Instituto de Matemática Pura e Aplicada - IMPA

PhD Thesis

High-Level Techniques for Landscape Creation

Leandro Moraes Valle Cruz

Advisor: Luiz Velho

Co-Advisor: Eric Galin

Rio de Janeiro

Março de 2015

Abstract

A complete solution for creating an entire planet by combining different types of elements from an intuitive specification is still an open problem. High level techniques are in the core of the solution for this issue. Its main focus is to achieve ways for a more intuitive specification and to provide smart modeling operations.

In this context, our work aims to contribute for the development of high level techniques for landscape creation by the proposal of two approaches. The first deals with the manipulation of the position of the landscape elements (according to some rules and targets), and the second is related to the generation of terrain models (through two new data-driven methods).

In particular, we will introduce an extendable method for resizing of a landscape specification keeping the overall appearance. It is based on the insertion and removal of objects (based on general purpose criteria) followed by a scene adjustment (enlargement or shrinking).

Furthermore, we will introduce two data-driven techniques for terrain synthesis inspired on by-example texture synthesis. Both techniques are developed over the same intuitive control approach: a guide and a categorization map. Furthermore, both methods take advantage of the geometric nature of the data for improving the synthesis. The first technique is a patch-based algorithm, with new optimization structures for patch choice, and a new patch insertion approach (both based on the geometric nature of the data). The second is a pixel-based method, based on a graph of exemplars in multirresolution.

Resumo

A criação de um planeta virtual completo, através da combinação de diferentes tipos de elementos, a partir de uma especificação intuitiva, ainda é um problema em aberto na área de modelagem de paisagens virtuais. As técnicas de Alto Nível estão no centro da solução dessa tarefa. Seu foco é obter meios que possibilitem um controle intuitivo de principais características do modelo, bem como, a partir de operações de modelagem inteligentes.

Nesse contexto, contribuimos para a criação dessas técnicas de alto nível através de duas abordagens. A primeira é uma técnica para manipulação da posição dos objetos de uma paisagem (de acordo com algumas regras e objetivos); e a segunda se refere a geração de modelos de terrenos (através de dois métodos guiados por exemplares).

Em particular, apresentaremos um método extensível para o redimensionamento de uma especificação vetorial de uma paisagem, mantendo a aparência global do modelo. Isto é realizado através de inserções e remoções de objetos (baseado em critérios generalistas), seguido de ajustes na cena (alargamento ou encolhimento).

Além disso, apresentaremos dois métodos para síntese de terreno guiados por exemplares, inspirados em técnicas de síntese de textura. Um dos algoritmos é baseado em *patches*, enquanto o outro é baseado em *pixels*. Ambos apresentam uma abordagem de controle intuitiva baseada em um guia e em um mapa de categorias. Além disso, aprimoramos algumas etapas utilizadas em algoritmos clássicos para sintese de textura, para o contexto de síntese de terreno, tirando proveito da natureza geométrica do nosso dado.

Contents

Chapter 1

Introduction

In the last three decades, the modeling of virtual landscapes presented meaning advances. Nowadays, we can see extremely realistic models in movies, animations, and games. Nevertheless, the creation of these models is still widely laborious.

The advances in landscape modeling is due to the advances of the computer graphics methods (general methods, and those for modeling and visualization); of the use of better hardwares, which enable the implementation of more complex algorithms; and the availability of large data sets, from which we can obtain informations whereof make the modeling more efficient.

The improvement of computer graphics and the technological advances have afforded the creation of quite sophisticated methods of modeling. Indeed, some sub-areas of geometric modeling are well established. In particular, the modeling of manufactured objects, one of the first studied area, already has a wide variety of techniques able of producing very precise pieces. Nevertheless, the characteristics of manufactured objects is completely different than those of landscapes. On one hand, it is common to create CAD-CAM modeling from very precise specification of measures of each part. On the other hand, the landscape models are very large and with many imprecise details. Therefore, a landscape requires a more general specification, highlighting the most important features of the model.

As a result, a recent trend of landscape modeling, as well as of geometric modeling for general purpose, is to use approaches which admit a more general control, from a

specification of macro structures, and automating the creation of details. However, the known methods tend to have only one of the following two characteristics: (1) being realistic and precise, but depending on a lot of manual work for specification; or (2) from a more intuitive specification, creating a model visually good, but not precise, efficient to transmit an idea about the shape, but with some unnatural features.

When parts of the model are automatically generated, it happens many times a loss of control about some features. In these cases, it is necessary a re-work of designers to fix it or to create those structures.

In order to solve this problem, it has recently emerged many researches trying to generate a more realistic and quite precise model from an intuitive specification. We can call these approaches *High Level techniques* because they are based on a smart control of representations related to more specific features. For example, a method for creating a landscape whereof we define the topography only by describing where will have mountains, where will be smooth, and how is the percentage of water and land, is an example of a high level approach. On the other hand, create a landscape from defining parameters of procedural methods, or by changing of the position of control points, is a low level approach. Nevertheless, the creation of high level techniques is still an issue in geometric modeling, in spite of some good results have been achieved for very specific scopes.

One of the main focus for high level techniques is to achieve manners for a more intuitive specification. Many methods get inspiring in analogies with activities already practiced outside of the modeling environment (for instance: drawing and painting). In fact, these analogies allow the user to understand easily the behavior of the method, and thus, to obtain the desired result, by applying less effort. From this specification, it is coded many operations whereof the semantic is based on a more abstract representation.

The definition of these representations, as well, the creation of operations with a more exquisite semantic is not trivial. Whereas, there are more and more available data about landscape elements. We can use it for adding extra informations for synthesis about structures not explicitly specified. The challenge regards to, besides of the difficult about dealing with a large amount of data, extract a more manageable

representation easier to be used for synthesis. Notwithstanding, there are some techniques based in simple information from real datasets (in general, small data sets, or even, a single data). To the best of our knowledge, there is no sophisticated techniques for analyzing data base of elements present in landscapes producing high level representations that would allow richer operations. Moreover, the processing of these large datasets is also a scientific and technological challenge.

In this context, our work aims to contribute for creating high level techniques for landscape generation, by analyzing the state of art, evaluating the pros and cons about the related techniques, and by analyzing the trends and challenges of this topic. In this direction, we will introduce two approaches: one for handling with vector specification of landscapes, and another for creating terrain based in real data.

In Chapter 2, we will present an overview of techniques for landscape creation. This topic will be presented by a comparison of images and landscape elements. In particular, we will show the relation between terrain creation and texture synthesis, and landscape specification and vector graphics.

In Chapter 3, we will introduce our method for landscape specification resizing. Besides of the shrinking and enlargement techniques, we will describe the particularities of the method, like the creation of paths, masks and distance field, and some general approaches for choosing the objects which will be inserted or removed.

In Chapter 4, we will show two adaptation of by-example texture synthesis techniques for terrain creation derived from the Markov Model. In particular, these methods emphasize how to guide the synthesis for creating some coherent terrain structures, while preserving the exemplar features, and how to use the geometric particularities of the model to improve the result.

In Chapter 5, we will show the main trends for the development of methods for landscape creation: being high-level. We will present the main challenges for the creation of this kind of approaches, and how to use the tools already known.

Finally, in Conclusion, we will highlight the high level characteristics of our methods; mention how they can be combined; how they contribute for the development of landscape creation techniques being more intuitive and able of generating huge models (like an entire planet); and we will point to some future works.

1.1 Related Works

The conceptual part of this thesis (Chapter 2) shows an analysis of the state of art of landscape creation methods, relating them with image methods. This analysis concludes that these methods are going in the direction of development of high level techniques. Some other works present similar analysis but with other emphasis.

Smelik et al. [1] introduced a quite complete survey about techniques for creating elements present in a virtual world. They quickly mention the trends of these techniques of becoming more and more intuitive and dealing with existent data base of predefined model, but do not point precisely in the high-level direction.

Natalie et al. [2] presented a survey about terrain modeling and creation of subsurface geology. This work present some interesting terrain representation, based in geological layers, that can be used for more precise simulation. However, the emphasis of our work is only in the topography of the terrain (the highest layers).

Mariethoz and Lefebvre [3] introduced the similarities between the approaches of multiple-point geostatistics and by-exemplar texture synthesis. In our work, we keep focus only in computer graphics techniques, but we present a more wide discussion about techniques for terrain generation and texture synthesis.

Other works present a quite complete overview about some particular topic we have approached. For instance, Ebert et al. [4] published a book about procedural methods for modeling and texture creation (including techniques for creating fractal planets). Lagae et al. [5] present a survey about noise functions. They are used for procedural texture synthesis, but can be used for terrain generation. Wei et al. [6] present a survey about by-example texture synthesis. Some of these techniques also can be adapted for terrain generation.

The novelty of this thesis are the methods concerned to the manipulation of landscape specification and two data-driven terrain synthesis. The first is related to vector graphics and image retargeting methods. The second is related to texture synthesis and by-example terrain creation.

Image retargeting has been widely studied in the last years. Valquero et al. [7] presented a survey about this topic. The main motivation of this problem is to

change the dimensions of an image to fit in a new region. It is becoming increasingly important due to the large variety of displays where the same image may be visualized. These displays differ in size and aspect ratio. Because a simple scaling or cropping is too limited, these methods try to perform a content-based resizing. Thus, they go towards defining a good fitting.

Image resizing can also be useful in photography. Liu et al. [8] introduced a method to change the composition of the objects in some image to increase the aesthetic value. Dekkers and Kobbelt [9] extended the image retargeting approach to mesh deformation. They proposed a method that combines elastic and plastic deformation using the seam carving concept in the mesh editing context.

The content-based resizing problem is the same problem that we approach in this thesis. Nevertheless, because the difference of the contexts (images and vector landscapes) there are differences between our method and those previously introduced for image resizing.

Our resizing approach is performed by applying shifts in the objects according to some inserted or removed paths in the scene. It is similar to the Seam Carving method for image resizing, introduced by Avidan et al. [10]. This approach is the base of our method to shrink and to enlarge the scene. The difference between our approach and the above methods [8, 9, 10] is, because we are dealing with a straightforward model, our method can use a simpler metric, but also producing good results.

Furthermore, we added rules for control the placement of objects into the scene (seam carving does not use semantic about the scene). There are many approaches for spreading objects in some specific models. However, all of these methods are based in specific rules according to the element type. Because of this, in this work we do not introduce complex methods (methods with hard constraints) for this operation, but we will present some general rules and show how to extend our method to use other particular rules.

The procedural generation of cities is often hierarchical. For instance, it is possible first to create the city frontier and the neighborhood boundaries, after placing some landmarks, such avenues, streets, special buildings, squares, city blocks, and finally to create buildings and houses. Parish and Muller [11] introduced a seminal procedural

method to create cities using L-systems. Recently, many other methods were presented using pattern-based approach, agent-simulations, tensor fields, etc. Watson et al. [12] presented an overview of this topic. In the same direction, the procedural definition of furniture layouts uses shape grammars [13], subdivision methods [14], tree-based or graph-based methods [15], among others.

Yeh et al. [16] introduced an approach for the retargeting of scenes based on Markov chain Monte Carlo. Their model is based on a factor graph (a graph that encodes constraints as factors). It is a global approach applied on a specification with strong relations between the objects presented in the scene.

The layout of an internal scene (spreading of furniture) and the buildings of a city are strongly constrained. Nevertheless, there are many contexts where the constraints related to the position of the objects are weaker, for instance: the position of cities in a map, mountains in a landscape, houses and other buildings in a rural area (or in a medieval scenario), trees in a field, etc. The previously mentioned researches [11, 12, 13, 14, 15, 16] work well in the first situation, while the criteria we will present work better in the second one.

It is important to highlight that the vector model has structural information which can be very useful in many of these methods. Thus, we can have a base model used as a seed of a procedural technique, and we can resize this model and use it again to generate another with different dimensions.

This landscape specification can be also used for controlling of our data-driven methods. These techniques were inspired by texture synthesis approaches (one is a patch-based and the other is pixel-based), with focus in the improvement of the control and the synthesis quality (by taking advantage of the geometric nature of the terrain).

Patch-based synthesis methods [17] create a model, by taking an exemplar, by covering the target using patches taken from the exemplar, placed with some predefined overlapping over the already synthesized parts of model. The choice of which patch will be chosen depends on some metric based on comparison of the overlapping regions and, eventually, comparing some features of a control model. After the choice of which patch will be inserted, a cut is calculated, into the overlapping region, separating

the old part and the new one. Our synthesis approach is similar, but include a more sophisticated control and some adaptation based on the nature of the data.

An extension of the Image Quilting [17] was presented by Lasram and Lefebvre [18] performing the insertion of a set of patches in parallel. The authors introduced an iterative method, such that, in each step it is necessary to choose a patch for each cell of the grid, define the cut and insert it in the case when there is an improvement of the synthesis quality (based on a specific metric). Our synthesis approach is sequential. But, it is seemingly easy to extend that method to use our control, patch choice and insertion approaches.

Zhou et al. [19] proposed a patch-based method for terrain synthesis, creating the new model by gluing patches of an exemplar. It also provides a map with the main desired ridges or valleys. During the synthesis, the patches are chosen by matching of a descriptor based on these features. This control approach is based in features on an intermediate scale. Despite of our main concern is to control features in a higher scale, we also perform an evaluation of the valleys for improving the matching of this feature.

Freeman et al. [20] proposed a super-resolution approach for image synthesis. This approach chooses the patches by comparing a low frequency on a base image (like our guide) and it matches the high frequency in an overlapping region. Our approach is similar, improving the control and the patch choice and insertion.

Furthermore, there are many pixel-based methods for texture synthesis that can be used for terrain synthesis [21]. But, in general, they are concerned to create a big homogeneous model with the features contained on a provided exemplar. Han et al. [22] proposed a pixel-based approach using in a graph of exemplars in multiscale. This new structure allows the synthesis to create an heterogeneous model, and possibly with an infinity resolution.

To the best of our knowledge, there are few data-driven methods for terrain synthesis. We believe that the availability of terrain data is not entirely explored. Furthermore, there are other important tools useful for creating a high-level technique for landscape creation. In this thesis, we will approach this problem and show how our methods can improve the state of art in this direction.

1.2 Contributions

The main contributions of this thesis are:

- A technique for resizing a vector-based specification of a landscape satisfying some predefined rules according to the initial model [23].
- Two exemplar-based techniques for terrain generation, which are adaptation of classical texture synthesis methods, taking advantage of the geometric nature of the data, and improving the control synthesis process [24, 25].
- An analysis of the state of art about techniques for landscape creation, emphasizing the relation between the creation of images, and the generation and manipulation of elements present in a virtual environment; converging to the identification of the main challenges for the development of high level techniques for this purpose.

In particular, we will introduce a method for resizing of a landscape specification keeping the overall appearance [23]. This method is based on the insertion and removal of objects followed by a scene adjustment (enlargement or shrinking). Furthermore, we will introduce straightforward and effective methods for insertion and removal of objects in the scene based on general criteria.

Moreover, we will introduce two data-driven techniques for terrain synthesis inspired on by-example texture synthesis methods. The focus of these approaches is in an intuitive synthesis control, and in the improvement of the matching of terrain features. We take advantage of some geometric feature of the model, and provide new control structures. We also choose the exemplars, from a large set of exemplars, according to some control structures (like a guide containing the desired low scale, a categorization of the exemplars, and a map of distribution of these categories). The first method is a patch-based technique [25]. The choice of patches is performed by a chaining of a set of descriptors (some for processing acceleration, and others for the improvement of matching, like matching of valleys), and the insertion can be performed by the classical cut and paste, or by an interpolation into the overlapping

region. Furthermore, we perform a vertical translation of the patch to reduce the overlapping error. The second method is a pixel-based technique, including a way to create a graph of exemplars, used for synthesis, and a new way to control high scale features [24].

Finally, we will talk about the main challenges, and point to the trends for the development of high level techniques for landscape creation. We highlight the lack of methods that combine different type of elements, and create elements of different scales. We believe the new generation of techniques for creation of landscapes will have these characteristics (at least, methods with these characteristics will be able of creating good models). From this analysis, we show the high level features of our introduced methods, and to point to future works.

Chapter 2

Concepts of Landscape and Images

An *image* is a function $f : \mathbb{R}^2 \longrightarrow \mathbb{R}^n$, whose dimension of the codomain depends on the representation of the visual attributes (for example, the codomain of an RGB image has dimension equals to three, while the one of a grayscale image is one). A picture shown in a graphic device is the discretization of f in a regular set of samples. The result of the discretization is called *Digital Image*.

The function f is associated to a geometric model. Sometimes, we do not know this model, but only the samples. In this case, we can recreate f from this set through an interpolation process. This procedure is called *reconstruction*.

In certain cases, we can define f from a set of geometric structures and the respective visual attributes. This representation is called *Vector Graphics*. The discretization process of a Vector Graphics to obtain the respective Digital Image is called *rasterization*.

The digital image creation is not a simple evaluation of f. It depends on filtering, some analysis of the samples, among other image operations. A complete explanation about images, rasterization, sampling and reconstruction was provided by Andrew S. Glassner [26].

Similar to images, the main structure of the terrain topography can be represented by a function $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$. This model is called Digital Elevation Model (DEM) [27]. Thus, there is a seemingly trivial association between a grayscale image and this terrain representation. Because of this similarity, we can use some image processing

techniques for terrain processing. In particular, textures are a kind of image more related with terrains. In Section 2.1, we will present a review about terrain and texture synthesis techniques, and how we can adapt texture approaches for terrain creation.

Analogously, there is a relation between a landscape representation and a vector graphics. In general, large landscape models are composed by a set of elements: the terrain topography, trees, rivers, houses, buildings, roads, etc. The landscape representation is a map specifying the position of each element and some other attributes (for example, the respective 3D, a scale, etc). This map is described in a similar way than a vector graphics image. In Section 2.2, we will describe how to represent a landscape and its relation with vector graphics.

2.1 Terrain and Textures

In Computer Graphics, the term *texture* are related to the group of images that represents the material of some object (its appearance). The study about textures comprises some problems like synthesis, mapping, reparameterization, etc. In this text, our concern is only about texture synthesis.

Despite of the advances in texture synthesis in the last four decades, this is still not a well solved problem. For practical applications, it is still necessary many manual work. Nowadays, there is a specific kind of professional, named texture artist, who creates textures for objects and landscapes like those used in games and movies. The work of these professionals consists in creating textures with the known synthesis techniques, creating samples (used in exemplar-based approaches), and applying of manual corrections in the result.

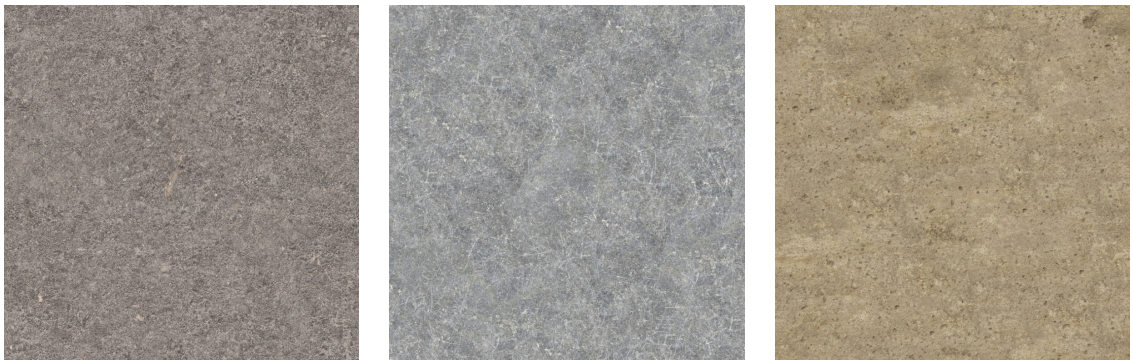
In this section, we will present a texture classification based on the structures and organization of texture elements (Section 2.1.1), and talk about the synthesis approaches and what are challenges for developing of synthesis methods for some class of textures (Section 2.1.2). Furthermore, we will talk about bridges between textures and terrain synthesis (Section 2.1.3).

2.1.1 Texture Classification

Despite of we are limiting the texture only for the images representing the appearance of the material of some object, there are several kinds of textures, with different classes of features. However, there is no widely accepted classification for types of texture. Sylvain Lefebvre [28] presented a classification based on the structures and organization of texture elements. The structures can be more or less stationary and the organization can be more or less regular. The classes are:

1. Unstructured - stochastic
2. Structured - stochastic
3. Structured - organized
4. Near-regular and regular

The *Unstructured - stochastic* textures are the type better studied. It is a stationary model, with a homogeneous appearance, no structures (i.e. there is no edges in the image separating different features). There are some procedural methods based on noise that efficiently create this kind of texture.



(a) (b) (c)

Figure 2.1: Examples of *Unstructured - stochastic* textures

(a) (b) (c)

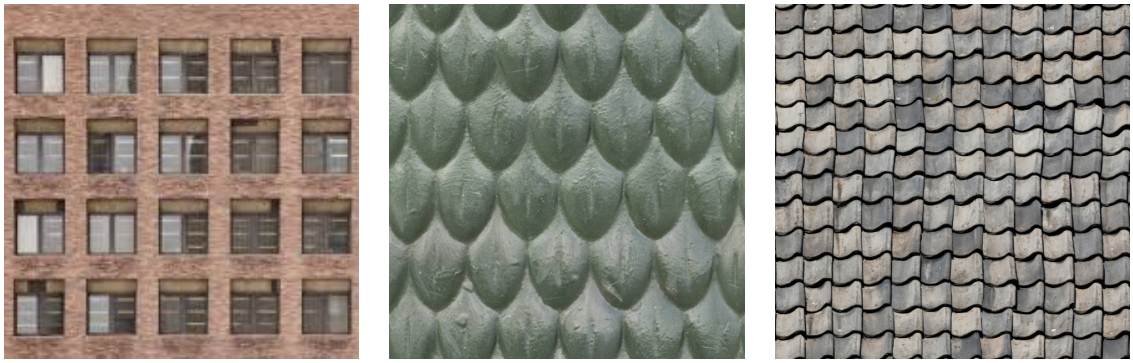
Figure 2.2: Examples of *Structured - stochastic* textures

(a) (b) (c)

Figure 2.3: Examples of *Structured - organized* textures

The *Structured - stochastic* textures contain a globally stochastic appearance, but with local structures. These textures contain some edges boundaring some small structures, but the distribution of these structures is globally homogeneous. Some exemplar-based techniques for texture synthesis are very efficient for the synthesis of this kind of textures.

The *Structured - organized* textures contains some structures strongly defined, i.e., some deterministic patterns. This class contains a rule for elements distributions whose human beings can easily recognize, but can be tough to infer computationally.



(a) (b) (c)

Figure 2.4: Examples of *Near-regular and regular* textures

Because of this, most of classical texture synthesis methods fail for reproducing it. Methods able of reproducing this organization depend on an extra information to guide the synthesis respecting the distribution rules.

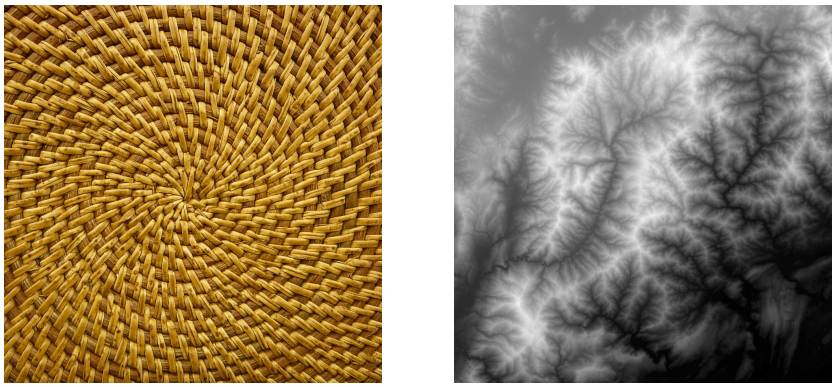
The *Near-regular and regular* textures are a specific type of structured-organized textures. In this case, the organization of the structures presents some degree of periodicity. Despite of this periodic organization, the image is not necessarily periodic, because some structures can appear from different appearance. Data-driven approaches can synthesize these patterns, keeping the periodic organization, without a periodic appearance.

This is a classification of homogeneous textures. By homogeneity we refer to textures containing the same distribution of elements in the entire area. These elements can be structured or not, and the distribution can be organized or not, but the overall appearance of the model is the same. On the other hand, heterogeneous model are those containing a variation of the types of structures and organization. Figure 2.5 illustrates these difference.

We can think about terrain as a structured-organized model. The structures refer to the landforms and the organization is given by geomorphological phenomena. The synthesis of both aspects (structure and organization) has no complete solution. The data-driven methods points to a good creation of local structures, but it is necessary

(a) (b)

Figure 2.5: Examples of homogeneous (a) and heterogeneous (b) textures



(a) (b)

Figure 2.6: Examples of an organized texture (a) and a terrain (b)

to control the process to create a heterogeneous model respecting some natural rules about the landform organization. Figure 2.6 shows a comparison of these models.

Thinking in large landscapes, we also have to include a heterogeneity aspect in the synthesis. Few texture synthesis methods presents approaches for this task. Along this text we will discuss how to control the terrain synthesis methods for synthesize a heterogeneous terrain model.

2.1.2 Texture Synthesis

We can split the texture synthesis approaches in two classes: *procedural* and *exemplar-based*. The first approach is based on mathematical rules able of creating the features of the model. The second one creates a model by reproducing in the target model features from a provided exemplar.

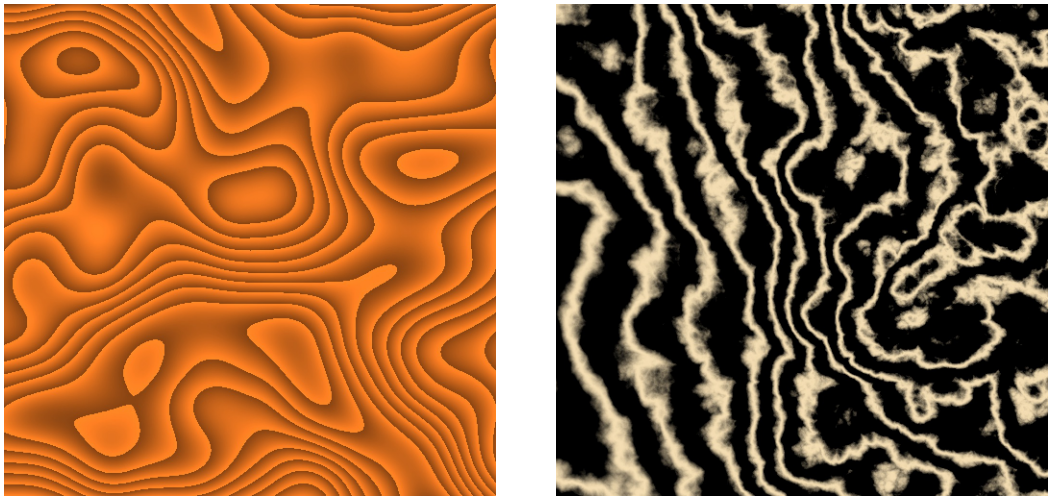
One of the first procedural texture method was proposed by Peachey, in 1985 [29]. He named his approach by *Solid texturing*. It consists in defining a mathematical function, that can be evaluated in a 3D region, that represents the appearance of a material. At the same time, Ken Perlin [30] proposed a texturing method based on a noise function (nowadays named *Perlin Noise*). Most of procedural methods are based on noise functions. Thenceforth, many other approaches were proposed [31, 5]. This class of methods is efficient for synthesizing a stochastic texture from a few parameters. Furthermore, it is seemingly easy to be parallelized.

A procedural texture obtained from a noise function is not necessarily continuous, because it is possible compose a noise function with a non-continuous function. But, the result is homogeneous and keeps an overall pattern, without spurious edges. Because of these features, it is widely used in applications as video games for generating texture for stochastic parts.

The biggest challenge of procedural approaches is the creation of the function which produces some expected features. For instance, it is not trivial define a function able of creating the appearance of a marble, or of a wood (Figure 2.7). In general, the definition of those functions is based on trial and errors, and it is strongly dependent of the experience of the designer for choosing some functions, of which when they are composed produces a texture similar to the expected phenomenon.

In the last two decades, the exemplar-based methods were widely studied, and nowadays we have good methods for synthesizing Structured-Stochastic textures. The ability of reproducing structures present in the exemplar allows these techniques create textures containing structures richer than those created by procedural models.

The exemplar-based techniques can be divided in three categories: pixel-based, patch-based and optimization-based [6]. The first exemplar-based techniques were



(a) Wood (b) Marble

Figure 2.7: Procedural texture created using Perlin Noise

pixel-based [32, 33, 34]. This approach determines the color of the pixel (or whichever characteristic) by analyzing the already synthesized neighboring pixels. Patch-based approaches glue patches (set of connected pixels), also by analyzing the already synthesized neighbors. Optimization-based approaches use a greedy algorithm for determining all pixels, at the same time, by optimizing an energy function. In general, this approach is too expensive, and it is only used for generating a small texture satisfying some property (for example, being tillable). We will keep the focus in the first two approaches.

Figure 2.8 shows a pixel-based scheme. For synthesizing a new pixel, its pre-synthesized neighborhood is compared with similar sets of pixels in the given exemplar (a set of pixel with the same distribution). Finally, the pixel respective to the most similar neighborhood is chosen and placed in the texture. This procedure is repeated until there is a pixel not synthesized.

This approach is efficient for creating stationary texture. In the case of textures with some structure, the strict use of this strategy is not good enough (by *strict* we means without an auxiliary information). An important improvement is to synthesize

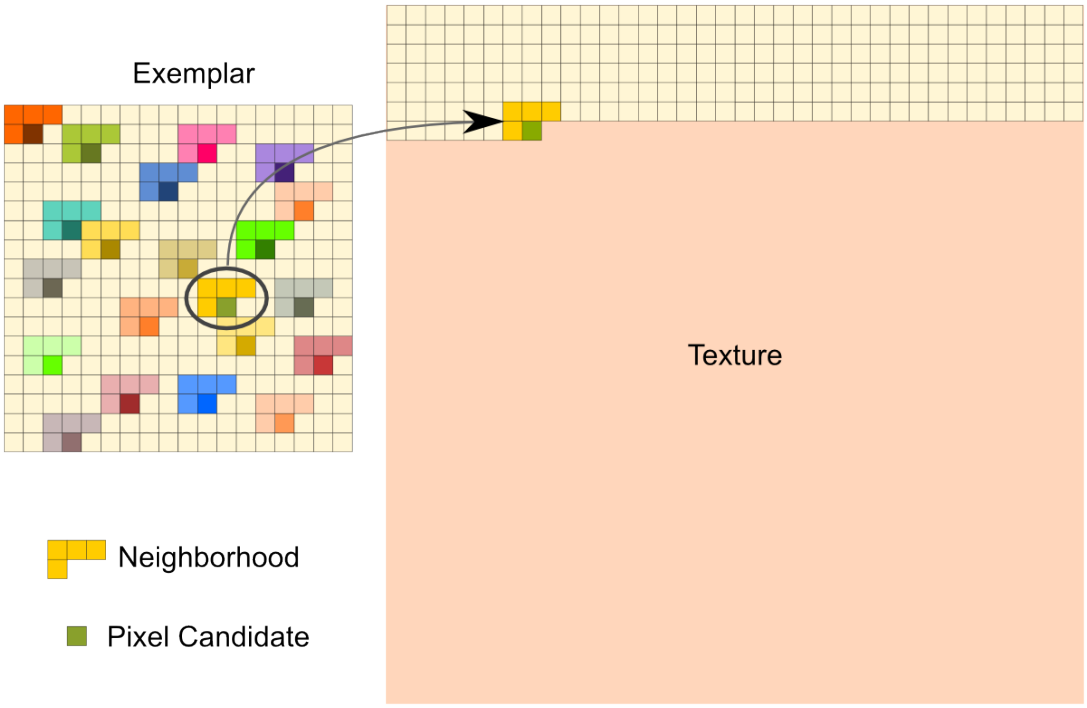


Figure 2.8: Pixel-based scheme

textures in multiscale. The improvement happens because non-stationary textures has features in different scales, and this new approach can reproduce them.

The first multiscale method was proposed by Heeger et al. [33]. However, this method was based in matching of histograms, in this way transferring some textures features by changing in the Cumulative Distribution Function (CDF). For each scale, it changes the pixels for coercing the CDF of the texture to be similar of the CDF of input exemplar.

Heeger et al. [33] proposed the use of a Steerable Pyramid for the multiscale decomposition. De Bonet [35] also proposed a multiscale scheme, but he used a filter bank of first and second Gaussian derivatives and the Laplacian operator for the decomposition. He stated that a local texture measure (an approximation for the human perceptual texture discriminability) can be obtained from these operators.

Wei and Levoy [36] proposed an approach which does not depends on a probabilistic

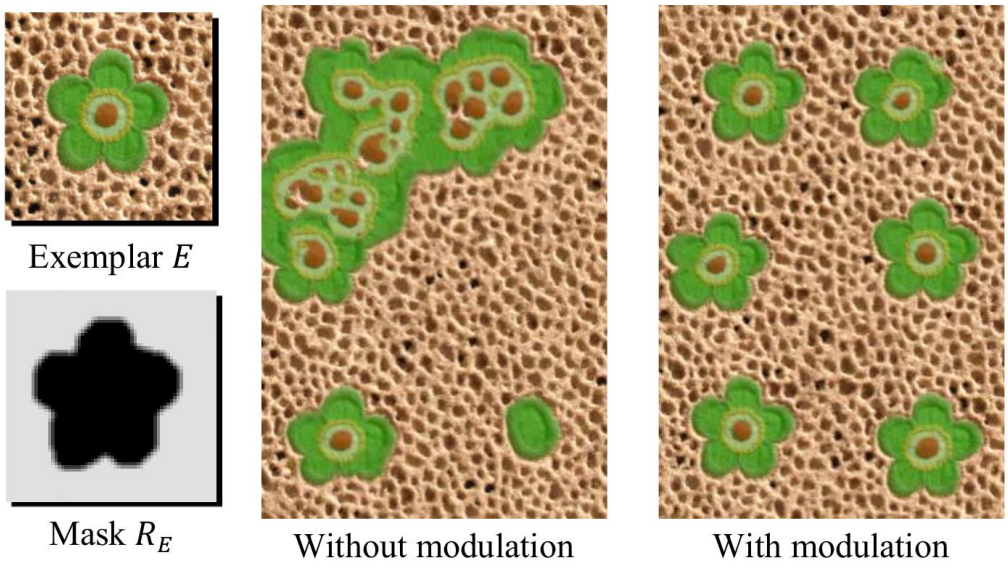


Figure 2.9: Multiscale pixel-based synthesis using spatial modulation, based on a provided mask (image from: [21])

model. They synthesize a texture in multiscale, beginning by the coarsest level. They initialize this texture level with random valid colors, and apply a correction step comparing the neighborhood of the pixel with neighborhood in exemplar.

The synthesis can follow an order, and only to consider the pixels previously corrected. It guarantees a more coherent neighborhood. Many methods proposed different order for pixel choice, and different shape for neighboring. Wey and Levoy also proposed a multiscale technique order-independent [37], i.e., they introduced a technique for defining the pixel neighborhood whose is independent of the synthesis order.

Lefebvre and Hoppe [21] extended this technique allowing a parallel and controllable synthesis. They replaced the exemplar pyramid by a Gaussian stack, increasing the amount of candidate pixels, and so, improving the texture result. Furthermore, they use a jitter step on the pixel coordinate map creating a randomness on texture structure. They also proposed the use of a mask for preservation of structures. The mask is used for creating a spatial modulation of jitter, i.e., the jitter can not send

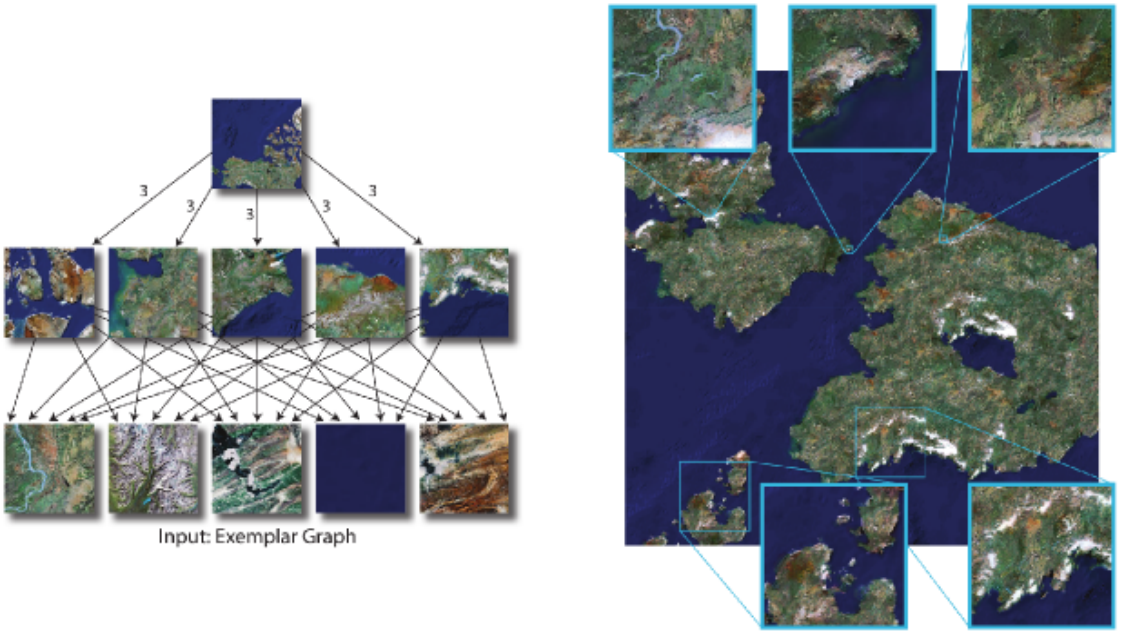


Figure 2.10: Multiscale pixel-based synthesis using a graph containing multiples exemplars (images from: [22])

a pixel from one type of structure region to other. Figure 2.9 shows an example of texture resulted from a spatial modulation. The jitter control (avoiding big jittering in finner levels) and the mask of structures (spatial modulation) allow this method to create texture with more sophisticate structures distributed in a non-organized way.

The synthesis of a heterogeneous texture is still an issue in computer graphics. This problem was addressed in some works [38, 39]. But for while, one of the best result in this direction was proposed by Han et al. [22]. In this work, the authors create a heterogeneous textures from a set of exemplars organized in a graph where edges are related to the difference of scale. They adapted the synthesis to consider all stacks which contains a level in the current scale. Figure 2.10 shows an example of image obtained for the provided graph of exemplars.

The pixel-based synthesis is a parallelizable problem [21, 22]. For achieving a good performance, besides evaluating all pixels in parallel, it is possible to use some

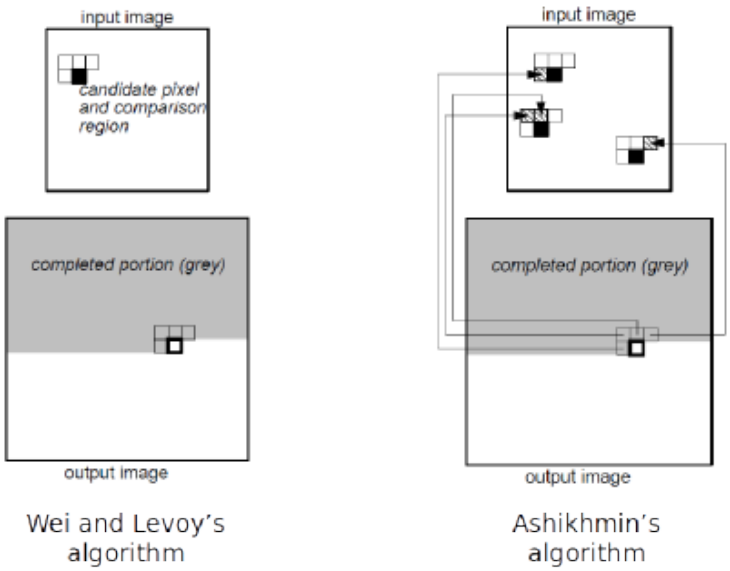


Figure 2.11: Coherent pixel search (image from: [40])

strategies to make faster the pixel choice. First, it is possible using Principal Component Analysis (PCA) for reducing the dimension of the neighborhood representation [21]. Other acceleration strategy is to use a kd-tree for comparison of the pixel neighborhood with all neighborhoods in exemplar [20]. Another one is to avoid the comparison of this neighborhood with all those possibles. It is performed by using a coherent comparison, named *k-coherence*. This coherence is achieved through an analysis of the neighborhood [40], as shown in Figure 2.11. The constant k refers to the distance (in L^∞ norm) from neighbors to the evaluated pixel. And this strategy is independent of the shape of neighborhood.

After the introduction of pixel-based methods, some researchers proposed the increasing of the size of the synthesis unit: from a pixel to a patch (a block of pixels). This new approach consists in creating a texture by gluing, in an adequate way, blocks of pixels. It fostered some improvements in the process and in the quality of the texture appearance. The first improvement refers to performance. In general, it is faster to create a texture by gluing blocks of pixels than to set pixel-by-pixel. The

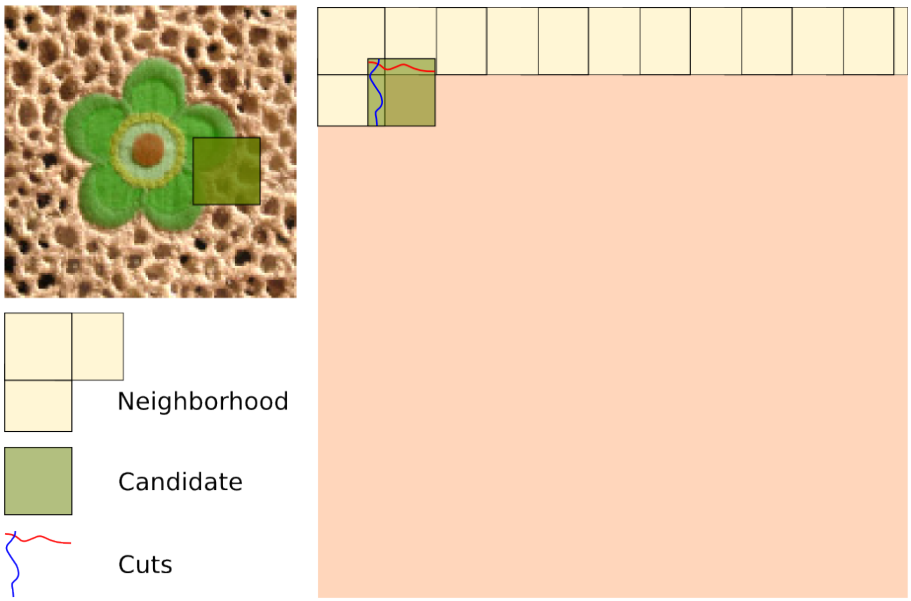


Figure 2.12: Patch-based scheme

second one refers to keep some structures present in the exemplar.

Xu et al. [41] were the first to propose a patch-based approach. Their method consists in creating a texture from a regular tiling (regular grid of patches from the exemplar; possibly the patch is the entire exemplar). In this method, the tiling is modified by moving random small patches. To avoid mismatched features, they created an empty contour around the patch and synthesized the region using the pixel-based approach proposed by Efros and Leung [34].

Another patch-based scheme was proposed by Praun et al. [42]. Their main goal was the creation of an atlas for texture mapping. Because of this, their patches were not a regular block. They mapped the surface texture by gluing interesting patches from the exemplar until it is completely covered. The patches are placed with an overlapping, and this region is blended by an interpolation process.

Efros and Freeman [17] proposed a patch-based approach widely extended in other researches, named *Image Quilting*. The basic idea is shown in Figure 2.12. The idea is to fill the texture with a set of patches placed with a predefined overlapping.

In the overlap region, define a cut separating the already synthesized texture from the new patch. Another patch-based approach was proposed by Liang et al. [43]. The difference of their method and the Image Quilting is they blend the overlapping region, instead of cutting the frontier.

Lasram and Lefebvre [18] proposed a parallel version of the Image Quilting. Their quilting version is based on placing circular patches over regions with higher seam error. These patches are transformed to polar space, where is defined the cut using Dynamic Programming. Besides the cut, they perform an offset of the pixels to improve the alignment of the texture structures.

Besides of the Quilting formulation, Efros and Freeman [17] introduced an approach for synthesis control, named *Texture Transfer*. From a map with some desired features, they guide the choice of the patches to control the organization of the structures in the texture. The texture transfer is similar to *Image Analogies* [44].

An important issue in patch-based techniques is to avoid the breaking of features in the patch seaming. A workaround to avoid discontinuity along the cut is to blend the region close of this path. However, it is not a perfect solution when features contain hard edges (that is smoothed with the blending). The difficulty for a good feature matching is the definition of an adequate representation of this structure. Wu and Yu [45] proposed an approach for recognizing the features and improve the matching. Furthermore, they proposed a way to deform the patch, improving even more the alignment.

Another approach for improving the patch gluing was proposed by Nealen and Alexa [46]. Basically, they create a texture using Image Quilting technique, find points with big seam errors and re-synthesize the region around these points using a pixel-based method. An important contribution of this work is the metric for evaluation of the seam.

In this section, we presented a small review about texture synthesis. More complete discussion about this subject was present by Ebert et al. [4], by Wei et al. [6], and by Lagae et al. [5]

2.1.3 Terrain Synthesis

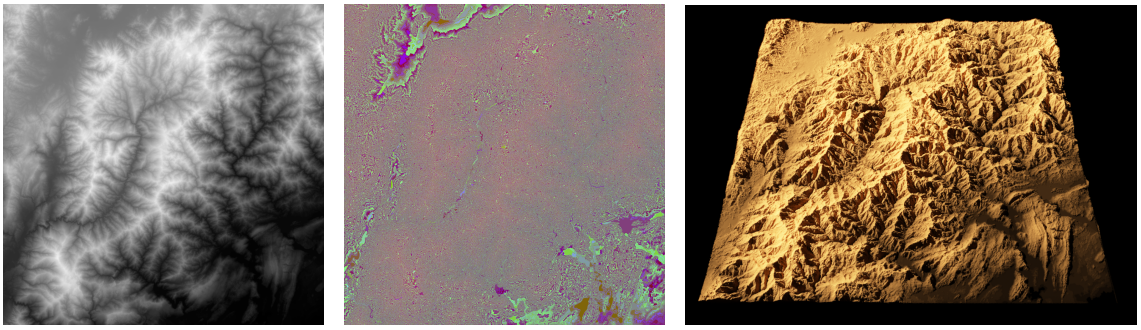
Most of procedural techniques for terrain synthesis are based on fractal or noise function. Benoit B. Mandelbrot was the first to present an approach for synthesizing natural elements using Brownian motion [47, 48]. He created hills and coastlines from fractal surfaces. Fournier et al. [49] introduced a straightforward method for creating of fractal curves and surfaces, based on midpoint displacement.

Gavin Miller [27] proposed an extension of the approach proposed by Fournier et al. [49]. His main contribution was the introduction of the use of Digital Elevation Models (DEM) for terrain representation. It can efficiently describe the main structures of a terrain, but cannot describe caves, overhangs and vertical slopes. Despite of this limitation, the most common data representation for terrains is a regular grid containing the elevation value for each point (a matrix of height). Other more general representations, able of dealing with the cases whereof DEM cannot represent, are meshes [50], implicit modeling [51], and a mixing of DEM and voxels [52].

Beyond the use of image techniques for terrain synthesis, it is possible to use an image for coding the DEM data. It is important to use an image coding format which does not have loss of information during saving and loading. Figure 2.13 shows an example of image whereof the height was coded in the RGB channels of a PNG Image (we coded the height from -10000m to 10000m, with a precision of 1.19mm).

Furthermore, it is possible to use a noise function, used in texture synthesis methods, for terrain creation. Procedural texture approaches have been adapted for terrain synthesis [53, 54]. It is useful for creating the stochastic features of terrains (like details). However, when the terrain is generated only by combining of noise functions or fractals the result does not respect most of natural phenomena.

Another approach for terrain generation is to use simulation methods. Kelley et al. [55] were the first to present a simulation technique for this purpose. Their method was based in an erosion by flowing water. They create a stream network of rivers which is used to guide the surface erosion. Musgrave et al. [56] also proposed a technique based on hydraulic and thermal erosion. This method was extended by Benes and Forbach [52], whose presented a new terrain representation based in



(a) (b) (c)

Figure 2.13: Terrain Representation: (a) the grayscale visualization of the DEM, (b) RGB DEM coding, (c) and 3D visualization of this model.

a layered data representation and introduced new properties for the erosion model presented by Musgrave et al. [56]. Despite of the use of a limited mathematical model (in comparison to the amount of phenomena which indeed occur in nature), they proposed an approach able of synthesizing a more natural terrain than those created using fractal or noise functions.

Kristof et al. [57] introduced a hydraulic erosion approach using Smoothed Particle Hydrodynamics (SPH). The particle motion changes the sediment position creating the terrain shape. More recently, Genevaux et al. [53] introduced another system inspired in hydrology. They create a river network graph from an user sketch, and apply hydraulic erosion in a noise model. Peytavie et al. [51] introduced a framework for terrain representation able of dealing with models containing caves, overhangs, loose rocks, etc. This synthesis is based on sculpting an implicit surface. It is based in a structure with multi-layers of different materials, and thus, this model is able of handling with more sophisticate geomorphological features (an extension of the representation proposed by Benes et al. [52]). Another similar approach was introduced by Stava et al. [58]. They proposed an interactive hydraulic erosion method over a layer-based terrain representation.

Terrain is a huge element of a landscape model. Because of this, there is a trade off between synthesis control and automation of the synthesis. Most of procedural and

simulation-based methods are not well controlled, but very efficient for synthesizing large models. Simulation methods are the most powerful tools for creating natural phenomena. Most of these methods part from a procedural model and create some features inspired in some natural phenomena. This approach is reasonable because procedural methods can efficiently generate the terrain details. However, they fail on creation of main features. So, this approach merges the strength of procedural methods (details) and simulation (features). On the other hand, they are parameterized in a non intuitive way. To improve the control, some sketch-based and data-driven methods were recently proposed.

One of the first sketch-based landscape approach was *Harold*, a method proposed by Cohen et al. [59]. In this tool, the terrain creation is performed from a silhouette drawing. A more sophisticate approach was proposed by Gain et al. [60]. In this method, the authors create a landform from silhouette, base and shadow curves. Tasse et al. [61] proposed a sketch-based terrain editing tool based on the approach used by artists for landscape painting (also specifying the hills silhouette). In this method, the user can interactively control the erosion for simulating a small-scale phenomena. Another similar technique was proposed by Passos and Igarashi [62]. In this method, the user draws the landform silhouette and the system searches in a database of pre-defined models for the one that contains, in some point of view, a similar silhouette and add it to the terrain model.

Many softwares for terrain synthesis provide brush-based tools for sculpting terrain features. In general, sculpting techniques and sketch-based approaches have some similarities. In both, it is necessary to create a 3D model from a small primitive specification (in the case of brushes are a set of points, and in the case of sketches are curves). Carpentier and Bidarra [63] proposed a technique that combines the strength of brush-based (quite similar to sketch-based) and procedural methods for sculpting a terrain using height field brushes.

Despite of the advantage of full controlling, purely sketch-based approaches require too much user effort. Furthermore, sketches are good for defining the main features (like ridges, valleys, rivers, boundaries of landforms, etc.), but this is not a good tool for small features (like erosion, loose rocks, sedimentation, etc.). A trend is a

Figure 2.14: Terrain created from a sketch-based specification and using a diffusion equation (images from [64])

combining of sketch-based approaches for specification of macro features (and maybe other control structures), and simulation or a data-driven for creating terrain features.

Hnaidi et al. [64] introduced a technique for terrain synthesis based on diffusion equation. The input of this method is a vector-based feature curves, i.e., a sketch associated with an information related to some features (like elevation, slope angle and roughness). Figure 2.14 shows an example of model obtained using this technique. This method was extended by Bernhardt et al. [65] improving the sketch-based control and providing a CPU-GPU coupled computation.

Simulation methods are very efficient for generating erosion phenomena and creating rivers networks. Nevertheless, the known techniques are based on small-scale phenomena, because the difficulty of defining the simulation rules. Furthermore, they part from a previous model (in general, a procedural generated model).

Most of exemplar-based terrain modeling methods are widely inspired by image processing approaches. Because of the importance of this topic for this thesis, we will dedicate the next section for presenting a better discussion about data-driven terrain synthesis.

The relation between terrain and textures is explored not only in Computer Graphics researches, but also in Geology and Geostatistics. Natali et al. [2] show a survey about Modeling Terrains and Subsurface Geology. Furthermore, Mariethoz and Lefebvre [3] present relations between multiple-point geostatistics and texture synthesis. Furthermore, Smelik et al. [66, 1] present a discussion about terrain modeling, and the creation of other elements present in landscapes.

2.1.4 Data-driven Terrain Synthesis

As we have mentioned, the similarity between a DEM and an image allows the use of image processing methods for terrain modeling. For example, Lefebvre and Hoppe [21] presented an example of their pixel-based method for texture synthesis for generating a terrain modeling. In this section, we will show some methods that are adaptation of exemplar-based texture synthesis techniques, or use some image approaches for a data-driven terrain synthesis.

Dachsbacher et al. [67] presented an adaptation of the non-parametric sampling method introduced by Efros and Leung [34] for terrain creation. The authors take advantage of the geometric nature of terrain and match the neighborhood of the pixel by a comparison of these values and the respective derivative. They argue that derivatives allow the method capture the anisotropy of the terrain features.

Brosz et al. [68] proposed a multirresolution approach for combining two terrain models. The first is the exemplar containing the high scale; and the second contains the low-scale, and receives the details from the exemplar. This transference is based on a multirresolution decomposition of the models. The matching of areas is based on Image Quilting [17].

One of the most important data-driven terrain synthesis method was proposed by Zhou et al. [19] (Figure 2.15). In this method, they create a new model from an analysis of ridges or valleys, two important terrain features. They find the curves related to these structures of a provided exemplar using the Profile recognition and Polygon breaking Algorithm (PPA) [69]. Also, it is provided a sketch-based map with the desired features. Then, they choose the patches from the exemplar by a comparison of features of the input and the target map. After choosing the patch, they find a cut to separate the pre-synthesized part of the new patch using Graph-Cut, and blend it using the Poisson equation. More recently, this approach was extended by Tasse et al. [70]. They enhanced the patch merging approach for obtaining a closer model of the provided sketch.

Nowadays, the simulation approach is properly well developed, i.e., there are many methods able of creating small-scale features of a terrain, according to some

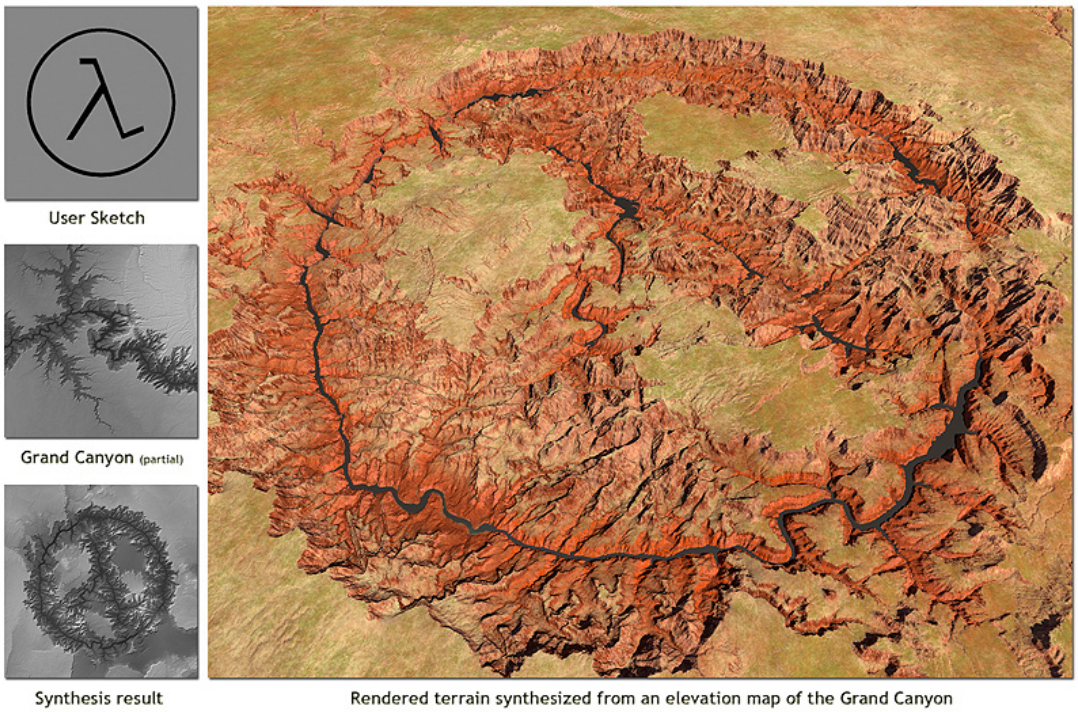


Figure 2.15: Terrain created from a sketch-based specification and a provided exemplar (images from [19])

geomorphological property, over a pre-defined model. However, the set of chosen features is not complete (because some phenomena are not well known), and thus, the created model has some degree of lack of realism. On the other hand, data-driven methods take advantage of combining pieces of a real exemplar, and thus, recreate most of terrain features, even without an explicit specification of the desired features. To the best of our knowledge, there are few works related to a data-driven terrain synthesis. And they do not explored entirely the possibilities of this approach. Beside of this, we believe that it is a trends that will be better developed in the next years.

2.2 Landscapes and Vector Graphics

As previously mentioned, the most common terrain representation (DEM) is quite similar to the digital image representation. Analogously, the landscape representation has some similarities with a vector graphics.

A Vector Graphics is a representation of the function $f : \mathbb{R}^2 \longrightarrow \mathbb{R}^n$, which defines the image. From this representation, we can obtain a digital image of a desired resolution. This representation is composed by geometric primitives (like points, curves and polygons), and their respective visual attributes (like color, thickness of the contour, gradient, etc.), and possibly a digital image (or pieces of it).

The landscape representation has requirement similar to those of vector graphics. The specification process of a landscape consists in defining a terrain, the water bodies (like rivers, lakes, and seas), the atmosphere (sky, clouds, fog, etc.), and the placement of each object into the scene. Among these objects, we can mention buildings, houses, landforms, roads, bridges, etc.

For describing a landscape, we have to define the geometry and visual attributes of each object into the scene. In general, these objects are previously modeled (i.e., in the landscape description has only a reference for the full geometry model of each object). So, the object description only contains the position and orientation into the scene (according to a pre-defined established coordinate system), and possibly a scale factor. Furthermore, it is possible to add some more specific properties related to the application which will use this landscape. As well as, it is possible to add in the description some global properties for scene (for example: initial camera position, fog, weather, etc.).

It is common graphic applications which use landscapes containing several objects with identical geometry and attributes, differing only by position (and maybe, by also orientation and scale). We will call this class of *objects* with similar properties by *elements*.

An example of technique for rendering and editing of a vector-based landscape was proposed by Bruneton and Neyret [71]. They present a method for filling a large landscape using roads, lakes and fields. Each object is described by vectors and

associated to a shader which specifies the appearance.

The vector-based landscape representation is an useful resource for high-level approaches. It allows to keep the focus on describing the main features of the environment, instead to determine geometric details of the object. On the other hand, of course, there are many particularities related to the context. For example, Smelik et al. [72] proposed a high-level approach which combines semantic-based modeling and procedural approaches for populate a huge virtual world, described in vector-based layers.

A good review about techniques for modeling and placement of the objects usually present in a landscape was introduced by Smelik et al. [1]. We do not intend present more details about this subject because it is beyond the scope of this text.

Chapter 3

Landscape Specification Resizing

As we introduced in the previous chapter, large landscape models, such as those used in games, animations or some kind of simulations, are composed by a set of elements: the terrain topography, trees, rivers, houses, buildings, roads, etc. The 3D model is created from a specification model, i.e., a vector model specifying the position of each object and some other attributes.

In this chapter, we will present a technique for resizing a given landscape specification. Our technique is based on the vector model of the scene. The main goal of this approach is to change the dimensions of the model preserving the overall appearance. Figure 3.1 illustrates this context. Figures 3.1a and 3.1b show the original model and the 3D landscape, respectively. Analogously, Figures 3.1c and 3.1d show the enlarged version of this model.

The motivation for this method was the following problem: *In a game or in a simulation, it may be necessary to change the size of the scene according to the number of the characters. For instance, we can have a virtual environment being explored initially by N people, and in another situation, by only half of those. In this second case, it is not good to keep the same space, because the characters will be too far from each other. Thus, it is interesting to reduce the landscape area for returning to the initial proportion. Nevertheless, we do not want to change the overall appearance. In contrast, we want to obtain a smaller scenario by maintaining the same experience.*

In general, a simple scaling of the model is not good enough, because it can

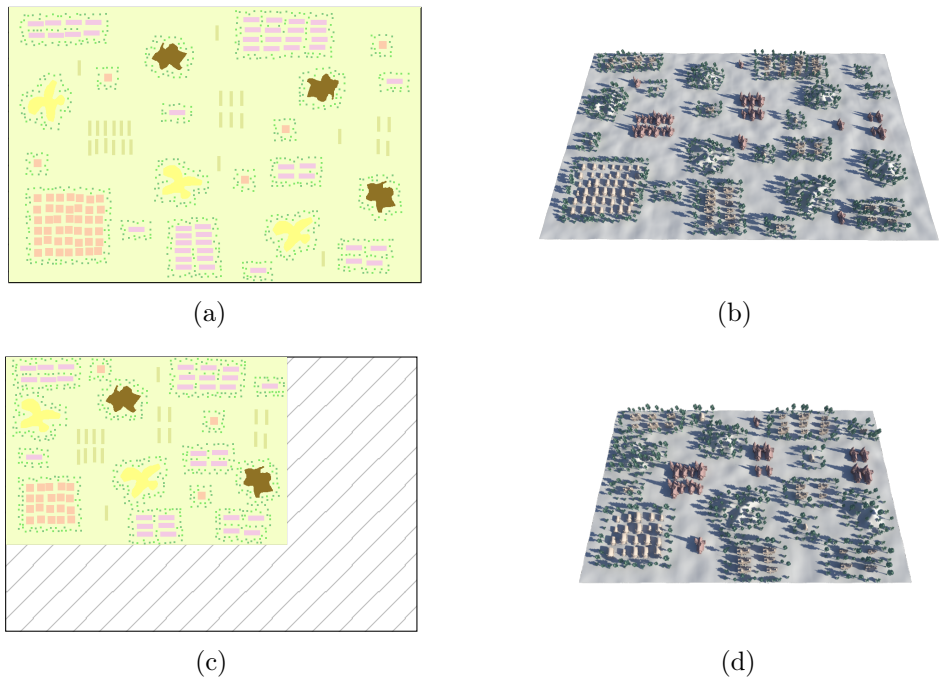
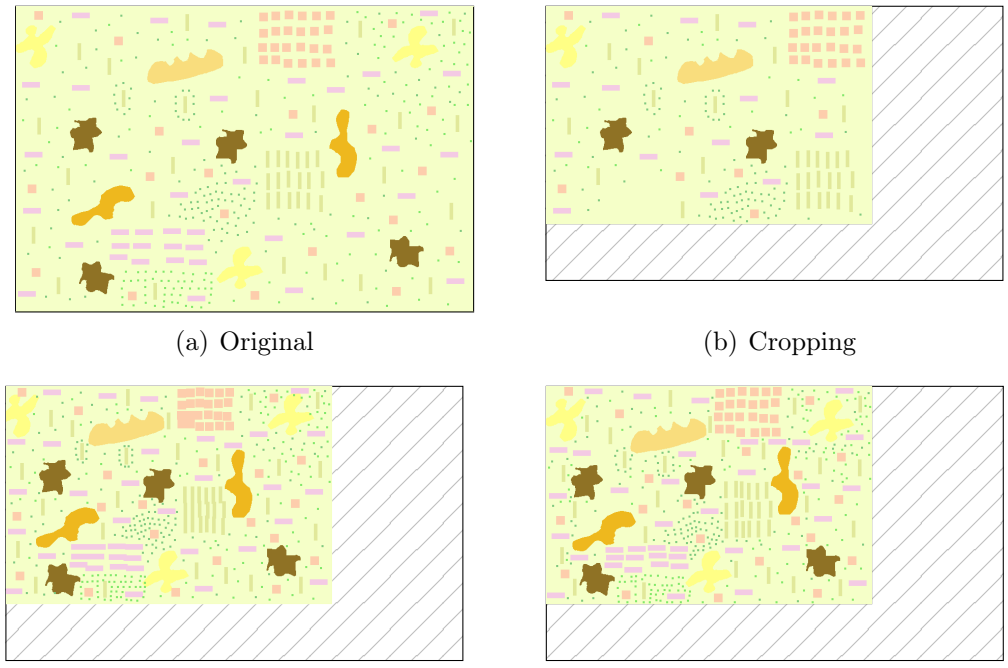


Figure 3.1: Landscape Resizing: specification model (left) and the respective 3D model (right)

produce some overlapping of objects. As well, a cropping of the model can remove some global feature of the landscape. Figure 3.2b and 3.2c illustrates these naive approaches. On the other hand, a content-aware shrinking obtains a much better resizing results (Figure 3.2d).

Inspired by the traditional seam carving approach [10], we have developed our method for dealing with vector-based landscape specification. It is important to highlight which this model is piecewise constant. Thus, our seam carving approach is simpler than the methods used in image retargeting [10] or geometry carving [9], even so, it still produces good results.

The core of our method works like a traditional seam carving. Its main functionality is to shrink or to enlarge the model by removal or by insertion of paths. Nevertheless, beyond replacing of objects, we will present how to insert and how to remove some objects in the landscape according to some criteria. Furthermore, this method can be easily extended, i.e. these criteria may be replaced by other technique of the state of



(a) Original (b) Cropping

(c) Scaling (d) Shrinking

Figure 3.2: Comparison between a simple cropping (b) or scaling (c) and our resizing approach (d)

art for spreading objects into a landscape.

The main contributions of this chapter are:

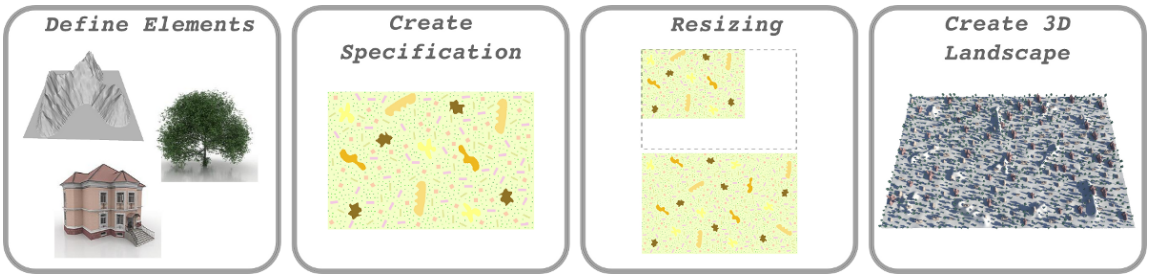
- An extendable method for shrinking and enlargement of a vector-based landscape specification model, keeping the overall appearance
- General-purpose approaches for object placement
- Strategies for keeping some properties about object distribution

This chapter has three main parts. The first one introduces the landscape resizing method. In Section 3.1, we will present an overview of the technique and, in Section 3.2, we will explain the resizing approaches. The second part, in Section 3.3, regards to the discussion about some implementation details. Finally, in Section 3.4, we will present some results and additional applications of our resizing approach.

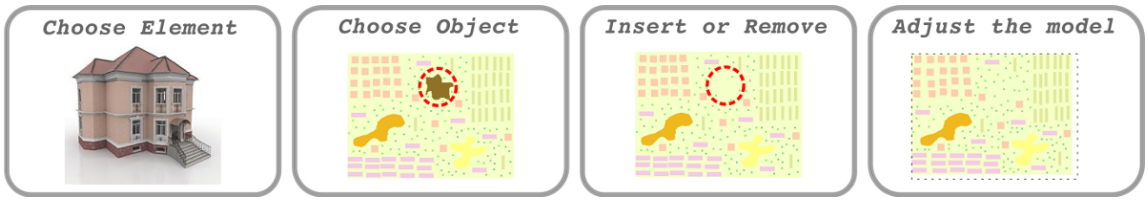
3.1 Overview

The input and output of our method is the specification of a landscape. From the specification we create a 2D map containing a set of polygons, each one relative to an object into the scene. Thus, the specification is a vector-based model very similar to a vector graphics. Beyond the 2D shape of the object, it is possible to add other specific attributes for objects, elements, or for the scene. These attributes are related to the resizing process, or to some other specificity of the application. We will mention along the text when these attributes are necessary, and how to use them. Figure 3.1 shows an example of landscape map created from a specification model. We will use primarily this representation for discuss about our method because it is able to present the entire appearance of the scene, without distraction related to visualization.

In general, a 3D landscape is created from a pre-defined vector model, created on the fly using procedural techniques, or by a combination of both approaches. Our



(a)



(b)

Figure 3.3: The pipeline of landscape creation (a) and the pipeline of our resizing approach (b)

method fits in the third category. Our input and output are vector models, and we have some procedural approaches in the resizing pipeline.

Figure 4.10 shows the overview of the landscape creation (4.10a) and highlights our resizing pipeline (4.10b). We will only discuss the third step of this pipeline, shown in Figure 4.10a. We assume that the model of each element is previously available (as discussed in Chapter 2). The second step regards to the creation of the specification model. It is an initial version of the landscape created by an user, or by a procedural method. This model is the input and output of the resizing method described in Figure 4.10b. Finally, we use the specification to create the 3D landscape in some viewer.

One type of element may be copied in different positions of the specification. For instance, some kind of a tree may be placed at many positions of the scene. Because of this, we will refer to *element* as the set of *objects*, with similar attributes (like 3D shape), and the objects are instances of elements, each one placed at different position.

Figure 4.10b shows the resizing pipeline. The enlargement and the shrinking are performed similarly. We first choose which element will be removed or inserted. After, we choose which object of this element will be removed, or where we will insert a new object. And so, we perform the insertion or removal.

These operations can create some holes in the model. Because of this, we have to adjust the entire model after resizing. This adjustment is a shifting of all objects after paths in the landscape region.

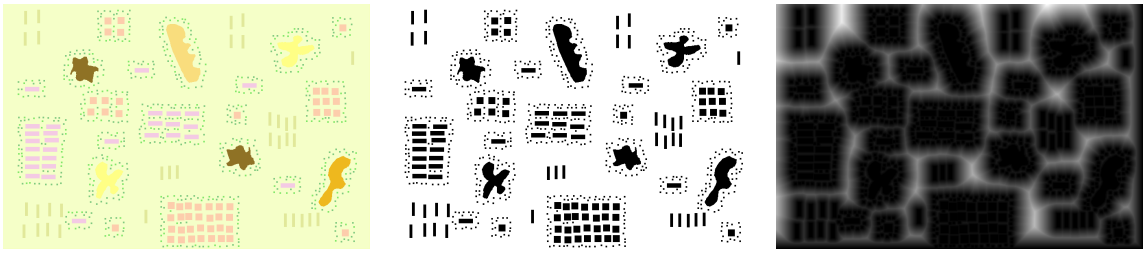
Even we are dealing with a vector model, we will perform all operations in the pixel space (an image). Most of these operations concern to the exterior area of the specification (the regions that do not belong to any area related to some object). Basically, these operations are: path creation, computation of the distance field, creation of object mask, and translation of the objects.

3.2 Landscape Resizing

In this section, we will discuss the resizing pipeline shown in Figure 4.10b. We will focus on the choice for the element will be inserted or removed, and on the need for adjustments of the model to keep the initial aesthetic.

In Section 3.3.4, we will discuss about the choices related to the objects, and we will show some approaches to choose the one will be removed, and another to choose where we can insert a new object. Nevertheless, it is possible to use more specific approaches related to some specific landscape context [1].

Both shrinking and enlargement are based in a cost function created over the scene. For creating this function, we first define a mask of all objects into the scene. After, we create a distance field of the exterior area, i.e., for each point not marked in the mask we calculate the minimum distance until those regions. For points inside of any region we define a negative cost, which the norm is greater than the biggest dimension of the landscape. The intuition of this function is to avoid the creation of a path passing for the interior of some object region. For this purpose, we privilege points far of all regions (i.e., points with big values in distance field). Figure 3.4 illustrates these elements used for creating the distance function.



(a) map (b) mask (c) distance field

Figure 3.4: Structures used for creating the cost function.

3.2.1 Shrinking of Landscapes

The shrinking process reduces the landscape dimensions while keeping consistent the overall appearance. The Algorithm 1 describes this method. It consists on repeatedly choosing for objects which will be removed, its removal, and the adapting of the model for this removal. Figure 3.5 illustrates these steps.

We are using a greedy approach to choose which object will be removed, aiming to keep the proportion the amount of objects related to each element. We first define a set C_s of candidate elements, containing objects which may be removed. Thus, we choose one object of this element. We can perform a random choice, or use a better approach discussed in Section 3.3.4. These procedures are performed in the functions *chooseElement* and *chooseObject* respectively, in Algorithm 1.

We define the set of candidate elements which can be removed as:

$$C_s = \{i; R_s(i) > (1.0 - \epsilon) \max_j R_s(j)\} \tag{3.1}$$

The function $R_s(i) = \frac{E_{curr}(i)}{E_{init}(i)}$ is the ratio of the current amount of objects of this element $E_{curr}(i)$ over the respective initial amount $E_{init}(i)$. And $\epsilon > 0$ is a small tolerance to enable the insertion of elements with values close to the maximum. This constant inserts a randomness in the element choice, but it is not mandatory.

After the choice of the object, we remove it in two steps. First, we remove the object from the respective element list (and thus, from the landscape). Nevertheless,

Algorithm 1 Shrinking:

```
1: procedure SHRINK(desiredArea)
2:     while currentArea() > desiredArea do
3:         e = chooseElement()
4:         o = chooseObject(e)
5:         removeObject(o)
6:         compactSpecification()
7:     end while
8: end procedure
```

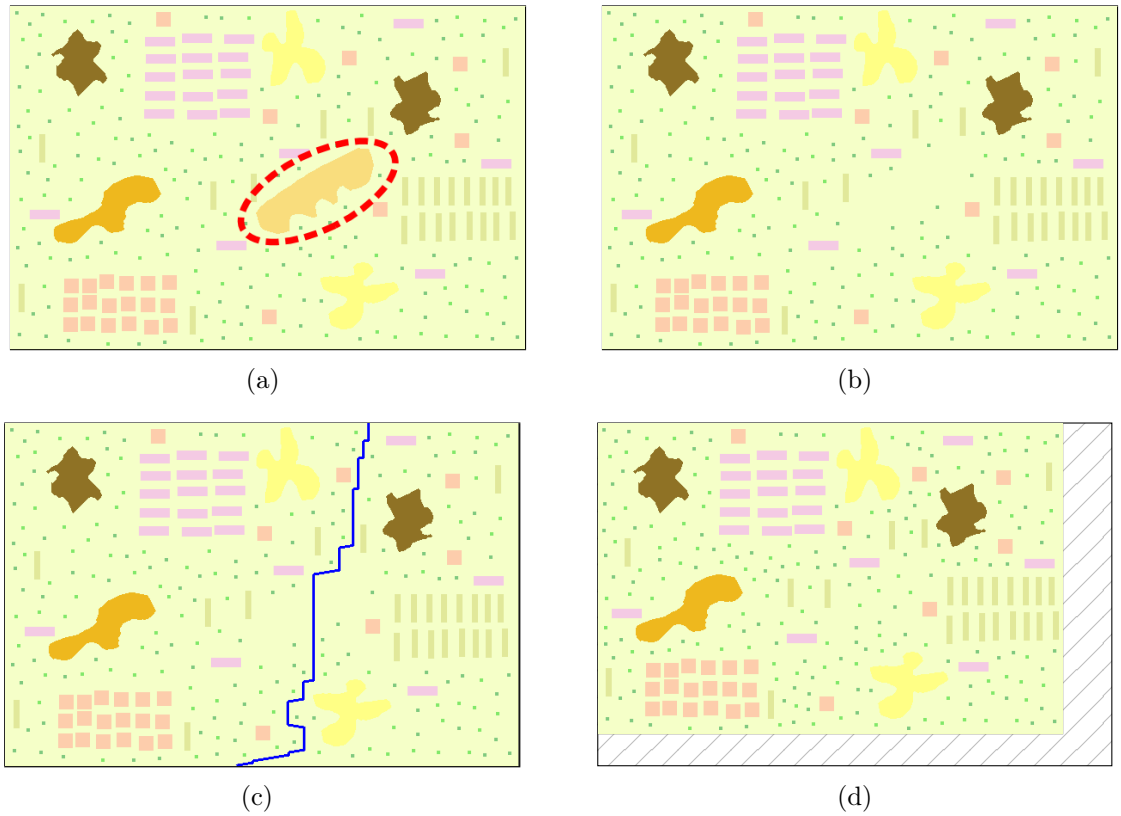


Figure 3.5: The shrinking starts from the choice for the object will be removed (a); after the choice we remove it, however it produces a hole in the model (b). After some removals of paths (c), we obtain the same initial aesthetic (d)

this removal creates a hole into the map. Thus, the second step is the adjustment of the specification to remove (or to shrink) this hole. It is performed in the function *compactSpecification* in the Algorithm 1.

The compaction of the model is the main procedure of the shrinking method. It is introduced in Algorithm 2. The main idea is while we can define a valid path (i.e. a path such that the cost is equals to zero) we remove it. To obtain a trade off between maximize the amount of path removals and a reasonable processing cost for the creation of a path, we combine an approach based in Dynamic Programming and a Greedy Algorithm.

The idea of the function *calculatePath*, described in Algorithm 3, is to use as minimum as possible a Dynamic Programming (DP) problem for defining the origin of all candidate paths, and use a greedy algorithm to create them. Because the DP procedure is an expensive step, we avoid to use it for all path creation. Once we have calculated the DP tables we can try to remove all paths containing cost equals to zero. However, after the removal of the first path, the table containing the previous point of the path is not anymore consistent. Thus, we use a Greedy Algorithm to define the path (favoring the previous point defined in the DP previous table, when it is possible). The Dynamic Programming method and the Greedy Algorithm for path creation is described in Subsection 3.3.1.

Once defined the path, its removal consists in moving all the objects after the path one step to the left, in the vertical paths case, or upward in the horizontal paths case (function *removePath*). Furthermore, it is necessary to update some structures, like DP tables, object mask, and the distance field (function *updateStructures*). They will be better described in Subsection 3.3.1, 3.3.2 and 3.3.3 respectively.

We can create paths horizontally and vertically. The choice for what type of paths will be created is based on the intention to preserve the initial aspect ratio or in order to achieve a desired aspect ratio. Furthermore, the compaction is performed while there are some paths with cost equal to zero. We do it, even though by the removal of a path we obtain a model with area smaller than the desired. But, the complete removal is necessary to keep the overall appearance.

When it is performed the DP procedure for defining all paths candidates, in

Algorithm 2 Compact Specification:

```
1: procedure COMPACTSPECIFICATION
2:     path = calculatePath()
3:     while path.isValid() do
4:         removePath(path)
5:         updateStructures(path)
6:         path = calculatePath()
7:     end while
8: end procedure
```

general, many candidates are marked, i.e., there are many possible paths with cost equals to zero. We observed that is very frequent many path candidates passing for a common region. In this way, if we remove the paths according to the sequence they appear then after all removals this region will disappear. The problem is the objects near of this region will be very close to each other, and thus, this region will be more compact than the rest of the model. To avoid this situation we can choose a candidate randomly in the candidates set (instead to follow the sequence related to position). In this way, some paths passing by the initial region will stop being a candidate after removal of paths that pass close of this region.

After all paths removal we can evaluate the quality of the result based in the predefined proportion and distribution of object amount. As previously mentioned, these rules are very dependent of the application context. In Subsection 3.3.4 we will discuss about how to choose the element, and how to evaluate if the final result is good enough.

Algorithm 3 Calculate Path:

```
1: procedure CALCULATEPATH
2:     if pathCandidates.size() == 0 then
3:         calculatePathCandidates()
4:     end if
5:     while pathCandidates.size() > 0 do
6:         idx = choosePath()
7:         path = createGreedyPath(idx)
8:         pathCandidates.remove(idx)
9:         if path.isValid() then
10:            return path
11:        end if
12:    end while
13:    return createEmptyPath()
14: end procedure
```

3.2.2 Enlargement of Landscapes

Analogously to the shrinking process, the landscape enlargement increases its dimensions while keeping the overall appearance. The Algorithm 4 describes this method. It is also performed by repeatedly choosing objects which will be inserted, and the adapting of the model to this insertion. Figure 3.6 illustrates these steps.

We also use a greedy algorithm for choosing the element such that we will add

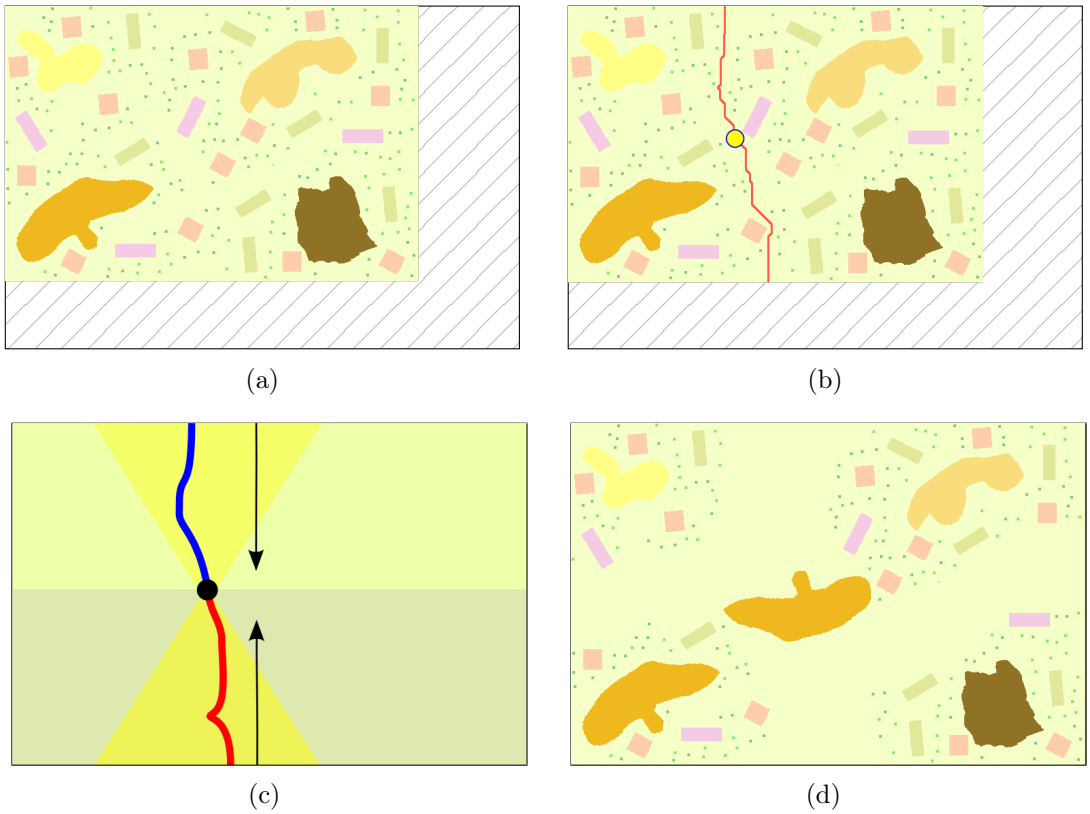


Figure 3.6: Given a model (a), we choose some object to be inserted and its position (the yellow circle in (b)). Then, it is necessary to create an empty space (to avoid overlapping) to insert the object (d). This is performed by insertion of paths passing by this point. Each path is created by solving two DP problems: one for the region above the point, and another one for the region below (c).

a new object. This choice is realized in order to keep the proportion of the amount of element's objects in the specification. In this case, the set C_e of the candidate elements which can be inserted is:

$$C_e = \{i; R_e(i) > (1.0 - \epsilon) \max_j R_e(j)\} \tag{3.2}$$

where the ratio function is $R_e(i) = R_s(i)^{-1} = \frac{E_{init}(i)}{E_{curr}(i)}$.

The function *choosePosition* defines where the object will be inserted. This choice can be global (an evaluation of the entire scene) or dependent of the element. The second case depends on the particularity of the application. Because of this, we will keep focus in the first approach. An example of a general global choice is to insert a new object of this element in the point farthest of all other objects. We obtain this information looking for the maximum value in the distance field created in the exterior regions of specification.

After choosing the element and the position of the object, we check if there is enough space to insert it. It is performed by comparing two masks: the specification mask and another similar containing only the new object (both with same dimensions). It is performed in function *intersects* on Algorithm 4.

If there is an intersection between the new object and some other, we have to adjust the model by performing some vertical or a horizontal shifts of two steps for

Algorithm 4 Increase:

```
1: procedure INCREASE(desired_area)
2:     while currentArea() < desired_area do
3:         e = chooseElements()
4:         pos = choosePosition(e)
5:         while intersects() do
6:             pos = shiftElements(pos)
7:         end while
8:         insertElement(e, pos)
9:         updateStructures()
10:    end while
11: end procedure
```

all elements after this position (by "position", we mean the center of the region), and of one step for the position of the new object, until there is no more intersection. Nevertheless, this approach creates some spurious artifacts (such as holes) and destroys some desired patterns in the scene. A strategy to avoid this breaking of patterns is to use the composition of objects (described in Subsection 3.4.3).

Another strategy to minimize the incidence of holes is to get the path more distant of all other objects (a path passing on the ridges of the distance field). But, this path has to pass for the chosen position. To achieve it, we divide the path creation in two path creation problems. The first problem calculates a path in the region below of the point (growing in a bottom-up way), and the other one calculates the path in the region above of it (growing in a top-down way), both passing by this point. The final path is a concatenation of the path created in the region below, with the inverse of the path created in the region above.

Different than the shrinking case, we are searching for paths with big cost (big amount of sum of distances). This path can pass into the object region (in spite of this region does not collaborate to the path cost). So, we set these regions as zero. In this way, the path passing by the new object position will always exists.

We can constraint the tables of DP problems (cost and neighbors) to the viable region, i.e., the region where it is possible to have a piece of a path starting from the new object position. It is a triangular region, such that each row depends on k (amount of possible neighbors of each path point). Figure 3.6c illustrates this idea. Another possibility is to use the greedy algorithm for path creation (because in this case the optimality is less sensitive than in the shrinking case).

When there is enough space for the new object, it is inserted. We repeat this procedure until the desired new size is obtained. Figure 3.6 illustrates this procedure.

To optimize the processing, we also do not recalculate all structures in all steps. The function *shiftElements* perform a shift in the map, and thus in the object mask. So, this operation depends only on the amount of objects in the scene (while the creation of the entire mask also depends on the image resolution). The distance field can be also updated in a smart way. We will describe it in Subsection 3.3.3.

3.3 Implementation Details

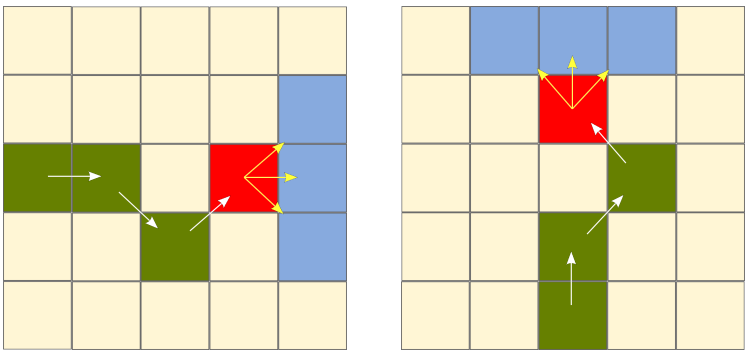
3.3.1 Path Creation

The path creation problem is an important subject for this work. Beyond the use for landscape resizing, it is also used for cutting and seaming of patches, in out Patch-based method for terrain synthesis (Chapter 4). In this section, we will briefly discuss two approaches for solving this problem.

More specifically, the problem we want to solve consists in defining a set of neighbor points (*the path*), in a regular rectangular grid R, growing in the same direction (vertically or horizontally). The choice for the points is performed according to a cost function f, such that we know the value over all points of R.

There are some ways for solving this problem, but we will keep focus in two approaches. The first is based in a Dynamic Programming problem, and the second one is based in a Greedy algorithm. The main difference between both is that the first obtains a globally optimal path (i.e., a path with the smallest cost according to f), but is computationally more expensive. The second one is cheaper, but it is based in local decision, and so, there is no guarantee about optimality.

The greedy algorithm creates the path by choosing the smallest next neighbor



(a) (b)

Figure 3.7: Path definition

point, growing horizontally (Figure 3.7a) or vertically (Figure 3.7b). The x-th vertical path will begin at point $(x, 0)$. As well, the y-th horizontal path will begin at point $(0, y)$. In each step, we have to decide the next path point, between k possibilities (k is preferably odd to keep the symmetry).

Figure 3.7 shows an example of path creation using the greedy approach for $k = 3$. In this case, to perform the horizontal growth, if the current point is (x, y) (red pixel in Figure 3.7a) the next one will be $(x + 1, y - 1)$, $(x + 1, y)$ or $(x + 1, y + 1)$ (blue pixels in Figure 3.7a). The vertical growth is analogous, as shown in Figure 3.7b.

We can optimize the path creation by the following observation: when two paths have an intersection at a common point, thus these paths will be equal from this point on. In this way, we can save a table of previous greedy decision, and the respective path index. Thus, when we get a point previously marked in this table we can copy the rest of the respective path.

Algorithm 5 Create Path using DP:

```
1:  procedure createPathDP(i, j)
2:      for j = 0; j < m; j++ do
3:          c(0, j) = f(0, j)
4:      end for
5:      for i = 1; i < n; i++ do
6:          for j = 0; j < m; j++ do
7:              c(i, j) = min(c(i − 1, j − 1), c(i − 1, j), c(i − 1, j + 1)) + f(i, j)
8:          end for
9:      end for
10:     for i = n − 1; i >= 0; i−− do
11:         for j = 0; j < m; j++ do
12:             m = argmin(c(i − 1, j − 1), c(i − 1, j), c(i − 1, j + 1))
13:             n(i, j) = m − 1
14:         end for
15:     end for
16:     idx = minimumCost(c)
17:     return definePath(idx, n)
18: end procedure
```

If the dimension of the region R is $n \times m$, then the Dynamic Programming approach is performed using three tables of the same size. The first is for describing the cost function f, the second c with the accumulated cost, and another n to determine who is the cheapest previous neighbor.

The algorithm consists in three parts. The first is the creation of the cost table, performed in a bottom-up way. The accumulated cost is the minimum cost of the previous neighbors plus the cost of the point. The second step is the creation of the previous neighbor table, in a top-down way. The previous neighbor is that of smallest accumulated cost. Once we have defined the accumulated cost, the previous neighbor is that which contains the minimum value. The previous point of (x, y) is $(x + d, y - 1)$, where $d \in \{-\frac{k}{2}, ..., \frac{k}{2}\}$, and we save the value of d in this table. Finally, the third step is a search for the minimum accumulated cost value and its respective path. Algorithm 5 shows this algorithm.

The value of the constant k is strongly dependent of the context. For instance, for image carving, as well as for cutting and seaming of terrain patches (as present in Chapter 4) the high variation of pixels implies in to choose more carefully this value (i.e., avoiding big values) because it can produce spurious artifacts (like strong discontinuities). In landscape resizing case, the model is piecewise constant, and thus we can use large values for k. The unique constraint is related to the processing cost, because larger k values imply more processing. We are taking k as an odd number close to 5% of the minimal dimension of the map.

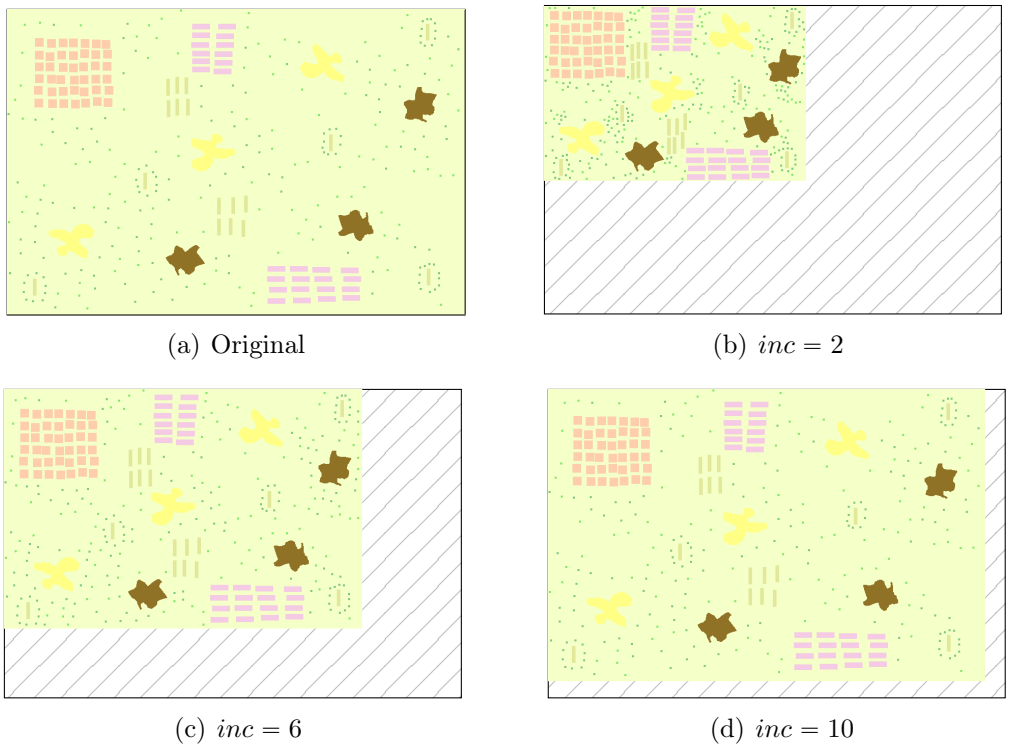
For landscape shrinking, the DP cost table is created to favor paths in the exterior region. The cost to insert a new path point is zero for points being outside of these regions and a big positive value otherwise. Thus, using DP, we calculate paths that do not intersect these regions (paths with cost equals to zero). Moreover, the update of these tables after a path removal is a shift analogous to those performed for the objects into the map, i.e., all entries after the path is translated one step backward (and after all translation the table dimension is reduced). Again, it is important to highlight that after this update these tables are not anymore consistent. However, in practice it is a good inspiration about the behavior of the path cost, and thus we can use it to guide the greedy algorithm for path creation.

3.3.2 Object Mask

All operations proposed are based on the object mask. The first idea of the mask is to define the region where the 3D model will be positioned in the landscape. Another desired effect is to avoid two objects being too close after resizing.

The cost table of the DP procedure is created from this model. The mask maps the area which contains the model in an image (with some predetermined resolution). It contains filled polygons related to the shapes of the planar region of each object. After the mask creation, we define the distance field used to define the path cost.

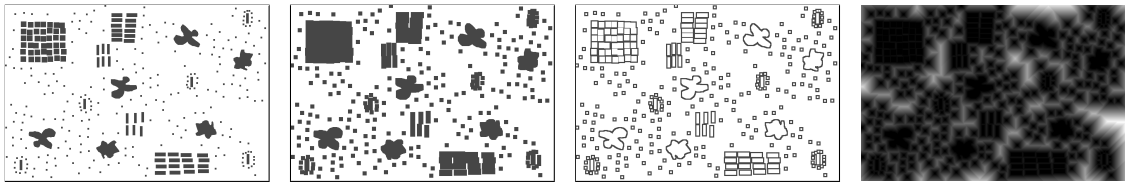
We use the mask to decide if a point is inside of some object region. This image,



(a) Original (b) $inc = 2$

(c) $inc = 6$ (d) $inc = 10$

Figure 3.8: The mask increasing avoids that two adjacent objects stay glued (big increasing big distance between adjacent objects). A comparison is shown in (b), (c) and (d).



(a) Original Mask (b) Increased Mask (c) Increased border (d) Distance Field

Figure 3.9: The implementation details related to the mask increasing and object's distance field

that can be efficiently rendered, avoids a geometric approach (in general, expensive) to perform this decision. Figure 3.9 shows an example of the use of the mask.

To avoid that the objects being glued, we scale the object region during mask creation. As a consequence, during the creation of DP cost table, we avoid the paths passing between two near objects. Of course, it will decrease the amount of possible paths, and thus, it will decrease the compactability.

To get a strip with a fixed width (in pixel space) along the boundary of the object region we opt for scale regions using the distance field. If the distance of a pixel is less than a threshold then it is included into the mask. A geometric scaling does not create this enlargement of a strip with settled width in non-convex cases.

Figure 3.8 shows three examples of different masks. Figure 3.9a shows the original model. The following images show the size reduction using mask increasing of *inc* pixels. Observe that the objects in Figure 3.8d are not glued as in Figure 3.8b. However, the size decreasing was smaller.

The update of the mask is performed by shifts analogous as described in Section 3.2. The guarantee that there is no overlapping of regions is if two regions are glued (without overlapping) then it is not possible to have a path with zero cost passing in this part. So, both regions will be in the same side of the path and thus will have the same translation, i.e. they will be moved the same amount of steps in the same direction (and thus, do not causing overlapping).

3.3.3 Distance Field

As previously mentioned, we are using distance field for creating the cost table of the Dynamic Programming procedure for path creation, and for increasing the area related to the object region. More precisely the procedure for creation of the distance field is: *Given a set of samples R of the region over which we will create the distance field, a distance function d, and a set of points S, we have to define a mapping L associating each point $p \in R$ to a nearest point of S. Hence, the distance field is determined by $d(L(p))$.* It is a general definition for this problem. For our purpose, we will use $S \subset R$ the union of all points inside of some object region and d the Euclidean distance.

The brute force approach consists in the comparison of each point of R with each point of F. On one hand, this approach creates the exact distance field, on the other hand, it has a high processing cost. Another strategy consists in creating the field by morphological operations. This method is cheaper than the previous one. But, it only creates a correct Distance Field when the distance d is the L^1 norm or L^∞. For L^2 norm (Euclidean distance) the result is an approximation. This approximation can be good enough, but using the next strategy we can get a field with a negligible error, by a reasonable processing cost.

We are creating the distance field using the algorithm introduced by Per-Erik Danielsson [73]. This method creates the distance field in a sequential algorithm, i.e., the cost of this algorithm is linear in the size of the field. It is performed in four steps where for each point $p \in R$ the nearest point of S is defined by an analysis on the neighborhood of p. These steps depend on an ordering of the points of R (by increasing x, increasing y, decreasing x, and decreasing y).

To optimize the calculation of the distance field after a path removal, we mark some points which can have changing of the distance and only recalculate these values. This marking is realized when the point $p = (x, y) \in R$ and its nearest point $L(p) = (\bar{x}, \bar{y}) \in S$ are in a different side of the path, when $d(p) < \frac{k}{2}(x - \bar{x})$ (for vertical paths), or $d(p) < \frac{k}{2}(y - \bar{y})$ (for horizontal paths).

3.3.4 Choosing Objects

Different landscape applications depend on different rules for spreading of objects. For instance, Deussen et al. [74] introduced a technique for spreading trees according to a provided DEM. Analogously, Prusinkiewicz et al. [75] present a method for generating rivers with a height map subdivision. These techniques for placing objects are completely different than a technique for defining a city layout [11], or for creating the interior of a building (designing the furniture layout) [13]. Because of this variety, we opt for context-free rules for object management that can be used for a general purpose landscape application, avoiding those more specific rules (context dependent).

An example of general criterion for inserting an object of some element is consider it has a high probability of incidence close to objects of some type of element. Figure 3.11 shows an example where objects represented by the green points were inserted closed to some kind of objects.

Algorithm 6 shows how to choose the position for a new object of an element e that is more likely to be close to the elements in the set A. It is an example of implementation for the function *choosePosition* used in Algorithm 1. This method considers a new object o of a given element e is attracted by a set of objects A of certain elements. The first three steps ensure o will be placed close enough to some point of A. We create a mask of the neighborhood of all objects in A, defining the region where o will be inserted. It guarantees that o will be placed in the right place. Finally, we choose some candidate positions (to insert o), and use the distance field inside of this neighborhood to weight the choice. Bigger distance bigger the probability

Algorithm 6 Choose Object Position

```
1: procedure CHOOSEPOSITION(e)
2:     A = getCloseElement(e)
3:     neighborhood = createNeighborhoodMask(A)
4:     candidates = chooseCandidateObjects(neighborhood)
5:     createDistaceField(neighborhood)
6:     return getObjectPosition(candidates)
7: end procedure
```

of choosing the object. This maximum distance criterion avoids big changes in the scene, because the new object will be placed in the position with more space, and thus will be necessary less or no shifts.

Beyond the maximum distance in the neighborhood of candidates we can maximize the distribution of inserted elements, the distribution of the objects of e, the distribution of objects of elements similar to e, etc. All these rules should be used in an optimization problem in function *getObjectPosition*.

The choice of which object will be removed is very important to ensure all holes in the scene will disappear after of the path removal. An example of this criterion is to keep the distribution of the removed objects well spread in the specification space. For this purpose, Algorithm 7 presents a way to get the object by combining the following criteria (similar to object insertion): get the one which the maximum distance in its neighborhood is the smallest between the candidates, keep the distribution of the objects of the element e, of the objects of elements similar to e, and of the previously removed object.

As previously mentioned, all criteria used for choosing the object position are based in an optimization problem. More precisely, these algorithms are based in the solution of the following optimization problem:

$$W(e.o(i)) = \alpha N(e.o(i)) + \beta O(e.o(i)) + \gamma E(e,i) - \delta R(e,i) \tag{3.3}$$

where N is a function of the maximum distance in the neighborhood of $e.o(i)$, O is the sum of the distance between the i-th object and the others e's objects, E is the

Algorithm 7 Choose Object for Removal

```
1: procedure CHOOSEOBJECT(e)
2:     C = getObjectCandidates(e)
3:     neighborhood = createNeighborhoodMask(C)
4:     createDistaceField(neighborhood)
5:     maxDist = chooseCandidateObjects(neighborhood)
6:     return getObjectForRemoval(e, C, maxDist)
7: end procedure
```

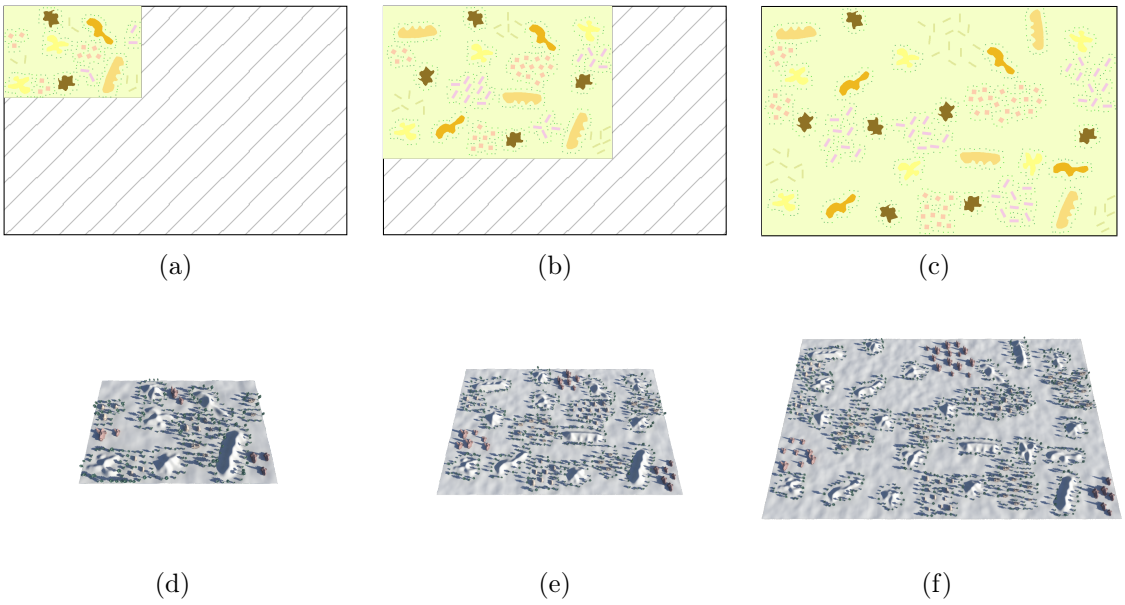


Figure 3.10: Specification Resizing: original model ((b) and (e)), shrunk version ((a) and (d)) and enlarged one ((c) and (f))

sum of the distance between this object and all other objects of elements similar to e (for instance, we can have more than one elements of the tree type), and R is the sum of the distance between this object and all other removed objects of the elements similar to e. Observe that we want to maximize the last term, so we have to subtract the weighted value of R.

The choice of the weights is empirical. We opt for the following rules:

- α is directly proportional to the log of the amount of e's objects
- β is inversely proportional to the log of the amount of e's objects
- γ is inversely proportional to the log of the amount of objects of the elements similar to e
- δ is inversely proportional to the log of the amount of removed objects of the elements similar to e

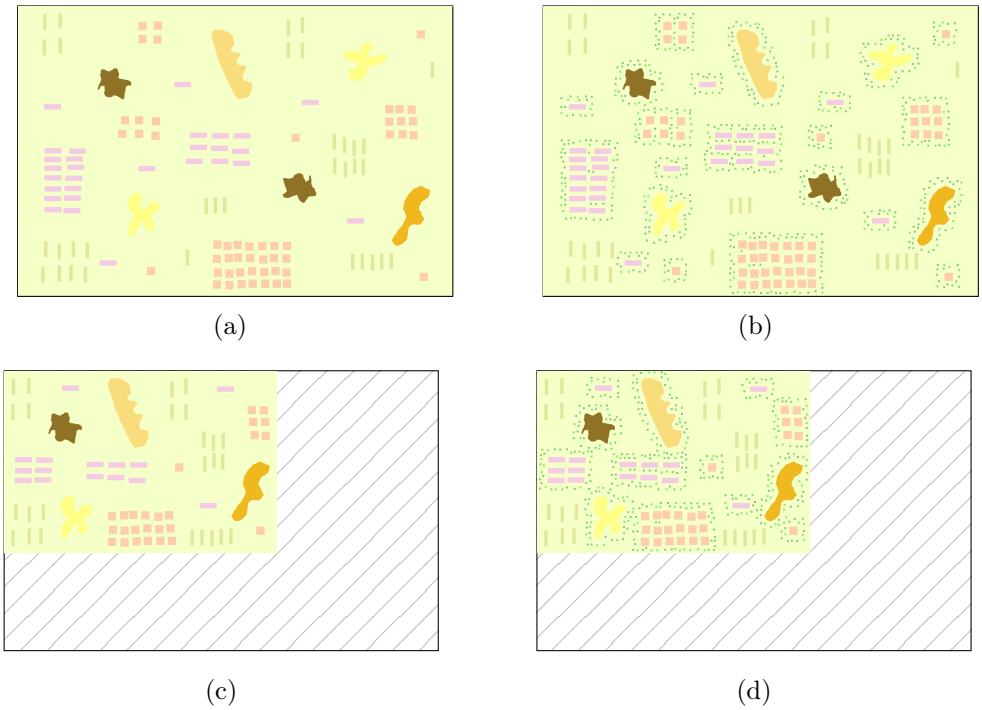


Figure 3.11: Using a basic model in the resizing, and after that, inserting detail in the scene

Finally, this approach is based on the following optimization problem:

$$o = \arg\min_{i \in C} W(e.o(i)) \tag{3.4}$$

The evaluated neighborhood can be a circle centered in the object position, or a region created using the increased mask approach (present in Subsection 3.3.2). If all objects of an element have the same shape then the first approach is fair, i.e. the external neighborhood area is the same for all objects. But, in the case of some element has objects with different shapes it is better to use the second approach, when the external area of each object will be a belt produced using the region increasing.

Our technique is easily extendable to add other insertion and removal criterion. For this purpose, it is only necessary to change the algorithms present in this subsection. We present ways to keep distribution of objects and use an attraction metaphor. Other possible general rules that can be easily used are clusters of objects, repulsion of

objects (i.e., objects of an element is highly unlikely close of objects of some elements), grids, or the distribution of objects inside of of a region, or along a curve. These approaches will be discussed in Chapter 5.

An important requirement for the resizing approach is to keep the overall appearance. For this purpose, we have to use rules for inserting and removal of objects coherent to those used for creating the specification. Because the specification can be created manually, the definition of which criteria were used can be extremely hard.

The criteria introduced in this section are easy for performing the insertion and removal, and also, it is simple to define rules for evaluating if a model is respecting these criteria. Beyond keeping the proportion of objects of each element, we can easily measure the attraction and distribution. Unfortunately, it is not the case for more sophisticated rules. It can be very hard to evaluate if the final result is coherent with what was being expected.

For this purpose, it is necessary to use Machine Learning techniques for creating a database of possible rules present in a scene and try to match patterns present in the specification with these rules. To improve this analysis, the knowledge of relation between objects in some landscapes can be very useful. Fortunately, this kind of information is more and more available. In Chapter 5 we will improve this discussion.

	Shrinked	***Original***	***Enlarged***
Resolution	525 x 352	900 x 600	1350 x 869
Aspect Ratio	1.49	1.5	1.55
Max distance	18	18	18
mean distance	13.3099	12.1935	12.4643
Free forms	0.882353	1.11628	1.46341
Rectangles	11.8487	12.186	11.9512
Points	87.2689	86.6977	86.5854

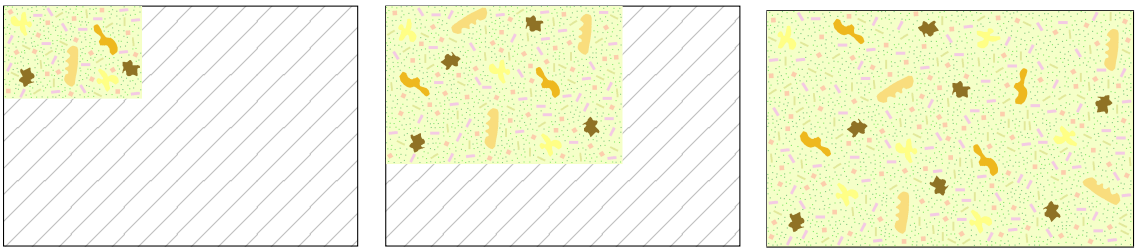
Table 1: Resizing metrics

3.4 Results

In this section, we will present some results obtained using our resizing approaches, some small variations of our methods, and how to evaluate the quality of the results. This problem is a constant in procedural modeling approaches for landscapes creation (as well as, for many computer graphics techniques). The visual aspect is the most relevant criterion for most techniques. Some approaches consider mathematical or statistical models related to some natural phenomena. We will also keep focus in metrics to reinforce the visual quality of our results.

Figure 3.12 shows an example where the original model (3.12b) was shrunk (3.12a) and enlarged (3.12c). Table 1 shows how the values related to the metric previously cited were changed by the resizing. The last three rows mean the percentage of amount of the objects of each type of element.

In some landscape models, some objects are more important than others (for instance, a hill is more important than a common tree). The result shown in Figure 3.11 was created considering this fact. Figure 3.11a shows a base specification, with the most important elements. The final specification (Figure 3.11b) is an extended version, under which we add other less important objects. The resizing was performed in the base model and kept the most important structures of the scene (the shrunk version shown in Figure 3.11c). As well, after the resizing, we add other objects to compose the final scene (3.11d).



(a) shrunk version (b) original model (c) enlarged model

Figure 3.12: Example of resizing

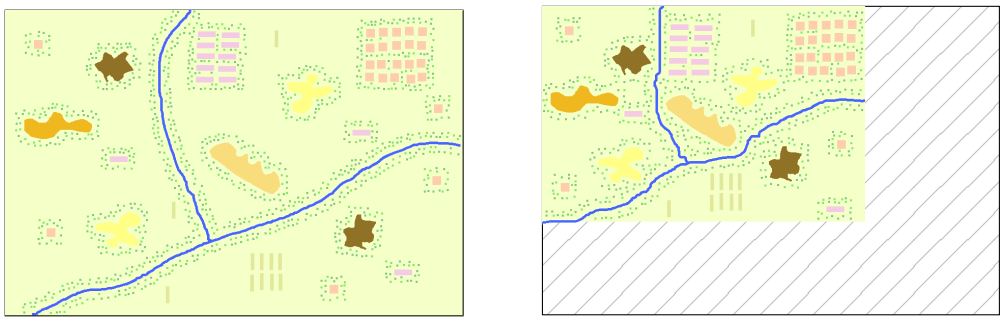
3.4.1 Resizing of Curves

In general, most elements in a landscape can be represented either by a polygon, a free form closed curve or a point. On the other hand, the best representation for some elements can be a curve (or a set of curves). It happens, for instance, with the representation of rivers and roads.

This kind of element is not well fitted in our resizing approach. The main reason is, it is very easy to a have a landscape where there are no paths that do not cross the curve (e.g. Figure 3.13). Hence, we do not add these curves in the mask, and thus, we can have paths crossing the curve. On the other hand, because our curve representation is the entire set of points of the curve (i.e., we do not use a control point representation) the shifts are performed at the same way.

This straightforward way avoids that the changes realized in the curves (shifts) create overlapping with other objects. Nevertheless, it is possible to create some artifacts in the curve.

An improvement for this method is to represent the curve as a spline curve and perform the changes of the control point position carefully to avoid overlapping. Another possibility is to keep a low resolution curve and perform midpoint displacement subdivision [49] to add detail in the river (but again, performing it carefully to avoid overlapping).



(a) (b)

Figure 3.13: Objects represented by curves

3.4.2 Retargeting of Landscapes

In general, landscapes are presented by a rectangular model. In the case of planets, the model is represented by a set of rectangular charts (an atlas). However, it is possible to work with different kinds of shapes. For instance, it happens when we are working with a limited model, such as, an island or a biome.

Our method can be easily adapted to deal with non-rectangular models. We deal with this situation using the following two approaches:

- Using a parameterization to transform the rectangular domain in the desired shape (and its inverse function)
- Using a closed free form curve to define the boundary of the landscape

In the first situation, we only have to change the space where the operations are performed, i.e., the creation of the the increased masks and the distance field is performed in the model transformed by the parameterization. Nevertheless, the paths are defined in the rectangular domain. For this purpose, we create the DP cost table by evaluating the transformed points, i.e., each point of the discretization of the domain is transformed, and then, the cost is evaluated in the image of this function.

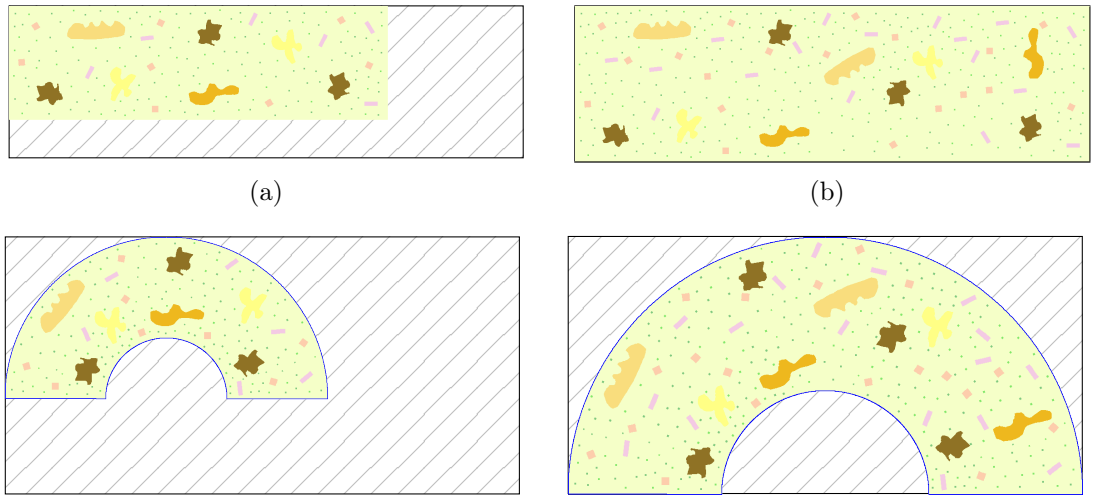
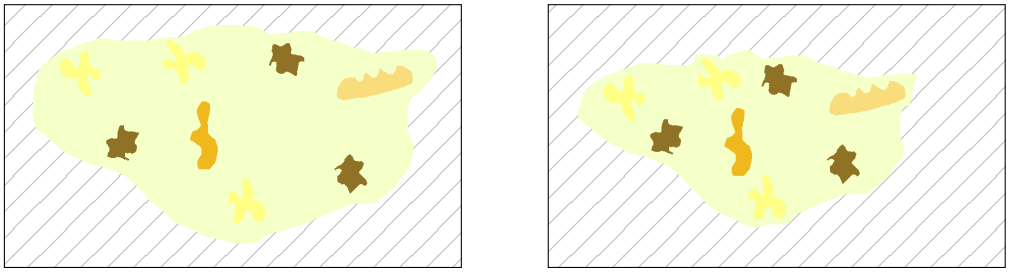
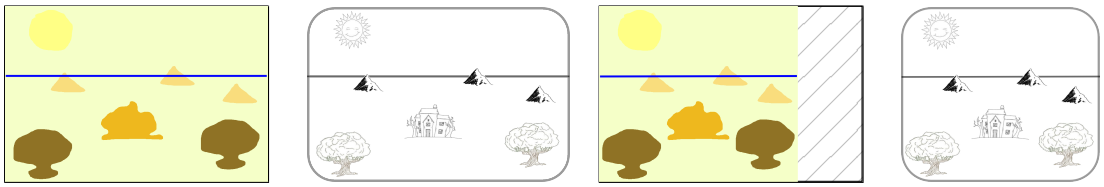


Figure 3.14: Landscape retargeting using parameterization



(a) (b)

Figure 3.15: Resizing a free form landscape



(a) (b) (c) (d)

Figure 3.16: Resizing a Vector Image

Figure 3.14 shows an example of the original model, (3.14a) transformed by a given parameterization (3.14c), being enlarged (3.14b and 3.14d) using this approach.

In the case of a free form curve defining the landscape boundary the difference is to constraint all operations to the specified region. It is performed using an auxiliary mask to define this region. For this purpose, the DP problem is adapted: the points in the area outside of the landscape region receive a penalty (much smaller than the value used in the interior of the object's region). We perform the resizing while we can create paths mainly passing by the landscape region. Figure 3.15 shows an example of the shrinking of a free form area.

Furthermore, a future work is to adapt our method for changing some feature of the model and adapt the rest of the specification to this changing (similar to the approach of Yeh [16]). For instance, if we want to move an object, or a set of objects, in the scene. Another possibility is we define a region and define how we want to deform it. Thus, the method has to move all objects to achieve this retargeting.

3.4.3 Composed Objects

There are some contexts where we can handle with composed objects, i.e., objects composed by a set of other objects. For example, objects composing a pattern in a scene can be considered only one object (whereof the mask is their convex hull) and, in this case, the resizing method will break this pattern.

This kind of objects can represent more sophisticated elements, as cities, chain of mountains, forests, etc. It introduces a concept of multi-layer in our approach (better discussed in Chapter 5).

We can manage with this kind of objects using a common mask that involves the region which contains them and translate them all together during the resizing. On the other hand, the retargeting of free form landscapes opens the possibility to resize this kind of object. Figure 3.15 shows an example of resizing of the composed objects (sets of rectangles placed in an almost regular grid).

3.4.4 Resizing of Vector Images

Our landscape resizing technique is easily adapted for the vector-images context. The unique modification is to change the planar shape used in masks for some component of the image.

In the image case, it is possible to extend this method to add some semantic in shifts operations. For instance, to limit the movement of some objects to some part of the image (e.g. some objects can be constrained to stay below of the skyline, or decrease the size of the shape when it get closer of the skyline).

Figure 3.16 shows an example where we shrunk an initial vector image. In this example, we only decreased the width to keep the depth sensation. After the deformation using the same approach used for landscape resizing, we used the specification model to create an image. We replaced each mask for a small image creating the final model.

Chapter 4

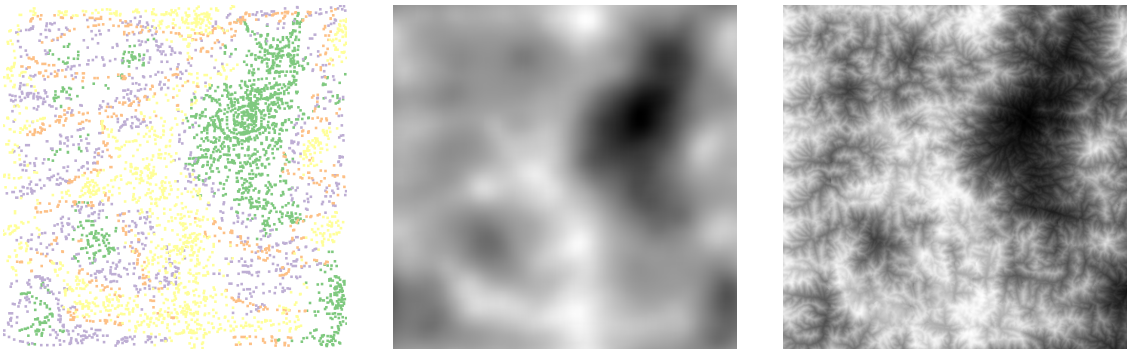
Data-driven Terrain Synthesis

The basic idea of data-driven approaches is to create a new model T from an exemplar (or a set of exemplars) E. These types of methods have the ability of creating structures resembling those present on E, and thus, they are more robust to synthesize non-homogeneous textures.

As previously mentioned, there is a trivial association between images and the terrain representation by Digital Elevation Model (DEM). In general, terrains are locally homogeneous, and they contain some global structures that are matter of priority for being synthesized (i.e., terrains with structured textures).

Data-driven approaches are more and more used because its ability of reproducing structures from the exemplar, even we do not have an explicit representation for them. In our case, the proposed methods are able of generating a terrain with macro structures (like big landforms, and the definition of the coarse height average) specified by our control approach, the meso structures (like the ridges, valleys, and big erosion phenomena) by the Markovian approach, and the micro structured (details) by coping pixels from exemplar.

The focus of our methods is in the generation of macro features by the use of control structures. The meso and micro structures are obtained by intrinsic characteristics of data-driven synthesis. Moreover, we use a valley descriptor for improving the matching of this feature, a vertical translation of the chosen patch to reduce the overlapping error, and a specific interpolation to remove seam discontinuities.



(a) (b) (c)

Figure 4.1: Specification (a) and generated guide (b); and the synthesized DEM (c)

The main contributions of this chapter are:

- An approach for data-driven synthesis control based on two strcutures: guide and categorization map. We show how to create and use the guide, with the coarse structures of the target; and how to create an exemplar categorization and the map for associating regions of the target with these classes;
- A criterion for validation of the exemplars according to the guide, and a rule for choosing a minimum set of exemplars, into a large data set of exemplars, able to cover the guide;
- A patch-based algorithm including, besides of control, a new optimization structure for patch choice (for accelerating the processing, and improve the feature matching), and a new patch insertion approach (reducing the overlapping error and removing the seam discontinuities), both based on the geometric nature of the data;
- A pixel-based algorithm including, besides of control of macro features, a strategy for graph creation

In Section 4.1, we will present the Markovian approach for data-driven synthesis. After, we will present our control approach (section 4.2). In Section 4.3, we will show our patch-based method [25], and in Section 4.4, our pixel-based approach.

4.1 Markov Model for Data-driven Synthesis

A Markov Chain is a stochastic process with discrete states whose each one only depends on the immediately previous. That is, if $S = \{s_1, s_2, ...\}$ is the space of states, and the firsts states of a chain were $X_1, ..., X_n$, then X_{n+1} has to satisfy the Markov Property:

$$P(X_{n+1} = s|X_1, ..., X_n) = P(X_{n+1} = s|X_n) \tag{4.1}$$

We can describe entirely a Markov Chain by the space of states (a countable set, possibly finite), an initial state and a transition matrix (whereof $a_{ij} = P(X_{n+1} = s_j|X_n = s_i)$).

The Markov Chain provides us a way to model changing in an unidimensional process. The Markov Random Fields (MRF) is an extension of the Markov Chain. It is a random process, satisfying the Markov Property, described by an undirected graph $G = (V, E)$.

In the context of image processing, as well as, of terrain synthesis, the vertices from the set V of the graph G represent a pixel of an image, or a set of neighbor pixels. For data-driven synthesis purpose, we will describe these vertices as a matrix $X = [X_{ij}]$. The edges E depend on the application (in our case, we will consider as the neighbors all eight pixels adjacent to the site). Furthermore, each site X_{ij} has a neighborhood N_{ij} satisfying a symmetric relation ($S_{kl} \in N_{ij} \iff S_{ij} \in N_{kl}$) and $X_{ij} \notin N_{ij}$.

For synthesis purpose, a region where we intend to place a new pixel (or a set of pixels) is determined by an evaluation of its neighborhood N_{ij} (or eventually, $N_{ij} \bigcup \{X_{ij}\}$). This representation defines a *Local Conditional Probability Density Function* (LCPDF). Hence, it determines $P(X_{ij} = x|X_\lambda = x_\lambda, \forall\lambda \in N_{ij})$ (and thus, the LCPDF defines the transition matrix of the MRF).

The LCPDF is based on a 2D histogram, such that, the elements are associated to a value of a possible state and a possible neighborhood. If F is the histogram of an image I, the LCPDF is given by:

$$P(X_{ij}|N_{ij}) = \frac{F(X_{ij}, N_{ij})}{\sum_{x_{kl} \in I} F(x_{kl}, N_{ij})} \tag{4.2}$$

Unfortunately, the amount of data of F is very large (despite of many values are zero) and the cost of the search can be prohibitively high. This approach was tested for grayscale homogeneous image producing quite good results [76].

An improvement for this approach, named FRAME (Filters, Random Fields and Maximum Entropy), was proposed by Zhu et al. [77]. The authors proposed to synthesize a texture by using of a special kind of MRF: the Gibbs Random Fields. In this model the probability distribution is strictly positive. It uses a filter bank, based on histograms, and Maximum Entropy principle for defining the marginal distribution of the probability distribution used on Gibbs sampling.

A workaround for synthesizing a texture using MRF avoiding problem of creating the probability function was proposed by Efros and Leung [34]. In this approach, instead of creating the probability distribution based on a histogram of all possible values of a pixel and its neighbors, the authors decide which pixel will be chosen comparing its neighborhood with neighborhoods on a provided exemplar.

This approach has been used in many different works, including pixel-based and patch-based approaches. The decrease of processing and memory cost enabled more sophisticated methods able to handle non-homogeneous textures. We will present two data-driven terrain synthesis methods following this approach.

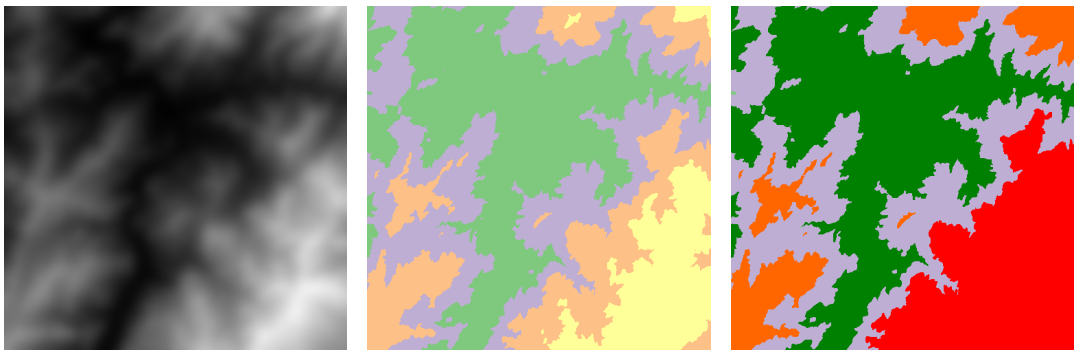
The analysis of neighborhood guarantees a coherence of the data. Of course, an exemplar has not all types of neighborhood, and thus the best choice for X_{n+1} is the element with the smallest matching error (or an element from a set of candidates with small error). Unfortunately, more structured the exemplar more incidence of mismatching. In other words, for unstructured and stochastic textures, this approach is almost perfect, but for structured and/or organized textures it is necessary to improve the matching and to guide the synthesis for better results.

4.2 Synthesis Control

The methods we will introduce in this chapter can be controlled by similar structures. They control primarily the synthesis of macro features of the synthesized model. The meso and micro features are obtained by the data-driven Markov-based approach. In this section, we will explain how to use and create the control structures.

The guide is the most important structure for controlling of macro features. It contains the macro structures desired (e.g. where there are mountains, canyons, big plateaus, etc.). It also contains an implicit categorization of the exemplars, since it has a wide range of height, and so each exemplars, in general, can be only used in some parts of the model. Also, it is a continuous model, and so it helps the method to guarantee a continuous synthesis. Furthermore, it guarantees that the patch choice respects a coherent flow of the data.

We can create the guide by filtering an existing terrain model (when we want the same macro features, but with different details). Another possibility is to use sketches to specify the macro structures [61]. Furthermore, we can use brushes or seeds based on features extracted from the exemplars. Figure 4.1 shows an example of a guide created from a set of seeds related to four different classes of heights. Another possibility is to use some rules used in our landscape specification method to spread



(a) Guide (b) Four height clusters (c) Map with three categories

Figure 4.2: Example of the categorization map

some coarse landform in the scene, and use this model as the guide.

The map of categories can be based on a subdivision of the guide by clustering of heights. The clustering process creates a set of closed regions associated to the respective class. This map can be created by setting which regions will be associated with each category of exemplars. Of course, this association must be enough to cover the entire guide, and must have intersection to be able of synthesizing the transition areas. Furthermore, in the association of regions and categories we can relate regions of a same height to different categories, and we can relate a category to more than one level of heights (since the exemplars have the entire range of heights). Figure 4.2 shows an example of map a created with these features.

Another important strategy to control some features of the model is the choice of an adequate set of exemplars. If the amount of the data is too small there is not enough information to perform an adequate synthesis. However, a huge amount of redundant data increases significantly the processing cost.

In general, in texture synthesis work, the user provides the exemplars. In our approach, it is extremely important that the exemplars be able to cover the guide. So, to avoid many trial and errors, we perform an input validation. It is performed by checking if all blocks in the guide are related to a minimum amount of patches from exemplars. In general, we ask for this minimum be greater than five hundred candidates, to have many alternatives for the next criteria.

In the cases where the synthesis is performed without a guide, we have to guarantee that the exemplars are compatible: i.e. all exemplars have an enough amount of patches in the same range of height of patches in other exemplars. It is necessary that the synthesis uses patches from all exemplars without big discontinuities. When we use a guide, an adequate coverture has this validation implicitly. The compatibility of patches is also performed by the comparison of blocks of piece (analogous to the comparison with the guide).

Figure 4.3 illustrates an execution of our greedy algorithm for choosing a small set of exemplars, between a large data base, able of cover the guide (Algorithm 8). Our algorithm is based in a provided connected *graph*, where the vertices are the exemplars, and two vertices are connected by an edge if they are compatible. For

each vertex, we define a value related to how many patches of this exemplar are compatible with the guide.

Firstly, we get the sub graph of exemplars, such that, each one minimally covers the guide (green points). So, we find for the exemplar I (blue point), into the graph, with the biggest coverture (regarding to the guide). This exemplar is appended to the set of chosen exemplars E (light blue points), and all other exemplars compatible with I is added to a candidate list C (purple points). While the guide is not covered by E, we find in C by the exemplar with the biggest coverture and repeat the initial steps.

The constraint of the graph to exemplars with a minimum coverture regarding to the guide does not produce a connected subgraph. In cases when there is no more possible compatible exemplars to add to C, we remove all elements of E of the *subgraph* and repeat the algorithm.

Furthermore, we cluster the patches from exemplars according to the blocks of average of pieces, and create the categorization according to this clustering. With this division the user can create the categorization map.

Algorithm 8 Choosing Exemplars:

```
1: procedure CHOOSEEXEMPLARS(graph, guide)
2:     subgraph = cover(guide)
3:     I = biggestCoverture(graph)
4:     E = I
5:     C = compatibleExemplars(I)
6:     while !E.cover(guide) do
7:         I = biggestCoverture(C)
8:         E = E ∪ {I}
9:         C = C ∪ compatibleExemplars(I, subgraph)
10:    end while
11: end procedure
```

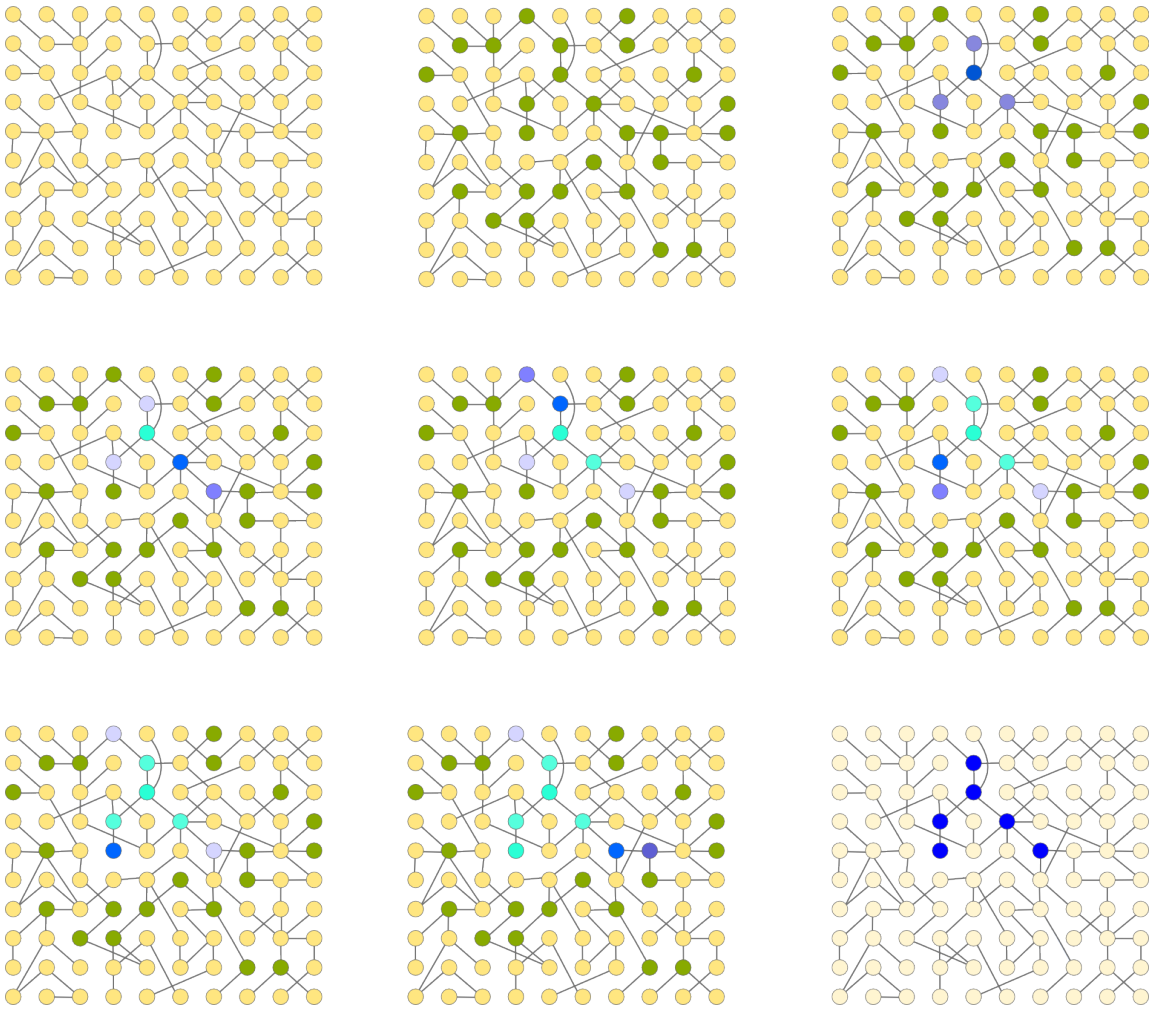
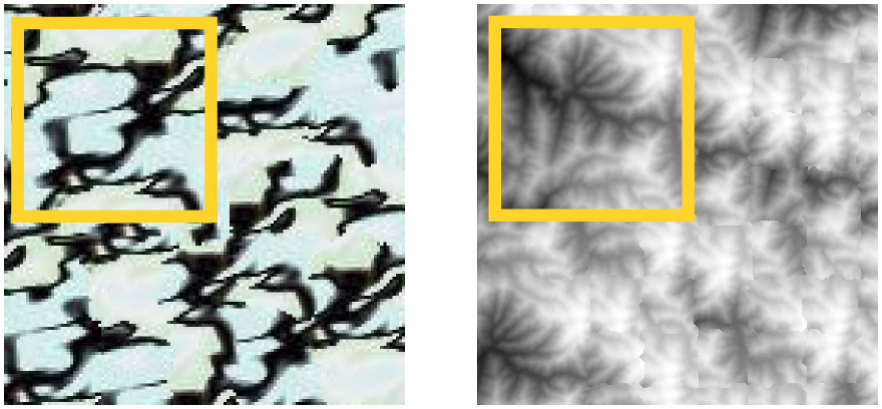


Figure 4.3: Example execution of the algorithm for choosing the exemplars which cover a provided guide.

4.3 Patch-based Terrain Synthesis

This method is based on a sequential choice and insertion of patches. Figure 4.5 shows the pipeline of our method. Its essence is the creation of a grid of overlapped regions onto the target, and for each part of this grid to find a patch well fitted to the pre-synthesized parts. This search is performed by analyzing each possible patch of the exemplars, based on some rules.

We will present a patch-based technique for terrain synthesis [25]. This approach is inspired in classical patch-based texture synthesis techniques, above all, the Image Quilting [17] and the Super-Resolution Images [20]. However, when the Image Quilting method is directly applied to generate a terrain, the produced result is not as good as those generated for textures. Figure 4.4 shows an example of this comparison. The overlapped patches are not well fitted and the global topographic features (such as mountains and valleys) have a non natural distribution. It is because the terrain exemplar is a non-stationary model, and so, it does not have information enough to create a new adequate model. Thus, our patch-based approach aims to add some constraints for dealing with the specificities of terrains to obtain better results, and



(a) (b)

Figure 4.4: Results generated using Image Quilting method. The model were generated from the exemplar (the yellow top-left box).

use a bigger input (to add to the input variations of the exemplars, e.g rotations; use more than one exemplar and/or larger exemplars).

Our input is a set of exemplars and the control models. The first control model is the guide: a DEM, with the same resolution of the target model, containing the desired low frequency. The second is the categorization map: a mask splitted in some regions, and associating each region with a class of exemplars (they are pre-categorized according to some feature).

The synthesis of each patch begins by definition of candidates to be inserted, that is, all possible patches compatible to the respective region of the target (relative to some rule). The rules for candidates definition will be discussed on Subsection 4.3.1. The initial candidates set has a large number of possible patches, chosen with high level criteria (able to choose the candidates by a low cost comparison) aiming to match the coarse features of the patches and the guide.

After applying these high level criteria for defining the candidates, we will choose the patches with a good matching of features into the overlapping region, by the use of low level criteria, i.e., rules for performing a more detailed comparison (and thus more expensive). Furthermore, it is necessary to define the cut to split the pre-synthesized part from the chosen patch (presented in Subsection 4.3.2), or to use another way for blending the overlapping region (presented in Subsection 4.3.3).

We will also talk, on Subsection 4.3.2, about the classical cutting and pasting approach for patch insertion, and the blending into the overlapping region to remove seam discontinuities, applied after the patch insertion. Furthermore, because we know that our data is a terrain, we can perform some small vertical translation on the patches, to improve the matching, before the insertion. We will present this approach on Subsection 4.3.3.1.

Finally, on Subsection 4.3.4, we will talk about the processing optimizations. We will describe the acceleration structures, created in the preprocessing step, and how they optimize the patch choice.

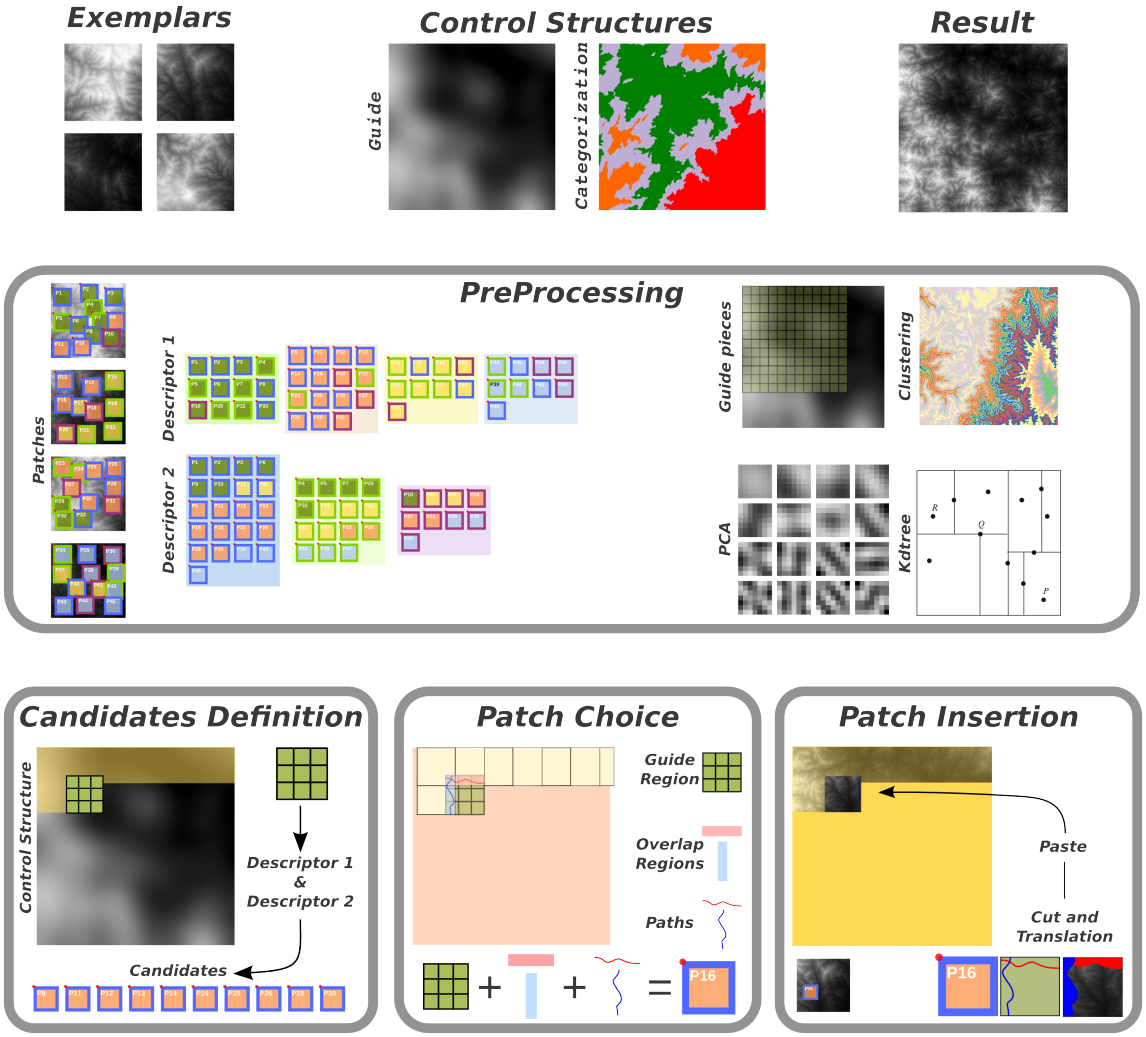


Figure 4.5: The method begins by the input definition: exemplars, and control structures. Following, in the synthesis step, we choose a patch for each part of the grid. The result is a terrain model, with the predefined size and features.

4.3.1 Definition of the Patch Candidates

The choice for which patch will be inserted in the target model is divided in two steps named *candidates definition* and *patch choice*. The first step concerns to define a set of patches well fitted to the control structures. The second step searches for the best patch, between the candidates previously defined, according to the matching of meso and micro features. In this section we will focus on the first step. The patch choice is based in an optimization problem, that will be present in Subsection 4.3.2.

In general, in patch-based approaches we search for the best one by comparison of pixels in the overlapped region between the pre-synthesized part and each possible patch in exemplars. However, depending on the amount of input data (for instance, if we are using many large exemplars), the brute force approach is very slow (or even, prohibitive).

To improve the processing, we are proposing to use besides the known acceleration strategies (like use PCA and Nearest Neighbor), a combination of high level criteria for defining a subset of candidates by a small cost, with low level criteria for performing an adequate matching.

We name the criteria used for defining the candidates by *high level*, when they base their decision by information related to coarse features, and thus we can choose many candidates (or discard a big amount of data) by a low cost processing. The criteria used for patch choice are named *low level*, because they perform a finner analysis for matching meso features and details.

The evaluation of high level criteria is performed by a *chain of processing*, i.e., a serialization of successive removal of patches from the candidates set. This processing refers to an implementation to obtain the intersection of all high-level criteria.

The high level criteria are able of choosing quickly which patches are well fitted to the control by the use of some acceleration structures. These structures are created based on some descriptor D that represents a patch by a tuple of k numbers (where k is much smaller than the patch size). These structures are *Hash Tables* H whose keys are k-tuples obtained by clustering the representation of patches given by D. Indeed, many patches have the same description by D, thus we define the set of candidates

by comparison of only few k-tuples.

For defining the candidates related to the position p in the target, we evaluate the same position in the control structures, according to some descriptor. We have a descriptor D_i, and the respective H_i, for each control structure. During the synthesis, the region being synthesized is also represented by D_i and it is compared to the keys of H_i to determine which clusters are closer (possibly more than one). Hence, we obtain the candidates resembling to the control, according to the feature described by D_i, without analyzing individually all possible patches.

The first criterion is the *categorization*. This descriptor is based in an association of each region in the categorization map, with one of the classes of exemplars. It defines as candidates the patches in the exemplars of the respective category. Each category is represented by an integer number, and so, during the synthesis we will only compare this small representation of the data. Furthermore, it is important to highlight that the categorization defines as candidates a very big amount of patches. Because of this, it is necessary to use other rules more constrained. Moreover, when the categorization map is not provided, we can assume we are using a trivial categorization: all exemplars belong to the same category, and the map is constant.

The second criterion is the *guidance*. It is based on the coarse scale of the data. We can compare the average height of the guide region with the one of the patch, but it is a poor comparison, inasmuch as extremely different distributions of heights can have the same average. To improve the comparison, we divide this region in a block of $N \times N$ pieces, and we compare the average of each piece. These pieces are created using a box filter, and we opt for $N = 3$. In our tests, bigger values for N have caused over-constrained matching (few candidates). This descriptor can be achieved by downsampling the guide and the exemplars. Another structure is related to a more compact description of the overlapping regions, that is, to clusterize the data in the region to obtain a smaller representation. We tested the clustering using the PCA representation (described on Subsection 4.3.4) and the average of blocks. The PCA is a global representation, while the blocks is a better representation of the local features. Thus, the second approach provided a better cut definition (so, we choose it).

4.3.2 Patch Choice

The candidates are patches well globally fitted to the desired macro features. They are chosen according to the provided control structures. But, we also have to guarantee that meso (valleys) and micro (details) features are also being well matched. For this purpose, we will use low level criteria (and thus, more expensive tests).

This choice is achieved by comparison of the overlapping region matching O, and the cost of insertion I. It is performed by a solution of an optimization problem based on the following the energy function M:

$$M(p, c) = \alpha O(p, c) + \beta I(p, c) + \gamma V(p, c) \tag{4.3}$$

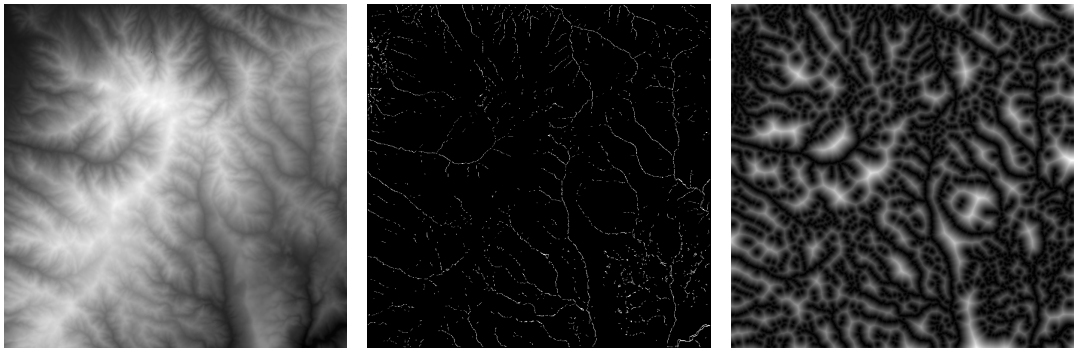
where α, β and γ are given parameters. The function O is the Euclidean distance of the chunk of pixels (with a pre-defined size) from the point p into T and c into E. The function I depends on the insertion approach (it will be discussed in the next section). And the function V compares the distances of the valleys descriptor. The patch $\bar{c}_p \in C$ which will be inserted in the position p is the solution of the following problem:

$$\bar{c}_p = \texttt{argmin}\ M(p, c), \texttt{ subject to } c \in C \tag{4.4}$$

4.3.2.1 Valley Descriptor

The *Valley Descriptor* is responsible for the improvement of the matching of meso features (the valley). We set a point as a valley if it is a local minimum in some direction (we evaluate it in vertical, horizontal and in both diagonals). We calculate the valleys of all exemplars, and also calculate a distance field of this marked points. Figure 4.6 shows an example of valleys extracted from a provided terrain, and the respective distance field.

Analogously, after each patch insertion, we update the target valley by evaluating the generating terrain with this descriptor. Thus, the function V is the local comparison of the distance field of the target with the distance field of the valleys of the patch candidate.



(a) Terrain (b) Valleys (c) Distance Field

Figure 4.6: Valleys Descriptor

Despite of the best valley matching can worsen the overlapping matching, it improves the visual quality of the model, because it glues features that are very important for the understanding of the model. Figure 4.7 shows a comparison of two example of patch insertion: one by overlapping matching and another by overlapping and valley matching. The second case is visually much better because the harmony of terrain features. For improving the transition between patches, we will show in the next section some strategies for patch insertion.

4.3.3 Patch Insertion

The classical approach for patch insertion consists in dividing the overlapping region in an area of the pre-synthesized part and the new patch by defining a cut in this region and paste the cut patch in the respective area [17]. In this subsection we will present how to insert a new patch by using of this cut and paste strategy, or by blend this transition. In particular, to improve the first approach, we will show how to perform a vertical translation in the new patch.

If the value of O, in equation 4.3, is close to zero, then both data in the overlapping regions are very close. In this case, a simple greedy algorithm for defining the cut will be good enough. Otherwise, it is better to use a more accurate method, like the

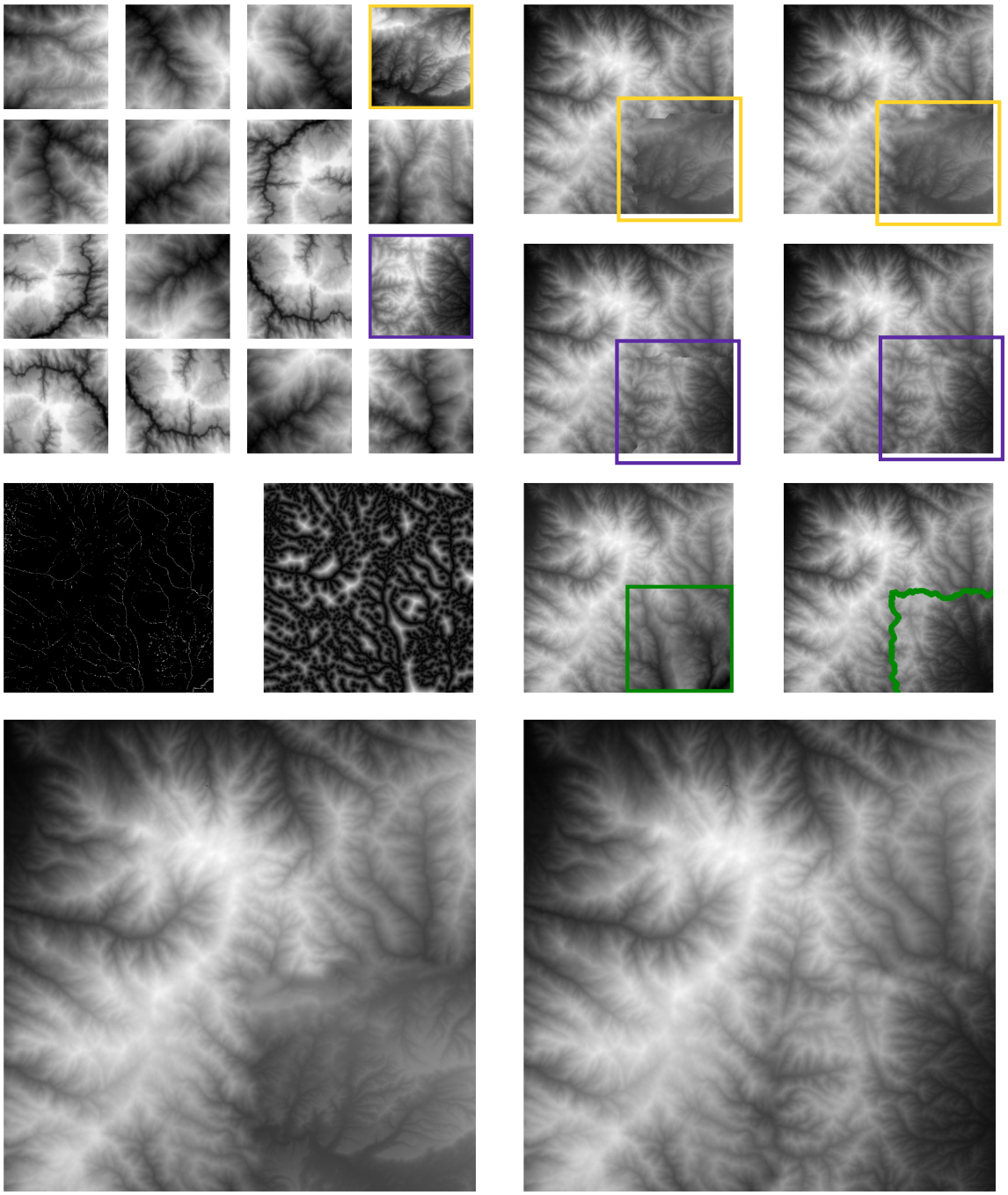


Figure 4.7: The yellow square shows an example of patch chosen by overlapping matching while the purple shows an example of valley matching. In both cases, we present an insertion with cut and paste and interpolation by tubular interpolation.

one using Dynamic Programming present in Chapter 3.

In general, the priority for the overlapping matching can harm the cutting. The choice of big β, in equation 4.3, favors a better overlapping matching. However, when this matching is not close enough to zero, the cutting can contain big discontinuities (an spurious artifact in the target model).

Another option for the patch choice is to apply a blending of the transition area using Poisson Blending [19], or using an interpolation to smooth the transition. In both cases the calculation of the path can be avoided (I is a null function). Because the search for a good overlapping matching, a straightforward interpolation for blending the transition of patches produces good results.

Another solution is to define a tubular neighborhood along the cut and perform the interpolation inside of this region. It is a linear interpolation parameterized by the distance from the cut. When we are using the Valley Descriptor, this insertion approach create an almost imperceptible transition between the patches.

Figure 4.8 shows examples of patch insertion using cut and paste, interpolation on the overlapping region, and tubular interpolation. The use of tubular interpolation present the better visual result, but it increases the processing, because it is necessary to define the cut, calculate the tubular region using a distance field in this region, and perform the interpolation (which is more expensive than the paste insertion). Despite of the processing cost, we believe it is the best option.

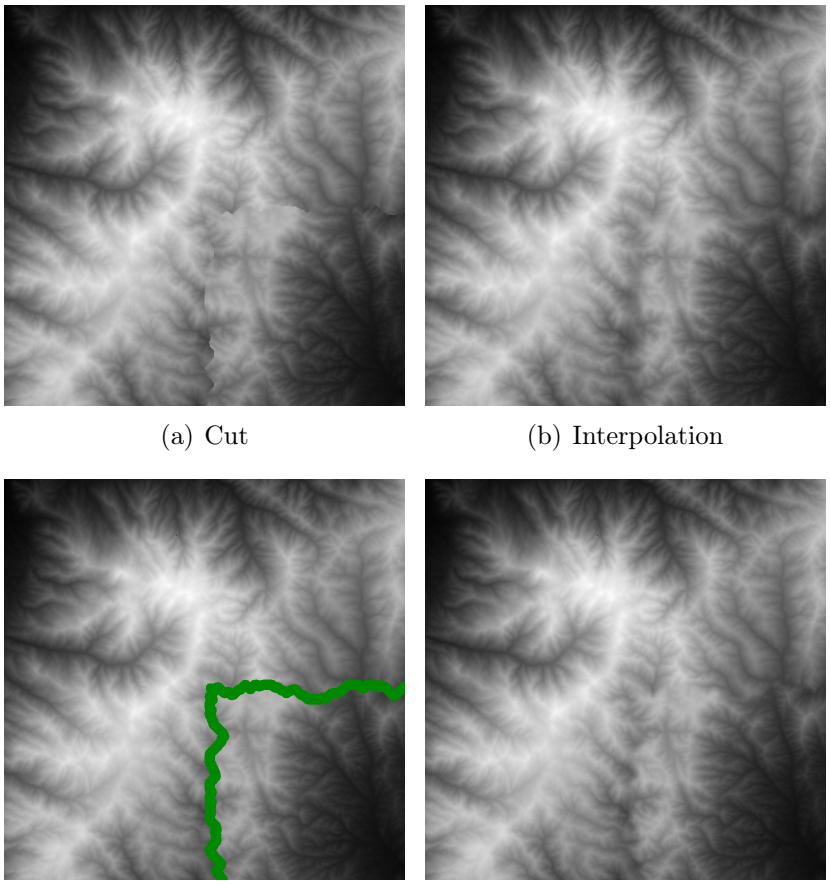
Another strategy for improving the patch insertion, guaranteeing an adequate overlapping matching, is to apply a vertical translation of the patch to reduce the overlapping matching error (i.e., to add a value for the height of all pixels in the patch). The calculation of the optimal translation will be explained in the next subsection.

Another strategy very promising is the approach proposed by Nealen and Alexa [46]. They introduced a metric for evaluating points in the overlapping area, and those with big mismatching are marked for a being regenerated using the pixel-based approach. A future work for our research is to find a metric better fitted to the terrain characteristics (take advantage of its geometric nature, and some natural features).

4.3.3.1 Vertical Translation

Some works apply a patch deformation to improve the matching [18]. However, we avoided this kind of deformation because it could create unnatural artifacts (unnatural in the sense, forms that are impossible to exist in the nature).

However, we can take advantage of the geometric nature of the data and apply some rigid transformation into the patch to reduce the seam error. The patch representation is a descriptor invariant to translations in the domain. Furthermore, we compare the region in the target with all similar candidates in the exemplar, and



(a) Cut (b) Interpolation

(c) Tubular region (d) Tubular Interpolation

Figure 4.8: Interpolation approaches

so all possible translations of the patch over the target have been implicitly performed. Moreover, we can perform a translation in the *height* direction.

So, let p the position in the target where we will insert the patch, and c the patch candidate (in some exemplar). The height translation, that will minimize the distance between both patches, can be achieved by solving the following optimization problem:

$$\bar{h} = \min_{h \in \mathbb{R}} \|B(T,p) - V(B(E,c),h)\|^2 \tag{4.5}$$

where $B(A,x)$ is the function which returns the block (with a predefined size) of A originated in the position x, and $V(A,h)$ the height translation of all points of the DEM (or the block) A by the scalar h.

It is a convex unconstrained optimization problem. So, the solution of this problem is given by:

$$\bar{h} = \frac{1}{nm} \sum_{i=0}^{n} \sum_{j=0}^{m} \Big(B(T,p)_{i,j} - B(E,c)_{i,j} \Big) \tag{4.6}$$

Another advantage of the vertical translation is that we can create a model with a height range bigger than the input range. Of course, it is not interesting to increase a lot this range, because natural phenomena depends on the height. But, small translations are not in contradiction with nature. For the same reason, even in the synthesis height range, we will not accept big translations, i.e. if $|h| > \lambda$ the translation will be not considered. However, the choice of patches according to the guide guarantees that the translation of the chosen patch is small.

4.3.4 Optimizations

Besides of the clustering of the data according to the categorization, we also clusterize them according to the guidance (descriptor for matching patches according to the guide), and according the overlapping regions (vertical and horizontal). All of these acceleration structures are created in the preprocessing stage.

We perform the clustering using the k-means algorithm. The choice of k aims to avoid small clusters, otherwise we could have small candidate sets (and then, an

inefficient patch choice).

Another optimization regards the overlapping region comparison. We reduce their dimension by use the of Principal Component Analysis - PCA [21], and find the best matching by the Nearest Neighbor method [20]. The PCA basis and kd-tree used in the second method is also created in the preprocessing stage.

Finally, in the optimization performed in the patch choice step, when we are using cutting, the most expensive calculation refers to the path creation. So, we can solve this problem in two steps. The first one keeps the candidates close to the optimal one defined by O. And, the second step calculates the path for all remainder candidates and pick the best. In this case, we only perform the most expensive step for few candidates.

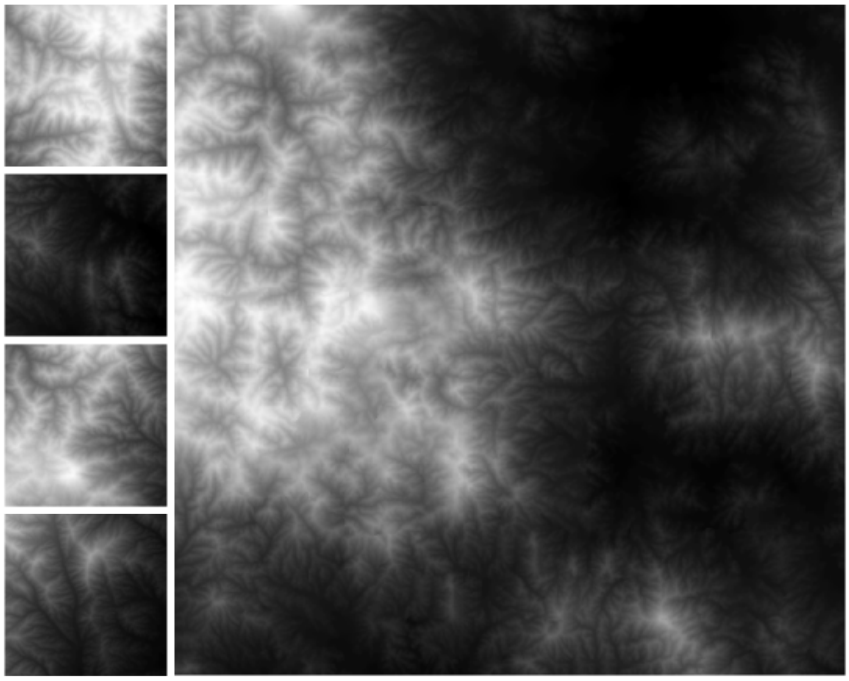


Figure 4.9: A terrain synthesized using our approach

4.4 Pixel-based Terrain Synthesis

The first idea for texture synthesis using a Markovian approach was based in pixels, instead of patches. Many works present techniques for creating a texture by matching the neighborhood of a given pixel in target texture with all neighborhoods in the exemplar. This approach is very efficient for generating stationary or quasi-stationary models. However, using additional control models it is possible to generate some structured and organized textures.

Lefebvre et al. [21] introduced a method very robust, by combining many techniques previously introduced, and providing novelties about parallelization, improving the randomness control, and introducing the Gaussian Stack, a new data structure used for matching neighborhoods in multiscale.

An advance for this method was proposed by Han et al. [22]. They create a texture from a graph of exemplars with one or more samples, and a relation about scale transition. This approach improves the ability of creating structured and organized textures, with heterogeneous patterns. The organization is achieved by coarse scale samples, and the structures are created by finner scales in exemplars.

In our work, we will use the method introduced by Han et al. [22], adding a control by a guide, and present how to create the graph of exemplars. This approach also take advantage of the geometric nature of the data (DEM), because of this, it is not trivial to extend it for colored images. Nevertheless, we can create simultaneously the DEM and the texture of a model applying different metrics for different channels of data.

In our method, the control of macro structures is performed by analyzing the guide, instead of a random process over coarse scale of the graph. The meso structures are generated by matching of pixel neighborhood, and the details is given by copying the exemplar pixels.

In Subsection 4.4.1, we will present an overview about this pixel-based method. After, we will present how we improve the control by the use of a guide, and how to take advantage of the geometric nature of the terrain data.

4.4.1 The Algorithm

Figure 4.10 shows the pipeline of our method. We begin by specifying the input data: the set of samples (RGB+DEM), the graph (which can be a single exemplar), the jittering of each level, and the control structures. The input is then processed to create auxiliary data structures, such as the Gaussian stacks [21]. Finally, we perform the synthesis by creating the output from the coarsest resolution to the finest.

The basis of this algorithm was proposed by Wei and Levoy [36]. In that work, they synthesize a texture in multirresolution. They begin by initializing the coarsest level with random colors collected from the input exemplar. For each new resolution level, the method applies upsampling followed by a correction phase.

The correction step consists in searching into the input exemplar for the closest neighborhood, by some given metric, to the neighborhood of the pixel in the texture level being synthesized. This choice may not be unique, since many neighborhoods in the input can share the same distance to the texture neighborhood being corrected. In this case, we can choose randomly or base our decision on some other criterion. Besides that, we can choose not only from the set of closest neighborhoods, but from a slight larger set of the closer neighborhoods. This change introduces more variability and can produce interesting results.

Our multirresolution representation of the exemplar is the Gaussian Stack [21]. If the input is a square patch with side 2^L then the stack has $L + 1$ levels. The finest level ($l = L$) is equals to the input, and the coarsest level ($l = 0$) contains only a constant value. All intermediary levels are representations of the input at different resolutions or levels of detail. Generally speaking, level l has half the amount of

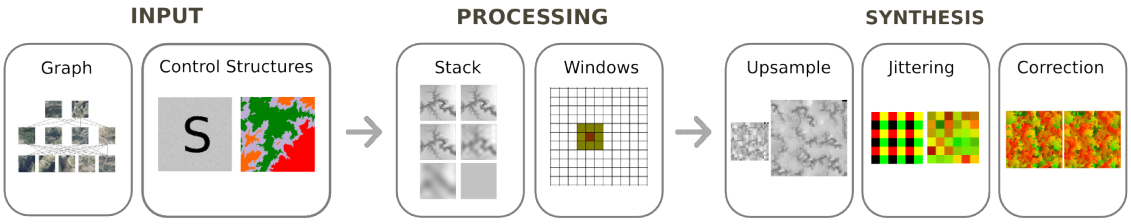


Figure 4.10: Pipeline of our method

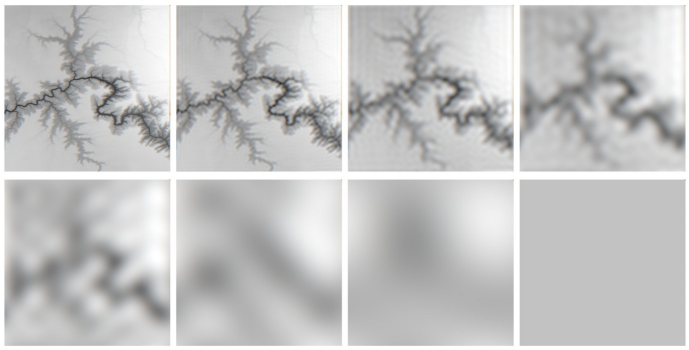


Figure 4.11: The Gaussian stack of an input exemplar of size 128×128, with 8 levels.

frequency information than level $l + 1$, that is, level l is level $l + 1$ convolved with a low-pass filter. In our filter, we first transform the data for the frequency domain (by the use of a Fast Fourier Transform - FFT), after we cut frequencies greater than $\frac{1}{2^l}H$, where H is the highest frequency, and finally, we go back to the space domain by the inverse of FFT. Figure 4.11 shows an exemplar stack.

The core of the method, according to Lefebvre and Hoppe [21], consists in the following three steps: (1) upsample the texture level; (2) jittering the coordinates, and (3) correct the pixels. The correction step may be performed more than once. This is not a convergent process. Because of this, it is not clear how many times it should be repeated for optimal results. For the generated examples shown in this paper, we have performed two correction steps.

4.4.1.1 Neighborhoods

The neighborhood is a $k \times k$ window around of the pixel. We choose k odd in order to have the evaluated pixel at the center of the window. The neighborhood matching is made using an Euclidean norm in vectors of dimension $k \times k \times d$, where d is the number of channels of the data. This is a naive norm, since it does not take in account the specific domain of the problem. Lasram and Lefebvre [18] use a metric comparing coordinates and colors for textures. Nevertheless, we believe that it is possible to find a better metric to match neighborhoods and to evaluate the quality of the results, by taking advantage of the geometric nature of terrains (it is a future work).

Once the neighborhood size is defined, we can improve the matching performance reducing its dimensionality using PCA [21]. We select the eigenvectors that represent the directions containing at least 95% of the energy of the input.

To the best of our knowledge, in all texture synthesis works the window size is an arbitrary parameter. If this parameter is small, the neighborhoods do not capture structural informations of the exemplar, and the matching is inefficient. If it is too large, the matching process is too expensive, and the result can not show a desired variability. We calculate the optimal size of the window as the one that minimizes the ratio between the PCA dimension and the neighborhood dimension ($k \times k \times d$). In this section, the generated terrains were created from exemplars which size is 128×128 or 256×256. In these cases, we found for the optimal window size with $k \in \{3, 5, 7, 9\}$.

In many examples of texture synthesis works, the authors use toroidal exemplars to avoid the creation of spurious artifacts. These artifacts are created due to discontinuity on the borders of the exemplar. However, this is not an alternative for terrain synthesis, because the type of exemplars we use do not have this characteristic. To alleviate this problem, we use, in the same spirit as Lefebvre and Hoppe [21], larger exemplars, employing only the data inside a predefined safety margin.

We have created some terrains using the presented approach. Figure 4.14 shows one of the results. The first line shows the exemplars and the generated result. The second line shows the rendering of a piece of this terrain. On left we are showing only geometry and in right a complete rendering.

Figure 4.12: Guide: (left) exemplar, (middle) sketch-based guide, and (right) gluing approach.

4.4.2 Control

The graph of exemplars provides a control of features by selecting exemplars and their scale relations. Furthermore, for improving the control, we can use the guide, similar to our approach presented for patch-based terrain synthesis. We represent the guide by the same way we represent a terrain model, using a DEM. Furthermore, it is important to ensure that the scale of guide is the same of the terrain.

The graph provides an implicit categorization. But, it is not easy to explicitly control it during the graph creation. Nevertheless, we can also use the categorization map for explicitly defining the choice of which exemplars can be used in which parts. To use this categorization, we have to create a kd-tree for each class, and perform the neighborhood matching in the specific structure.

The macro structures of the model is also controlled by the guide. In the processing stage, the guide is decomposed in a multirresolution pyramid (Laplacian Pyramid). For each level of the synthesis, we add the respective resolution information from the Laplacian pyramid. Following, we apply the correction. This way we maintain the multirresolution approach, but we introduce a new step in the synthesis pipeline. This process offer a control of the synthesized result, ensuring that the macro structure of

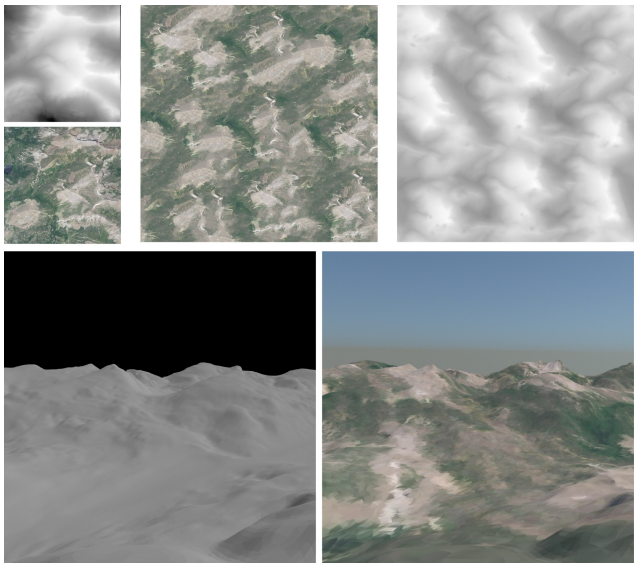


Figure 4.13: Synthesized Terrain.

the guide will appear in the synthesized terrain.

Besides of the strategies presented on Section 4.2, we also can create a guide by gluing features of the exemplar, possibly warping them to fit the desired output. Figure 4.12 shows some results created with guide generated by brushes and another created by gluing patches from the exemplar.

The neighborhood representation is invariant to translations. In order to successfully execute the guided synthesis, we need to be able to create neighborhoods that do not exist in the input, unless we apply some sort of transformation, such as rotations, on the input. For that reason, the translation invariance is not general enough for the matching process.

To simulate a representation that is invariant to rotation, we have added rotated versions of the exemplars (we use 4 rotated versions). It has aggregated more informations for the correction step, and improved the quality of the results. This is not a solution. We still do not have a representation invariant to rotations. A future work for this research is to search for a better neighborhood representation.

4.4.3 The Graph Creation

Han et al. [22] assume the graph of exemplars as an input of the method. However, this structure is a sensitive data of this approach (i.e., if the graph has some inconsistence, the resulted model can be very weird). Furthermore, its creation is a non trivial task. In this section, we will introduce an approach for creating a graph of exemplars from a big set of samples.

The input of the graph creation is a set of exemplars divided in different scales. We have to guarantee the following three properties:

- each resolution of the guide has to be covered by the exemplars on the respective level
- the range of the heights of a level l has to contain the range of the level $l-1$
- the exemplars of each level have to be compatible

The first property is achieved by the union of the interval of height of all exemplars in each level. This union has to contain the interval of height of the respective level of the guide. The second property is analogous, but the union has to contain the interval of the previous level.

The third property is based in the *compatibility* property. We say two exemplars are compatible when the amount of pixels in the intersection of height range is greater than a given parameter λ. Using this property, we can create a graph of compatibility, where two vertices (exemplars) are connected by an edge when they are compatible. Hence, we say a set of exemplars is compatible when the the graph of compatibility is connected.

In this way, from a large data set of exemplars candidates divided in resolution levels, we can randomly choose a set of exemplars and test if they satisfies these properties. Until these properties are not satisfied, we can search for another exemplar able of improving the current state, in the sense of satisfying the properties. As well as, we do not want to have a set of exemplars very large, so if two exemplars are highly compatible (i.e., their range of heights is very similar), thus we remove one of them (rather, the one with the smallest range). Although, this approach is not enough to get an optimal set of exemplars satisfying those properties. As a future work, we intend to use an algorithm for defining an optimal click of the graph (or close of the optimal) satisfying those properties.

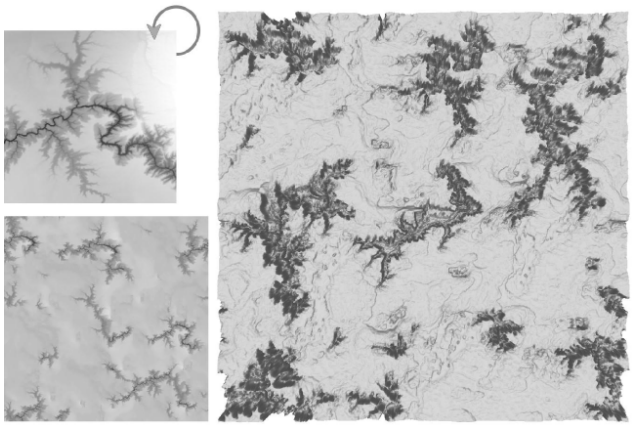


Figure 4.14: An example of a graph with an exemplar with an auto similarity

Chapter 5

High Level Methods

The approaches for modeling of landscape elements are quite well developed [1]. There are good techniques for creating trees, roads, isolated hills, lakes, cities, rivers, buildings, etc. Nevertheless, the creation of a landscape by an appropriate combination of different type of models was still not properly explored.

We believe that the next generation of these techniques will be based on a high level specification, containing the main desired features for guiding the synthesis. The challenge for this change of paradigm is the creation of adequate representations.

The need for high-level methods is due to the huge amount of work required to create a large landscape. We are considering *high-level* methods those which provide a smart control for creation or editing low level representations. In other words, these methods create an interface between the user and the techniques, allowing the creation of an entire landscape from a general specification (defining the main features of the model). They wrap specific techniques and provide for the user a more natural control, by smart operations.

In this chapter, we intend to discuss the direction we believe the landscape modeling is going to, to argue about which known tools can be used for the development of these methods, and present some examples of improvement for this area. Furthermore, we will mention how our methods can improve the achievement of this goal.

5.1 High Level Tools

The intuitive control is one of the most desired characteristic for a high-level approach. A widely used intuitive tool is the *sketch* [19, 60, 64, 61]. In the landscape context, we can use sketches for specification of rivers, mountain ridges, valleys, silhouettes and base contour of hills, trends of objects placement, etc. It is a very good tool for describing main features of the object, and to control some editing operations.

Other high level tool are the *semantic rules*, that is, operations able of managing low level representations in a smart way. An example of this approach was proposed by Nealen et al. [78]. The authors introduced a sketch-based technique for mesh editing. From a trace drawn in a plane placed over the object (specifying the changes in the object silhouette) the mesh is deformed (the vertices in the neighborhood of the sketch are moved) to fit on the desired silhouette. It is an example of high level method, because they have a semantic rule for controlling the changing of the position of a set of vertices (low level representation) based on an intuitive control (sketching the new silhouette).

Sorkine et. al [79] present other example for mesh editing. They deform an object by defining the region of interest and using a smart handle for defining interactively the manipulation. This technique is based in the control of the Laplacian Operator from an easy and intuitive handle.

Another example of high level technique for general purpose geometric modeling is the Beautification [80]. In this work, freehand drawings are interpreted as straight line segments based on geometric constraint, such as, connectivity, parallelism, perpendicularity, congruence, symmetry and interval equality. Instead of determine (precisely) the position of control points, the user can sketch lines and the system will interpret it and show all possible segments which fit to this input. This work is an example of another high level tool: *suggestions*.

The Beautification idea is based on the biggest challenge for high level techniques: the system has to predict what the user wants from a simple input. Naturally, this prediction can produce some mistakes. Thus, the high level representations must guarantee that the main desired features are properly defined, and thus the generated

model really contains what is mostly expected (avoiding big mistakes).

Classical methods are controlled by non-intuitive parameters (for example, the fractal dimension [49], noise functions [30], diffusion parameters [64], etc.). The changing of control approach was a big issue in the past, and fortunately we currently have techniques providing a human-friendly manner for modeling control [19, 53].

These rules are associated with procedural methods. Many procedural methods are based on non-intuitive parameters, but once the parameters are chosen, they can be smartly used to fill certain area.

In Chapter 3, we mentioned the use of some specific rules related to the application the landscape specification will be used. Some examples of high level approach for defining these rules are:

- Create a grid of objects in a pre-defined area (specified by a sketch)
- Create a cluster of objects in a pre-defined area (specified by a sketch)
- place objects over a sketch
- place objects close to a sketch

The last three approaches can spread uniformly the elements in the respective area, or to use some distribution function (for example, placing objects more concentrated in some regions than in others). Furthermore, we can add a constraint (deterministic or probabilistic) to avoid two objects being very close.

Despite of the simplicities of these rules, it is not trivial to create criteria for determining which objects in the landscape fit in one of them. The definition of better rules and criteria depends on Machine Learning approaches, and it is currently beyond the scope of this research. Nevertheless, it is a promising future work for our research about handling landscape specification.

Additional knowledge about the context where the objects are placed is very useful for defining this type of criteria (for example, to place buildings in a city [11], or furnitures in an interior building [15, 81]). Another strategy is to use very constrained distributions [16].

However, we believe that better approaches can be obtained by analyzing large data sets. In Chapter 2 we show some works for terrain synthesis based in real data. Nevertheless, all these works are based in single samples provided for the method. Better approaches depends on better descriptors (high level descriptors) for the data.

An intermediate stage for the development of high level descriptors is the definition of which characteristics can be used to decompose the model in smaller units respecting some similarities. Examples of these features are: clustering of height, roughness of the model, ridges and valleys (length, density, etc.), rivers, roads, area of cities, etc.

These features can be used to infer some statistic behavior of some data and from this try to reproduce them. An example of application (a future work), assuming a terrain can be created by the Midpoint Displacement Mapping algorithm [49], we can decompose a terrain sample according to fractal scale. And thus, we can create another model using the same statistic of the input. It is a future work we intend to explore as soon as possible.

Ian Parberry presented recently a work in this direction [82]. He introduced a method for generating a terrain, that calibrate the parameters of a noise function by an analysis of a given real exemplar. Nevertheless, we believe that this work presents a shallow approach for this problem. The big problem is the used procedural method is not able of creating many terrain features. Furthermore, they do not present a way for distributing different sets of rules in the terrain (they produce a homogeneous model).

Finally, it is important to highlight that for describing a big model with many specific features will be (always) necessary a big specification. Nevertheless, the goal of high level techniques is move the focus from the creation of details (different than the modeling by moving control points, by the use of brushes, or controlling finner parameters) to keep focus in what is essential for the model, and describe it in a natural way.

Furthermore, for combining different types of elements, in different scales, maybe by combining of different techniques, we believe that the use of a multi-level (or multiscale) representation for the landscape is the best approach. In Next Section, we will discuss about it.

5.2 Multilevel Landscape Modeling

Despite of the variety and quality of methods for creating specific kind of elements, few of them consider simultaneously more than one type during the synthesis. For example, Deussen et.al. [74] introduced a technique for creating a plant ecosystem according to a provided terrain model. Another example was introduced by Galin et al. [83]. The authors proposed a procedural method for generating roads connecting cities over a given terrain model.

We believe the best way for creating a huge landscape, or an entire planet, is by combining some (or many) techniques related to different kind of objects, in different scales. To the best of our knowledge, the method proposed by Smelik et al. [72] is the most advanced technique in this direction. However, it is not a definitive approach. All introduced combination of elements are limited and hardly specific, that is, they are able to reproduce few phenomena.

We also believe that the best way to take advantage of this combination of techniques to the maximum is to create a multi-level representation, that will be evaluated in a top-down way. In this case, the features of landscape have to be described in several levels (or layers). Figure 5.1 shows a proposal of main levels for creating a huge landscape. The first level is the biggest feature: the continents

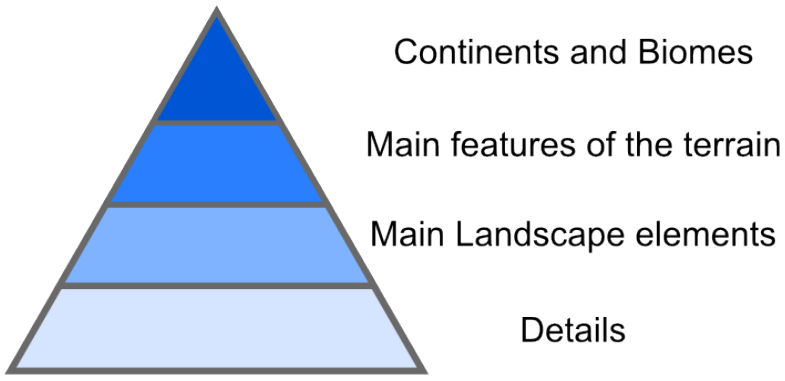


Figure 5.1: Example of Multilevel landscape decomposition

subdivided in biomes (like forests, deserts, grasslands, tundras, etc). The second level is the main features of the terrain: main chain of ridges and valleys; water bodies like rivers and lakes; and big cities. The third is the main landscape elements: landforms, buildings, trees, roads, etc. And finally, the finest level add details in the model (like flowers and grass, loose rocks, decorative elements, etc.).

Of course, it is an example of landscape decomposition that can be changed according to the definition of methods for generating each level. However, we really believe in the power of this approach for creating an entire planet by combining different methods for different types of objects.

This multilevel approach can be performed in a composition of regions, each one satisfying a set of common rules. The definition of these regions is performed by defining the place and shape and the set of containing elements (and their respective properties). Furthermore, it is necessary to define how is the transition between adjacent regions.

Chapter 6

Conclusion

The creation of virtual landscapes is a widely studied topic in computer graphics, because of the large number of possible applications. Despite of this topic has been studied for almost four decades, there is no definitive solution that creates a complex landscape, with many different features, according to nature, and controlled in a reasonably intuitive way.

The creation of an entire planet involves many research topics. We are focused on terrain synthesis and on the management of objects into the landscape. Despite our approaches have not been tested on the generation of an entire and complex planet from an intuitive specification, we are contributing for the improvement of the area in direction of achieving those goals.

Our main contribution is, from an analysis of the state of art, pointing to the expected future of the area, presenting its trends and challenges, to introduce a method for landscape specification management and two data-driven terrain synthesis. Furthermore, the development of this analysis and of these new proposed methods, provided us many ideas for future work. We have mentioned some of them in the previous chapters, and we will present others in this section.

We introduced a method to resize a landscape specification keeping the overall appearance. This method is based on the insertion and removal of objects followed by a scene adjustment (enlargement or shrinking). We have adapted the Seam Carving approach used for image and mesh editing to a simpler context: a vector-based

landscape model. Because our model is piecewise constant, we achieve good results using a straightforward metric. Furthermore, we have introduced simple and effective general methods to insert and to remove objects in the scene. Moreover, our method can be easily extended to use the state of the art techniques for spreading objects in a scene.

The main application of our method is to change the dimensions of a vector model to adapt the respective virtual environment, for instance, to be explored by a different number of characters. Furthermore, it is possible to use this approach to create large landscapes with the same overall appearance of a small model created by the user.

A drawback of our method is the resizing is not invertible. Thus, it is not possible to back to the exact previous configuration after the resizing. However, all modifications keep the initial appearance.

Furthermore, this method is sensitive to the definition of the rules used for generating the scene. A natural future work of this research is to define other constraints for the insertion and removal of objects. But mainly, we have to research criteria to determine which rules (from a set of rules) are being used in a given specification.

Another good improvement is to perform a specification retargeting, moving some set of elements from a place to another, or changing (and possibly moving) the area where they are inserted. We believe that the movement of a set of objects can be performed considering them as a composed object, and by using paths for translation.

By combining our approach with good methods to insert and remove objects, we obtained a powerful tool for the creation of huge landscapes. Moreover, by recursively creating structured elements, the proposed approach can be applied in multi-resolution. Thus, it could be used for the creation and editing of virtual worlds from a high level perspective: a very difficult problem in this area.

Moreover, we have introduced two data-driven methods for terrain synthesis. Both techniques are inspired on texture synthesis approaches, providing new structures for synthesis control, and taking advantage of the geometric nature of the data for improving the quality of results.

We have presented some techniques for creation of control structures. However, we

intend to improve this research by the definition of new methods, mainly by creation of tools able of reproducing some coarse features of a terrain.

Nealen and Alexa [46] introduced a hybrid technique for texture synthesis, that combines a patch-based approach with a pixel based. They create the texture like the Image Quilting method, but improve the seam error using the pixel-based technique. We believe this combination is very promising. Besides of the improvement of the seam, we believe we can synthesize some coarse levels using a patch-based approach, and finalize the synthesis using a pixel-based one. It is due the ability of the first approach to synthesize the macro structures, and the second to synthesize meso and micro. Furthermore, the metric of seam error proposed by Nealen and Alexa regards to texture. We believe we can get a more adequate metric for terrain.

Finally, the main goal of this thesis is to create high level techniques for landscape creation. Our method for specification resizing contributes to this goal because from a set of provided rules for object insertion, and an initial specification, we can create a larger model with the desired features. Furthermore, our data-driven methods are focused on improvement of the synthesis control (in a high level way). Our control structures provide an intuitive control of coarse features, and we introduced some methods for choosing adequately the exemplars from a large data set. Despite of the advances for terrain synthesis, there is a lack of high level methods for complex models creation, and methods that combines different kinds of nature elements. The main future work for this research is to pursuit these goals. We intend to route our research for creation of entire planets, and identify, in a more precise way, what are the main challenges of this topic.

Bibliography

[1] R. Smelik, T. Tutenel, R. Bidarra, and B. Benes, "A survey on procedural modeling for virtual worlds." Computer Graphics Forum, 2014.

[2] M. Natali, E. M. Lidal, J. Parulek, I. Viola, and D. Patel, "Modeling terrains and subsurface geology." EuroGraphics 2013 State of the Art Reports, 2013.

[3] G. Mariethoz and S. Lefebvre, "Bridges between multiple-point geostatistics and texture synthesis." Computers and Geosciences, 2014.

[4] D. Ebert, F. K. Musgrave, D. Peachey, K. Perlyn, and S. Worley, *Texturing and Modeling: A procedural approach.* Academic Press, 1998.

[5] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. Ebert, J. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions." Computer Graphics Forum, 2010.

[6] L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk, "State of the art in example-based texture synthesis." Eurographics State of the Art Reports, 2009.

[7] D. Vaquero, M. Turk, K. Pulli, M. Tico, and N. Gelfand, "A survey of image retargeting techniques." In Proceedings of Applications of Digital Image Processing, 2010.

[8] L. Liu, R. Chen, L. Wolf, and D. Cohen-Or, "Optimizing photo composition." Eurographics, 2010.

[9] E. Dekkers and L. Kobbelt, "Geometry seam carving." SIAM Conference on Geometric and Physical Modeling, 2013.

[10] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing." ACM SIGGRAPH, 2007.

[11] Y. Parish and P. Muller, "Procedural modeling of cities." Conference on Computer Graphics and Interactive Techniques, 2001.

[12] B. Watson, P. Muller, P. Wonka, C. Sexton, O. Veryovka, and A. Fuller, "Procedural urban modeling in practice." IEEE Computer Graphics and Applications, 2008.

[13] A. Rau-Chaplin, B. MacKay-Lyons, and P. F. Spierenburg, "The lahave house project: Towards an automated architectural design service." International Conference on Computer-Aided Design, 1996.

[14] E. Hahn, P. Bose, and A. Whitehead, "Persistent realtime building interior generation." ACM Symposium on Videogames, 2006.

[15] J. Martin, "Procedural house generation: A method for dynamically generating floor plans." Symposium on Interactive 3D Graphics and Games, 2006.

[16] Y.-T. Yeh, L. Yang, M. Watson, N. D. Goodman, and P. Hanrahan, "Synthesizing open worlds with constraints using locally annealed reversible jump mcmc." ACM SIGGRAPH, 2012.

[17] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer." ACM SIGGRAPH, 2001.

[18] A. Lasram and S. Lefebvre, "Parallel patch-based texture synthesis." High Performance Graphics conference proceedings, 2012.

[19] H. Zhou, J. Sun, G. Turk, and J. Rehg, "Terrain synthesis from digital elevation models." IEEE Transactions on Visualization and Computer Graphics, 2007.

[20] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super-resolution." Computer Graphics and Applications, 2002.

[21] S. Lefebvre and H. Hoppe, "Parallel controllable texture synthesis." ACM SIGGRAPH, 2005.

[22] C. Han, E. Risser, R. Ramamoorthi, and E. Grinspun, "Multiscale texture synthesis." ACM SIGGRAPH, 2008.

[23] L. Cruz, L. Velho, D. Lucio, E. Galin, A. Peytavie, and E. Guerin, "Landscape specification resizing." CLEI, 2014.

[24] L. Cruz, F. Ganacim, L. Velho, L. H. de Figueiredo, and D. Lucio, "Exemplar-based terrain synthesis." WIP - SIBGRAPI, 2013.

[25] L. Cruz, L. Velho, E. Galin, A. Peytavie, and E. Guerin, "Patch-based terrain synthesis." GRAPP, 2015.

[26] A. S. Glassner, *Principles of Digital Image Synthesis.* Morgan Kaufmann, 2011.

[27] G. S. P. Miller, "The definition and rendering of terrain maps." ACM SIGGRAPH, 1986.

[28] S. Lefebvre, "Runtime texture synthesis." Habilitation thesis - Université de Lorraine, 2014.

[29] D. R. Peachey, "Solid texturing of complex surfaces." ACM SIGGRAPH, 1985.

[30] K. Perlin, "An image synthesizer." ACM SIGGRAPH, 1985.

[31] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion." ACM SIGGRAPH, 1991.

[32] K. Popat and R. W. Picard, "Novel cluster-based probability model for texture synthesis, classi cation, and compression." In Proceedings of SPIE Visual Communications and Image Processing, 1993.

[33] D. J. Heeger and J. R. Bergen, "Pyramid-based texture analysis/synthesis." Proceedings of Conference on Computer Graphics and Interactive Techniques, 1995.

[34] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling." IEEE International Conference on Computer Vision, 1999.

[35] J. S. D. Bonet, "Multiresolution sampling procedure for analysis and synthesis of texture images." Conference on Computer graphics and interactive techniques, 1997.

[36] L.-Y. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization." ACM SIGGRAPH, 2000.

[37] ——, "Order-independent texture synthesis." Technical Report, 2002.

[38] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum, "Synthesis of progressively-variant textures on arbitrary surfaces." ACM SIGGRAPH, 2003.

[39] A. Zalesny, V. Ferrari, G. Caenen, and L. V. Gool, "Composite texture synthesis." International Journal of Computer Vision, 2005.

[40] M. Ashikhmin, "Synthesizing natural textures." In Proceedings of the Symposium on Interactive 3D graphics, 2001.

[41] Y.-Q. Xu, B. Guo, and H. Shum, "Chaos mosaic: Fast and memory efficient texture synthesis." Technical Report, 2000.

[42] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures." In Proceedings of the Conference on Computer Graphics and Interactive Techniques, 2000.

[43] L. Liang, C. Liu, Y. Xu, B. Guo, and H.-Y. Shum, "Real-time texture synthesis by patch-based sampling." ACM Transactions on Graphics, 2001.

[44] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies." ACM SIGGRAPH, 2001.

[45] Q. Wu and Y. Yu, "Feature matching and deformation for texture synthesis." ACM SIGGRAPH, 2004.

[46] A. Nealen and M. Alexa, "Hybrid texture synthesis." Eurographics Symposium on Rendering, 2003.

[47] B. B. Mandelbrot, "Stochastic models for the earth's relief, the shape and the fractal dimension of the coastlines, and the number-area rule for islands." In Proceedings of the National Academy of Sciences (USA), 1975.

[48] ——, *The Fractal Geometry of Nature*. Freeman and Co, 1983.

[49] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models." Communications of the ACM, 1982.

[50] M. Gamito and F. K. Musgrave, "Procedural landscapes with overhangs." In Proceedings of 10th Portuguese Computer Graphics Meeting, 2001.

[51] A. Peytavie, E. Galin, S. Merillou, and J. Grosjean, "Arches - a framework for modeling complex terrains." Computer Graphics Forum, 2009.

[52] B. Benes and R. Forsbach, "Layered data representation for visual simulation of terrain erosion." In Proceedings of the 17th Spring conference on Computer graphics, 2001.

[53] J.-D. Genevaux, E. Galin, E. Guerin, A. Peytavie, and B. Benes, "Terrain generation using procedural models based on hydrology." ACM SIGGRAPH, 2013.

[54] D. M. de Carli, C. T. Pozzer, F. Bevilacqua, and V. Schetinger, "Procedural generation of 3d canyons." SIBGRAPI, 2014.

[55] A. Kelley, M. Malin, and G. Nielson, "Terrain simulation using a model of stream erosion." In Proceedings of Conference on Computer Graphics and Interactive Techniques, 1988.

[56] F. K. Musgrave, C. E. Kolb, and R. S. Mace, "The synthesis and rendering of eroded fractal terrains." ACM SIGGRAPH, 1989.

[57] P. Kristof, B. Benes, J. Krivanek, and O. Stava, "Hydraulic erosion using smoothed particle hydrodynamics." Computer Graphics Forum, 2009.

[58] O. Stava, B. Benes, M. Brisbinn, and J. Krivanek, "Interactive terrain modeling using hydraulic erosion." Symposium on Computer Animation, 2008.

[59] J. M. Cohen, J. F. Hughes, and R. C. Zeleznik, "Harold: a world made of drawings." NPAR, 2000.

[60] J. Gain, P. Marais, and W. Straber, "Terrain sketching." Symposium on Interactive 3D graphics and games, 2009.

[61] F. P. Tasse, A. Emilien, M.-P. Cani, S. Hahmann, and A. Bernhardt, "First person sketch-based terrain editing." Graphics Interface Conference, 2014.

[62] V. A. dos Passos and T. Igarashi, "Landsketch: A first person point-of-view example-based terrain modeling approach." Symposium on Sketch-Based Interfaces and Modeling, 2013.

[63] G. de Carpentier and R. Bidarra, "Interactive gpu-based procedural heightfield brushes." International Conference on Foundations of Digital Games, 2009.

[64] H. Hnaidi, E. Guérin, S. Akkouche, A. Peytavie, and E. Galin, "Feature based terrain generation using diffusion equation." In Proceedings of Pacific Graphic, 2010.

[65] A. Bernhardt, A. Maximo, L. Velho, H. Hnaidi, and M.-P. Cani, "Real-time terrain modeling using cpu–gpu coupled computation." In Proceedings of Conference on Graphics, Patterns and Images, 2011,.

[66] R. Smelik, K. J. D. Kraker, S. A. Groenewegen, T. Tutenel, and R. Bidarra, "A survey of procedural methods for terrain modelling." CASA, 2009.

[67] C. Dachsbacher, M. Meyer, and M. Stamminger, "Height-field synthesis by non-parametric sampling." In Proceedings of Vision, Modeling and Visualization, 2005.

[68] J. Brosz, F. Samavati, and M. C. Sousa, "Terrain synthesis by-example." In Proceedings of International Conference on Computer Graphics Theory and Applications, 2006.

[69] Y.-C. Chang, G.-S. Song, and S.-K. Hsu, "Automatic extraction of ridge and valley axes using the profile recognition and polygon-breaking algorithm." Computers and Geosciences, 1998.

[70] F. P. Tasse, J. Gain, and P. Marais, "Enhanced texture-based terrain synthesis on graphics hardware." Computer Graphics Forum, 2012.

[71] E. Bruneton and F. Neyret, "Real-time rendering and editing of vector-based terrains." Eurographics, 2008.

[72] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, "A declarative approach to procedural modeling of virtual worlds." Computers and Graphics, 2011.

[73] P.-E. Danielsson, "Euclidean distance mapping." Computer Graphics and image Processing, 1980.

[74] O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems." ACM SIGGRAPH 1998, 1998.

[75] P. Prusinkiewicz and M. Hammel, "A fractal model of mountains with rivers." In Proceeding of Graphics Interface, 1993.

[76] R. Paget and D. Longstaff, "A nonparametric multiscale markov random field model for synthesising natural textures." International Symposium on Signal Processing and its Applications, 1996.

[77] S. C. Zhu, Y. Wu, and D. Mumford, "Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling." International Journal of Computer Vision, 1998.

[78] A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or, "A sketch-based interface for detail-preserving mesh editing." ACM SIGGRAPH, 2005.

[79] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Roessl, and H.-P. Seidel, "Laplacian surface editing." Symposium on Geometry Processing, 2004.

[80] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka, "Interactive beautification: A technique for rapid geometric design." Symposium on User Interface Software and Technology, 1997.

[81] P. Merrell, E. Schkufza, Z. Li, M. Agrawala, and V. Koltun, "Interactive furniture layout using interior design guidelines." ACM SIGGRAPH, 2011.

[82] I. Parberry, "Designer worlds: Procedural generation of infinite terrain from real-world elevation data." Journal of Computer Graphics Techniques, 2014.

[83] E. Galin, A. Peytavie, E. Guerin, and B. Benes, "Authoring hierarchical road networks." In Proceedings of Pacific Graphics, 2011.