.22: Front, side and top views of the textured envelope mesh (top) and the resulting detailed crown with relief perturbation (bottom).

Due to the assumptions we made during the crown envelope stage, our method does not generate convincing shapes for certain types of trees (like some species of firs or palm trees), which have large-scale branch structures on the photograph silhouette but that should not lie on the same plane as they are facing towards or away from the camera. In order to improve the overall shape of these trees, we allow the user to combine some rotated copies of the crown envelope (as an example see Figure 6.23). In order to do so, we compute the union of the rotated copies of the crown envelope, and then add the relief perturbation using the cubemap. Note that, although each rotated envelope copy is identical, the added relief will be distinct at different points on the union surface.

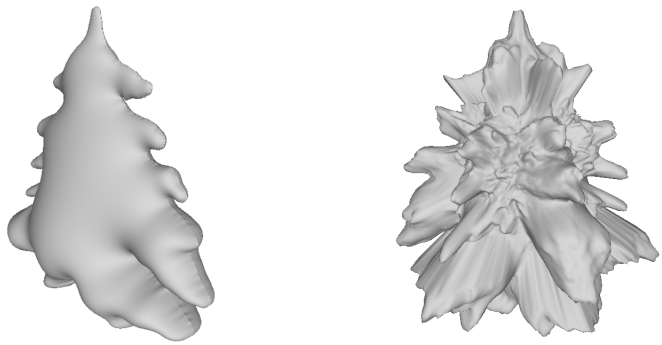


Figure 6.23: Base envelope mesh computed from a segmentation (left). Tree crown created from three rotated copies of the envelope and relief perturbation (right).

6.3.3 Billboard clouds from a photograph

The crown envelope with relief perturbation and color texture like in Figure 6.22 bottom row, provides a convincing representation for medium and far distance trees. However, on closer views, there is lack of more detailed texture and parallax effects due to branches and leaves occluding each others as we move around the tree, an effect difficult to reproduce on a single surface as we have used so far. Billboard clouds are a common representation for volumetric tree foliage: each small branch or group of leaves is represented as a textured quad or a very simple planar mesh, and given enough billboards the tree can look very realistic. We will now introduce an approach for generating a billboard cloud from the segmented input photograph of the tree crown.

The first step is to obtain a set of billboard textures from the photograph. A naive approach could be just random sampling of regions inside the crown area, ideally with a fractal contour. This strategy, however, would not be taking into account the existing shapes and groups that as humans we can easily perceive on the picture. There are some methods that partition an image into *superpixels*, groups of connected pixels representing perceptually meaningful regions of the image. In particular, we used SLIC [ASS+12]. This method, based on k-means clustering, has two parameters: the approximate number N of regions to compute in the image, and the compactness c, which balances between spatial proximity and color similarity. Figure 6.24 compares different values of N and c. We empirically found that $N = 256$ and $c = 0.35$ yields good results for the several 1024×1024 pictures in our library, and thus adopted these values as constants in our pipeline.

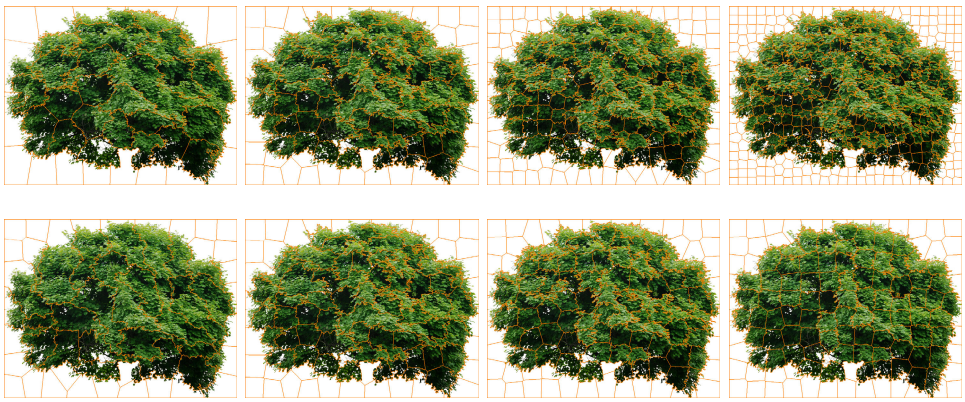


Figure 6.24: Comparison of different SLIC parameters. Top row: $N = 64$, 128, 256, 512 and $c = 0.3$. Bottom row: $c = 0.1$, 0.3, 0.5, 0.9 and $N = 128$.

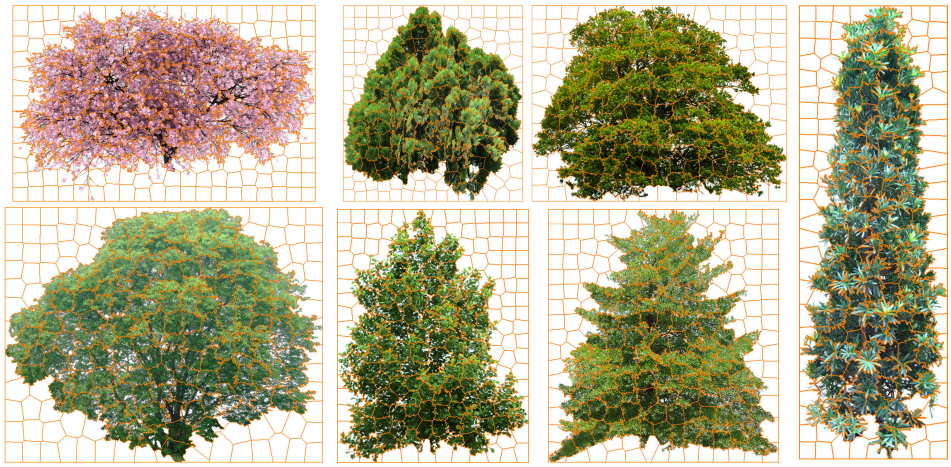


Figure 6.25: Segmentation into superpixels for different trees, using SLIC with $N = 256$ and compactness 0.35.

Figure 6.25 shows the segmentations obtained on different tree pictures using $N = 256$ and $c = 0.35$. As we can see, there are regions with long rectilinear edges on the boundary. This is undesirable; when we render the billboard cloud, sharp straight edges will be easily perceived by the viewer and will reduce the plausibility of the tree model, as vegetation has a very fractal appearance. Reducing the compactness would yield more fractal edges, but as we show in Figure 6.24 the regions can largely differ in size, which we do not want either. We tried looking for a graph cut in the region around the straight edges to obtain a possible better segmentation, but we realized that these rectilinear edges tend to appear on dark or homogeneous areas, and thus graph cuts do not offer much improvement over the SLIC segmentation. Another simple idea, just adding fractal noise to perturb the straight edges, results in segments that look quite different to the rest of the contour.

Then, we observed that we already had the information about how the contour of a region, i.e. a group of branches and leaves, should look like: like the external contour of the tree crown. Therefore, we substitute each straight edge on a region contour by some curve taken from the silhouette of the tree crown following these two contraints: the relative positions of the endpoints of the silhouette segment must coincide with the endpoints of the straight line (up to a small tolerance), and the segment must not cross the straight line. This second constraint is optional, but we added in order to ensure that each pixel from the original photograph is represented at least in one region. This way, the frontal view of the billboard cloud will look almost identical to the input photo-

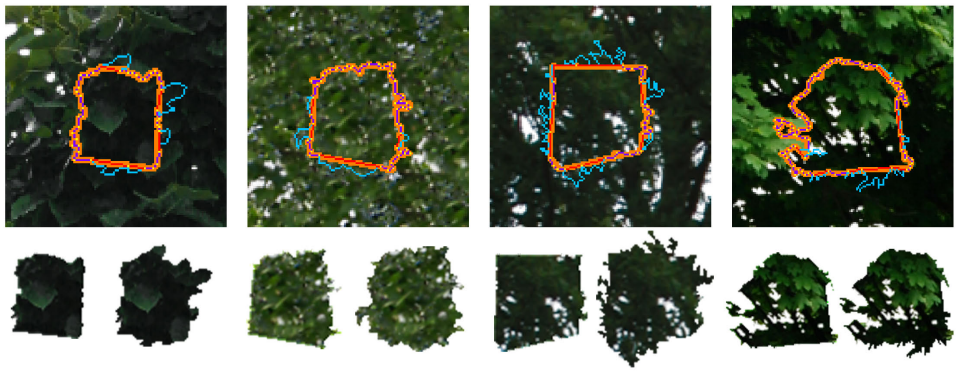


Figure 6.26: Improving region segmentation by substituting straight edges with segments taken from the silhouette of the tree crown. Orange: pixelwise contour drawn twice as wide for visualization. Purple: polygonal contour obtained as a simplification of the pixelwise contour [TC89], with contour points in yellow. Red: segments for which the distance between endpoints is greater than a threshold (8 pixels). Blue: substitution contour extracted from the silhouette. The bottom row compares the cropped billboards without and with edge replacement.

graph. Figure 6.26 shows some improved contours using this silhouette segments replacement strategy and provides more details about how we obtain it.

The next step is placing these billboards to populate the tree crown. Since we want the front view to be as close as possible to the original photograph, we will simply respect the relative positions of the billboards, and use the height of the crown envelope at their center position to assign an initial height for the first (frontal) layer of billboards. We improve this positioning by deepening the height of those billboards with mean luminance smaller than the mean luminance of the crown. The reason is that we do not want dark billboards appearing on the silhouette when rendering the tree from lateral views, because we interpret darker areas as more occluded and the billboards on the silhouette do not tend to have occluders, thus producing unnatural views of the tree. By forcing dark billboards to be deeper in the crown, we are also creating some big cavities in the otherwise smoother surface of the envelope, and the next layers of billboards will be visible through them providing convincing parallax effects when rotating around the crown.

The rear layer of the crown is obtained similarly, but now we horizontally flip the relative positions of the billboards. Note that we are not flipping the texture of the billboards, just their positions. If we did, we would see the rear side as a flipped version of the front side, seeming unrealistic when inspecting the tree and

looking symmetric from a lateral view. Finally, we randomly select billboards and place them at random positions following a blue noise-like distribution inside the crown volume approximated by the frontal and rear layers. We also lower the luminance of these billboards according to their depth inside the crown. Figure 6.27 shows some results, although a video or interactive inspection is needed in order to perceive the parallax effects that our representation produces.

About the generation times, the SLIC implementation provided in the Python package `scikit-image` with $N = 256$ and $c = 0.35$, plus the improvement of the region contours, takes from 20 to 50 seconds depending on the tree. Billboard placement is executed in less than a second for models with about 500 billboards.

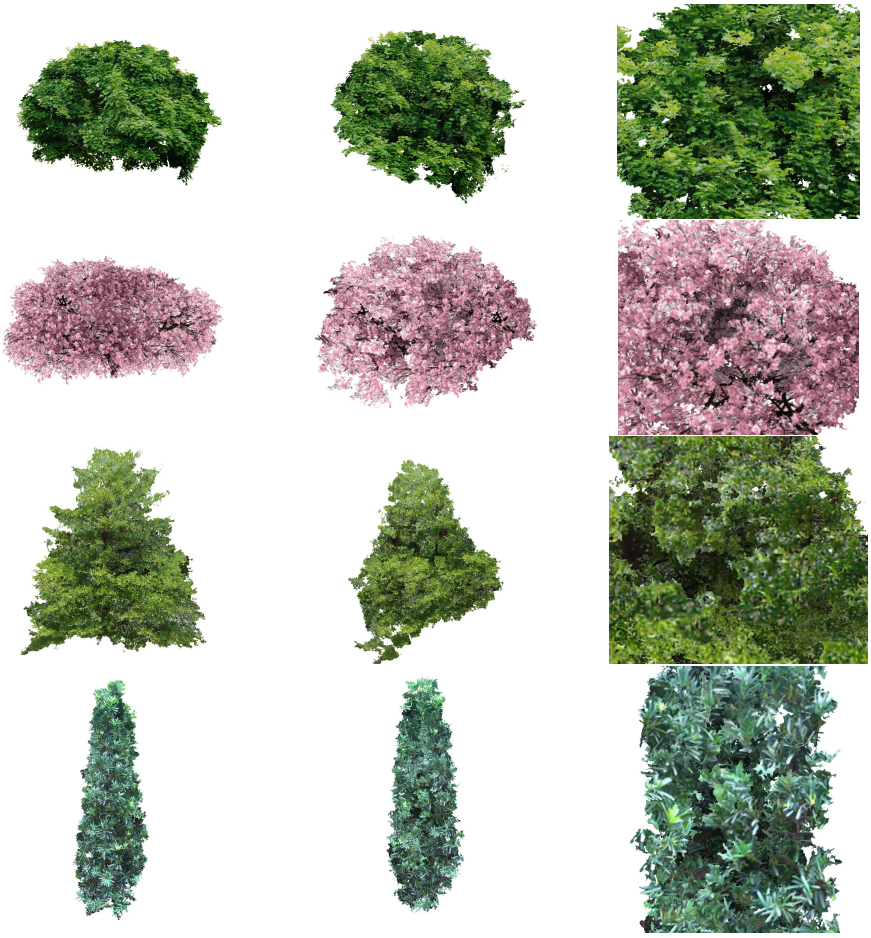


Figure 6.27: Examples of automatically generated billboard clouds. Note how the frontal view (first column) looks almost exactly as the original photograph (see Figure 6.25), and how the deeper billboards also appear darker.

6.3.4 Branches, leaves and detailed tree models

The billboard cloud representation provides a good representation of the photograph that also allows for volumetric and parallax effects, but the level of detail is still insufficient for closer views.

In order to support highly detailed models, a branching structure for the crown can be generated with resource competition algorithms, for example the Space Colonization algorithm by Runions et al. [RLP07]. Given the volume occupied by a crown, for example the space inside our crown envelope representations (we will use the envelope with relief perturbation), we generate a set S of N attraction points using a blue noise distribution in this volume. Then, starting from a fixed root node – we simply define a downwards displacement from the crown center of mass – the Space Colonization algorithm generates a branching structure iteratively by extruding new edges of average length D from current nodes towards their closest attraction points. The level of detail of the generated branching structures can be controlled with the number N of the attraction points placed around the volume, and the *kill distance* d_k that determines the minimum distance between attraction points and the nodes in the branching structure in order to be kept in S.

Fully-detailed tree models can be generated by increasing the refinement of the branching structure. Figure 6.28 shows a couple of tree models generated using this approach in about 90 seconds (Python code running on Blender).

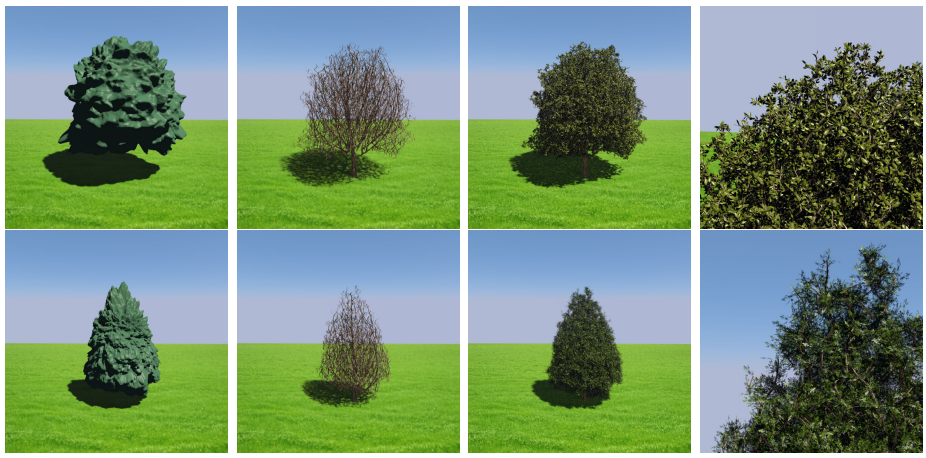


Figure 6.28: Fully-detailed models created by generating attraction points within the crown envelope and then using a space colonization algorithm. The images show the crown envelope, the branching system, and the final tree (full/detail).

Notice that the final shape of the tree models closely matches that of the crown envelopes. In these examples, we used $N = 10,000$ attraction points, $d_k = 0.01r$ (r being the average radius of the crown envelope) to ensure that the branching system provides a dense coverage of the crown volume, and an *internode length* $D = 0.5k_d$. The resulting models have around 300 K textured quads (leaves) plus about 100 K quads (branching system), so their use should be reserved to very close-up views.

In order to keep real time framerates during rendering, we will use much simpler branching structures, like the ones shown in Figure 6.30, and larger billboards representing groups of leaves and small branches. These billboards are created by instantiating multiple copies of a leaflet, a photograph of leaves or small branches – it could even be a 3D mesh if available. For a fully automatic pipeline, we would need to specify or derive some correspondence between the species from the photographs and the type of leaves associated with them. In our work, we have manually provided a picture as input, which also has the advantage of being able to generate additional model variations from the same crown shape just by changing the associated kind of leaves.

The distribution and arrangement of leaflets inside each billboard is also based on the Space Colonization algorithm with an arbitrary input volume. We found that using simple shapes like spheres or cones yields convincing results (see Figure 6.29). Conical volumes yield billboards representing branchlets that will be connected from the base of the cone to the branching structure. Spherical clouds, although less realistic when seen in isolation, can be rotated arbitrarily around the connecting point - their center - and thus can be oriented at runtime to face the viewer at any time, achieving more variety of leaves on the tree with fewer billboards.

Figure 6.29: Input leaflet (left) and automatically generated billboards using Space Colonization in a conical (center) and a spherical (right) point clouds.

We can then reconstruct with the Space Colonization algorithm much simpler tree skeletons than the ones we depicted in Figure 6.28, and attach the branchlet billboards to the nodes of the branching structure, as shown in Figure 6.30. For $N = 1,000$, $d_k = 0.15r$ and $D = (0.5 \pm 0.25)d_k$, our C++ implementation executed in 50 - 200 ms.

Figure 6.30: Tree skeletons generated from the crown envelope and rendered branches with billboards attached. We painted the branchlets with transparency and without depth test to represent their density and show the branches.

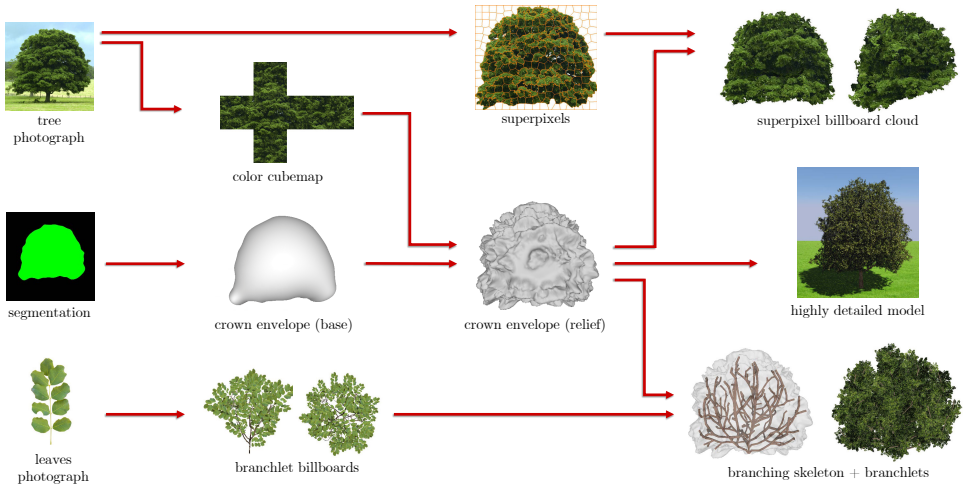


Figure 6.31: Summary of the different reconstruction steps and resulting models.

6.4 Real-time rendering of forest scenes

In the previous section, we have introduced a handful of different tree models, all of them derived from a single picture of the tree. Recall that our goal is to render mountain and forest landscapes with models of trees inspired on local species, and these models should integrate it into real-time applications like virtual globe navigation or video games.

Next, we describe how to combine these different representations into a level of detail (LOD) strategy to allow for real-time rendering of tens of thousands of trees. But first, we still need to introduce an additional data structure: the Radial Distance Map (RDM). Then, we will see how we use it to smoothly transition between representations. Finally, we will analyze the performance and costs of our proposed models.

6.4.1 Radial Distance Map

One of the major drawbacks of LOD techniques is the popping effect that occurs when transitioning between different rendering techniques or model representations. This effect is yet more noticeable when the shape of the object changes abruptly. Therefore, in order to provide better and smooth transitions, we want to ensure that a particular tree always has the same shape; i.e. we need to represent the shape of the tree crown and be able to tell if a point is inside or outside it.

So far, we already have a representation of the boundary between the interior and exterior space of the tree crown: the crown envelope mesh (from now on, we refer to the detailed version with relief perturbation). However, testing whether a point is inside an arbitrary mesh is an operation too expensive for real-time rendering of thousands of tree instances. We could approximate the crown volume using a voxel grid, i.e. a 3D texture, and the interior test would reduce to a simple texture look-up, but the memory cost in this case would be prohibitive for many different models.

We propose to represent the crown surface using a radial function $r = f(C, \varphi)$, where C is the crown center, φ a 3D direction, and r is the (maximum) radial distance from C to the surface of the crown. The drawback of a radial representation is that it only allows for star-shaped surfaces, but in practice we can still approximate a large variety of crown shapes, as shown in Figure 6.32. In fact, the crown envelopes with relief perturbation shown in Figures 6.19 (last two columns), 6.22 (bottom row) and 6.23 (right), for example, were already showing

the RDM representation. On the other hand, the two major advantages of this representation are:

- The interior test of a point p can be performed with a simple comparison: p is interior if $||p - C|| < f(C, p - C)$.
- We can efficiently store and query f from the GPU using a cubemap, which we will name *Radial Distance Map*.

Moreover, since we know for any point inside the volume the distance to the center and to the surface, we can profit on this information when designing LOD strategies as well as shading algorithms. For example, we use the ratio $||p - C||/f(C, p - C)$ to darken billboards inside the crown.

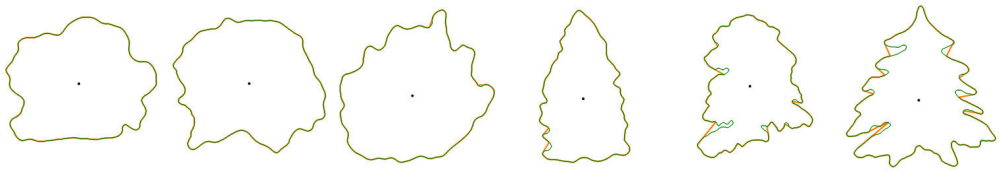


Figure 6.32: Crown contours (green) and star-shaped contour (orange) defined by their computed radial function from the center of mass (black dot).

In the next sections, we will see how we use the radial distance map to generate different Level of Detail representations and smoothly transition between them.

6.4.2 Direct rendering of the RDM

Our first rendering method implements a direct visualization of the RDM through fragment-based relief mapping. Relief mapping approaches [POC05] cannot compete with current tessellation engines when close-up views of detailed geometry are required, but offer excellent performance for distant objects (especially in deferred shading setups) since their rendering cost is output-sensitive, i.e. the cost directly depends on the number of covered pixels [BPA15]. This is thus the LOD we use for rendering distant trees.

The classic approach for relief mapping [POC05] is to compute the ray-heightfield intersection by sampling points along the ray, first using linear search to find a sample inside the object, and then refining the intersection point through binary search. For each sampled point, the height of the sample is compared with

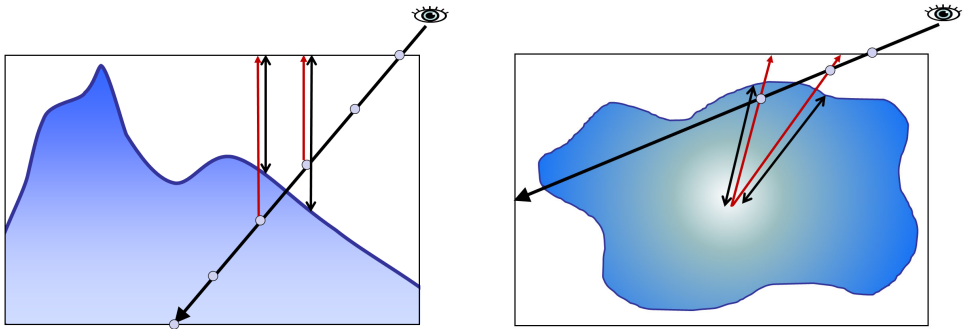


Figure 6.33: Computing ray intersections for a height field (*left*) and a radial distance map (*right*). The red arrows indicate the direction of projection; black arrows show the distance stored in the texture, which is checked against the real distance at the ray sample.

the depth stored in the relief map to determine whether the sample is inside or outside the heightfield (Figure 6.33 left).

We apply this idea to radial distance maps (Figure 6.33 right). Each individual tree is rendered as an instance of a simple geometry; we used an icosahedron as it better fits the shape of most tree crowns than a bounding box. This proxy geometry will initiate the ray traversal on the fragment shader. When traversing the ray along the viewing direction, ray samples are computed in world space (instead of tangent space). For a ray sample P, we compute $\vec{v} = (P - C)/R_{\max}$ (C is the crown center and $R_{\max}$ is the maximum crown radius), and check if P is inside the crown by comparing $||\vec{v}||$ with the radial distance value stored for direction $\vec{v}$. The number of linear steps, their length, and the number of binary steps depend on the screen-projected radius of the bounding sphere. Additionally, since we know the minimum and maximum value of the radial distance map, $R_{\min}$ and $R_{\max}$, if the perpendicular distance d between C and the ray is greater than $R_{\max}$, we can early discard this fragment without raycasting. Similarly, if d is smaller than $R_{\min}$, we know that this ray will hit the crown. In this case, we compute the first hit with a sphere of radius $R_{\min}$ and skip the linear search.

We considered two options for computing the fragment color. The simplest option is to get the color directly from the color cube map. However, crown parts along the same radial direction get the same color from the cube map, resulting in some texture stretching that might be noticeable in trees with prominent branches and abrupt relief changes. Alternatively, we can get the fragment color from the central part of the synthetic 2D texture (Figure 6.20), using the same texture coordinates we would use for a planar billboard. This second option

provides higher quality images, but the re-usage of the same texture portion for all view directions becomes apparent when rotating the camera around a close-up tree. We thus use this second option only for distant trees. Notice that with both options the rendered tree silhouette matches that of the RDM.

6.4.3 Clipped billboard clouds

For closer views we add high-frequency details by drawing a billboard cloud with an RGBA texture showing leaves and small branches. Billboard centers are placed on points of the branching structure generated using the Space Colonization algorithm (see Section 6.3.4 and Figure 6.30). A single vertex buffer is shared among all tree instances of the same species. The actual number of points to render (converted into quads in the geometry shader) varies for each instance and depending on the distance to the viewer.

In order to preserve the overall crown shape of the species, we clip the geometry outside the crown volume. Again, a point-inside-crown test requires a single texture lookup to the RDM. This clipping operation can be performed at different granularity levels. Discarding complete billboards based on their center (e.g. in the geometry shader) is the simplest option, but tends to produce popping artifacts around the tree silhouette when large textures are used. Discarding individual fragments faithfully preserves the crown shape (except for transparent texels in the leaf texture), but tends to cut individual leaves in the texture.

A better option is to discard fragments based on the centroid of the individual leaves in the texture (Figure 6.34). This requires additional segmentation information where individual leaves point towards the cell centroid. We automatically generate this information when we create the billboards of the branchlets. This clip technique yields better silhouettes at the only expense of an extra texture lookup, so we use it when trees approach the camera.

Apart from the RGBA color and leaf centroid, we also store other information in the billboards (see Figure 6.35). Normals and precomputed ambient occlusion are used during shading. Depth is used to reduce the artifacts when two billboards cross: by modifying the depth of the fragments we avoid the intersection of two billboards to render as a line, making it less visible and reducing popping effects. The type bit (shown in black and gray for clarity) separates the leaves from the branches, and we use this information during specular lighting to avoid highlights on the branches. Finally, the texture coordinates refer to the coordinates used to instantiate the original photograph (see Figure 6.29). For very close trees, when the billboard texels become greater than a pixel, at the

cost of an additional texture lookup we can use these coordinates to recover the details from the original leaves photograph, which has higher resolution than the billboard.

Figure 6.34: Rendering clipped billboard clouds: (a) leaf texture; (b) segmentation showing individual leaf centroids, (c) per-fragment clipping, (d) per-leaf clipping, based on the inside/outside classification of the leaf centroid.

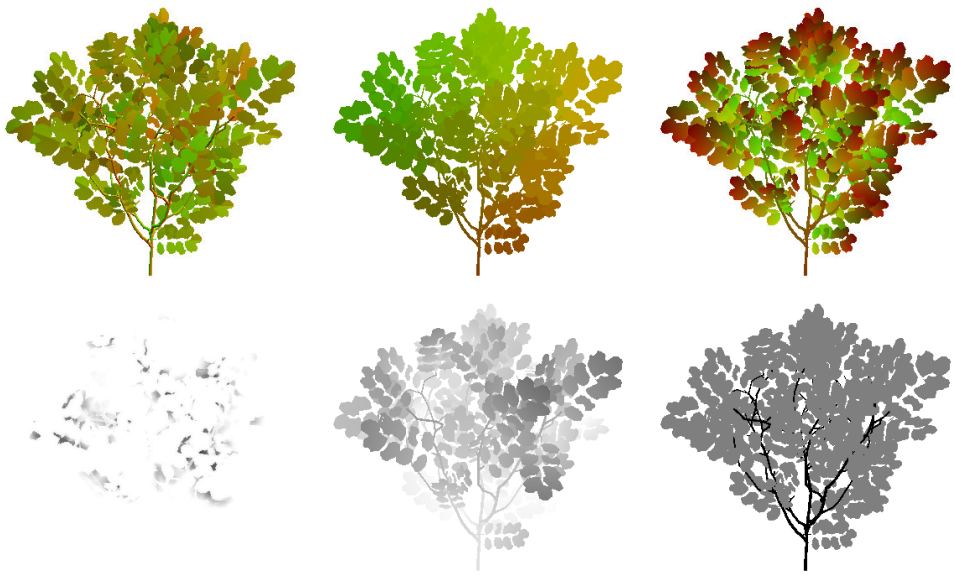


Figure 6.35: Information stored for the generated leaves billboards besides the RGBA components. Left to right and top to bottom: normals, leaf centroids, texture coordinates, ambient occlusion, depth, type bit.

6.4.4 LOD transition

The cost of drawing a tree using either representation can be adjusted according to its distance, thus providing a mechanism for LOD within each representation. When using the direct rendering of the RDM through relief mapping, the step size used during the linear search is inversely proportional to the screen-space projection of the tree. For the rendering through clipped billboards clouds, we reduce the number of billboards with the distance, and increase the size of the remaining ones, as in [CHP+07]. To minimize popping billboards, this pruning starts at some distance threshold (we used 40-60 m). When the distance is close to another threshold (160 m), we only use about 16-20 billboards. During the initialization stage, we sort the billboard anchor positions such that the i-th anchor is the one that maximizes the distance to all the previous anchors. This way, even with only a few billboards, we can still cover the tree surface.

Our proposed representations also allow for a smooth transition between them without need for cross-fading, i.e. rendering both representations at the same time with complementary alpha values. Billboard clouds far from the camera use only a few billboards, but they have been enlarged enough such that per-pixel clipping preserves the same crown silhouette that will be drawn with the RDM. Matching the texture and shading of both representations can also be done if we ray-march from the billboard fragment position to the camera, checking the RDM at each step. However, doing so for each fragment generated by the billboard clouds would be prohibitively expensive. Instead, we use the radial distance r in the direction of the crown center to the fragment position – which we have already fetched before to check whether to keep or discard the fragment – and compute an approximate surface position as the intersection of the view ray with a spherical crown of radius r. As trees get closer to the camera, we gradually blend from this shading that approximates the one of the RDM, to the shading based on the leaves billboards. Figure 6.36 shows different representations for the same tree.

Figure 6.36: LOD progression: close view, 256 billboard cloud, 16 billboard cloud with enlarged billboards and color from the cubemap, direct render of the RDM with the color cubemap, and direct render of the RDM with projected photo.

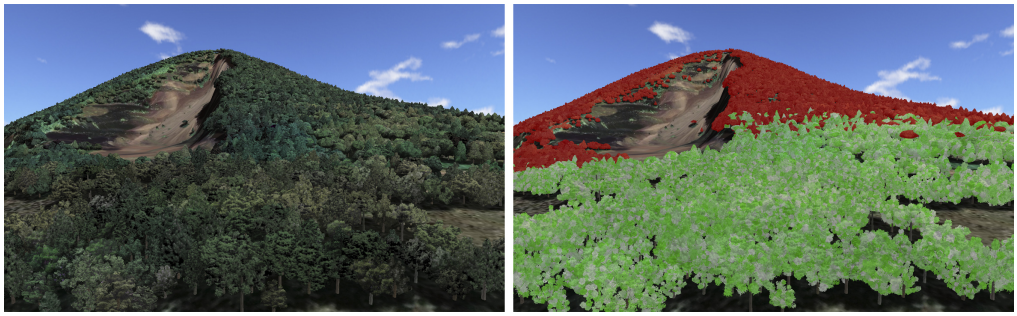


Figure 6.37: Scene (left) and LOD visualization (right). Trees rendered using the RDM appear in red, individual billboards are shown in different tones of green. LOD switch distance was set to 160 ± 40 m.

Illumination and shading algorithms can also be made adaptive with the distance. Note that the RDM can be directly integrated with shadowing algorithms like Shadow Mapping, and can project and receive shadows. We also simulate some self-shadowing and diffuse term simply with the radial direction of the crown surface, obscuring the half facing away from to the light. When we transition to billboard clouds, we will also use the radial direction and distance to the crown center to obscure the billboards according to the side of the tree they are with respect to the light direction, as well as how deep they are inside the crown. Moreover, on close billboards, we compute the Phong model per fragment by modifying the billboard plane normal with the normal vector encoded in the additional channels (Figure 6.35 as well as adding the precomputed ambient occlusion term.

Finally, we also add a certain jittering to the LOD switch distance in order to make the visual wavefront effect – produced by even small changes in color/shading – less coherent and thus less noticeable. Therefore, trees do not switch representation at a fixed distance but at a randomly sampled distance in an interval, as illustrated in Figure 6.37.

6.4.5 Rendering performance

We measured the rendering performance of our tree representations on a dense forest scene (Figure 6.38). We randomly placed 50,000 trees in areas of the aerial image segmented as tree (depicted in green), following a blue noise distribution. Tree instances had varying sizes. We chose a view with a large number of visible trees (42K trees) with a broad range of distances (up to about 1,000 m).

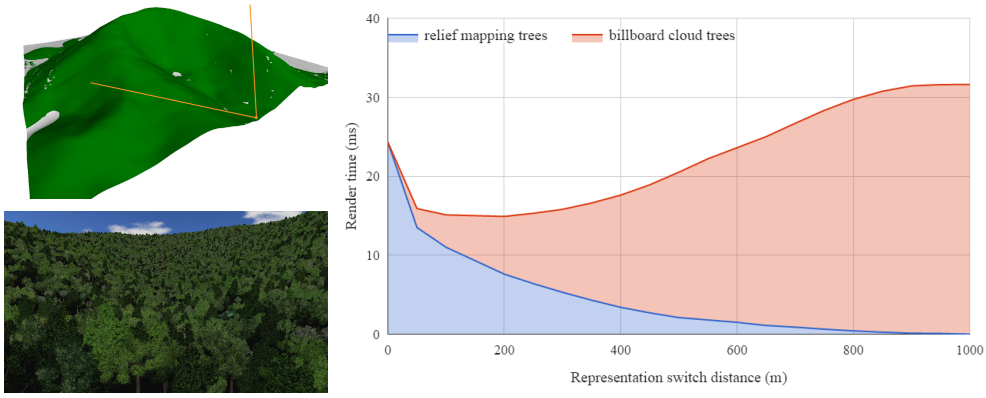


Figure 6.38: Left, test scene used for performance analysis with 42K trees inside the view frustum. Right, accumulated rendering times of both LOD representations depending on the switch distance.

Rendering times were measured with different distance thresholds (d) for the LOD switch. Figure 6.38 shows average rendering times on NVIDIA GTX 970 hardware on a high-quality setting: Full HD (1920×1080) resolution, $8\times$ multisampling, and alpha-to-coverage enabled. All tree instances within d distance were rendered using the billboard cloud representation, using a varying number of billboards per instance (from 16 to 144) according to the screen-projected area. Tree instances beyond d were rendered using the direct (relief-mapping) representation.

For the detailed representation, the throughput varied from 700 trees/ms (only the closest trees being rendered with the detailed representation) to 1,300 trees/ms (all trees rendered as billboard clouds). For the direct rendering, the throughput varied from 1,700 trees/ms (all trees rendered through relief mapping) to 12,000 trees/ms (only distant trees rendered with relief mapping). For distant trees, the relief mapping was up to one order of magnitude faster than billboard clouds, mainly due to its output sensitive nature.

From a visual quality point of view, we found out that the distance threshold d should be greater than 150 m (for smaller values the artifacts of the relief representation become too apparent). From a performance point of view, optimal d values were about 200 m. The reason of a minimum point in the accumulated rendering time is due to the relief representation becoming inefficient for tree instances covering a large part of the viewport, whereas the per-instance cost for the detailed representation remains roughly constant once they reach the minimum number of billboards. Frame rates for $d = 200$ m were above 60 fps even for this dense forest scene and the ones in Figure 6.39.

Memory footprint

For each tree species we need to store the following data: a RDM encoded as a single-channel cubemap (6×256×256), an RGB color texture (1024×768), a small number of billboard textures (512 × 512, encoding RGBA channels plus 8 bytes per pixel for the additional information), and a branching structure (about 200-400 3D points). Note that with only the RDM and the color texture (4.5 MB of uncompressed data) we can already render trees with the direct representations. For the detailed representation we used four billboard textures per species, each texture taking 3 MB of uncompressed data.

The total memory footprint per species is less than 17 MB. Using standard compressed texture formats, the footprint per species is about 5 MB, which is negligible when compared with the aerial imagery. Therefore, our approach is thus suitable for network streaming.

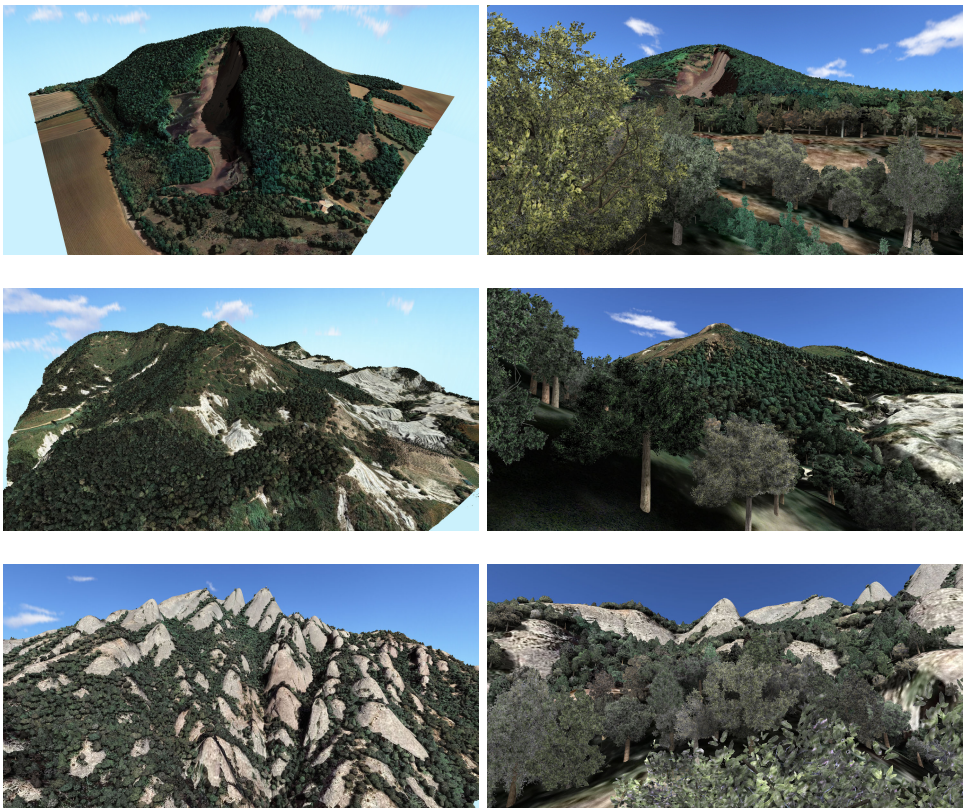


Figure 6.39: Forest landscapes populated with thousands of trees, rendered at Full HD in real-time (60 fps) with shadows and simple wind animation effects. Top to bottom: Croscat (46k trees and 10k shrubs), Creu de Gurb (31k trees and 112k shrubs) and Montserrat (32k trees and 2k shrubs).

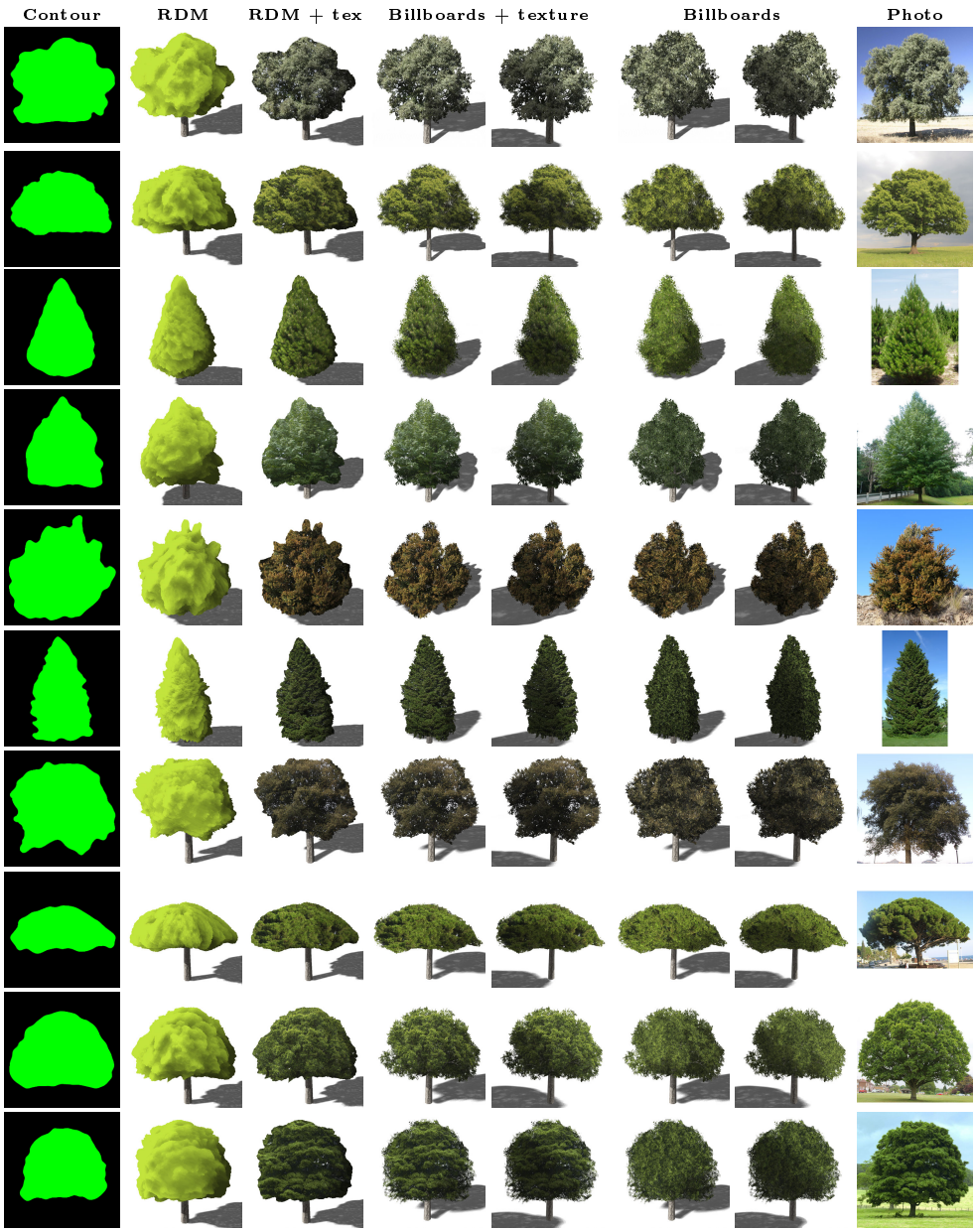


Figure 6.40: Examples of trees rendered using our representations. From left to right: segmentation mask, front view of the volume represented by the radial distance map, the same view using the color texture, two lightings of the billboard cloud textured from the color texture, two lightings of the billboard cloud using the color from the billboards, and the original photograph. Shadow patterns on the ground are achieved by modulating shaded areas with a pattern texture.

6.5 Summary

We started this chapter with the motivation of generating plausible vegetation and populate forest landscapes with tree models resembling local species, while keeping in mind real-time applications such as virtual globe navigation or video games. We claim that our initial goals have been accomplished, as we summarize our contributions as follows:

- An algorithm for the automatic generation of tree image variations starting from a collection of example pictures. The tree variations we generate can be used to author fictional tree images and hybrid specimens, as well as to create variations of the same type of tree to prevent identical copies of the same billboard from being discovered by users.
- We described a method to obtain an approximate envelope for the volume of the tree crown, from a single picture. Using this volume we can obtain different kinds of models for the pictures tree, with varying complexity and degree of realism: a billboard cloud extracted from image superpixels, a complete highly detailed branching structure, or a combination of branchlet billboards and a simple branching structure.
- We introduced the Radial Distance Map as an efficient representation for rendering tree crowns. We showed how the RDM can be directly rendered on far and medium distance views, supporting efficient rending from arbitrary view directions and shading. For closer views, we use the simple branching structure with branchlet billboards, which also allows for animation effects such as wind. By leveraging the RDM, we explained how to transition between LOD while preserving the overall shape and shading.
- We show that realistic forest scenes containing several thousands of trees can be rendered in real time using our proposed models. Moreover, the RDM is encoded in a compact, image-based, and easy to compress representation, suitable for network streaming.

6.6 Publications

Our results on tree modeling and rendering have led to a journal article and a presentation in a local conference. We are currently working on extending the tree variations algorithm and we would like to publish it as another article.

- O. Argudo, A. Chica, and C. Andújar. "Single-picture reconstruction and rendering of trees for plausible vegetation synthesis". In: *Computers & Graphics* 57 (2016), pp. 55–67
- O. Argudo, C. Andújar, and A. Chica. "Tree Variations". In: *CEIG - Spanish Computer Graphics Conference*. 2017

7

Conclusions and future work

7.1 Conclusions

As we stated in the introduction, our primary goal was the leverage of currently available public datasets in order to create detailed 3D scenes from real-world locations, focusing on natural scenarios such as forests and mountains, and providing a level of detail adequate for ground-level navigation views. We believe our different contributions on the many topics we have seen from Chapters 4 to 6 provide a valuable set of tools and techniques towards accomplishing this goal.

We have designed a usable pipeline to efficiently segment aerial images of a terrain set. With little effort and in short time, users can create a training set for the arbitrary classes they may need, train a model, and classify the pixels in the photograph. The result is a set of probability maps per class, as well as their composition as a labeled pixel map. These outputs can be directly used as input on procedural detail generators or synthesis algorithms.

The problem of the low resolution on publicly available DEM has been addressed through a Convolutional Neural Network that performs super-resolution, effectively doubling it, if the aerial image is also available. For those cases where aerial imagery is not available or does not have enough resolution, our proposed network is still capable of enhancing the terrain details based on the DEM itself. Moreover, we also proposed a dictionary-based approach that replaces patches of the terrain with high-resolution patches from a set of exemplars. This approach can also be used to synthesize coherent additional layers on a terrain, such as vegetation densities or distributions, thus reusing more complete datasets on

other regions that lack all the information we would like to have about it.

Finally, we proposed techniques for generating a variety of tree models taking as input a set of pictures from the species found in the area of interest. As a first step, this picture set can be enlarged through the synthesis of image variations, and each image can be then converted into a relief representation suitable for extracting a full tree model and rendering it efficiently, as we have demonstrated on landscapes covered with tens of thousands of trees.

Putting it all together, it is possible to download a terrain as low resolution DEM with its corresponding aerial image, increase the resolution of this DEM, identify where vegetation, water, or bare ground is located, synthesize coherent extra layers – like vegetation species distribution –, and place a variety of tree models on it that resemble the local species.

In conclusion, while there is still plenty of room for improvements and additional work, we believe we accomplished our global objective.

7.2 Future work

There are many ideas and avenues of work that would be interesting to follow.

Regarding the segmentation, a first extension could be using regression to estimate the height of vegetation, for example training from Digital Surface Models or LiDAR data. This could also allow to identify individual trees in the photograph and define the position where each of them is placed. It would also be interesting to design tools for facilitating the manual segmentation of the aerial images and constructing a good training set. Additionally, improvements on classification algorithms and hardware could lead to very fast – ideally interactive – segmentation. Meanwhile, a hierarchical or incremental classification could be implemented, showing pixel labels to the user as soon as they are computed. In that case, the training set examples could be provided through an interface, in such a way that the resulting segmentation is updated after each user action (e.g. including a new region sample for a class, defining a new class, correcting the label of a region, etc.). Moreover, real-time segmentation would allow on the fly classification during a streaming of DEM and aerial imagery like in virtual globe applications, with detail enhancement taking place on the client side.

On the terrain synthesis part, one of the current limitations of our dictionary-based method is that it cannot be directly used to generate constructs such as road networks or cities, as there is no direct relation between a patch and its

neighbors. Exploring more complex functions to enforce coherence also between neighboring patches is a problem worth investigating as future work. For example, as suggested by a reviewer, we could blend neighboring patches by finding the best cut in the overlapping area. We agree this is an interesting idea as future research on the topic, and might also improve the results on more structured data patches.

Related to the DEM super-resolution work, a possible extension would be testing if the use of hyper-spectral images – instead of visible RGB – helps with the super-resolution enhancement. Additionally it could be worth to exploit the information of DTMs (Digital Terrain Models) along with DSMs (Digital Surface Models) wherever available. The proposed algorithm could benefit from treating differently any objects on the ground (vegetation and others) and captured in the DSM (not in the DTM). Another possibility would be to benefit from the availability of multiple aerial images of the terrain taken at different years, as lighting conditions might be different and could reveal more DEM details.

Finally, on the plausible vegetation section, an interesting avenue for future work is the support of sparse crowns and multiple-nuclei foliage, since our crown reconstruction algorithm is aimed for a single crown nucleus. Moreover, reconstructing branch structures seen in the photograph was left out in our work, and is certainly a useful extension. For the tree variations algorithm, the Mean Value Coordinates allow multiple polygons, and we could explore how to find morphing when target and source contours are not topologically equivalent. Another improvement of our proposed algorithms would be to compute which subset of the given exemplar trees are sufficiently compatible for their combination, storing it as a graph, and generating variations and models automatically using it as a guide. Moreover, given the recent popularity of Generative Adversarial Networks as generative models, it could be worth to explore their capabilities on generating tree models and exploring a latent space for variations.

7.3 Publications list

The contributions from this thesis have led to the following articles, four of them in indexed journals:

- O. Argudo, A. Chica, and C. Andújar. "Single-picture reconstruction and rendering of trees for plausible vegetation synthesis". In: *Computers & Graphics* 57 (2016), pp. 55–67
- O. Argudo et al. "Segmentation of Aerial Images for Plausible Detail Synthesis". In: *Computers & Graphics* 71 (2018), pp. 23–34
- O. Argudo et al. "Coherent multi-layer landscape synthesis". In: *The Visual Computer* 33.6 (2017), pp. 1005–1015
- O. Argudo, C. Andújar, and A. Chica. "Tree Variations". In: *CEIG - Spanish Computer Graphics Conference*. 2017
- O. Argudo, A. Chica, and C. Andújar. "Terrain Super-resolution through Aerial Imagery and Fully Convolutional Networks". In: *Computer Graphics Forum* 37.2 (2018), pp. 101–110

An overview of this thesis has also been presented as a short talk and poster at the Eurographics 2018 Doctoral Consortium.

References

[AAC+17] O. Argudo, C. Andújar, A. Chica, E. Guérin, J. Digne, A. Peytavie, and E. Galin. "Coherent multi-layer landscape synthesis". In: *The Visual Computer* 33.6 (2017), pp. 1005–1015.

[AAC17] O. Argudo, C. Andújar, and A. Chica. "Tree Variations". In: *CEIG - Spanish Computer Graphics Conference*. 2017.

[ACA16] O. Argudo, A. Chica, and C. Andújar. "Single-picture reconstruction and rendering of trees for plausible vegetation synthesis". In: *Computers & Graphics* 57 (2016), pp. 55–67.

[ACA18] O. Argudo, A. Chica, and C. Andújar. "Terrain Super-resolution through Aerial Imagery and Fully Convolutional Networks". In: *Computer Graphics Forum* 37.2 (2018), pp. 101–110.

[ACC+18] O. Argudo, M. Comino, A. Chica, C. Andújar, and F. Lumbreras. "Segmentation of Aerial Images for Plausible Detail Synthesis". In: *Computers & Graphics* 71 (2018), pp. 23–34.

[ACV+14] C. Andújar, A. Chica, M. A. Vico, S. Moya, and P. Brunet. "Inexpensive Reconstruction and Rendering of Realistic Roadside Landscapes". In: *Computer Graphics Forum* 33.6 (2014), pp. 101–117.

[AD05] M. Alsweis and O. Deussen. "Modeling and Visualization of symmetric and asymmetric plant competition". In: *Eurographics Workshop on Natural Phenomena*. The Eurographics Association, 2005.

[AK84] M. Aono and T. Kunii. "Botanical Tree Image Generation". In: *IEEE Computer Graphics and Applications* 4.5 (1984), pp. 10–34.

[Aks08] S. Aksoy. "Spatial techniques for image classification". In: *Image processing for remote sensing* (2008), pp. 225–247.

[ASS+12] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (2012), pp. 2274–2282.

[BA83] P. J. Burt and E. H. Adelson. "The Laplacian Pyramid as a Compact Image Code". In: *IEEE Transactions on Communications* 31 (1983), pp. 532–540.

[BCD+08] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. "KNIME: The Konstanz Information Miner". In: *Data Analysis, Machine Learning and Applications: Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation e.V., Albert-Ludwigs-Universität Freiburg, March 7–9, 2007*. Springer Berlin Heidelberg, 2008, pp. 319–326.

[BCF+05] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen. "Realistic Real-Time Rendering of Landscapes Using Billboard Clouds". In: *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2005)* 24.3 (2005), pp. 507–516.

[BF01] B. Benes and R. Forsbach. "Layered Data Representation for Visual Simulation of Terrain Erosion". In: *Proceedings of the 17th Spring Conference on Computer Graphics*. SCCG '01. IEEE Computer Society, 2001, pp. 80–86.

[Blo85] J. Bloomenthal. "Modeling the Mighty Maple". In: *SIGGRAPH Comput. Graph.* 19.3 (1985), pp. 305–311.

[BMG06] F. Boudon, A. Meyer, and C. Godin. *Survey on Computer Representations of Trees for Realistic and Efficient Rendering*. Rapport de recherche 2301. LIRIS, 2006.

[BN12] E. Bruneton and F. Neyret. "Real-time Realistic Rendering and Lighting of Forests". In: *Computer Graphics Forum* 31.2pt1 (2012), pp. 373–382.

[BNB13] D. Bradley, D. Nowrouzezahrai, and P. Beardsley. "Image-based Reconstruction and Synthesis of Dense Foliage". In: *ACM Trans. Graph.* 32.4 (2013), 74:1–74:10.

[BPA15] A. Beacco, N. Pelechano, and C. Andujar. "A Survey of Real-Time Crowd Rendering". In: *Computer Graphics Forum* 35.8 (2015), pp. 32–50.

[Bre01] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[BSS07] J. Brosz, F. F. Samavati, and M. C. Sousa. "Terrain Synthesis By-Example". In: *Advances in Computer Graphics and Computer Vision: International Conferences VISAPP and GRAPP 2006, Setúbal, Portugal, February 25-28, 2006, Revised Selected Papers*. Springer Berlin Heidelberg, 2007, pp. 58–77.

[BTH+06] B. Beneš, V. Těšínský, J. Hornyš, and S. K. Bhatia. "Hydraulic erosion". In: *Computer Animation and Virtual Worlds* 17.2 (2006), pp. 99–108.

[ČB10] L. Čehovin and Z. Bosnić. "Empirical evaluation of feature selection methods in classification". In: *Intelligent data analysis* 14.3 (2010), pp. 265–281.

[CBC+16] G. Cordonnier, J. Braun, M.-P. Cani, B. Benes, É. Galin, A. Peytavie, and É. Guérin. "Large Scale Terrain Generation from Tectonic Uplift and Fluvial Erosion". In: *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics*. EG '16. 2016, pp. 165–175.

[CCB+17] G. Cordonnier, M. P. Cani, B. Benes, J. Braun, and E. Galin. "Sculpting Mountains: Interactive Terrain Modeling Based on Subsurface Geology". In: *IEEE Transactions on Visualization and Computer Graphics* PP.99 (2017), p. 1.

[CEG+18] G. Cordonnier, P. Ecormier, E. Galin, J. Gain, B. Benes, and M. Cani. "Interactive Generation of Time-evolving, Snow-Covered Landscapes with Avalanches". In: *Computer Graphics Forum* 37.2 (2018), pp. 497–509.

[CGG+17] G. Cordonnier, E. Galin, J. Gain, B. Benes, E. Guérin, A. Peytavie, and M.-P. Cani. "Authoring Landscapes by Combining Ecosystem and Terrain Erosion Simulation". In: *ACM Trans. Graph.* 36.4 (2017), 134:1–134:12.

[Che14] A. M. Cheriyadat. "Unsupervised Feature Learning for Aerial Scene Classification". In: *Geoscience and Remote Sensing, IEEE Transactions on* 52.1 (2014), pp. 439–451.

[Cho16] F. Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357.

[CHP+07] R. L. Cook, J. Halstead, M. Planck, and D. Ryu. "Stochastic Simplification of Aggregate Detail". In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. ACM, 2007.

[CMF98] N. Chiba, K. Muraoka, and K. Fujita. "An erosion model based on velocity fields for the visual simulation of mountain scenery". In: *The Journal of Visualization and Computer Animation* 9.4 (1998), pp. 185–194.

[Coh67] D. Cohen. "Computer simulation of biological pattern generation processes". In: *Nature* 216 (1967), pp. 246–248.

[COR+16] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 3213–3223.

[CP10] M. Clasen and S. Prohaska. "Image-Error-Based Level of Detail for Landscape Visualization." In: *Proc. VMV 2010*. Vol. 10. Citeseer. 2010, pp. 267–274.

[CPK+18] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2018), pp. 834–848.

[CPS+15] M. Castelluccio, G. Poggi, C. Sansone, and L. Verdoliva. "Land Use Classification in Remote Sensing Images by Convolutional Neural Networks". In: *CoRR* abs/1508.0 (2015).

[CTK16] D. Cho, Y.-W. Tai, and I. Kweon. "Natural Image Matting using Deep Convolutional Neural Networks". In: *ECCV*. 2016.

[CZP+18] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: *arXiv:1802.02611* (2018).

[DBS06] C. Dachsbacher, T. Bolch, and M. Stamminger. "Procedural Reproduction of Terrain Textures with Geographic Data". In: *In: L. Kobbelt ; T. Kuhlen ; T. Aach ; R. Westermann (Hrsg.) : Vision, Modelling and Visualization (Vision, Modelling and Visualization Aachen Nov 22.-24). Berlin : Akademische Verlagsgesellschaft*. 2006, pp. 105–112.

[DDS+03] X. Décoret, F. Durand, F. X. Sillion, and J. Dorsey. "Billboard Clouds for Extreme Model Simplification". In: *ACM SIGGRAPH 2003 Papers*. ACM, 2003, pp. 689–696.

[DHL+02] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, P. Prusinkiewicz, C. Colditz, M. Stamminger, and G. Drettakis. "Interactive Visualization of Complex Plant Ecosystems". In: *Proceedings of the Conference on Visualization '02*. VIS '02. IEEE Computer Society, 2002, pp. 219–226.

[DHL+98] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. "Realistic Modeling and Rendering of Plant Ecosystems". In: *Proceedings of the 25th Annual Conference on*

Computer Graphics and Interactive Techniques. SIGGRAPH '98. ACM, 1998, pp. 275–286.

[DL06] O. Deussen and B. Lintermann. *Digital Design of Nature: Computer Generated Plants and Organics*. 1st. Springer-Verlag Berlin Heidelberg, 2006.

[DLH+16] C. Dong, C. C. Loy, K. He, and X. Tang. "Image Super-Resolution Using Deep Convolutional Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.2 (2016), pp. 295–307.

[DN04] P. Decaudin and F. Neyret. "Rendering Forest Scenes in Real-Time". In: *Rendering Techniques '04 (Eurographics Symposium on Rendering)*. 2004, pp. 93–102.

[DN09] P. Decaudin and F. Neyret. "Volumetric Billboards". In: *Computer Graphics Forum* 28.8 (2009), pp. 2079–2089.

[DVS03] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. "Sequential Point Trees". In: *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. ACM, 2003, pp. 657–662.

[EBP+12] A. Emilien, A. Bernhardt, A. Peytavie, M.-P. Cani, and E. Galin. "Procedural generation of villages on arbitrary terrains". In: *The Visual Computer* 28.6-8 (2012), pp. 809–818.

[EF01] A. A. Efros and W. T. Freeman. "Image Quilting for Texture Synthesis and Transfer". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. ACM, 2001, pp. 341–346.

[EMP+98] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach*. 3rd. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann Publishers Inc., 1998.

[EVC+15] A. Emilien, U. Vimont, M.-P. Cani, P. Poulin, and B. Benes. "WorldBrush: Interactive Example-based Synthesis of Procedural Virtual Worlds". In: *ACM Trans. Graph.* 34.4 (2015), 106:1–106:11.

[FBW+13] B. Fröhlich, E. Bach, I. Walde, S. Hese, C. Schmullius, and J. Denzler. "Land Cover Classification of Satellite Images using Contextual Information". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* II-3/W1 (2013), pp. 1–6.

[FFC82] A. Fournier, D. Fussell, and L. Carpenter. "Computer Rendering of Stochastic Models". In: *Commun. ACM* 25.6 (1982), pp. 371–384.

[FRD13] B. Fröhlich, E. Rodner, and J. Denzler. "Semantic Segmentation with Millions of Features: Integrating Multiple Cues in a Combined Random Forest Approach". In: *Computer Vision – ACCV 2012*. Vol. 7724. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 218–231.

[FUM05] A. L. Fuhrmann, E. Umlauf, and S. Mantler. "Extreme Model Simplification for Forest Rendering". In: *Proceedings of the First Eurographics Conference on Natural Phenomena*. NPH'05. Eurographics Association, 2005, pp. 57–67.

[GDG+16] E. Guérin, J. Digne, E. Galin, and A. Peytavie. "Sparse representation of terrains for procedural modeling". In: *Computer Graphics Forum (proceedings of Eurographics 2016)* 35.2 (2016), pp. 177–187.

[GDG+17] E. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez. "Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks". In: *ACM Transactions on Graphics (proceedings of Siggraph Asia 2017)* 36.6 (2017).

[GGG+13] J.-D. Génevaux, É. Galin, É. Guérin, A. Peytavie, and B. Beneš. "Terrain Generation Using Procedural Models Based on Hydrology". In: *ACM Transaction on Graphics* 32.4 (2013), 143:1–143:13.

[GGP+15] J.-D. Génevaux, E. Galin, A. Peytavie, E. Guérin, C. Briquet, F. Grosbellet, and B. Benes. "Terrain Modelling from Feature Primitives". In: *Computer Graphics Forum* 34.6 (2015), pp. 198–210.

[GLC+17] J. Gain, H. Long, G. Cordonnier, and M.-P. Cani. "EcoBrush: Interactive Control of Visually Consistent Large-Scale Ecosystems". In: *Computer Graphics Forum* 36.2 (2017), pp. 63–73.

[GM77] R. A. Gingold and J. J. Monaghan. "Smoothed particle hydrodynamics: theory and application to non-spherical stars". In: *Monthly notices of the royal astronomical society* 181.3 (1977), pp. 375–389.

[GMB+13] J. Guénard, G. Morin, F. Boudon, and V. Charvillat. "Reconstructing Plants in 3D from a Single Image Using Analysis-by-Synthesis". In: *Advances in Visual Computing*. Springer, 2013, pp. 322–332.

[GMM15] J. Gain, B. Merry, and P. Marais. "Parallel, Realistic and Controllable Terrain Synthesis". In: *Comput. Graph. Forum* 34.2 (2015), pp. 105–116.

[GMN05] G. Gilet, A. Meyer, and F. Neyret. "Point-based rendering of trees". In: *Eurographics Workshop on Natural Phenomena*. 2005, pp. 67–72.

[GMS09] J. Gain, P. Marais, and W. Straßer. "Terrain Sketching". In: *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*. I3D '09. 2009, pp. 31–38.

[GOO+17] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez. "A Review on Deep Learning Techniques Applied to Semantic Segmentation". In: *CoRR* abs/1704.06857 (2017). arXiv: 1704.06857.

[GSS05] I. Garcia, M. Sbert, and L. Szirmay-Kalos. "Tree rendering with billboard clouds". In: *Proceedings of Third Hungarian Conference on Computer Graphics and Geometry*. 2005, pp. 9–15.

[GTB15] M. Gryka, M. Terry, and G. J. Brostow. "Learning to Remove Soft Shadows". In: *ACM Transactions on Graphics* 34.5 (2015), 153:1–153:15.

[GW07] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Pearson, 2007.

[GWM+08] M. Glencross, G. J. Ward, F. Melendez, C. Jay, J. Liu, and R. Hubbold. "A Perceptually Validated Model for Surface Depth Hallucination". In: *ACM SIGGRAPH 2008 Papers*. Los Angeles, California: ACM, 2008, 59:1–59:8.

[Har01] P. Harrison. "A Non-Hierarchical Procedure for Re-Synthesis of Complex Textures". In: *WSCG*. 2001.

[HF06] K. Hormann and M. S. Floater. "Mean Value Coordinates for Arbitrary Planar Polygons". In: *ACM Trans. Graph.* 25.4 (2006), pp. 1424–1441.

[HGA+10] H. Hnaidi, É. Guérin, S. Akkouche, A. Peytavie, and É. Galin. "Feature based terrain generation using diffusion equation". In: *Computer Graphics Forum* 29.7 (2010), pp. 2179–2186.

[HLZ+17] S. Hu, Z. Li, Z. Zhang, D. He, and M. Wimmer. "Efficient tree modeling from airborne LiDAR point clouds". In: *Computers & Graphics* 67.Supplement C (2017), pp. 1–13.

[Hon71] H. Honda. "Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body". In: *Journal of Theoretical Biology* 31.2 (1971), pp. 331–338.

[IMT99] T. Igarashi, S. Matsuoka, and H. Tanaka. "Teddy: a sketching interface for 3D freeform design". In: *ACM SIGGRAPH 1999*. ACM. 1999, pp. 409–416.

[IZZ+16] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-image translation with conditional adversarial networks". In: *arXiv preprint arXiv:1611.07004* (2016).

[Jak00] A. Jakulin. "Interactive vegetation rendering with slicing and blending". In: *Proc. of Eurographics (Short Presentations)*. 2000.

[JSD+14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia* (2014), pp. 675–678.

[KB14] D. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[KBK+09] P. Krištof, B. Beneš, J. Křivánek, and O. Šťáva. "Hydraulic erosion using smoothed particle hydrodynamics". In: *Computer Graphics Forum*. Vol. 28(2). 2009, pp. 219–228.

[KCV+98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. "Interactive Multi-resolution Modeling on Arbitrary Meshes". In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. 1998, pp. 105–114.

[KK11] P. Krähenbühl and V. Koltun. "Efficient inference in fully connected crfs with gaussian edge potentials". In: *Advances in neural information processing systems*. 2011, pp. 109–117.

[KKM16] J. Kim, J. Kwon Lee, and K. Mu Lee. "Accurate image super-resolution using very deep convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1646–1654.

[KMN+69] F. J. Kriegler, W. A. Malila, R. F. Nalepka, and W. Richardson. "Preprocessing transformations and their effects on multispectral recognition". In: *Proceedings of the Symposium on Remote Sensing of Environment '69*. 1969, pp. 97–131.

[KMN88] A. D. Kelley, M. C. Malin, and G. M. Nielson. "Terrain simulation using a model of stream erosion". In: *Computer Graphics* 22.4 (1988), pp. 263–268.

[Kru14] J. Kruschke. *Doing Bayesian Data Analysis*. 2nd. Academic Press, 2014.

[KS15] Š. Kohek and D. Strnad. "Interactive synthesis of self-organizing tree models on the GPU". In: *Computing* 97.2 (2015), pp. 145–169.

[KS17] Š. Kohek and D. Strnad. "Interactive Large-Scale Procedural Forest Construction and Visualization Based on Particle Flow Simulation". In: *Computer Graphics Forum* (2017), pp. 389–402.

[KSE+03] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. "Graphcut Textures: Image and Video Synthesis Using Graph Cuts". In: *ACM Trans. Graph.* 22.3 (2003), pp. 277–286.

[LD99] B. Lintermann and O. Deussen. "Interactive Modeling of Plants". In: *IEEE Comput. Graph. Appl.* 19.1 (1999), pp. 56–65.

[LES+06] J. D. Lacewell, D. Edwards, P. Shirley, and W. B. Thompson. "Stochastic Billboard Clouds for Interactive Foliage Rendering". In: *Journal of Graphics Tools* 11.1 (2006), pp. 1–12.

[LH05] S. Lefebvre and H. Hoppe. "Parallel Controllable Texture Synthesis". In: *ACM Trans. Graph.* 24.3 (2005), pp. 777–786.

[Lin68] A. Lindenmayer. "Mathematical models for cellular interactions in development: Parts I and II". In: *Journal of theoretical biology* 18.3 (1968), pp. 280–315.

[LMP01] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2001, pp. 282–289.

[LPC+11] Y. Livny, S. Pirk, Z. Cheng, F. Yan, O. Deussen, D. Cohen-Or, and B. Chen. "Texture-lobes for Tree Modelling". In: *ACM Transactions on Graphics* 30.4 (2011), 53:1–53:10.

[LRB+12] S. Longay, A. Runions, F. Boudon, and P. Prusinkiewicz. "TreeSketch: Interactive Procedural Modeling of Trees on a Tablet". In: *Eurographics Workshop on Sketch-Based Interfaces and Modeling*. The Eurographics Association, 2012, pp. 107–120.

[LSD15] J. Long, E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[Luc77] L. B. Lucy. "A numerical approach to the testing of the fission hypothesis". In: *The astronomical journal* 82 (1977), pp. 1013–1024.

[LYO+10] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana. "Automatic Reconstruction of Tree Skeletal Structures from Point Clouds". In: *ACM Transactions on Graphics (TOG)* 29.6 (2010), 151:1–151:8.

[MKM89] F. K. Musgrave, C. E. Kolb, and R. S. Mace. "The Synthesis and Rendering of Eroded Fractal Terrains". In: *SIGGRAPH Comput. Graph.* 23.3 (1989), pp. 41–50.

[MP96] R. Měch and P. Prusinkiewicz. "Visual Models of Plants Interacting with Their Environment". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. ACM, 1996, pp. 397–410.

[MZ93] S. G. Mallat and Z. Zhang. "Matching pursuits with time-frequency dictionaries". In: *IEEE Transactions on Signal Processing* 41.12 (1993), pp. 3397–3415.

[Nag98] K. Nagashima. "Computer generation of eroded valley and mountain terrains". In: *The Visual Computer* 13.9-10 (1998), pp. 456–464.

[NFD07] B. Neubert, T. Franken, and O. Deussen. "Approximate Image-based Tree-modeling Using Particle Flows". In: *ACM Trans. Graph.* 26.3 (2007).

[NLP+13] M. Natali, E. M. Lidal, J. Parulek, I. Viola, and D. Patel. "Modeling Terrains and Subsurface Geology". In: *EuroGraphics 2013 State of the Art Reports (STARs)*. 2013, pp. 155–173.

[NPD+11] B. Neubert, S. Pirk, O. Deussen, and C. Dachsbacher. "Improved Model- and View-Dependent Pruning of Large Botanical Scenes". In: *Computer Graphics Forum* 30.6 (2011), pp. 1708–1718.

[OGG08] D. B. O'brien, M. R. Gupta, and R. M. Gray. "Cost-sensitive multi-class classification from probability estimates". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 712–719.

[OOI05] M. Okabe, S. Owada, and T. Igarashi. "Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing". In: *Computer Graphics Forum* 24.3 (2005), pp. 487–496.

[OPM02] T. Ojala, M. Pietikäinen, and T. Mäenpää. "Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 24.7 (2002), pp. 971–987.

[Opp86] P. E. Oppenheimer. "Real Time Design and Animation of Fractal Plants and Trees". In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '86. ACM, 1986, pp. 55–64.

[Pal07] W. Palubicki. *Fuzzy Plant Modeling with OpenGL- Novel Approaches in Simulating Phototropism and Environmental Conditions*. VDM Verlag, 2007.

[Per85] K. Perlin. "An Image Synthesizer". In: *SIGGRAPH Comput. Graph.* 19.3 (1985), pp. 287–296.

[PGG+09] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou. "Arches: a Framework for Modeling Complex Terrains." In: *Comput. Graph. Forum* 28.2 (2009), pp. 457–467.

[PH93] P. Prusinkiewicz and M. Hammel. "A fractal model of mountains with rivers". In: *Proc. Graphics Interface*. 1993, pp. 174–180.

[PHL+09] W. Palubicki, K. Horel, S. Longay, A. Runions, B. Lane, R. Měch, and P. Prusinkiewicz. "Self-organizing Tree Models for Image Synthesis". In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH '09. ACM, 2009, 58:1–58:10.

[PHM93] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness. "Animation of Plant Development". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. ACM, 1993, pp. 351–360.

[PJM+17] S. Pirk, T. Jarząbek Michałand Hädrich, D. L. Michels, and W. Palubicki. "Interactive Wood Combustion for Botanical Tree Models". In: *ACM Trans. Graph.* 36.6 (2017), 197:1–197:12.

[PJM94] P. Prusinkiewicz, M. James, and R. Měch. "Synthetic Topiary". In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. ACM, 1994, pp. 351–358.

[PL96] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., 1996.

[PMK+01] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. "The Use of Positional Information in the Modeling of Plants". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. ACM, 2001, pp. 289–300.

[PNH+14] S. Pirk, T. Niese, T. Hädrich, B. Benes, and O. Deussen. "Windy Trees: Computing Stress Response for Developmental Tree Models". In: *ACM Trans. Graph.* 33.6 (2014), 204:1–204:11.

[PNS15] O. A. B. Penatti, K. Nogueira, and J. A. dos Santos. "Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?" In: *Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2015, pp. 44–51.

[POC05] F. Policarpo, M. M. Oliveira, and J. L. D. Comba. "Real-time Relief Mapping on Arbitrary Polygonal Surfaces". In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D '05. Washington, District of Columbia: ACM, 2005, pp. 155–162.

[PSK+12] S. Pirk, O. Stava, J. Kratt, M. A. M. Said, B. Neubert, R. Měch, B. Benes, and O. Deussen. "Plastic Trees: Interactive Self-adapting Botanical Tree Models". In: *ACM Trans. Graph.* 31.4 (2012), 50:1–50:10.

[QNT+03] X. Qin, E. Nakamae, K. Tadamura, and Y. Nagai. "Fast Photo-Realistic Rendering of Trees in Daylight". In: *Computer Graphics Forum* 22.3 (2003), pp. 243–252.

[QTZ+06] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang. "Image-based Plant Modeling". In: *ACM Trans. Graph.* 25.3 (2006), pp. 599–604.

[RB85] W. T. Reeves and R. Blau. "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems". In: *SIGGRAPH Comput. Graph.* 19.3 (1985), pp. 313–322.

[RBJ+14] A. Roncat, C. Briese, J. Jansa, and N. Pfeifer. "Radiometrically Calibrated Features of Full-Waveform Lidar Point Clouds Based on Statistical Moments". In: *IEEE Geoscience and Remote Sensing Letters* 11.2 (2014), pp. 549–553.

[RCS+03] Y. Rodkaew, P. Chongstitvatana, S. Siripant, and C. Lursinsap. "Particle systems for plant modeling". In: *Plant growth modeling and applications* (2003), pp. 210–217.

[REF+88] P. de Reffye, C. Edelin, J. Françon, M. Jaeger, and C. Puech. "Plant Models Faithful to Botanical Structure and Development". In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '88. ACM, 1988, pp. 151–158.

[Rep05] A. Repenning. "Inflatable icons: Diffusion-based interactive extrusion of 2d images into 3d models". In: *Journal of Graphics, GPU, and Game Tools* 10.1 (2005), pp. 1–15.

[RFR04] L. Ruiz, A. Fdez-Sarría, and J. Recio. "Texture feature extraction for classification of remote sensing data using wavelet decomposition: a comparative study". In: *20th ISPRS Congress*. Vol. 35. 2004, pp. 1109–1114.

[RLP07] A. Runions, B. Lane, and P. Prusinkiewicz. "Modeling Trees with a Space Colonization Algorithm". In: *Proceedings of the Third Eurographics Conference on Natural Phenomena*. NPH'07. Eurographics Association, 2007, pp. 63–70.

[RMD04] A. Reche, I. Martin, and G. Drettakis. "Volumetric Reconstruction and Interactive Rendering of Trees from Photographs". In: *ACM Transactions on Graphics* 23.3 (2004), pp. 720–727.

[RMG+15] D. Ritchie, B. Mildenhall, N. D. Goodman, and P. Hanrahan. "Controlling procedural modeling programs with stochastically-ordered sequential monte carlo". In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 105.

[Roe07] S. Roettger. "NDVI-based Vegetation Rendering". In: *Proceedings of the Ninth IASTED International Conference on Computer Graphics and Imaging*. CGIM '07. ACTA Press, 2007, pp. 41–45.

[ŠBB+08] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek. "Interactive Terrain Modeling Using Hydraulic Erosion". In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '08. Eurographics Association, 2008, pp. 201–210.

[SBW06] J. Schneider, T. Boldte, and R. R. Westermann. "Real-time editing, synthesis, and rendering of infinite landscapes on GPUs". In: *Vision, Modeling and Visualization 2006*. 2006, p. 145.

[Sch12] K. Schindler. "An Overview and Comparison of Smooth Labeling Methods for Land-Cover Classification". In: *Geoscience and Remote Sensing, IEEE Transactions on* 50.11 (2012), pp. 4534–4545.

[SEZ+13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* abs/1312.6 (2013).

[SLJ+14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4 (2014).

[SPd10] J. A. dos Santos, O. A. B. Penatti, and R. da Silva Torres. "Evaluating the Potential of Texture and Color Descriptors for Remote Sensing Image Retrieval and Classification." In: *VISAPP (2)*. 2010, pp. 203–208.

[SPK+14] O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes. "Inverse Procedural Modelling of Trees". In: *Computer Graphics Forum* 33.6 (2014), pp. 118–131.

[SRD+01] I. Shlyakhter, M. Rozenoer, J. Dorsey, and S. Teller. "Reconstructing 3D Tree Models from Instrumented Photographs". In: *IEEE Computer Graphics and Applications* 21.3 (2001), pp. 53–61.

[SS01] G. Stockman and L. G. Shapiro. *Computer Vision*. 1st. Prentice Hall PTR, 2001.

[SS99] R. E. Schapire and Y. Singer. "Improved Boosting Algorithms Using Confidence-rated Predictions". In: *Machine Learning* 37.3 (1999), pp. 297–336.

[STB+14] R. M. Smelik, T. Tutenel, R. Bidarra, and B. Benes. "A Survey on Procedural Modelling for Virtual Worlds". In: *Computer Graphics Forum* 33.6 (2014), pp. 31–50.

[Suz+85] S. Suzuki et al. "Topological structural analysis of digitized binary images by border following". In: *Computer vision, graphics, and image processing* 30(1) (1985), pp. 32–46.

[SWP+13] T. Sturn, M. Wimmer, P. Purgathofer, and S. Fritz. "Landspotting - Games for Improving Global Land Cover". In: *Proceedings of Foundations of Digital Games Conference 2013 (FDG 2013)*. Chania, Greece, 2013, pp. 117–125.

[SZ14] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1 (2014).

[TC89] C. H. Teh and R. T. Chin. "On the Detection of Dominant Points on Digital Curves". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 11.8 (1989), pp. 859–872.

[TFX+08] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan. "Single Image Tree Modeling". In: *ACM Transactions on Graphics* 27.5 (2008), 108:1–108:7.

[TG07] J. A. Tropp and A. C. Gilbert. "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit". In: *IEEE Transactions on Information Theory* 53.12 (2007), pp. 4655–4666.

[TGY+09] J. O. Talton, D. Gibson, L. Yang, P. Hanrahan, and V. Koltun. "Exploratory Modeling with Collaborative Design Spaces". In: *ACM SIGGRAPH Asia 2009 Papers*. 2009, 167:1–167:10.

[TLL+11] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Měch, and V. Koltun. "Metropolis Procedural Modeling". In: *ACM Trans. Graph.* 30.2 (2011), 11:1–11:14.

[TMS12] P. Tokarczyk, J. Montoya, and K. Schindler. "An Evaluation of Feature Learning Methods for High Resolution Image Classification". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* I-3 (2012), pp. 389–394.

[TO03] A. Torralba and A. Oliva. "Statistics of natural image categories". In: *Network: computation in neural systems* 14.3 (2003), pp. 391–412.

[TWE11] P. Tania, C. D. W., and R. Erik. "A Survey of Image Statistics Relevant to Computer Graphics". In: *Computer Graphics Forum* 30.6 (2011), pp. 1761–1788.

[TWW+15] P. Tokarczyk, J. D. Wegner, S. Walk, and K. Schindler. "Features, Color Spaces, and Boosting: New Insights on Semantic Classification of Remote Sensing Images". In: *Geoscience and Remote Sensing, IEEE Transactions on* 53.1 (2015), pp. 280–295.

[TZW+07] P. Tan, G. Zeng, J. Wang, S. B. Kang, and L. Quan. "Image-based Tree Modeling". In: *ACM Transactions on Graphics* 26.3 (2007), p. 87.

[WBC+09] J. Wither, F. Boudon, M.-P. Cani, and C. Godin. "Structure from silhouettes: a new paradigm for fast sketch-based design of trees". In: *Computer Graphics Forum* 28.2 (2009), pp. 541–550.

[WCM+07] C. Wojtan, M. Carlson, P. J. Mucha, and G. Turk. "Animating Corrosion and Erosion". In: *Proceedings of the Third Eurographics Conference on Natural Phenomena*. NPH'07. Eurographics Association, 2007, pp. 15–22.

[WLK+09] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. "State of the Art in Example-based Texture Synthesis". In: *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009, pp. 93–117.

[WLX+18] G. Wang, H. Laga, N. Xie, J. Jia, and H. Tabia. "The Shape Space of 3D Botanical Tree Models". In: *ACM Trans. Graph.* 37.1 (2018), 7:1–7:18.

[WP95] J. Weber and J. Penn. "Creation and Rendering of Realistic Trees". In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. ACM, 1995, pp. 119–128.

[WWB+09] J. Whitehill, T.-f. Wu, J. Bergsma, J. R. Movellan, and P. L. Ruvolo. "Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise". In: *Advances in Neural Information Processing Systems 22*. Curran Associates, Inc., 2009, pp. 2035–2043.

[XGC07] H. Xu, N. Gossett, and B. Chen. "Knowledge and Heuristic-based Modeling of Laser-scanned Trees". In: *ACM Trans. Graph.* 26.4 (2007).

[XM15] L. Xu and D. Mould. "Procedural Tree Modeling with Guiding Vectors". In: *Computer Graphics Forum* 34.7 (2015), pp. 47–56.

[XYS+16] K. Xie, F. Yan, A. Sharf, O. Deussen, H. Huang, and B. Chen. "Tree Modeling with Real Tree-Parts Examples". In: *IEEE Transactions on Visualization and Computer Graphics* 22.12 (2016), pp. 2608–2618.

[YAM+15] M. E. Yumer, P. Asente, R. Mech, and L. B. Kara. "Procedural Modeling Using Autoencoder Networks". In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 2015, pp. 109–118.

[YYD+07] Q. Yang, R. Yang, J. Davis, and D. Nister. "Spatial-Depth Super Resolution for Range Images". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. 2007, pp. 1–8.

[ZST+07] H. Zhou, J. Sun, G. Turk, and J. M. Rehg. "Terrain Synthesis from Digital Elevation Models". In: *IEEE Transactions on Visualization and Computer Graphics* 13.4 (2007), pp. 834–848.

Websites

[1] *ASTER Global Digital Elevation Map.* https://asterweb.jpl.nasa.gov/gdem.asp.

[2] *CORINE Land Cover 2012.* http://land.copernicus.eu/pan-european/corine-land-cover/clc-2012.

[3] *ESA GlobCover.* http://due.esrin.esa.int/page_globcover.php.

[4] *Flickr: Explore everyone's photos on a Map.* http://www.flickr.com/map.

[5] *Geo-Wiki, Engaging Citizens in Environmental Modeling.* https://www.geo-wiki.org/.

[6] *Global Imagery Browse Services (GIBS).* http://earthdata.nasa.gov/about/science-system-description/eosdis-components/global-imagery-browse-services-gibs.

[7] *Global Land Cover 2000.* http://forobs.jrc.ec.europa.eu/products/glc2000/glc2000.php.

[8] *Global Multi-resolution Terrain Elevation Data 2010 (GMTED2010).* http://lta.cr.usgs.gov/GMTED2010.

[9] *Google Earth.* http://www.google.com/earth.

[10] *Google Street View.* http://www.google.com/streetview.

[11] *Global 30 Arc-Second Elevation (GTOPO30).* http://lta.cr.usgs.gov/GTOPO30.

[12] *Institut Cartogràfic i Geològic de Catalunya.* http://www.icgc.cat.

[13] *Instituto Geográfico Nacional.* http://www.ign.es.

[14] *Mammut Project360.* http://project360.mammut.ch.

[15] *Mapa de Cobertes de Sòl de Catalunya.* http://www.creaf.uab.es/mcsc/.

[16] *GLCF: MODIS Land Cover.* http://www.landcover.org/data/lc/.

[17] *National Agriculture Imagery Program (NAIP).* http://lta.cr.usgs.gov/node/300.

[18] *NASA World Wind.* http://worldwind.arc.nasa.gov.

[19] *OpenDEM Searcher.* http://opendem.info/opendemsearcher.html.

[20] *Open Topography*. http://www.opentopography.org/.

[21] *Plan Nacional de Ortofotografía Aérea*. http://pnoa.ign.es.

[22] *Südtiroler Bürgernetz GeoKatalog*. http://geokatalog.buergernetz.bz.it/geokatalog. Online; accessed 01 October 2017.

[23] *SIOSE: Sistema de Información de Ocupación del Suelo en España*. http://www.siose.es/.

[24] *Shuttle Radar Topography Mission*. http://www2.jpl.nasa.gov/srtm.

[25] G. Guennebaud, B. Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[26] P. Mineiro. *Modeling Mechanical Turk*. http://www.machinedlearnings.com/2011/01/modeling-mechanical-turk.html. Blog. 2011.

All the referenced websites have been correctly accessed on June 12, 2018.