Decoupling structure and appearance for example-driven detail synthesis

Marie-Claude Frasson

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS UFR Sciences

École Doctorale STIC

Thèse

pour obtenir le titre de

Docteur en Sciences

de l'UNIVERSITÉ de Nice - Sophia Antipolis

Spécialité : INFORMATIQUE

présentée par

Marie-Claude FRASSON

Decoupling Structure and Appearance for Example-Driven Detail Synthesis

Thèse dirigée par George DRETTAKIS
et préparée à l'INRIA Sophia-Antipolis au sein du projet REVES

Soutenue publiquement le vendredi 24 juin 2005
devant le jury composé de :

M.	Peter	SANDER	ESSI, France	Président
M.	Ken	PERLIN	New-York University, USA	Rapporteur
M.	Bernard	PÉROCHE	Université de Lyon, France	Rapporteur
M.	Fabrice	NEYRET	Laboratoire GRAVIR/IMAG-INRIA, France	Examinateur
M.	Pierre	ALLIEZ	INRIA Sophia-Antipolis, France	Examinateur
M.	George	DRETTAKIS	INRIA Sophia-Antipolis, France	Directeur

Contents

À ma famille et mes amis, des deux côtés de l'Atlantique…

Remerciements

5/09/98: Je pose la dernière touche à mon mémoire de M.Sc. au Canada. Moralité: plus jamais!
5/09/05: Me voici à nouveau en train d'écrire une page de remerciements pour la clôture d'un autre mémoire, en France: celui du doctorat! Qui l'eût cru? En tout cas sûrement pas moi mais la vie étant ce qui arrive à la place de ce que l'on planifie[1], je me retrouve devant cette page blanche, essoufflée après la course épuisante de quelques années que je viens d'achever, mais tellement soulagée d'y être finalement arrivé, tout en ayant parfois du mal à y croire!

Ces dernières années n'ont ainsi pas été de tout repos et si j'ai pu achever aujourd'hui ce mémoire que vous tenez entre les mains (ou entre celles de M. Acrobat), c'est essentiellement dû au personnes qui m'ont constamment soutenu, m'encourageant lorsque je me décourageais, ne me tenant pas rigueur pour ces longs mois sans nouvelles, en bref, ne m'oubliant pas malgré tout et ce, des deux côtés de l'Atlantique.

Ma thèse parle de structure, mes remerciements n'en auront pas vraiment et voici donc en vrac quelques remerciements et pensées...

Cette traversée de thèse, je l'ai vécue sous la direction du capitaine George. Tantôt houleux, tantôt calmes, les épisodes de cette épopée se sont succédés jusqu'à l'arrivée en terre des docteurs. Je le remercie vivement pour ses conseils judicieux, son soutien lors du sprint final (sur plusieurs mois) et surtout je lui suis reconnaissante de m'avoir poussé à finir avant que je commence à prendre racine dans mon bureau.

Je remercie mes deux rapporteurs, Ken (qui n'aime plus les aéroports depuis) et Bernard (d'une gentillesse exemplaire) d'avoir accepté de relire cette thèse ainsi que les autres membres du Jury: Peter (un président canadien pour soutenir un 24 juin, quelle belle image ce fut!), Fabrice (et sa multitude de corrections!) et Pierre. Je remercie également ce dernier pour toutes les longues discussions sur mon sujet et d'autres...

J'ai eu la chance d'effectuer ma thèse à l'INRIA de Sophia-Antipolis. Je pense qu'on ne peut rêver de meilleur cadre pour faire une thèse (Heureusement... avec le nombre de week-ends que j'y ai passé!). Je garderai un très beau souvenir du lieu (et de sa cafétéria!) et des gens qui le peuplent, tous plus gentils les uns que les autres. J'ai tout particulièrement apprécié les chouettes moments avec les gens du projet Reves: Nico et son assiette de carotte quotidienne et pour rigoler à mes bêtises, Manu et ses blagues qui tombent parfois à plat, Florent qui s'est exilé à Paris mais était virtuellement là, Guillaume et sa musique dans le couloir et surtout Gael et Alex pour leur soutien inébranlable et les nombreuses discussions et rigolades! Attention les garçons, il va falloir se débrouiller sans maman mc à partir de maintenant! J'en profite également pour remercier Agnès et Marie-Line pour leur aide précieuse en cette fin de thèse...

De l'INRIA, je tiens tout particulièrement à remercier les gens du CSD et surtout Thierry, mon ange gardien, sans qui je pense que j'aurais fini beaucoup plus tôt... mais sans thèse...

[1] Ce n'est pas de moi ;-)

En plus des soirées précieuses entre amis, deux activités m'ont permis de garder un semblant de sanité mentale: L'impro du vendredi soir et la voile du samedi matin. Merci aux lapineaux de l'AIA pour m'avoir aidé à complètement évacuer et encouragé dans mes personnages de pingouins québécois! Et le lendemain, réveil tôt mais pour une bonne cause: voguer dans la baie d'Antibes avec Claire, Fabienne, Cédric, Sylvain, Olivier, Armelle, Guilhem, Laurence, Marc(s), Nat et tous les autres du CNA! Merci pour votre soutien aussi!

J'ai eu la chance pendant ces années de thèse de me trouver au sein d'un super groupe d'amis avec qui j'ai partagé d'inoubliables soirées, repas, randos, week-ends, etc... (même si j'ai dû en refuser un paquet :o( ) Un merci particulier donc à Alberto et Damiana pour tout le piquant italien, Nicolas pour râler tout en étant adorable, Olivier pour se moquer des thésards et finalement pour vraiment comprendre avec Marina ce que c'est!, Tom et Sev qui m'ont prouvé qu'on pouvait mener de front vie familiale et vie entre copains, Benoit pour ses conseils souvent en or et pour m'avoir supporté avec mon chat, Fred et Magali, le couple secret et enfin Alain, un compatriote, pour me les avoir présenté!

Une petite soeur et maman à la fois m'a souvent épaulée et sans elle, la vie aurait été beaucoup moins rigolote au cours de cette thèse. Je tiens vraiment à remercier Céline! Entre mails interminables et organisation de soirées, anniversaires et week-ends et semaines entières passées à manger en tête-à-tête des plats micro-ondes parfois douteux dans la cafète sombre de l'INRIA, sa présence et son support m'ont été inestimables et j'espère que nous pourrons nous retrouver dans le futur, que ce soit pour un autre voyage (caramba!) ou pour un lieu commun.

D'autres amis m'ont également soutenu et je tiens à saluer Robin, Valda, Fanny, Yann, Béa, Agnès, Fabien, Ludo, Guillaume, Patrick, Marc, Claire, David et tous les autres d'ici que j'ai côtoyé à un moment ou à un autre. Je ne peux nommer tout le monde, même si je le voudrais bien, mais il y a aussi mes amis lointains (en distance), au Canada et aux US comme en France, dont les encouragements transgressaient la distance. Ils se reconnaîtront... Un bec en particulier à Caroline qui en plus, est venue me voir chaque année! Pardon de n'avoir donné que très peu de nouvelles et merci pour votre patience...

Un merci très spécial à Nico et Sébastien qui ont cru en moi alors que ce n'était pas vraiment mon cas à plusieurs occasions; cette thèse, je vous en dois un bon bout... et aussi à Alesk pour avoir été là, patient et compréhensif malgré les absences continues, pour ses picnics du dimanche, jamais trop loin de l'INRIA, pour les corrections, pour les câlins, pour plein de choses...

Je tiens à finir par ma famille. Ici, en France, mon oncle Henry et Fabienne pour tous les bons repas et le refuge des dernières semaines de rédaction. Et en Amérique du Nord, mes plus ardents supporters, papa, maman (dont l'aide pour plusieurs aspects de cette thèse m'a été très précieuse) et ma petite soeur Corinne, tous les trois toujours là malgré la distance et que j'aime profondément...

Chapter 1

Introduction

"Lorsque tu ne sais pas où tu vas, regarde d'où tu viens."

—Proverbe africain

For most of us, thirty years ago, computer graphics was not about realism and details, it was only about having something else than letters and numbers on a screen and being able to interact graphically with the computer. We could not become bored playing one of the first video games: *Pong* (cf image 1.1), and it consisted only in two lines moving up and down and a point traveling across the screen. Nowadays, computer power has evolved so much that highly realistic tennis games featuring nicely animated players are now possible (see image 1.1). Most importantly, we now *expect* such high degree of realism in the games we play, and also always would like more.

Figure 1.1 Left: The first "tennis" video game: Pong. Right: One of the latest tennis video game (Image from TopSpin [Stu]).

This high demand for realism does not concern only the games market but also all the sectors where computer graphics techniques are being increasingly used such as cinema or scientific visualization. People are more and more accustomed to seeing virtual 3D worlds and are thus very critical regarding the visual quality of these worlds. They want them to be as realistic, or believable, as possible. Researchers are thus always investigating new algorithms to make computer-generated images almost indistinguishable from real photographs. For some type of scenes[1], this goal has been achieved. However, for others, many artefacts remain. Most of the time, the simplicity of the models, the lack of details and the repetition of structures, patterns, characters, colors, are the defects of synthetic scenes which bother us.

[1]Some terms are defined in the glossary in appendix A. Their first appearance in the text is underlined.

We consider *details* to be anything that adds to the realism of a scene. It can be color, fine geometry, reflectance properties of surfaces, silhouettes, variability of information, etc. All the complexity which characterizes the real-world. In this thesis, we want to provide solutions to increase the realism of a scene by providing some of this detail complexity using real-world examples, while sacrificing as little as possible memory space or processing time.

1.1 Motivation

In order to respond to the ever-increasing demand of realism, a lot of effort is put into improving hardware and software components of the computer graphics pipeline (modeling, animation and rendering). With the recent technological advances in processing units and computer graphics cards, amazing speed increases have been attained for the rendering phase. However, the tools for modeling and animation have yet to follow. Indeed, as most computer generated (*CG*) scenes are hand-made, the final results depend on the skills of the artists involved and the tools they are using. Depending on the time and budget available, they take shortcuts when building these scenes: Copy-pasting models and textures to repeat them, giving the illusion of details using textures over coarse models, pre-computing lighting information as additional textures, etc. Furthermore, the target usage and the limitations of the available hardware can also place a restriction on the size of the scene (i.e., number of polygons, number of lights, number of textures, number of reflectance properties, amount of computation involved to create each image) and thus on the details it can contain. For example, a game scene has to be much "lighter" (in terms of demand in processing time) than a movie scene to be able to be displayed at an acceptable frame rate. A major bottleneck thus resides in the time to produce and edit detailed scenes.

To simulate missing geometric details, artists use a plethora of methods. The usual workflow they adopt consists in applying to a geometric model a series of *appearance layers*[2] whose combination defines the final aspect of the object. Each layer describes some surface (or material) properties of the object, such as color, reflectivity, displacement, translucency, etc. It can have various parameters which can be adjusted independently of the other layers. This notion of independence is very important in standard modeling workflow since it permits individual fine-tuning of each layer, avoiding possible impact on other layers. However, creating these appearance layers is not always an easy task. We describe next three problems we have identified in this process, which have guided the work in this thesis.

Problem #1: Structured Textures Synthesis

The first and most common layer added on top of a model is the color texture. An immense variety of texture models exists, ranging from procedurally-generated to image-based photo-realistic textures. Textures are typically used to add rich visual detail to a much simpler underlying mesh. However, they suffer from a number of problems due to their fixed-size (for non-procedural textures) and thus limited resolution, in addition to parametrization issues.

Two main classes of textures are particularly of interest: stochastic and structured textures. Stochastic textures are viewed as a realization of a 2D stochastic process whereas structured textures correspond to an arrangement of primitives according to a certain placement rule. Experiments have shown that human eyes are very sensitive to high-level features (especially their discontinuity) [Mar82], and tolerant to small deformations. Thus artefacts in structured textures

[2] In production, this stack of layers is called a shader due to historical reasons, but it no longer is exclusively related to shading.

are more easily detectable than in stochastic textures. Structured textures are often used for environment (background) objects, nonetheless the viewer still immediately notices repetition due to a tiling of a base texture, border problems at the junction of texture parts, parametrization issues, etc. Image 1.2 illustrates some of these common problems.

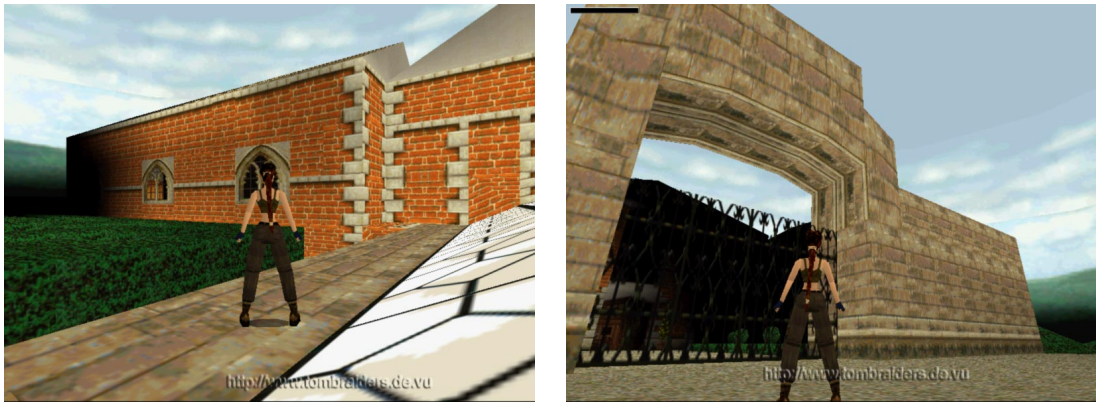

Figure 1.2 Texture mapping structured textures often exhibits annoying artefacts: parametrization problems, structure repetitions, limited resolution. (Images from [tom])

Ideally, we would like to be able to generate such structured textures without their inherent problems of junction and resolution (which will be detailed later). These can also be called "cellular textures" since they are typically a set of closed regions, such as a rock wall or a snake skin. We will use the term "structure" or "structured textures" for the most part in this text, with the exception of cases when the term "cellular texture" eases comprehension. To reproduce such kinds of textures, they can be drawn and tiled manually by a modeler but this rapidly becomes a lengthy and tedious process. Some structure generators exist but their results are often not predictable and their lack of control makes them not attractive for artists, who need to fine-tune their work at will. A practical solution would take an example of the expected result as input and provide a parameterizable procedural method to achieve a similar result. Indeed, textures extracted from the real-world are excellent sources of inspiration for designing virtual textures and the quality of their details should be taken advantage of as much as possible. The texture of the Pickering Castle walls (image 1.3) is an example of a structured texture we might want to reproduce in order to obtain a faithful virtual reconstitution of the completed castle. To synthesize a similar texture, an option would be to use texture synthesis algorithms which aim to replicate a texture starting by the analysis of a sample. However, even though the results on stochastic textures are satisfying, there is yet no good solution for structured textures.

The first problem we address in this thesis is thus the following: How can we analyze a given structured texture (typically coming from a real-world example), in order to be able to propagate it on various surfaces without getting the problems intrinsic to texture mapping? How can we define such a *structure generator*? Furthermore, we believe that completely automatic solutions take the artistic work off the hands of the artists. We need to relieve them of the tedious part of the work, while allowing them to influence the structured texture generation both globally and locally. Local modifications techniques avoid the need to regeneration of the entire structure, and facilitate rapid, iterative design.

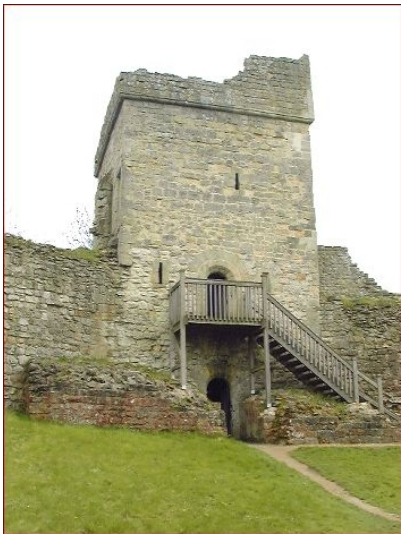

Figure 1.3 The Pickering castle. How can we virtually complete it and replicate the existing wall texture, in a convincing, non repetitive manner?

Problem #2: Coupling of Structure and Appearance

Consider the village of figure 1.4 (right) composed of several instances of the same house shown to its left. Using the same texture for every house will give a fake aspect to the whole scene. However, if we manage to synthesize a different stone wall structured texture to apply to each individual house using texture synthesis approaches such as [EF01] or [WL00], texture memory consumption will be very high since we can no longer share textures between objects. Procedural methods would be an alternative, but are notoriously hard to control and predict.

One important observation is that for structured textures such as these stone walls, *structure*, i.e., the form of the stones and their placement, and *appearance*, i.e., the texture and 3D details (e.g., relief) of each stone and of the mortar between them can (and should) be treated independently. We believe one of the reasons texture synthesis methods fail for such textures is because they attempt to handle them together, as a whole, without distinguishing them from other classes of textures.

Problem #3: 3D Detail Synthesis

To enhance the visual richness of these scenes, more advanced appearance layers must be used to map, for instance, surface normal, height or even lighting information onto the surface, avoiding the "flatness" and straight-silhouettes problems illustrated in figure 1.5. They allow the addition of subtle 3D information to the underlying color and geometry data. However, building appearance layers for similar patterns but arranged in a variety of locations and orientations can rapidly become very tedious for the artist, as he cannot rely on copy-pasting. There is no existing automatic method to relieve him from this task.

More generally, detail creation is a tedious process. Using traditional modeling tools to construct fine geometric details represents an enormous quantity of repetitive work. One solution can be to take a detail sample (for example, in the form of a picture or a measurement) and use it to enhance the hand-made coarse models by adding, generating or propagating this detail. However, each kind of detail has its characteristics and particularities that one has to understand in order to be able to regenerate it. In addition, the related data has to be editable and manipulable by artists for such a solution to be usable in practice.

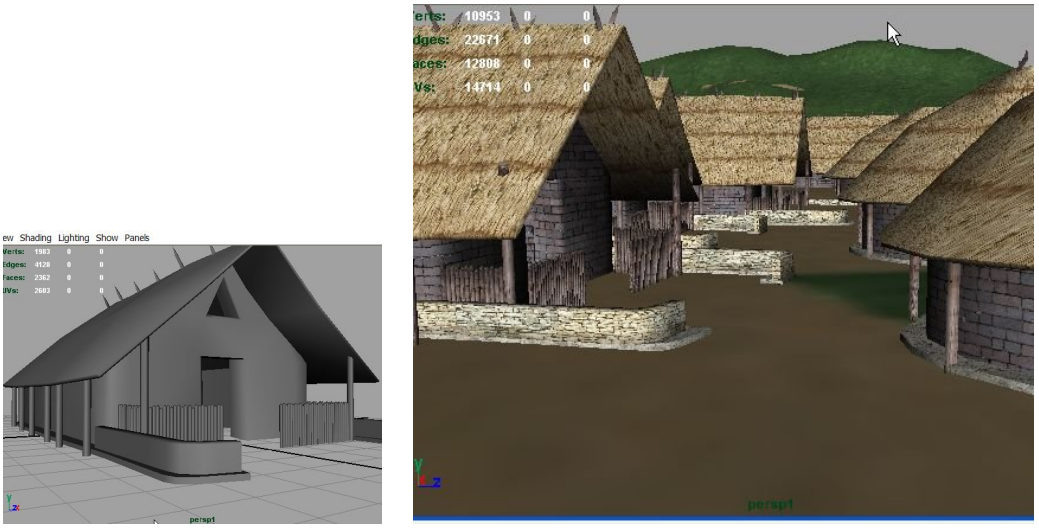

Figure 1.4 Left: Computer-Generated house yet to texture. Right: A village of similar houses with structured textures on their walls.

Figure 1.5 Left: Scene where the flatness of the structured textures is evident. Right: Additional appearance layers have been applied and realism has increased. (Images from Oliveira [dON00])

The third problem we address is thus how to ease the work of the user for the generation of small, but nonetheless crucial for realism details, either by generating appearance layers corresponding to synthesized structured textures, or more generally, by creating 3D information generators from example.

In summary, in this thesis, we investigate three themes revolving around the idea of exploiting real-world information to increase realism in 3D scenes:

1. How to synthesize structure from the analysis of a sample to avoid annoying repetitions.
2. How to generate realistic structure appearance using a compact description to be able to achieve real-time rendering, while avoiding repetitive textures.
3. How to propagate 3D information from example to relieve the artists from tedious detail creation tasks.

In the next section, we will state our diverse objectives to tackle the problems just described and the approaches we propose in order to reach them.

1.2 Objectives and proposed approaches

In this work, we address the main goal of improving realism in computer-generated scenes by providing solutions to analyze information from the real world (structure, appearance, 3D information) and using it to create user-tunable procedural "generators" of appearance layers which efficiently enrich the scene models. This constitutes a long-term goal of which this thesis sets the first elements.

The two first parts of this thesis focus on the synthesis of structured textures based on the analysis of real-world examples which are hard to reproduce using current techniques. We chose to address this particular problem by decoupling the structure and the appearance, which will give us a lot more flexibility. The third part exposes a framework for the recovery of 3D information from example and proposes an application of this approach to structured textures. The techniques exposed are applied to height information and generalized to other types of details.

1. Generating structure from example

Synthesizing a structure from an example is hard. We create structure generators to solve this problem, enabling us to propagate a given structured texture in a procedural manner, while allowing the user to influence the results. This is inherently a geometric problem, which we will treat as such. We try to analytically reproduce a given geometric structure represented as a 2D mesh of edges and vertices. Our objective is to extract a compact representation of the structure from the analysis of the sample. This representation allows us to replicate the structure on a target domain of varying shape and topology. The user is able to influence the synthesis and modify the result without having to regenerate everything.

The approach we propose consists in evaluating the anisotropy of each of the regions composing the structure by approximating them with a simple shape. This permits to build simple statistics to which we add some data related to the topology of the structure. This statistical data is exploited during the synthesis phase to generate new similarly-shaped regions in a two-step process. In a first step, the regions are sampled from the statistics and will progressively populate a target domain using a tiling method. Various types of constraints, input as maps for instance, can be taken into account during this stage. In a second step, the shape of the regions are optimized to resemble the original shapes as much as possible. The generated structure is editable, regions can be locally modified and the structure updates globally without the need for re-synthesis. The process is flexible; many extensions are possible and are described in chapters 3 and 6.

2. Generating 2D appearance from example

As stated earlier, when analyzing a given structured texture, we decouple the structure analysis from the appearance analysis as they are independent entities. Our main goal when analyzing the appearance of a structured texture is to create an appearance generator which is lightweight, i.e., which requires a small number of base/reference images and will render the result efficiently. It should be noted that we address only the 2D color texture at this stage which corresponds to the "2D appearance", the first appearance layer usually added to a model.

Since we want to produce a lightweight generator, we have to create reusable texture components which can be accessed many times, so as to use as little space as possible in texture memory. To achieve the same appearance as the example, these components have to be extracted from the original texture information, while taking care not to degrade the small details that are central to the richness of the texture. In order to achieve fast rendering, no new texture information can be

created during rendering, even though the target geometry can be different from the original. Our choice is to do as much processing as possible off-line, as a preprocess. Similarly to the structure generation, the user should be able to influence the process up to a certain point.

The approach we propose is inspired by the principle of *instancing* which consists in "recycling" textures by indexing many times in the same image to texture several models. We have as input an example of a structured texture with the corresponding geometric structure, composed of a set of closed regions. During the analysis phase, we extract texture elements from the region interiors and use them to build a set of *unified textures*. Note that contrary to traditional texture instancing, each region (e.g., a stone in a wall) indexes into a different part of this unified texture, thus avoiding repetition. We handle the region contours (e.g., mortar in a stone wall) differently by extracting texture parts around the edges and vertices and gathering them in texture atlases. We also store the structure geometric data and associate it with the extracted texture information.

The geometric structures resulting from the synthesis process previously described are composed of vertices and edges around which we create additional polygons. During the synthesis phase, we associate to each of them a texture from the atlases and sample the unified textures to generate the color information for the region interiors. Special care has to be taken to seamlessly blend all the texture portions. We decided to establish a tradeoff between the number of polygons and the number of textures utilized in order to achieve the space and speed requirements. We can afford polygons much more than texture space, especially if the textures are of high-quality. The user is allowed to modify the shared textures to create new appearance results. Structure generators can also be combined with appearance generators coming from different samples to synthesize brand new structured textures.

3. Generating 3D details from example

The last part of this thesis aims at recovering 3D information from images. Fine geometric details, reflectance properties etc., are imprinted into the textures and we should take advantage of this information, guided by the expert eyes of a user. This data can allow us to construct additional appearance layers to add more realism to our objects. Our goal is to generate *plausible* details, not necessarily exact details, using the available texture and model information and the guidance of the user with the aim of making our scenes more believable. We aspire to relieve the user from repetitive, tedious tasks such as creating appearance maps for structured textures, while leaving him the intentional, creative work.

We thus propose a generic detail recovery framework where information about desired details (geometry, reflectance, etc.) can be provided by the user. The system employs this extra information by analyzing the model to improve and generates the corresponding details appropriately. Specific results on displacement map recovery are described, and we can take advantage of any structure or texture information recovered in a previous step to improve the results. Insights for the recovery of other kinds of appearance layers are exposed. In a way, we are creating user-assisted detail generators. The main advantages of such generators is that they are procedural, and thus lightweight. They only contain a description of the missing information and how it should be reconstructed according to the underlying models/textures. Interactivity and user-input is a key feature in our approach as we are convinced that only a human user can provide the information required for a robust detail recovery, as current automatic methods are not mature enough and only the user knows what he really wants to recover.

1.3 Contributions

We summarize in this section the principal contributions of our work. These contributions, and additional ones, are described in more details in the conclusions.

- **Synthesis of detailed cellular textures from the analysis of examples**

  No method exists to generate detailed structured textures, notably cellular textures with cells of varying-shape, from the analysis of an example. We propose methods to recover their geometry as well as their appearance aspect, in 2D and 3D.

- **Decoupling structure and appearance**

  Structure and appearance are often independent entities and should be treated as such when analyzing structured textures in order to create generators which allow their reproduction. We explore this novel idea by describing our techniques to create *structure generators* in parallel with the creation of *appearance generators*.

- **Creation of structure generators**

  We present a new procedural technique to generate geometrical structures using statistics extracted from the analysis of a geometric input structure.

- **Creation of appearance generators**

  We designed techniques to extract reusable appearance components from a structured texture image, thus achieving significant memory gains. The texture of the structure regions interiors is decoupled from the texture of their contours (or separating material) in order not to mix unrelated materials. This offers the additional possibility to combine them with others coming from another structured texture.

- **Framework for detail generation**

  We expose a framework to alleviate the work of creating repetitive details by using the user and the available structure, model and texture information.

- **Leaving control to the user**

  In all the solutions presented, we left some control to the user over parameters and important parts of the process.

- **Texture synthesis**

  Our approach contributes to the texture synthesis research area as most current methods fail on structured textures. We also open avenues of thinking into how to add control to texture synthesis techniques, as this control is key to satisfactory results.

1.4 Thesis overview

This document is organized as follows.

Chapter 2: Related work This chapter describes the techniques to enrich geometric models with detail information, either real or simulated, using appearance layers. It presents state-of-the-art techniques to generate structured textures, ranging from geometric approaches up to cellular approaches. Various methods to propagate details from example, such as texture synthesis techniques, are reviewed before giving an overview of previous work focussing on integrating the user into the process of creating details.

Chapter 3 This chapter presents our new approach to analyze a geometric structure in order to extract a set of simple statistics, which are used in a two-stage synthesis phase to generate similar structures. We describe the tiling technique we use, showing its advantages in allowing control of the synthesis process through the use of constraint maps or interactive manipulation.

Chapter 4 This chapter describes our solution to extract appearance data from a structured texture in order to rapidly generate and display the appearance of another given structure. The creation of reusable unified textures and texture atlases are explained along with the mechanism to seamlessly blend all the texture extracts being used.

Chapter 5 This chapter presents our framework for semi-automatically propagating user-given detail information using available texture, structure and geometry information. It demonstrates results for the synthesis of displacement maps and describes additional layers which would allow to recover other kind of 3D information.

Conclusion We conclude the thesis by summarizing our contributions and discussing open avenues related to structure, appearance and detail synthesis.

Appendix A: Glossary Terms whose definition can lead to confusion are explained.

Appendix B: Sweeping approach Succinct description of a geometric approach for structure synthesis which was explored but did not meet our requirements of interactivity. Nevertheless, the concept is interesting.

Appendix C The French version of the introduction (chapter 1).

Appendix D The French version of the conclustion (chapter 6).

Chapter 2

Related Work

"If you have an apple and I have an apple and we exchange apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas."

—George Bernard Shaw

As we saw in the introduction, we have come to demand a very high level of realism in computer generated images. To achieve this realism, computer artists have to build 3D scenes containing a lot of lifelike *details* such as complex geometric models, advanced illumination effects or photo-realistic textures. The set of diverse details added to a model defines its final appearance.

The approaches we propose build up upon a great deal of previous work in disparate areas. In this chapter, we start by describing work related to enhancing 3D scenes with geometric details, either by truly modeling, or by simulating them. We then expose various methods designed to relieve the user from tedious recurring work by helping him propagate details in his scenes. In this thesis, we are particularly interested in the case of details arranged in a structured way. We thus devote an entire section to the different approaches taken to generate such structured details. Finally, since being able to control the results is essential for the end-users, we finish by presenting techniques developed to this end.

2.1 Enriching computer-generated 3D worlds with details

2.1.1 Adding geometrical details

The first stage when building a 3D scene for any computer graphics production (whether it be a movie, a video-game or a visualization of any sort) is *modeling*. The computer artist constructs models composed of polygons, curves or any other surface representation primitives. The complexity of the resulting scene, typically measured in polygon count for modeling, will be highly dependant on the skills and patience of the artists and the quality of tools used. Furthermore, the complexity he can *afford* depends on the target application. For interactive applications, the polygon count cannot be on the same order as that used for production movies for example, where rendering can be done off-line.

Geometry costs a lot even if graphic cards power is rising more each year. As a lot of the geometry used in a scene is not crucial (background, distant or out-of-focus objects), it is common to use simplification methods to avoid having to render all the polygons, especially the unimportant ones. Some authors (e.g., [Cla76] [FS93] [Hop96]) have developed simplification heuristics to automatically discard polygons according to their importance. For example, objects moving away from the viewer will progressively decay and it should not be noticeable. Computer artists use the same trick by building the same model at various resolutions and exchange one model for another appropriately. These techniques are part of the vast research area on the *level of detail* problem, whose associated literature is extensive.

Artists can also take advantage of the *geometry instancing* possibilities of the software/hardware they are using. This feature allows the *reuse* of existing geometry (for example, a character in a crowd scene) while allowing small geometry or texture modifications to avoid getting an exact clone of the original model. Modeling time is greatly reduced as only one model has to be constructed in detail.

Another common approach used to reduce the geometry complexity (and thus modeling time) is to use billboards (or impostors) [MS95] [DDSD03] instead of real geometry for distant or small complex objects. A billboard is a flat, usually square, polygon on which an image of the object is mapped to simulate this object. Billboards are used extensively in video games. However, it is impossible to interact with these objects and the simplification becomes quite evident from a close viewpoint.

Some image-based methods [DTM96] [POF98], using photographs as input, have been developed to be able to recover geometry using computer vision techniques and user input. The integration of the user is indispensable as techniques relying *only* on computer vision to recover a scene geometry are not robust enough and their results have to be post-processed in order to be usable for CG purposes. The final quality of the image-based modeling process depends on the time spent on the task, but generally results in medium-resolution models whose details are simulated by texture maps as depth information recovery is limited to the image resolution. The quality of image-based extracted textures used to complete the models is also dependent on the resolution, viewpoint and availability of the input images. Another way to get precise geometrical details is to use scanning techniques (range scanning, laser scanning, optical scanning). However, they suffer from the same problems as computer vision techniques: a lot of manual processing has to be done on the output data in order to clean it, sometimes simplify it and make it usable. Most of the time, the huge amount of data to process requires a lot of (man) time, effort and memory storage.

The techniques mentioned in this section all aim at reducing the modeling time and effort. However, it would be very tedious to model all the subtle details of a real-world surface like a rock wall but it could be possible to generate a realistic impression by allowing to *perceive* this complexity. This has always been an issue in graphics and the next section presents common techniques used to add detail at a cheaper cost by *simulating* it.

2.1.2 Simulating details

For a lot of applications we cannot afford the number of polygons required to be able to model real-life complexity. However, a few interesting techniques exists to increase visual detail without increasing geometric resolution. The process called *texture mapping* [Cat74] [BN76] [Bli78a]

[Hec86] [Hec89] [HS93] allows the addition of non-geometric details to geometric models. The appearance of an object surface can be stored in a discrete, two-dimensional array, which we call a *texture*. Data elements in the texture are referred to as *texels*, indexed by their *texture coordinates*. In order to reconstruct a texture, given an image and a geometric description for the surface of an object, each texel is projected onto the image according to a given parametrization. Texture mapping operations are being optimized on current graphic boards to speed up the process [HL90] [DWS$^+$88]. The values in the image are sampled at the projection locations. The data in a texture can be interpreted in various ways, some of which we give an overview of in the following.

Color Textures

The first and foremost purpose of a texture is to model the color attributes of a surface. In this case, the values it contains are RGB color values obtained from a drawing created with a painting software or a photograph of a real-world pattern on the surface of an object (photo-realistic textures). This kind of texture is widely used and has the advantage of being simple to acquire. However, as a texture is a 2D map of values bounded in space, it has the main disadvantage of being resolution-dependent. When texturing a surface larger than the available map with a fixed-size image, this image has to be repeated in a way that must appear indistinguishable to the viewer as simply scaling (up or down) the texture lowers its resolution and its visual quality degrades. Simple tiling often fails and artists have to deploy all their talent (and time) to seamlessly cover the surface with the given texture as the human eye is very sensitive to repetitive patterns [Mar82]. Such texture maps also face problems intrinsic to the texture mapping process (i.e., parametrization problems) if the surface to texture is complex and getting a correct non-distorted result can be very difficult.

Using a real-world photograph directly as a texture can give a stunning impression of detail. However, there are some drawbacks to using this approach. The details are imprinted into the texture and there is no possibility of having geometry-dependent effects like shadows, self-illumination, self-occlusion or surface properties effects like differences in reflectance. In addition, if the lighting in the 3D environment is different from the lighting used to capture the texture map, the final rendering will appear incorrect and thus unrealistic. When the texture image colors are blended with the computed lighting for the surface it is applied onto, the result can look inconsistent and flat to the viewer.

Procedural textures [Per85] [Wor96] [EMP$^+$94] offer a solution to this problem as they do not consist of images but rather in a procedure which generates a texture to infinity, and is thus resolution-independent. The texture colors are computed using the given function and a variety of interesting patterns can be obtained by tweaking the procedure parameters. Their main problem is the lack of control which is something very important for artists as they often do not like to be stuck with the result of a process without being able to modify it. Examples of procedural textures are shown in figure 2.1.

Solid texturing [Per85] [Pea85] [KK89] can be seen as a form of procedural texture mapping. The idea behind solid texturing is instead of mapping a texture to the surface of the object, the color at a point in texture space is given by a function $F(x, y, z)$. The texture is defined in a reference frame given by the texture space. The object can be placed anywhere in this 3D texture space and two mathematical transformations are required in order to compute the color at each surface point. See image upper-right image in figure 2.1.

Classification Looking at the diversity of possible textures, we can extract three main groups:

Stochastic textures A stochastic texture can be viewed as a sample of a two-dimensional stochastic process, which can be described in term of its statistical parameters. Stochastic textures thus have and irregular but visually homogeneous structure. Natural textures, such as images of grass, can sometimes be represented by stochastic textures. Perlin [Per85] generates stochastic textures (amongst others). See lower-left image of figure 2.1 for some examples.

Structured textures A structured texture is composed of a set of primitives, whose spatial organization (local and global) is determined by a set of placement rules. Examples of such textures include tilings of the plane, cellular structures such as tissue samples, pictures of brick and stone walls, etc. Worley [Wor96] generates structured textures procedurally. See bottom row of figure 2.1.

Others Any other that does not fall in any of the two above classes. For example, the textures that are a photograph of an object, mapped on a flat polygon. See upper-left image of figure 2.1.

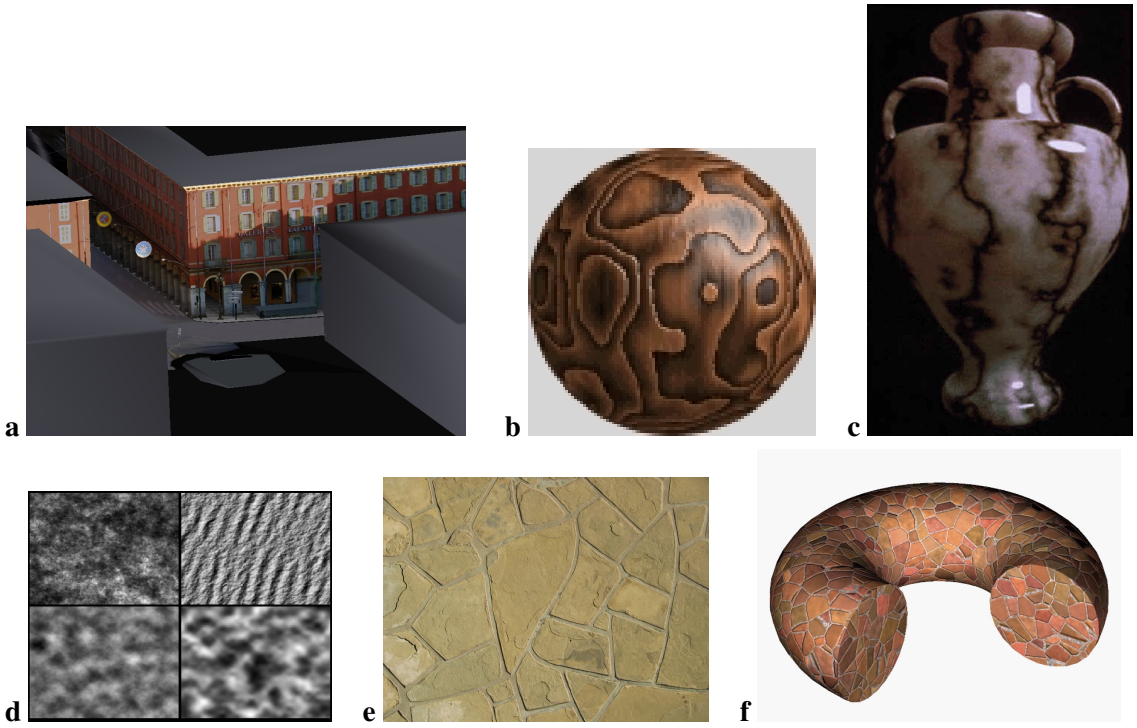

Figure 2.1 Top row: Left, texture based on a photograph; center: a procedural texture; right: a solid texture. Bottom row: Left, some stochastic textures; center, a structured texture; right, a procedurally generated structured texture.

If the model underneath a texture is pretty simple (i.e., not a lot of polygons), the flatness of the surface will be immediately noticeable when the surface is viewed closely or from a grazing angle. Many techniques have been proposed to overcome these limitations. Amongst the most widely known and used to simulate surface variations are bump mapping and displacement mapping which we describe next.

Bump and Displacement Textures

Two main techniques to simulate fine geometric detail on a simpler base surface, based on the notion of textures, are widely used in computer graphics: *bump mapping* and *displacement mapping*.

The principle of bump mapping [Bli78b] [Bli78a] is simple: a texture (the *bump map*) contains modifications which are applied to the normal vector during illumination computations, simulating lighting changes. As a result, highlights on the surface appear when lights are moved around due to the use of the modified normal in the lighting equation. However the geometry is left untouched and the flatness of its surface and silhouettes is easily noticed when the surface is viewed from very close, or from a grazing angle. Moreover, surface shadows originating from self-shadowing and highlights due to surface interreflections cannot be simulated.

Displacement mapping [Coo84] [CCC87] goes one step further by actually displacing the geometry according to height values contained in a displacement map. Each texel thus stores a distance instead of a color. This is typically a grayscale texture where black and white corresponds to maximal displacement values in the direction (and opposite direction) of the interpolated surface normal. The modifications of the geometry according to these values happen at rendering time, when the surface is tesselated in thousands (even millions) of micropolygons whose vertices are subsequently displaced according to the values in the map at that location. The lighting computations thus give a much more realistic look than bump mapping and silhouettes appear correctly. However, the process used to take so much space and time that it was not generally used in production. Recently, displacement mapping has been integrated in the graphics hardware [DKS01] and can thus become a viable alternative to bump mapping.

Figure 2.2 shows the difference between bump and displacement mapping. True relief is possible with displacement mapping. Notice especially the differences on the object silhouette.

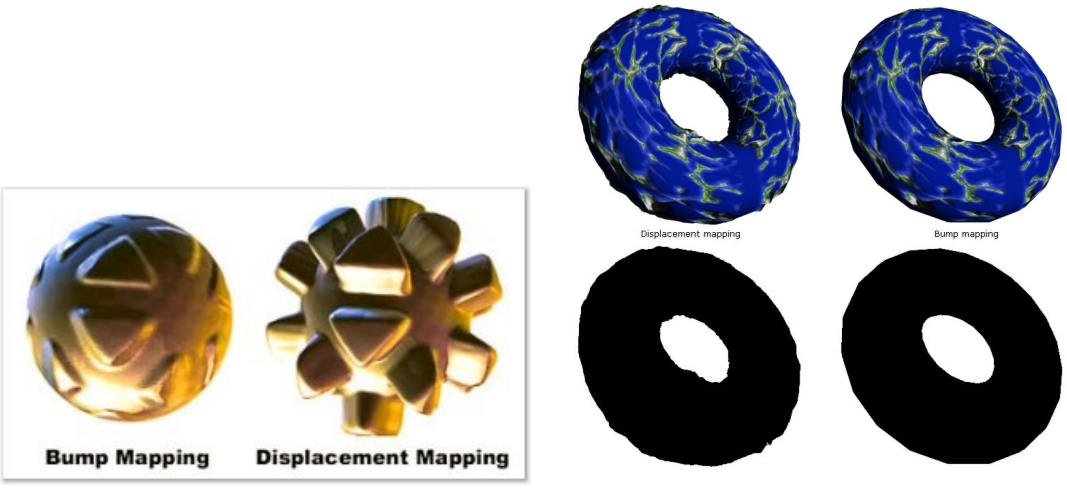

Figure 2.2 Bump mapping vs displacement mapping.

Bump and displacement maps can be either modelled by hand or generated procedurally. As there is no new geometry created but only a 2D map of values, the size of the new data is lightweight. However, creating such a map for a real world texture captured by a photograph is a difficult task. Some methods attempt to automatically generate these maps from a set of input images taken with various known light directions [RTG97] but they can only handle objects with small surface variations.

Augmented textures

Many interesting methods have been developed to augment basic texture objects with additional information to produce more realistic details. Without going into details and for the interested reader, we can enumerate the following techniques:

- Hypertextures [PH89]
- Reaction-Diffusion textures [WK91] [Tur91]
- Cellular textures [FLCB95] [Wor96] (described in section 2.3.2)
- Relief texture maps [dON00]
- Polynomial texture maps [MGW01]
- View-dependent texture mapping [DYB98]

All the techniques described in this section aim at simulating details by applying more or less complex textures to the scene models. However, to avoid having to repeat the process, some research has been done to generate detail automatically or imitate and propagate detail samples at will. We give an overview of these techniques in the next section.

2.2 Propagating details

Details are often considered as the high-frequency parts of images or geometric models. Creating these details can thus be a tedious process. Therefore, a natural demand is to be able to analyze examples of detail in order to seamlessly resynthesize similar details at will. In this section, we separate the techniques which aim at synthesizing textures from example, from the approaches which try to recover other kinds of details from example.

2.2.1 Detail recovery

Some authors have addressed the problems of enriching scenes with details by exploiting details already present in the scene or examples provided. We describe here some of these papers although the notion of detail is so vast that we cannot possibly cover all aspects of it. We will not describe approaches which build a specific model for a particular detail (for example, synthesis of bark [LN02]) but rather approaches that are more generic or which can be easily generalized.

Height A lot of research has been done to recover height information (e.g., bump maps) from example, either using a single image and some simplifying assumptions [DMG02] [LP00b] up to methods using sophisticated measurements techniques, multiple images or varying illumination [DvGNK99]. The main challenge faced by all these methods is the fact that the illumination is included in the images and decoupling the real color from the lighting color is still a critical research problem. In computer vision, a large literature is devoted to the "shape from shading" problem (e.g., [Bro89] [ZTCS99]) which is a vast research area.

Geometry Dischler and Ghazanfarpour [DG99] present an interesting analytical approach for the synthesis of macro-structured textures (i.e., textures with small oriented shapes on their surface). They recover the shape and appearance of these small geometric structures through the analysis of two orthogonal views and use this information to deform appropriately a series of generalized cylinders. The distributions, sizes and orientations of the macro-structures on the surface can be controlled interactively, which represents an improvement over automatic texture synthesis techniques which often distribute the features uniformly, without any possible user-interaction (such as the technique of Fleischer *et al.* [FLCB95] described in the next section). A result from their technique can be observed in figure 2.3.

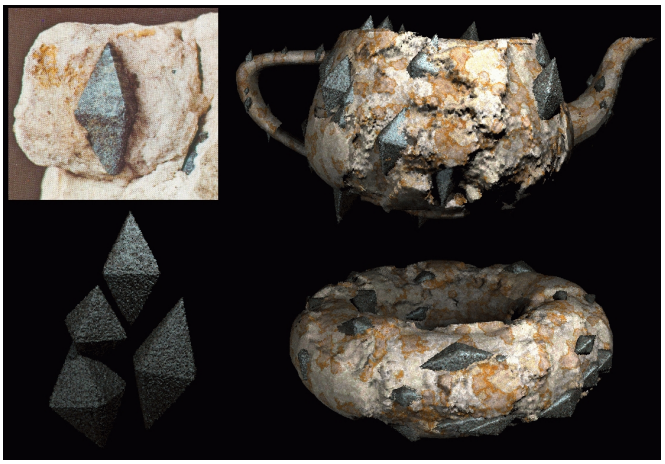

Figure 2.3 Interactive Image-Based modeling of macro-structured textures: The image in the top-left is analyzed to extract the shape, color and bump information of the macro-structure (seen below). The two models on the right were textured and rendered with the resulting macrostructured texture information. (Image from Dischler *et al.* [DG99])

Bhat *et al.* [BIT04] build on the image analogy approach [HJO$^+$01] (and thus texture synthesis) to learn a 3D geometric pattern from one example and copy this pattern to another model. The pixel neighborhood searching is transferred to a voxel search using user-defined vector fields on the input and output models. The geometry is directly transferred on the resulting model. Figure 2.4 shows a seahorse model whose geometry has been modified using the analogy from the cylinder pair above (with and without grooves).

Wang *et al.* [WWL$^+$03] present an appearance-based method to model the complex visual aspect of tree bark surfaces from a single image. From an input bark image and some user interaction they output a *plausible* height field and texture which allows the generation of similar bark. They perform a user-assisted texton analysis to segment the bark image into regions corresponding to different features. An interactive editing tool is then used to build a height field for each feature using similarity-based editing [BD02], which is adequate for a realistic rendering. A texture is generated using the height field, the input image and an heuristic to decouple the color from the illumination and map a color pixel to a height pixel. We think their approach fits the artists work methods and could be generalized to other kinds of detail.

Resolution Ismert and Bala [IBG03] aim to reintroduce high-frequency details in textures coming from image-based modeling techniques. These details have been lost due to the resampling process when applying the photograph to the virtual models, or because the original point of view of the camera was not optimal. They build on a texture synthesis technique [EL99] to synthesize missing detail (and not an entire texture) using a sampling-based metric to evaluate the location

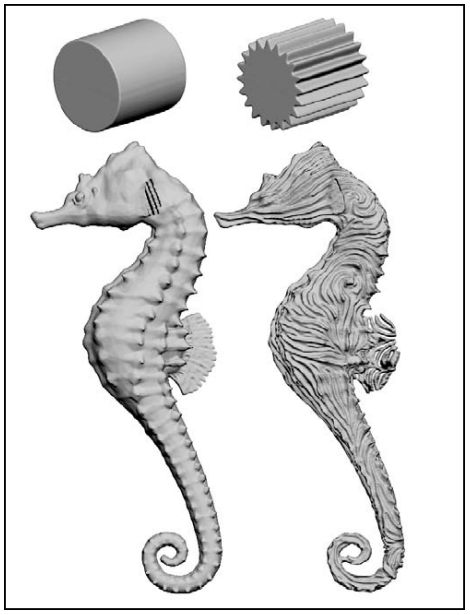

Figure 2.4 Geometric synthesis by example: The seahorse geometry is modified according to the analogy represented by the cylinder models pair, seen above. (Image from Bhat *et al.* [BIT04])

of regions of low-frequency data in the images. Their main contribution is the introduction of this metric, which is based on the Jacobian of the imaging transform. A result of their technique can be observed in figure 2.5, where they increased the resolution of image-based textures taken from a far viewpoint.

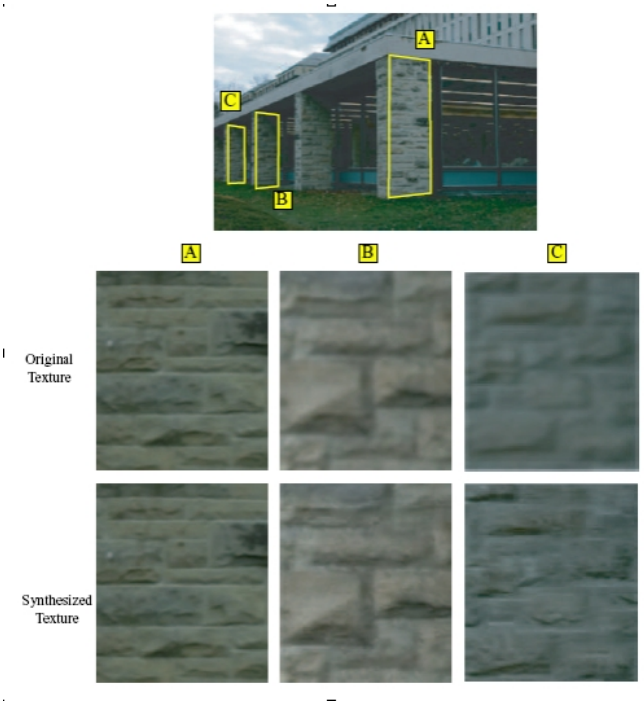

Figure 2.5 Detail Synthesis for Image-Based Texturing: Resolution lost in the farthest textures is reintroduced by using the more detailed texture of the front column. (Images from Ismert and Bala [IBG03])

Other authors have addresses this "super-resolution" problem using image analogies [HJO+01] to paste high-frequency details over their low-resolution version or more advanced learning models [FJP02].

2.2.2 Texture synthesis

Since humans are very sensitive to repetition in patterns, simple tiling of the texture is not always a good solution. Texture synthesis techniques aim at recovering the characteristics of an infinite texture by the analysis of an input sample, in order to generate output textures which have the appearance of the sources (see figure 2.6). Texture synthesis approaches have received a lot of attention in the recent years. Since they aim to synthesize and propagate details on arbitrarily large surfaces without too much effort, they fulfill a real need for the artists who usually build these textures by hand.

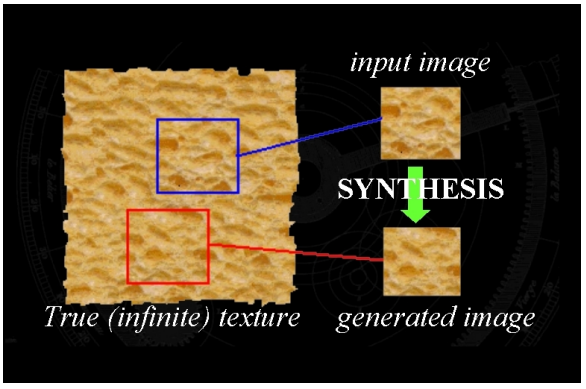

Figure 2.6 The texture synthesis approach. (Image from Efros *et al.* [EF01])

Texture synthesis approaches can be classified into three main categories: *statistical*, *pixel-based* and *block-based*.

Statistical approaches [Bon97] [HB95] try to recover the underlying statistical process at the origin of the (generally stochastic) texture in order to generate new images having the same kind of features. These techniques are efficient but do not preserve the local or global structure.

Pixel-based approaches [EL99] synthesize a new texture so that it is locally similar (pixel neighborhood comparisons) to an example texture patch. The new texture is generated pixel by pixel, and each pixel is determined so that local similarity (computed using the pixel neighborhood) is preserved between the example texture and the result image. This process is slow because of the large search space but optimizations have been developed [Ash01] [WL00]. They also do not preserve global structure.

Block or patch-based approaches [GSX00] [EF01] [LLX+01] [KSE+03] limit the search space by considering blocks of pixels instead of single pixels and apply special techniques to reduce overlap artefacts between the chosen patches. Depending on the patch size (fixed or not) and shape, they can preserve the local structure, to a certain extent, as well as some individual details. Some [KSE+03] are better than others in preserving global structure; overall these approaches tend to introduce unwanted visual artefacts at patch boundaries. In this work, we use the Efros *et al.* [EF01] approach and describe it in more detail in section 5.3.

Hybrid methods [NA03] combining the advantages of pixel based and patch based produce even better results for global structure conservation. Some approaches use multiple passes at increasingly finer resolution to improve the quality of their results.

Some results of these various approaches are presented in figure 2.7 and 2.9.

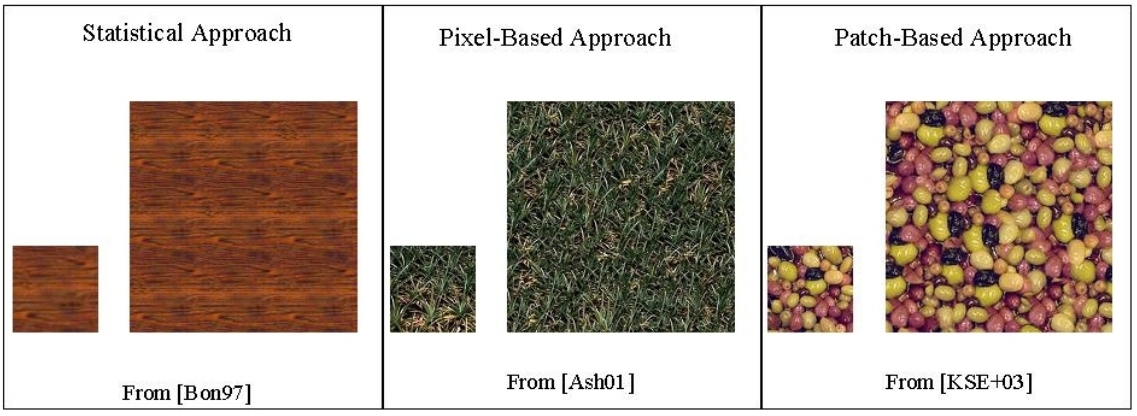

Figure 2.7 Results for the three-types of texture synthesis algorithms.

Some of these algorithms perform "constrained" texture synthesis [Ash01] [EF01] [HJO+01] (also called "texture transfer" by some authors) using, in addition to the input texture sample, a *guiding image* of the same size as the desired output texture. This image will influence the choice of the pixels (resp. blocks of pixels) during the synthesis. The candidate pixel (resp. block) has to respect synthesis constraints (such as ensuring local similarity with the already chosen pixels (resp. blocks)) but also has to match the color of pixels (resp. blocks) of the guiding image. This technique gives results such as the one in image 2.8 where the black and white photograph was used to control a texture synthesis using the small fabric sample texture, to give the results shown to the right.

These approaches first addressed 2D texture samples, producing 2D texture outputs that could then be applied to 3D models. However, the parametrization problems inherent to texture mapping imply that it is more efficient to directly generate the texture onto the models, taking the geometry into account during the process. Interesting techniques have been presented using this idea [PFH00] [SCA02] [NC99] [YHBZ01] [ZG04a].

One of the main problems of texture synthesis is that a lot of new information (pixels or blocks of pixels) is generated out of a small sample. However, all this synthesized data is already present in the sample image so a lot of memory is wasted. Soler *et al.* [SCA02] avoid this problem by only generating, for every mesh vertex of their models, texture coordinates indexing into the sample rather than rearranging existing color data and mapping a new, much larger, texture. Large models can be textured out of a small texture sample thus achieving important memory gains over traditional texture synthesis techniques and an increase in rendering efficiency.

Texture synthesis techniques look really promising by their results. However, to our knowledge, none of the existing published techniques are available in commercial packages and one might wonder why as the exposed results seem to solve the problem pretty well. One plausible

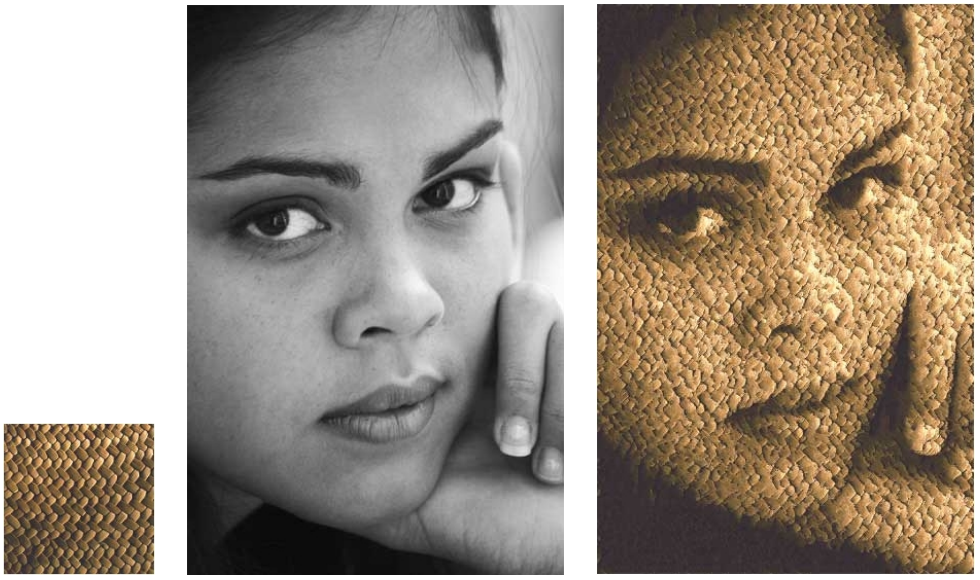

Figure 2.8 Using a guiding texture (middle) to influence the result (right) of the texture synthesis process using a sample (left). (Images from Freeman [EF01])

explanation is their lack of *usability*. The results of these approaches are still highly unpredictable and uncontrollable. After experimenting and investigating some of them, we summarize here the main problems that we encountered when using the current methods:

- The algorithms we tested are really sensitive to the choice of input parameters (e.g., size of block for block-based techniques, size of neighborhood for Markov field type of approaches, thresholds and non-intuitive parameters of all types, algorithm starting point) and they require an often tedious tweaking of these parameters for each different input texture they have to handle.

- Markov random fields models are known for their lack of stability. If the chosen pixels start to deviate too far from those present in the input texture, the algorithm can get stuck in a space where only garbage is produced and it is not until the end of the algorithm, when the result is displayed, that damage can be evaluated. Thus, a great deal of time can be lost especially if the problems have started appearing at the beginning of the algorithm, where it could have been corrected (or the execution cancelled) in time.

- Most current techniques fail on textures which display a high degree of structure. They often excessively blur or mismatch the objects boundaries. Figure 2.9 shows various examples of synthesis failures, where global structure is lost, boundaries are not preserved, etc.

- Statistical techniques assume some probability distributions of the texture pixels and aim to recover the underlying distributions so as to synthesize a texture having the same stochastic properties. However, this assumption cannot be true for structured textures.

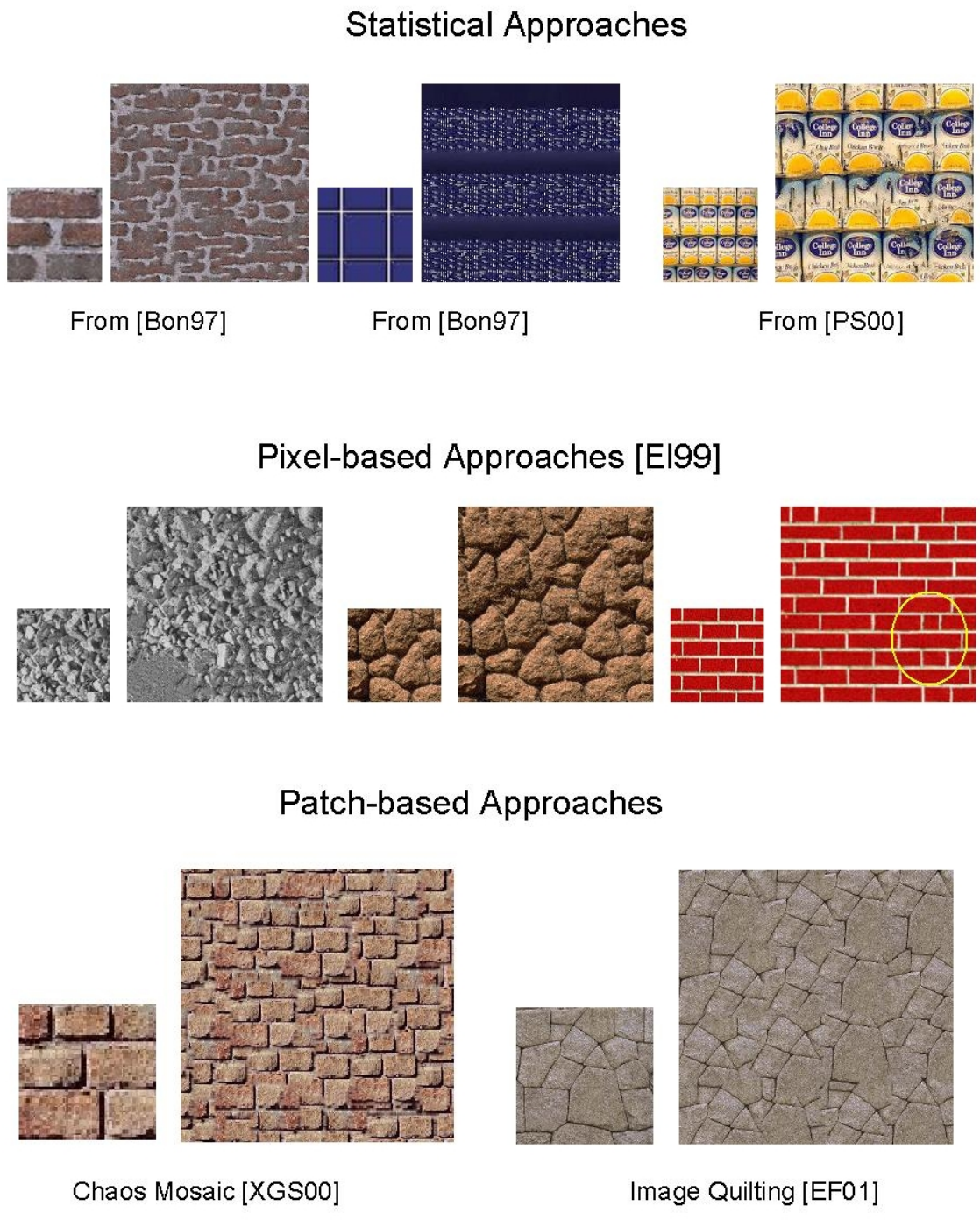

Figure 2.9 Some failures of current texture synthesis algorithms on patterns and structured details. Pattern coherence is lost, structure regions are merged, cut or invalid, boundaries are blurred or mismatched, algorithm becomes disoriented and produces garbage.

As we can see, the methods illustrated here fail on patterned and structured textures (for example, rock walls) and, to our knowledge, there is no texture synthesis algorithm which performs well on structured textures composed of irregular shapes. We believe this is mainly because a higher-order process has generated the structure and methods that operate at the pixel color level cannot capture back this process. The structure boundaries are often lost/ignored and the overall structure is destroyed. Two very recent works seem to address this problem in particular and get better results on structured textures than any other methods. They are described in more details in section 2.3.3.

Amongst all different types of textures and because of the current difficulties to propagate them, we focus in this thesis on textures composed by a series of variously-sized primitives, organizing the details with an underlying *structure*. There are plenty of such structured textures around us (see images 2.12 and 3.1 for some examples). The following sections give some background and insight on how structure and structured textures have been considered in research up to now.

2.3 Structuring detail

The word "structure" is very generic and can have a variety of meanings. In this work, we mainly adopt the following simple meaning: *A structure is an arrangement of parts. It is a manner of organization.*

Visually, the structures we are interested in are composed of two main elements: the primitives and their spatial organization. These two components are independent and we will treat them as such.

In the computer graphics field, structure is present, either in the geometry itself (2D or 3D) or in the patterns used to enhance it. A structure can be considered to be a semantic property of a texture. A brick is an object, a structure composed of bricks is a wall. In what follows, we describe some geometrical notions of structure and some elements from Computational Geometry. We then explore various ways that have been used to create structured textures in computer graphics and finally describe the methods that have attempted to build structured textures from the analysis of examples.

2.3.1 Geometrical aspect

Computational geometry mainly addresses problems in discrete geometry, where points, lines and segments are used to approximate continuous elements like curves. Discrete geometry is a basic tool for computer graphics, which also uses the same discrete structures to display discrete images in the form of pixel arrays.

The geometric properties of various types of structures have been studied extensively in computational geometry [OBSC00]). Geometrically speaking, a structure can be defined as a partition of the Euclidean space into disjoint space filling cells which we call *regions*. Depending on the domain considered, this partitioning can take various names. It can be a tesselation, a subdivision, a partition, an arrangement, a CW-complex, etc. In this work, we use the following definition: "A 2D *cellular subdivision* defines a partition of the plane into regions whose shape can be governed by some measure". More formally, a cellular subdivision can be defined as follows:

Let B^0 denote the interior of a set, B, in d-dimensional Euclidean space $\Re^d$. A subdivision, S , in $\Re^d$ is a countable set of regions $X_i \subset \Re^d$ such that:

1. ${X_i}^0 \cap {X_j}^0 = \emptyset$ for $i \neq j$.
2. $\cup_i X_i = \Re^d$
3. $\#\{X_i \cap B \neq \emptyset\} < \infty$ for all bounded $B \subset \Re^d$

The first two conditions specify that the regions partition the space, and have disjoint interiors. The third guarantees that the number of regions is countable. Indeed, we only consider finite subdivisions consisting of a fixed number of regions on a compact subset of the plane.

One of the most well-known and studied cellular subdivisions is the *Voronoi Diagram*, which we describe below. It is of particular interest to us, as we use a variant of a Voronoi Diagram to model our structures (chapter 3).

Voronoi Diagrams

A Voronoi diagram of a vertex set, called generators, is a subdivision of the plane into polygonal regions (some of which may be infinite), where each region (the Voronoi region) contains the set of points in the plane that are closer to its generator than to any other generator. Voronoi diagrams have a wide range of practical applications, and there are efficient well-known techniques for computing the Voronoi of a set of points. The *ordinary* Voronoi diagram uses a simple Euclidean distance metric and leads to convex regions of valency 3 (i.e., adjacent to three other regions). A typical Voronoi diagram is shown in figure 2.10, left.

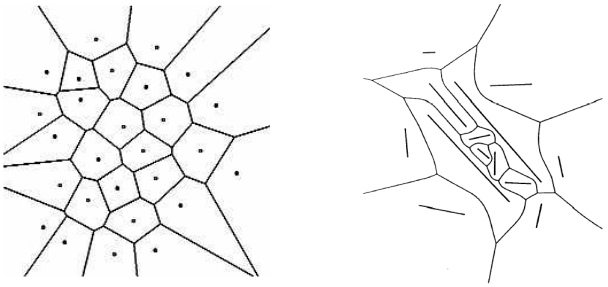

Figure 2.10 Left: A traditional Voronoi Diagram. Right: A Voronoi Diagram of line segments.

If the generators are line segments, instead of points, we can still compute a corresponding Voronoi diagram. We simply need to provide an appropriate distance function to compute the regions whose points are closer to their line segment generator than to any other. A Voronoi Diagram of line segments is shown on the right image of figure 2.10. This can be further generalized by letting the generators be any higher-order geometry, such as a polygon for instance. In this case, we need to provide a way to measure the distance between a point and the generator. When the generators are points, the Voronoi regions boundaries are straight lines. More complex generators can result in curved boundaries, and more interesting region shapes. These diagrams are referred to as *generalized* Voronoi diagrams.

In addition to the location and shape of the diagram generators, the shape of the Voronoi regions can be controlled through the distance metric used. Instead of the usual Euclidean distance one can consider the more general concept of convex distance functions, which leads to not necessarily convex Voronoi regions with valency $\neq 3$. For any convex shape, there is a corresponding convex distance function. If the convex shape is symmetric, then the resulting distance function is a metric. If not, the result can still be used to define a Voronoi diagram. The convex distance is useful to solve the problems that cannot be solved using the Euclidean metric. Ma [Ma00] describes how to compute a Voronoi diagram using a convex shape as a distance metric (see figure 2.11).

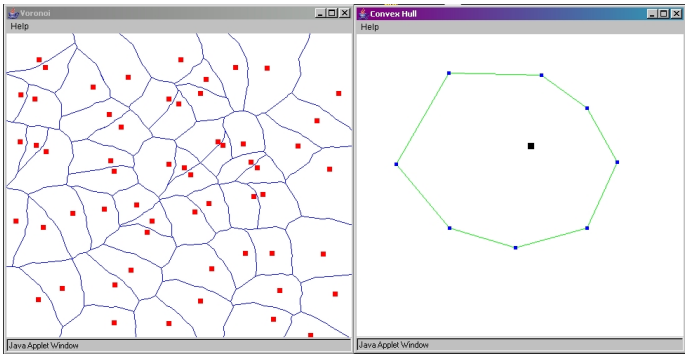

Figure 2.11 Voronoi diagram (left) computed using a convex distance defined by a convex polygon (right). (Images obtained using the applet of the GeometryLab [geo] of the University of Bonn)

2.3.2 Structured texture generation

If a graphic designer wants to create a texture of a stone wall or a pavement such as those illustrated in figure 2.12 (or any kind of structured texture such as the ones of image 3.1) in order to enhance a CG model, he has a few options, which we discuss.

Figure 2.12 Some real-world patterns which could be used to texture a virtual 3D model. (Top left image from Miyata *et al.* [MIS01])

Geometric approaches

An artist can draw the structure by hand and then fill the regions with a texture, e.g., a rock texture such as the one in figure 2.13. However, drawing the structure by hand can be very tedious if the surface to cover is large. Moreover, using only one texture for the appearance is not really realistic as its repetition will be noticeable, especially if the texture is small. The best thing would be to have a few similar textures with various colors and features to avoid the repetition artefacts. This, as well as further modifications, are very time-consuming tasks, any a posteriori modification of the structure incurring similar additional work.

Figure 2.13 A rock texture.

Another option could be to draw a set of different desired region shapes and try to pack them (by hand or using an algorithm) into the available space. However, if the shapes are not simple shapes such as constant-size circles or squares, this problem (known as the *polygon-packing* problem of which we can see an example in figure 2.14) is known to be NP-hard [FPT81].

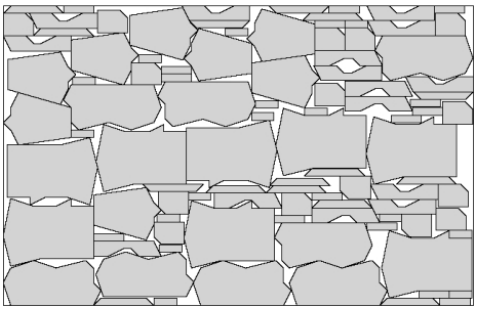

Figure 2.14 A polygon-packing problem example.

Finally, an artist can take a photograph of an existing structure or draw an example structure and use this sample to tile it on the target surface. However, repetition may still be an issue, there can be visible seams in the final texture and the border matching process can be a lot of work. Finally this sample can be used as input to the various texture synthesis algorithms proposed in the recent years (see section 2.2 for the description of such methods) but most of them fail on structured textures or are very difficult to tweak to get a good result.

Voronoi-based approaches

Voronoi diagrams constitute a very powerful tool for partitioning space into a set of regions. Many different kinds of Voronoi diagrams exist and the book by Okabe *et al.* [OBSC00] is an excellent reference on various types of Voronoi diagrams. As we saw in the previous section, a Voronoi Diagram can be built out of a set of generators associated with a distance metric. Variations can be obtained by modifying the shape of the generators or the distance metric used. Figure 2.15 shows two examples of Generalized Voronoi: the left one is a Weighted Voronoi, while the right one is also a Weighted Voronoi with variously shaped generators.

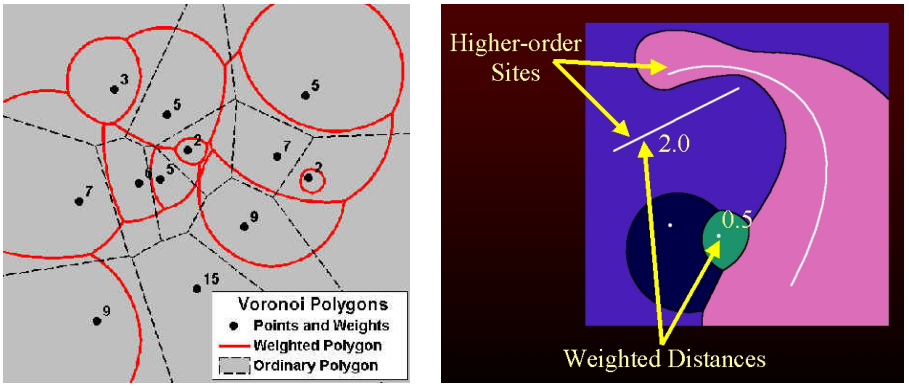

Figure 2.15 Two generalized Voronoi: variations on the distance metric and the shape of generators. (Image from Hoff [KEHKL+99])

The main problems of using Voronoi Generators of varied shapes in order to obtain a structure is that Generalized Voronoi Diagrams are very hard to compute analytically. Moreover, the more complex they are, the less predictable the resulting diagram aspect will be so they cannot really constitute a useful tool for artists to get a structure as they need to have control on the result of a tool.

However, some authors have used Voronoi Diagrams for specific tasks. Secord [Sec02b] uses Voronoi Diagrams to determine the position of stipples for the generation of stipple drawings from CG models. See figure 2.16, left. Hausner [Hau01] simulates decorative mosaics (figure 2.16, right) by computing a Centroidal Voronoi Diagrams (described in more detail in section 3.2.1) using a Manhattan distance metric. This metric causes the Voronoi Regions to have a square shape. The tiles are aligned along user-drawn direction fields by the Voronoi computation process and their color is sampled from an original image. His method gives nice results and is appropriate to its target application. However, it is not really generalizable to other shapes.

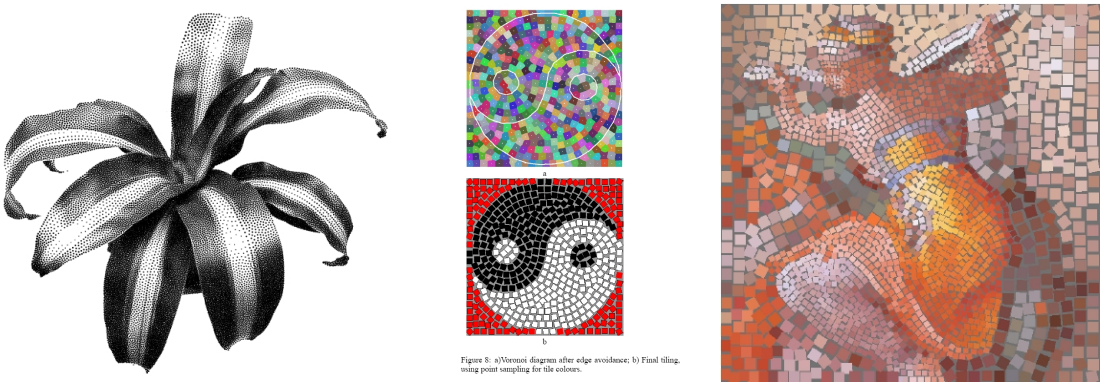

Figure 2.16 Left: Stipples are placed on a model using a Voronoi approach. (Image from Secord [Sec02b]). Right: Hausner uses a Centroidal Voronoi Diagram with a Manhattan metric to arrange square tiles on the edges of drawings to simulate mosaics (Images from Hausner [Hau01]).

Another interesting work using Voronoi Diagrams to generate structured textures is the one of Walter [WFM01] [WFR98]. He simulates the creation and growth of mammalian patterns (especially reticulated giraffes and tigers) using a biologically-plausible generation model. This procedural "Clonal Mosaic" model [WFR98] is based on cell-to-cell interactions and divisions

rules. Cells are modeled by points, whose Voronoi diagram is computed to generate the tesselation of the surface. Results visually similar to real-world patterns can be obtained, as can be seen on image 2.17. They state their approach could be easily generalizable to simulation over arbitrary surfaces.

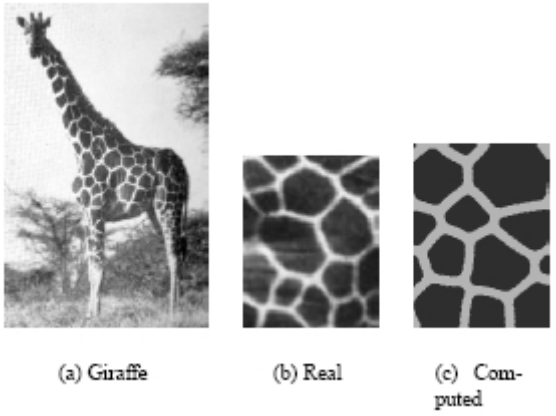

Figure 2.17 Generation of giraffe skin patterns using Voronoi diagrams (Image from Walter [WFM01]).

Procedural approaches

Several papers have addressed the problem of generating structures automatically, using a set of parameters to allow some variety in the results. This procedural approach seems like a natural one to take when thinking of the way the artists work to create their scenes.

Yessios [Yes79] presents methods to draft stone walls and wood patterns and output the corresponding two-dimensional line patterns. A first approach generates a structure stone-by-stone by generating rectangles one after the other using the available space. The structure is constructed in this way layer by layer, as we can observe in figure 2.18.

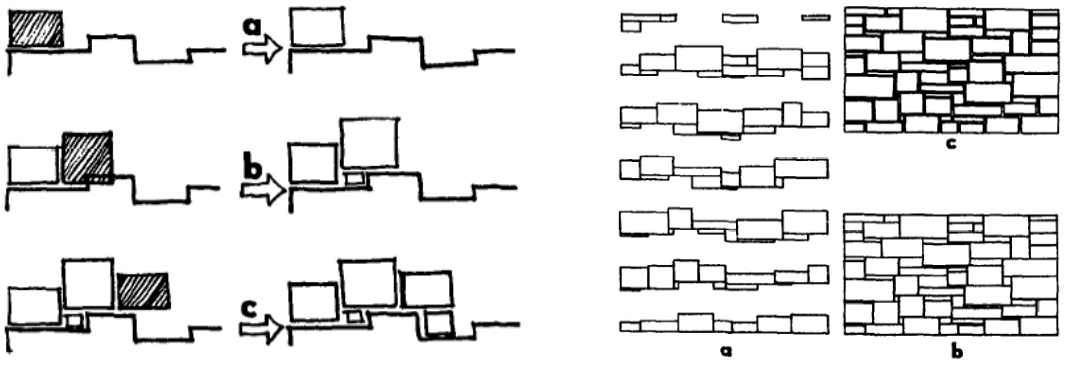

Figure 2.18 Yessios' technique to generate a wall stone-by-stone. Each layer is generated stone-by-stone (left and right, a) to build the final structure, whose interstice size can be varied (b,c) (Images from Yessios [Yes79]).

A second technique builds the structure by first building the layers, then adding the interstices and perturbed using noise functions. By varying the algorithm parameters, he can obtain a variety of wall-like line patterns as can be seen in figure 2.19. However, due to their construction process,

the structures are limited to layered structures and there is no possible individual control on the region placements and their shape.

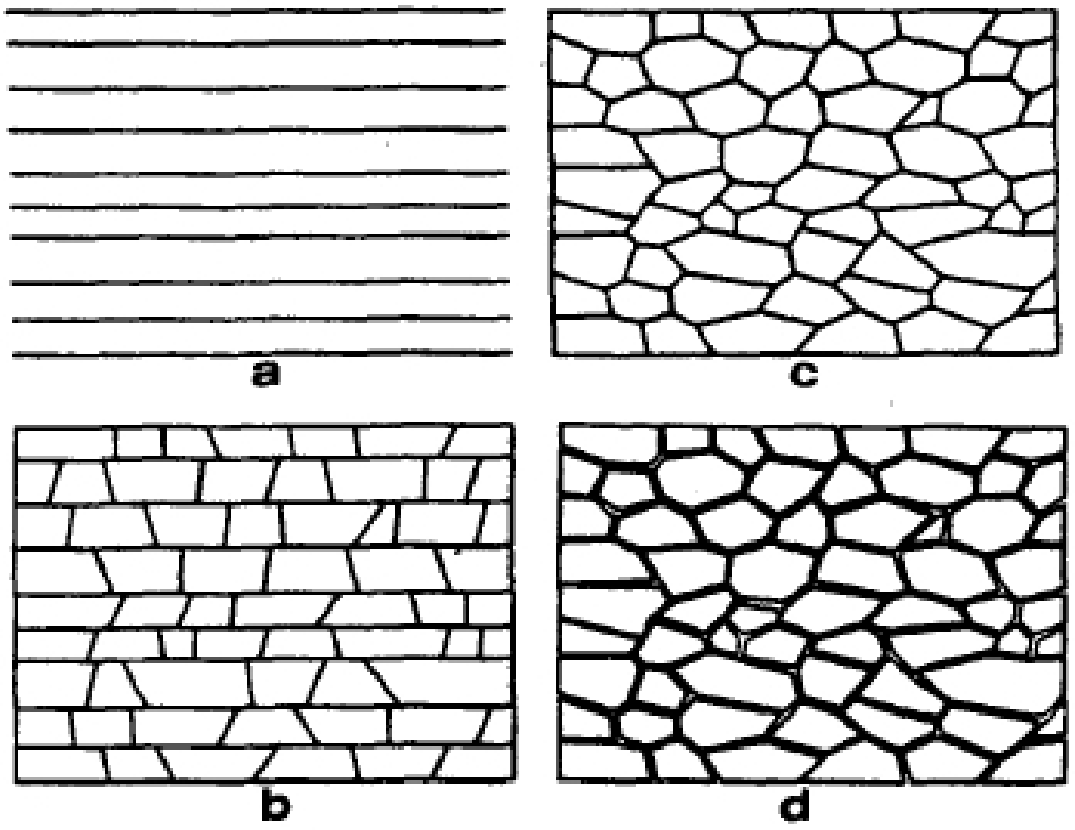

Figure 2.19 Yessios' technique to generate a wall line-by-line. Layers are generated first using 2D lines (a), then stones are added in each layer (b). Lines patterns are perturbed (c) and interstices are created (d) to lead to a final wall image. (Images from Yessios [Yes79]).

Miyata [Miy90] went further in this direction by adding a third dimension to the generated 2D structure and output 3D stone wall patterns that could then be applied to 3D models as seen in the right image of figure 2.20. The average size of stone, the roughness of its surface, the variance of its size are input parameters that can be used to vary the aspects of the results. Notice, however, the "gift wrap" aspect of the textured model.

Legakis *et al.* [LDG01] present an interesting procedural modeling approach to apply cellular textures to 3D models. In this work, the placement of the cells (bricks) is coupled with the underlying geometry onto which they are applied. By placing first the most constrained cells (corners, edges), they avoid problems that can happen at the model boundaries. However, the user sometimes has to guide the choice of the bricks and each new model to texture requires a new pattern generator to be written which is more of a programmer task than an artist's. See result images in figure 2.21.

Cellular approaches

In follow-up work of [Miy90], Miyata *et al.* [MIS01] adopted another approach to synthesize other kinds of structures. Pavements are generated according to some contour constraints and

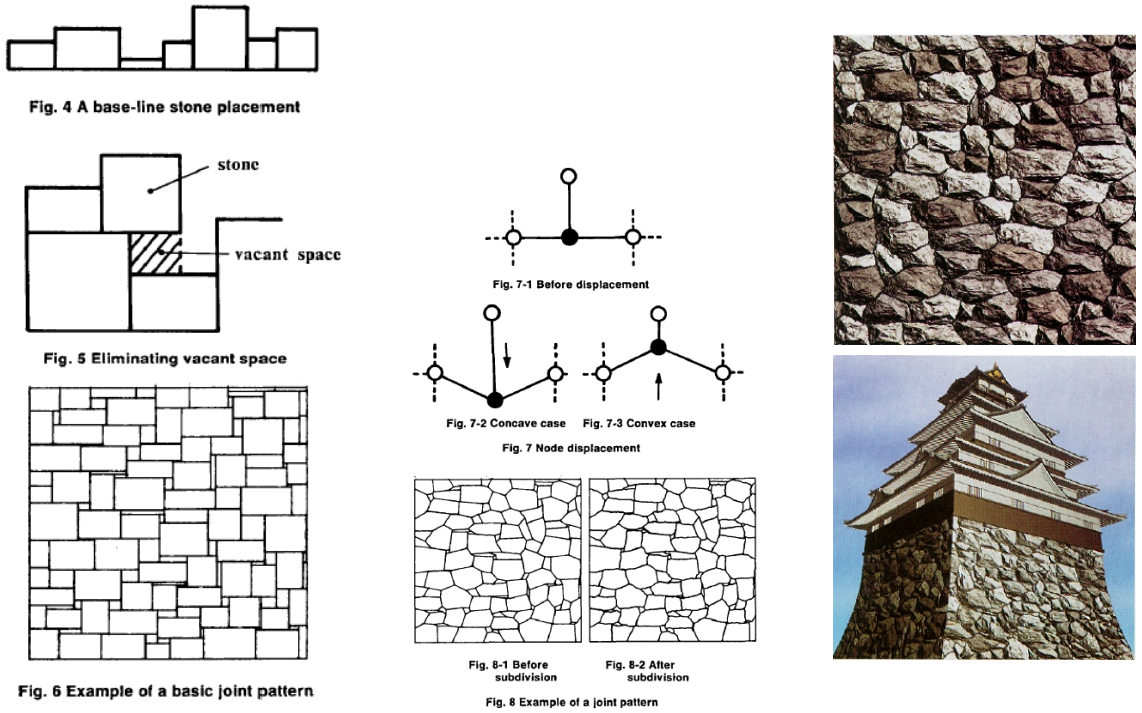

Figure 2.20 Left: The structure is built line by line by generating rectangles fitting in the available space. Middle: The edges are randomly perturbed to generate the stones. Right: The appearance (2D and 3D) is randomly generated and the final texture can be applied to a 3D model. (images from Miyata *et al.* [Miy90])

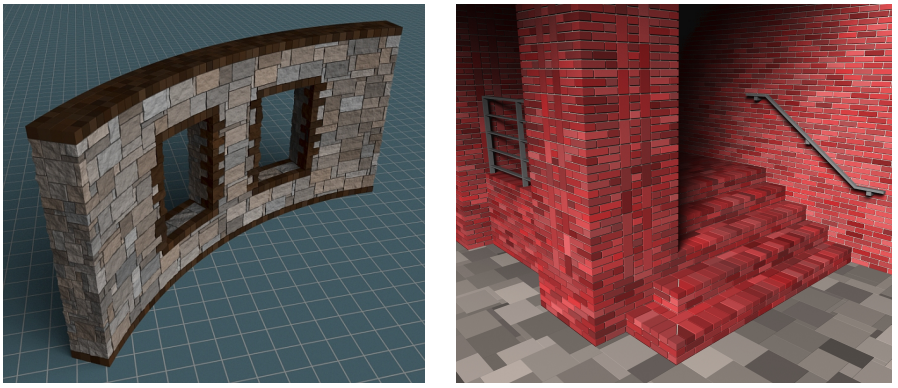

Figure 2.21 Models textured with two different pattern generators (Images from Legakis [LDG01]).

cell-packing techniques are used to make the 3D stones fit into the given contour, as seen in image 2.22. Cell-packing is a cellular approach which consists in applying particle system techniques to entities which constitute the final structure regions. Each particle has an energy potential which is computed from particle-to-particle distance and direction. The total energy potential is minimized iteratively until some convergence is reached. This technique has the advantage of being able to respect contour constraints for the target surface. However, it can only be applied to simple, similar shapes like circles or squares, whose contour can be disturbed afterwards to create more disparity.

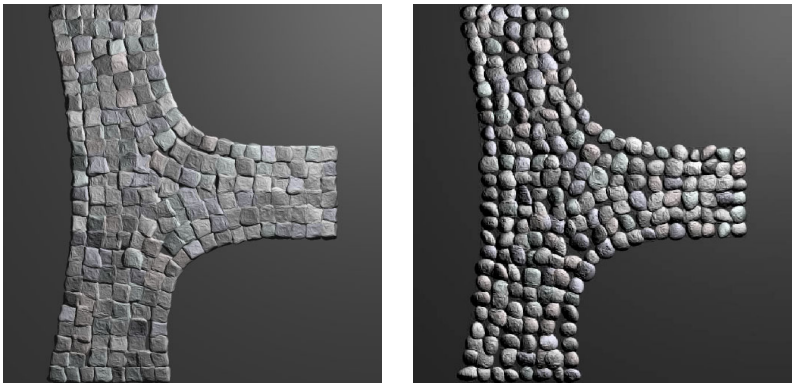

Figure 2.22 Cell-packing techniques used to generate pavements on a target domain. The stone aspect is generated randomly. (Images from Miyata *et al.* [MIS01])

Later, Itoh *et al.* [IMS03] applied and enhanced the cell-packing technique for the generation of organic textures such as animal skins (figure 2.23) by allowing the control of the anisotropy and directionality of the generated cells. The fine details of each generated 3D cell were created using fractal noise, generated according to a set of predefined rules. A big disadvantage of this technique is that a lot of geometry is created for a small texture sample, on the order of millions of triangle for a small texture patch.

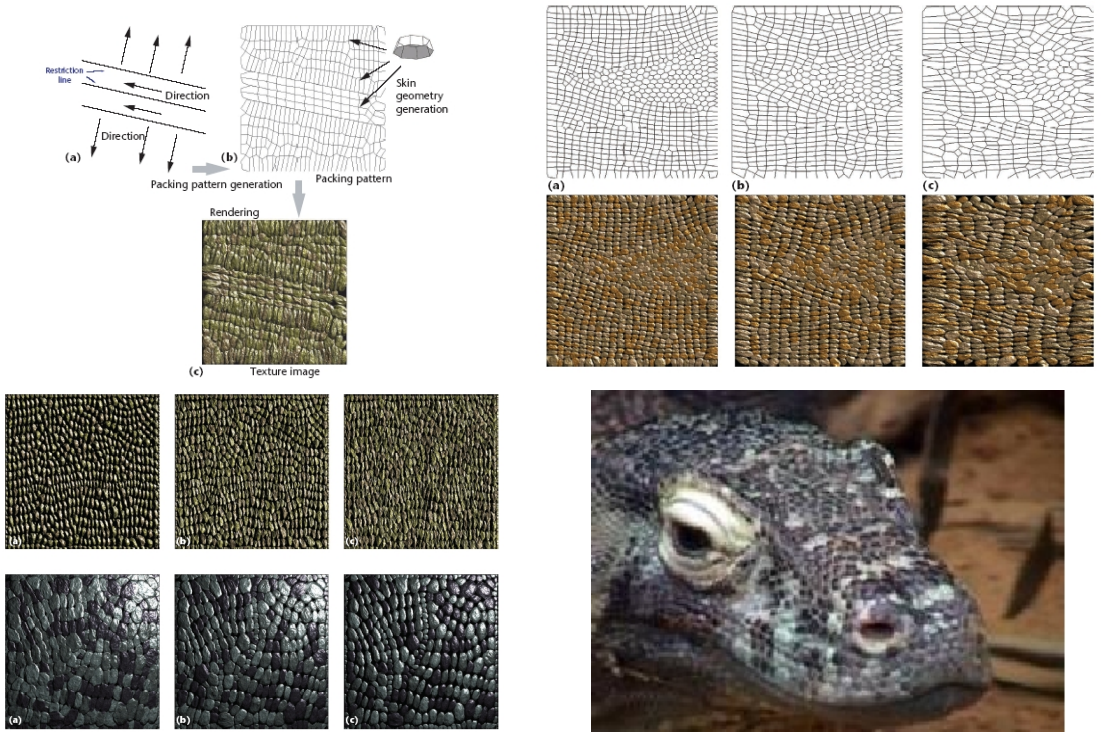

Figure 2.23 Generation of organic textures using cell-packing with anisotropy and directionality constraints. (Images from Itoh *et al.* [IMS03])

Fleischer *et al.* [FLCB95] also describe a cellular texture method to model surface details such as scales, feathers or thorns. Their method creates cellular particles constrained to lie on a 3D surface and whose arrangement (location, orientation) is computed using an energy-optimization

technique. The cells are then converted to 3D geometry using user-defined appearance parameters. Some results are shown in image 2.24. Particle systems work well for natural structured textures such as animal skins. However, the authors state that the programs ruling the interactions between cells are difficult to write and the results are hard to predict or control. Moreover, no global structure can be imposed on the cells arrangement.

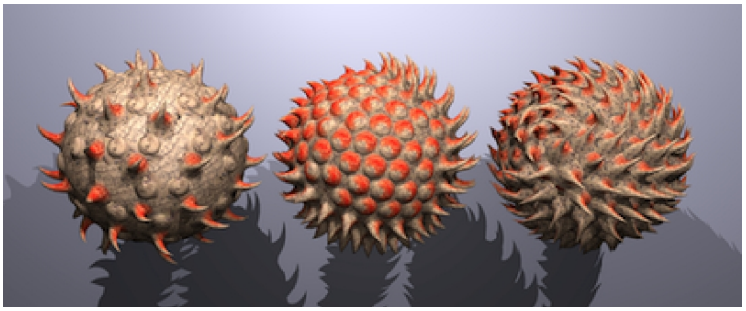

Figure 2.24 Fleischer generates parameterized 3D thorns using a cellular approach. (Image from Fleischer [FLCB95])

Worley [Wor96] introduced a cellular texture basis function to extend Perlin's turbulence functions [Per85]. This function takes as input a set of seed points and defines $F_n(P)$ as the distance from P to the n^{th} closest point. Many interesting results can be achieved by linearly combining the F_n. In particular, the form $F_2 - F_1$ implicitly defines a Voronoi diagram of the seed points. These functions can lead to nice-looking structured textures (see bottom-right image of figure 2.1) using implicit cells for rendering. However, these types of cells are not actual geometry and are thus strictly two-dimensional. We believe the inverse problem of retrieving the right combination of the basis functions to achieve a particular cellular texture to be an interesting research problem.

All the approaches described in this section are principally based on generative models, sometimes including a simulation step to place individual cells. However, to get a particular desired result - corresponding to a real-world sample for example - the models parameters have to be tuned precisely, and the simulations run again and again, in order to perhaps get a result approximatively similar to the sample. Some of these authors have mentioned as extensions the fact of trying to obtain the correct parameters by the analysis of an example. Only a few others have addressed this problem, as described in the next section.

2.3.3 Structure synthesis from example

The problem of generating a structure from the analysis of a sample given as an example of the desired overall look is not trivial. It is called an "inverse problem", where one tries to compute the parameters to give to a certain model to achieve a certain output result, as opposed to a "direct problem" where the result of a model is computed from the input of a given parameter set.

As we have seen in section 2.3.1, a wide range of visually different structures can be generated by computing the Voronoi diagrams of a set of generators, by varying the generator shapes (points, lines or other shapes), their density or the distance metric used in the algorithm. However, deducing these parameters from observation is not an easy task. Okabe *et al.* [OBSC00] explain a method to fit a Voronoi diagram to a simple polygonal tesselation. They associate a point generator to each region of the polygonal tesselation to approximate and model the problem as

a function to optimize. They compute the corresponding Voronoi diagram and the generator locations are progressively adjusted in order to maximize the common area between the obtained Voronoi regions and the original regions. The intersection area is modeled as an objective function and the best solution is found using the steepest descent method. This method applies to simple region shapes whose area can be easily computed. For regions whose boundaries are more complex, the objective function, if it exists, might be very intricate to define and solve. However, trying to fit Voronoi parameters (generator shape, density and metric) to a given complex structure is a very interesting, but very hard topic which could be investigated.

Lefebvre [LP00a] has addressed the problem of automatically extracting simple parameters from a photograph of a structured texture (brick wall (figure 2.25, left) or wood) in order to generate similar structures (figure 2.25, right). However, his technique recovers a few well defined parameters (height and width of bricks, size of gap between bricks, etc.) and is thus only applicable to regular brick walls.

Figure 2.25 Lefebvre recovers simple parameters from the observation of a brick wall (left) in order to synthesize new versions of it (right).

In early visual perception, the basic visual elements are referred to as *textons* and their study is important for a variety of problems (image coding, image modeling, etc.). Zhu *et al.* [ZeGWW02] define a texton as a small template patch consisting in a number of bases at some geometric and photometric configurations. They use these textons to analyse an image and build a generative model [eGZW01] by progressively adjusting the textons on the detected image features. The spatial relations of the bases serve to build a probabilistic model which can be learned and used to reconstruct the image and synthesize new images. The image on the left of figure 2.26 shows the result of the texton analysis phase of a brick wall, whereas the image on the right shows the synthesis step. The textons used are also shown. The approach is interesting and the concepts should be explored in order to be applied to computer graphics applications as the current results are not exploitable "as is".

In the same line of thought, Dischler *et al.* [JMKBD02] manually segment a texture image in various *texture particles* whose spatial arrangement is analyzed by computing the co-occurrences between neighboring particles. The texture can then be re-synthesized using this information and the individual particles by a "seeding procedure" which propagates the particles from one starting particle. The user can control the distributions of particles on the surface. However, their technique

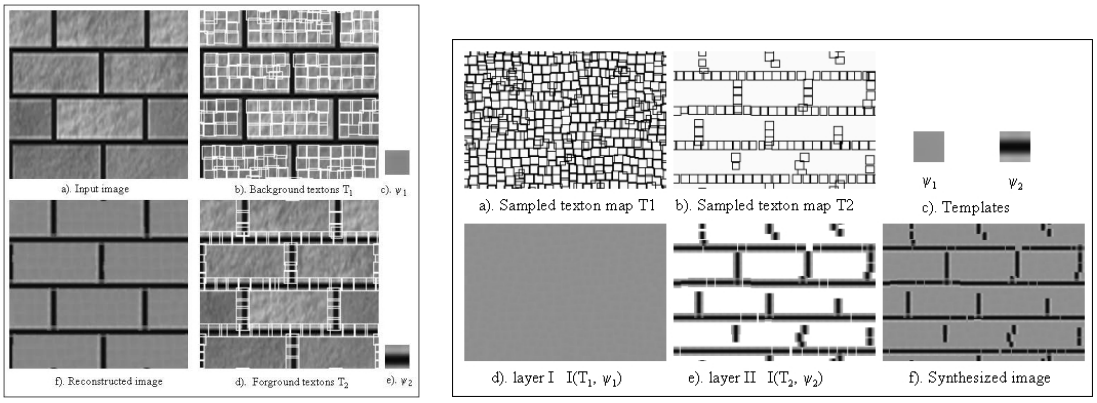

Figure 2.26 Texton learning. Left: Analysis step, discriminating the bricks from the mortar and building a probabilistic model. Right: Synthesis step, the templates are sampled to generate a similar image. (Images from Zhu *et al.* [eGZW01])

does not work for textures characterized by complex correlated spatial arrangements such as rock walls. One of their result is presented in image 2.27.

Figure 2.27 A sample texture (e.g., a photo) is used as input (left). It is manually decomposed into texture particles (center) in order to synthesize a new similar texture (right). (Image from Dischler *et al.* [JMKBD02])

Texture synthesis with structured textures

As described in the preceding section, texture synthesis techniques try to replicate a given sample but do not perform analysis other that of color or frequency, and thus will easily destroy any original existing visual structure (cf image 2.9). Some recent texture synthesis approaches have specifically addressed the problem of structured textures.

Wu and Yu [WY04] propose a method to correct the artefacts at the patch boundaries by using a *feature map* to store structural information of the high-level features and guide the texture synthesis. Curvilinear features are matched and aligned by measuring structural similarity. The feature map is extracted from the sample texture and synthesized at the same time as the new color texture, which it guides. However, the results shown are not much larger than the input texture used and the computation times are much longer than previous methods.

Kwatra *et al.* [KSE+03]'s graphcut texture synthesis approach is in our opinion one of the most interesting techniques to robustly generate a structured texture from a sample. They do not require a fixed block size but rather use varying-sized patches that they disperse and overlap on

the output texture domain, at various offsets. They represent the texture to be synthesized as a graph and reformulate the problem of choosing and blending the optimal patches as a min-cut problem, for which robust solutions already exist (e.g., the max-flow algorithm, which they use). However, for irregular textures, the problem is only displaced as there is no guarantee that the shape of the original regions will be preserved. In addition, for periodic textures, an incorrect choice of the overlap offset will destroy the periodicity of the texture.

Liu *et al.* describe a new class of texture which they call "near-regular textures" and define as textures which "deviate geometrically and photometrically from a regular congruent tiling". Image 2.28 presents their corresponding texture classification. Their method requires a user to manually initialize a texture synthesis process by indicating on a given sample of a near-regular texture the underlying deformed lattice that he supposes to be at its origin. He also has to indicate some information for a light-map extraction process and set the levels of geometry and color regularity. This information allows the recovery of deformation fields for both geometry and lighting so as to synthesize them jointly or separately. These recovered fields also permit some manipulation operations such as modifying the deformation undergone by the texture, varying the geometry and color parameters or deforming another near-regular texture in the same way (see figure 2.29 for some examples). Their approach is different and quite interesting. The integration of the user in the process enables a robust recovery of the deformation fields. The exposure of the analysis statistical parameters of their analysis permits the synthesis of other flavors of texture from the same input sample. However, their technique does not work for irregular textures such as stone walls, where the individual regions can greatly vary in size.

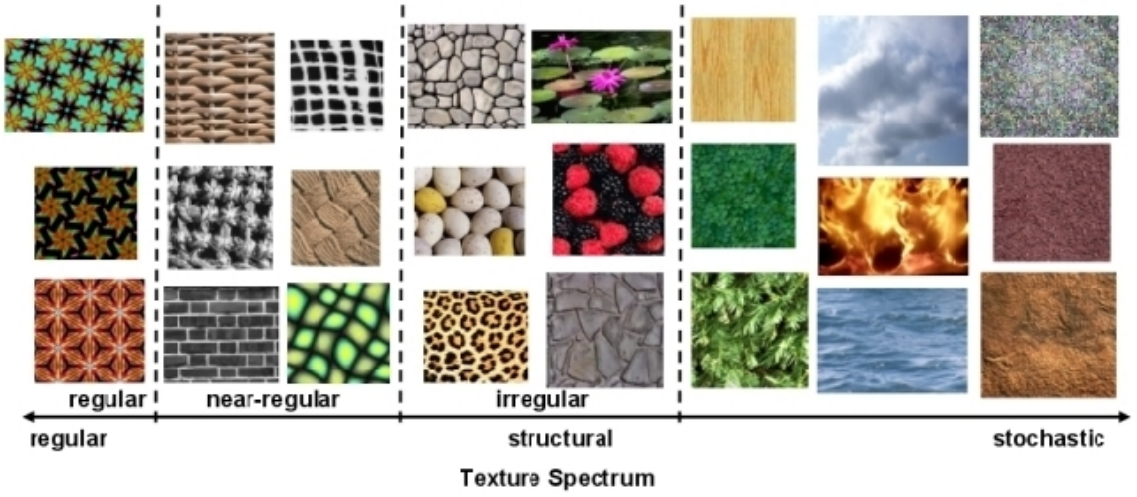

Figure 2.28 Texture classification according to Liu [LLH04].

Texture synthesis techniques have improved related to synthesizing regular patterns. However, anisotropic structure such as natural stone walls are still very hard to handle and we believe that no existing solution is currently satisfactory. Recovering patterns from example is a hard problem. Some authors have fairly good results because they choose to integrate the user in the process, either to influence, guide or correct the results. This aspect of user-integration is fundamental in computer graphics and we review in the next section some approaches which have included the user at some point.

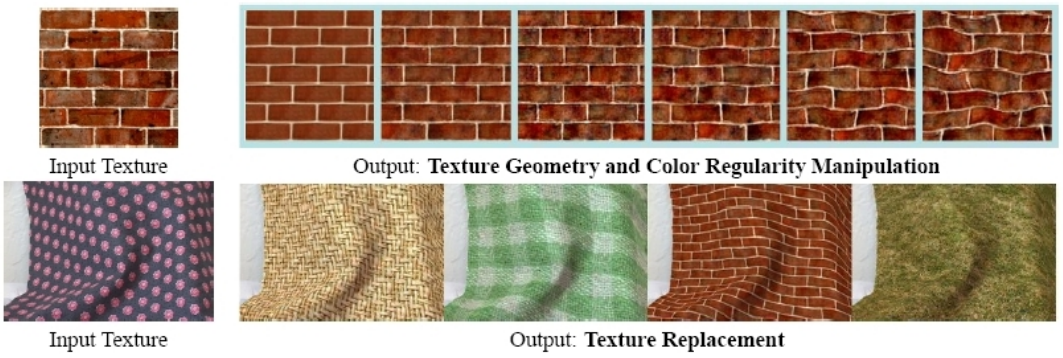

Figure 1: Top row: The regularity of the input texture is being manipulated along color and geometry dimensions, regularities decrease from left to right. Bottom row: Texture replacement where extracted geometric and lighting deformation fields from an input texture are applied to new textures.

Figure 2.29 Some results of Liu's near-regular texture analysis and synthesis (Image from Liu [LLH04]).

2.4 Controlling detail

In a typical computer graphics pipeline, an artist produces 3D images with the tools he is given access to. Thus, it is really of prime importance that the artist has as much control as possible over the output of any tool he is using. If everything is automatic, it is like telling a painter to push a button and its painting will appear automatically on its canvas: he will not use the tool. However we believe that a technique which alleviates its work when this work is far from creative (e.g., tedious, repetitive tasks) will be more than welcome.

Some authors have considered this aspect of control by creating techniques which integrate the user to give control on the output result, either by setting predictable parameters, influencing the process or being allowed to edit the result. We review here such works, directly related to the issues we address in this thesis. Some approaches previously described in this chapter also allowed some control (e.g., [WWL+03] [DG99]), we will not repeat them here.

Controlling texture synthesis

The lack of control in current texture synthesis algorithms makes them less attractive for usage in real-world productions. The guiding texture used in some texture synthesis algorithms [Ash01] [EF01] (as described in section 2.2.2) is an attempt at inserting control into an otherwise automatic process.

The approach of Hertzmann *et al.* [HJO+01] goes one step further by providing a training data *analogy*, consisting of a pair of images, to "learn" an image processing operation to apply to other images. The input to the algorithm is the training pair formed by a source image A with its filtered version A', and a target image B to which the algorithm should apply the same transformation as the one from A to A' to give a result image B'. The analogy is computed using existing pixel-based texture synthesis techniques, by building a search space of *feature vectors*. The feature vector for a pixel of A is composed of its pixel-neighborhood and the neighborhood of its corresponding pixel in A'. When synthesizing B', each pixel of B is replaced by the pixel from A' having the most similar feature vector. The authors display several results with operations such as embossing, artistic filters, image colorization, etc.

To simulate local variations in size of the texture elements in a large generated texture (such as a leopard skin, see image 2.30), Tonietto *et al.* [TW02] present a texture synthesis method, inspired from Wei and Levoy [WL00], which synthesizes its output by interpolating between versions of the same sample at different resolutions. Transition areas are defined between output regions which use the same input sample. Some results are shown in image 2.31. The idea of being able to control the output of a texture synthesis algorithm is crucial in order to make these techniques more usable. This is a first simple step in this direction as it is only limited to variations of the element size over fixed domains. Being able to control the placement, size, orientation and shape of the texture elements would be a big step forward.

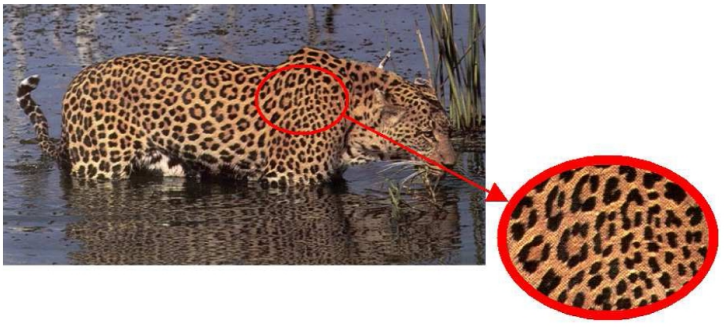

Figure 2.30 A leopard skin has spatially varying texture features. (Image from Tonietto *et al.* [TW02])

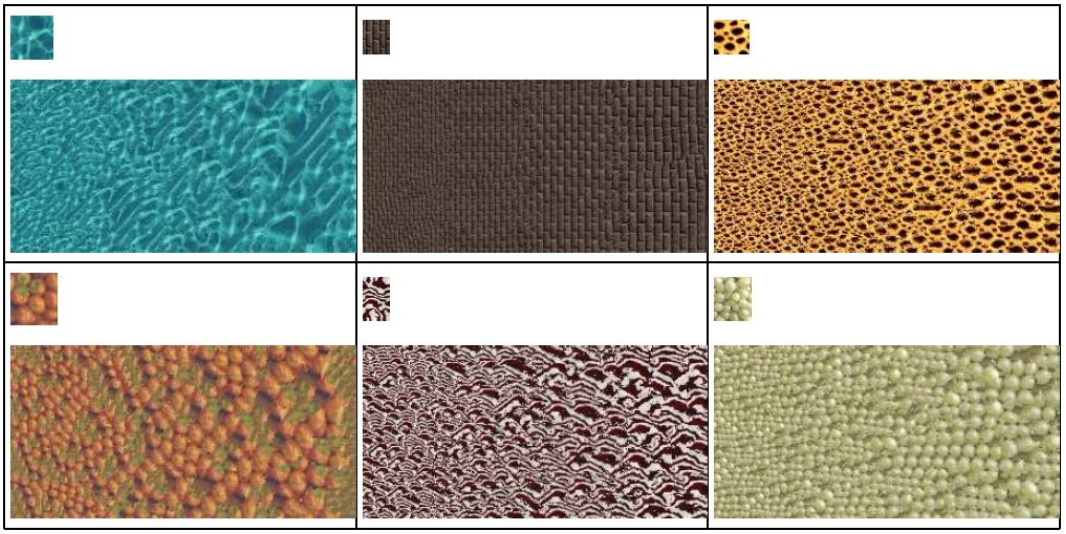

Figure 2.31 Some results of locally controlling the output element size in a synthesized texture. (Image from Tonietto *et al.* [TW02])

Taponecco *et al.* [TA04] take a similar approach by allowing the output of the texture synthesis to be influenced by a set of input filters. The output texture is synthesized with a pixel-based approach and the best matching pixels are selected from a set of input samples, whose properties (orientation, size) can be modified by an underlying filter. Some results are shown in image 2.32. Their work has ideas similar to our work on example-based detail propagation (chapter 5).

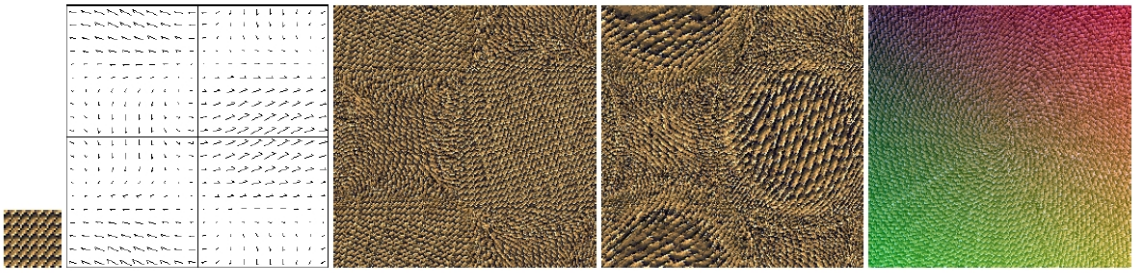

Figure 2.32 Results of the steerable texture synthesis approach. From left to right: The input sample is modified through a superimposed field: phase information is used to change the direction of the pattern, magnitude information is considered in addition to scale the original pattern. In the rightmost image, a critical point is visualized: the pattern is modified by rotation and color changes along several directions. (Image and caption from Taponecco *et al.* [TA04])

Procedural textures do not have the resolution problem of image-based textures. However, their output is not controllable and getting an exact desired result is often very difficult, even impossible in some cases. An interesting exception is the commercial software package [all] that permits local control and modification of procedural textures while maintaining the benefits of the generative approach.

Controlling appearance

Controlling the content of appearance layers is crucial for artists. Some authors have considered this requirement and provide techniques to fulfill this need, either by influencing the detail synthesis process, propagating changes or allowing interactive modifications.

Haeberli's paint-by-numbers approach [Hae90] has inspired a quantity of subsequent research works. In this pioneer paper, he presents an interactive paint system. In particular, he describes how to add a layer of control by using an image colors to parameterize the strokes of a paint brush. The "texture-by-numbers" technique of Hertzmann *et al.* [HJO+01] is inspired by their work. They label by hand various parts of an input image according to its texture components. When given a new labeling, the algorithm synthesizes a new image using the image original pixels and a texture synthesis technique getting its samples from the appropriate original labeled regions.

Hanrahan and Haeberli [HH90] introduced a new paradigm for direct WYSIWYG painting on 3D surfaces. Their tools let the artist design the texture maps directly onto 3D models by projecting screen-space paint strokes onto the 3D surface and then into texture space. This technique allowed for an instantaneous visualization of the result of the appearance layer and allowed the artists to concentrate more on the result than on the creation of the texture map itself, thus reducing the parametrization problems.

Brooks *et al.* [BD02] [BCD03] present an interesting approach to propagating local modifications. Their approach employs the same paradigms as texture synthesis for determining similarity of pixel neighborhoods. However, they use this similarity to replicate interactive editing operations (painting, cloning or warping pixels or regions of pixels) in order to create a new, different looking version of an original texture. Figure 2.33 shows some results.

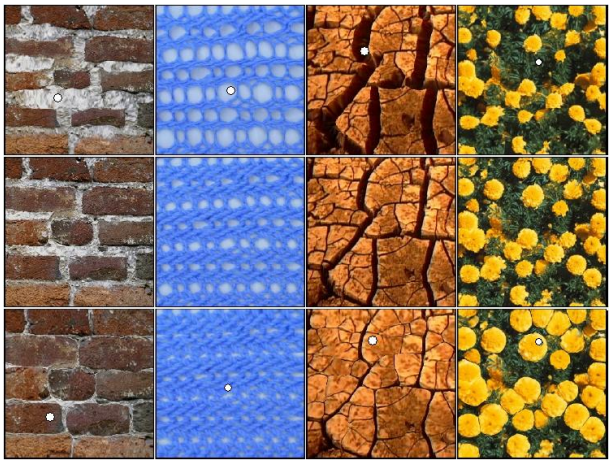

Figure 2.33 Similarity based editing. The original textures are in the middle row. The two other rows show the modified textures with the white circle indicating the origin of the modification. (Image from Brooks and Dogson [BD02])

This paper instigated Zelinka and Garland [ZG04b] [ZG03] to transfer this approach in 3D. Instead of determining similarities between pixel neighborhoods, they have to perform local surface comparisons, which is a much harder problem. We believe there is a lot of interesting future work in this direction.

2.5 Summary

In this chapter we identified previous work related to some of the problems we address in this thesis (as described in section 1.1), such as the synthesis of irregular structured textures and the recovery of details from example. There are also some interesting techniques which we build on or from which we transfer principles in order to build some solutions to these problems. In particular, as Ismert *et al.* [IBG03], we want to use photo-extracted textures as a guide for the reintroduction of details. We will see that we generalize this approach to using the underlying model as well. The image analogies framework [HJO$^+$01] principles also go along the same lines of thoughts. For the recovery of appearance information we aim to decrease memory usage when synthesizing information, in the same manner as Soler *et al.* [SCA02], and more generally, use instancing principles where possible.

Chapter 3

Generating 2D Structure from Example

"Reality is merely an illusion, albeit a very persistent one."

—Albert Einstein

Real-world structured textures can be found all around us, in masonry, tiling, biological tissues and many more, as we can see in the images of figure 3.1 and figure 2.12. Their geometric nature brings about many interesting challenges as we saw in chapter 2. In this section, we address the problem of analyzing such structures in order to get a compact description which will allow us to generate a similar version of them without it being a simple copy of the example. This can be seen as a 2D geometric problem (as seen in figure 3.2) which we will address as such. After stating the problem, we will explain the approach we chose, describe its avantages and limitations and show some of the results we obtained.

3.1 Motivation

In section 2.3, we gave the following definition for a geometric cellular structure: "A 2D **cellular subdivision** defines a partition of the plane into regions whose shape can be governed by some measure". For example, a Voronoi diagram is a partition where the points of any region are closer to the region generator than to any other generator. Thus the measure used is a distance metric.

Given a structured texture sample, for example a photograph of a stone wall, we can extract a geometric subdivision which allows us to separate the texture sample into its various primitives, i.e., the subdivision regions[1]. In the following, a region is composed of a number of *vertices*, linked by *edges*. Some subdivisions can exhibit a certain amount of *meta-structure*, i.e., the regions are not randomly dispersed but their arrangement seems to follow a simple underlying pattern (e.g., brick or stone walls). Some examples of the structures we want to handle can be seen in figure 3.3. The one on the right side has a layered meta-structure, as can be found in many man-made structures, like walls, and is inherent to the way they are constructed.

3.1.1 Problem statement

If we are given an example of a structured texture and we want to reproduce the corresponding geometric subdivision to create a visually similar texture to apply at will on any 3D model, we can think of a few approaches to try out. With a minimum amount of artistic skills, it is possible to

[1] In our examples, we drew this subdivision by hand on the photograph. It might be possible to implement image analysis techniques to obtain it automatically although our initial attempts were not successful.

Figure 3.1 Some structured textures found around us.

draw the subdivision by hand. But if the area to cover is large, it can rapidly become a very tedious task. To speed up the process, one can thus think of drawing a small part of the structure and tile it by hand or automatically (in the same way texture synthesis techniques aim to do). However, the human eye is very sensitive to patterns and repetition in drawings thus the final quality of the structure is proportional to the time spent to build a seamless tiling with a minimum of redundancy, matching the edges, patterns, etc. In addition, automatic texture synthesis techniques are not yet usable with irregular structured textures and there is no generic method available for synthesizing structured textures from examples, as we have seen in chapter 2.

Hence, we define the following problem:

Given a 2D subdivision of any type, how can we analyze its properties in order to generate a similar subdivision (i.e., having the same properties) ?

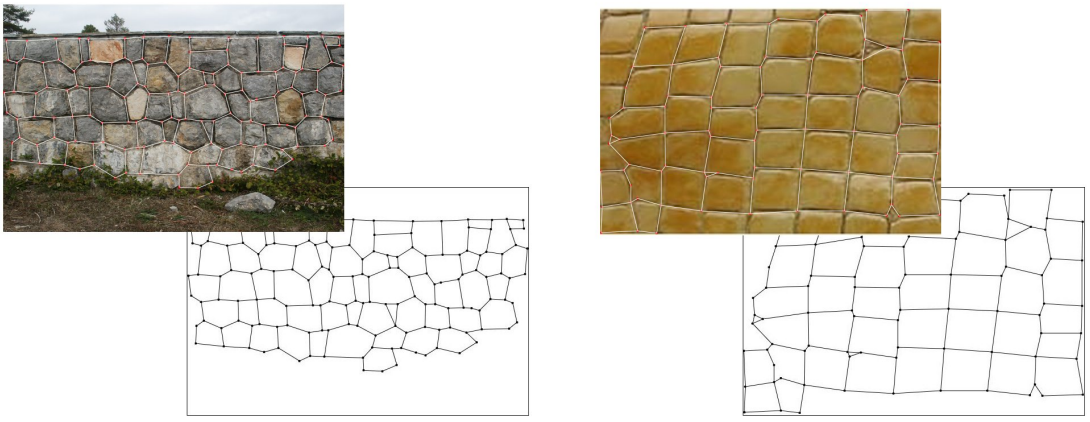

Figure 3.2 Structured textures and their geometric counterparts.

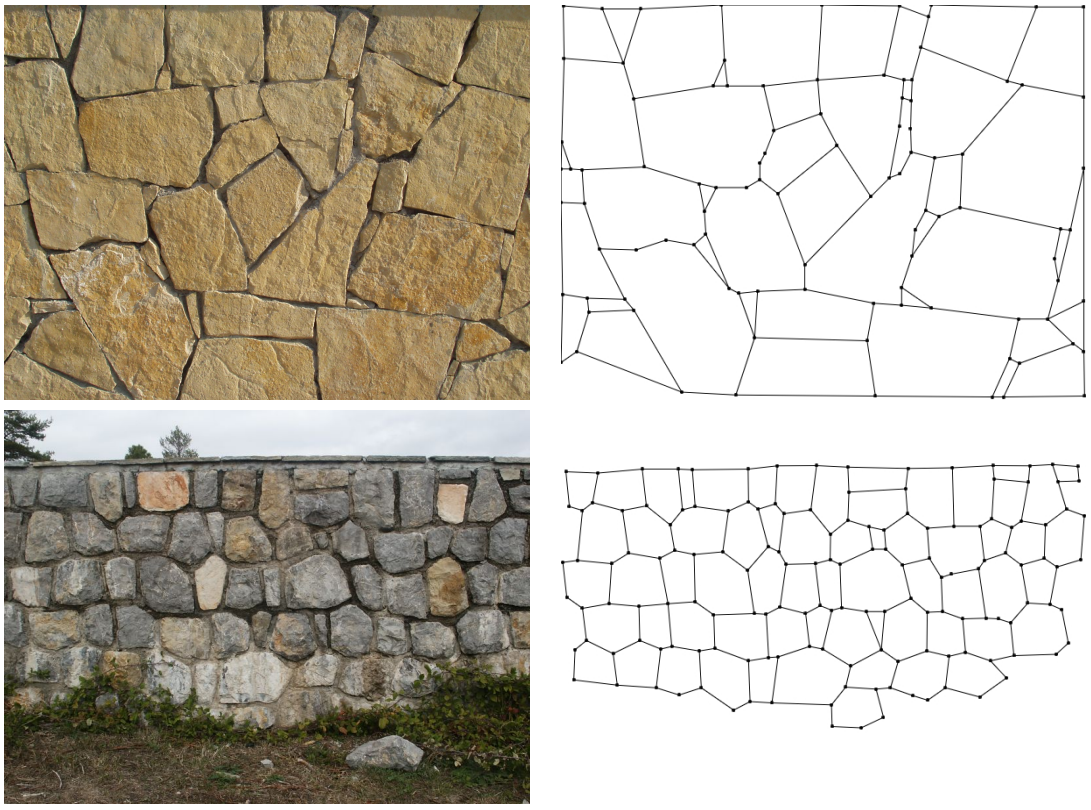

Figure 3.3 Two subdivision samples: the "INRIA wall" (top) and the "Levens wall" (bottom). The Levens wall exhibits a layered *meta-structure*.

In other words, how can we create a *generator* that will synthesize similar subdivisions from the analysis of a sample structure of finite size?

Having such a *structure generator* has undeniable advantages. A generator is a light, controllable, procedural method to generate indefinitely large samples of structures. If we are able to parameterize or influence this generator, we would have even more control on the generated sub-

divisions. These subdivisions can be used as a first step in the texturing process where the global structure is initially generated, followed by the creation of the details inside the structure regions, as we explain in the two next chapters.

3.1.2 Proposed approach

Our goal is to be able to analyze and re-synthesize structured textures ranging from completely randomly placed regions to subdivisions organized by some *simple* meta-structure.

We first analyze the given subdivision sample in order to extract meaningful information that will allow us to generate similar-looking subdivisions. Once we have collected this information, we would like to create a generator able to reproduce the same kind of subdivision. By "same kind", we mean a subdivision having the same statistical properties and whose aspect is similar to the original one (but not being its replica, nor a copy-paste of its subparts). This is clearly a subjective goal, but we believe that it is reasonable. In order to validate our results, we rely both on visual inspection and computation of some evaluation metrics.

The problem of generating a similar subdivision can be seen as a *sweeping* problem where one wants to synthesize the new subdivision in a certain direction so as to reproduce the subdivision sequentially. It can also be seen as a *tiling* problem as it really is generating a partition of space using regions whose statistical properties are partially known. After having evaluated a greedy, sweeping paradigm, we decided that a more global, tiling method would be more appropriate to solve our problem and especially address the particular need for interactivity and control in computer graphics applications. This is the method we are describing in this chapter. The sweeping approach we tried was not adequate as it did not allow us to make any modification a-posteriori (except local shape changes) or respect constraints during the process. Some backtracking was also necessary in order to correct synthesis problems, as is often the case with such greedy techniques. We designed another approach (described in appendix B) but did not implement it.

In the following, we start by presenting some information about the widely-known method of tiling a plane: Voronoi Diagrams, and the variant we are exploiting in this thesis. We then give an overview of our analysis-synthesis method, before describing each step in more details. Our solution must be able to reproduce the input structure on an infinite plane, possibly respecting some constraints. We also have to provide a way to measure the validity of the generated subdivisions.

3.2 Tiling the plane using Voronoi diagrams

By observing the subdivisions we want to handle, we can be tempted to categorize them into a class of Voronoi Diagrams as they respect the necessary conditions to fit into these classes: They effectively represent a complete partition (tiling) of the plane using disjoint regions. If we can cast our input into a certain class of Voronoi diagram, it should be possible to deduce some rules to replicate this class of Voronoi diagram at will.

After exposing some more theoretical aspects of Voronoi Diagrams (*VDs*) and the class of VDs that are of interest to us, we give an overview of our analysis-synthesis process. The next two sections detail these two main steps and are followed by sections showing some results and discussing the technique.

3.2.1 Discrete Voronoi diagrams

As we saw in section 2.3, the final aspect of a Voronoi diagram depends entirely on its input parameters: The density, location and shape of its generators and the distance metric used. The variety of possible results is immense and thus we think that we should be able to approximate a subdivision defined as in section 2.3.1 by using a Voronoi diagram and varying these parameters. However, deducing these parameters from analysis/observation is far from trivial and would constitute an entire research topic.

Computing a discrete Voronoi diagram

Computing the Voronoi diagram of a set of point generators with an Euclidean distance metric can be performed analytically. Indeed, we consider the plane as having an infinite number of points and we can find an exact solution for the region boundaries edges, which are in this case, line segments. However, computing a more complex kind of Voronoi can be very hard to do analytically [OBSC00]. In order to compute an exact representation of the Voronoi boundaries, high-degree curves have to be manipulated and intersected. The corresponding algorithms are often complex and hard to implement, and they suffer from robustness and precision problems. Nevertheless, it is possible to use approximate algorithms to compute a *discrete* approximation to a Voronoi Diagram (as described in [OBSC00]) which allows more flexibility to try out various distance metrics and generator shapes.

If we consider a portion of the plane as containing a finite number of points, where each point has integer coordinates (x, y) (i.e., corresponding to image pixels), we can compute such a discrete Voronoi Diagram using a technique inspired by the watershed transformation in mathematical morphology. Each generator is placed at a pixel position on the plane and is associated with a unique color. It constitutes a source for flooding the pixels around it using a square-shaped structuring element consisting of its 3×3 pixels neighborhood. The flooding direction and speed is determined by the distance metric used. Figure 3.4 shows the computation of an ordinary discrete Voronoi Diagram using this technique. The pixels around the generators are flooded equally in all directions, forming regions of pixels associated to their source generator and having the same color. When the pixels of two adjacent regions meet, a border is formed. In the end, we obtain a complete Voronoi diagram where each pixel is closer to its generator than to any other. Okabe *et al.* [OBSC00] state a set of assumptions corresponding to the above-described process, which collectively define what they call a *Voronoi Growth Model*. They describe many domains in which this model can be applied.

Trefftz *et al.* [TS03] propose a method to implement the computation of discrete Voronoi diagrams on parallel processors. We simulate such parallel processing by using a priority queue [All04] to store and process the pixels to handle. In the beginning, we are given a set of generators. The neighbors (eight pixels) of all the generators are inserted into the queue, along with the color of their generator, with a priority depending on the distance metric used: the pixels closer (according to the metric) to their generator are given a higher priority and will be treated before more distant pixels. The pixels are popped one by one from the queue and their neighbors are collected and inserted in the queue wit the same color assigned, until no more pixels are left and the priority queue is empty. This method is a flexible way to experiment with various types of generators and different metrics.

We implemented such a method and figure 3.5, right, shows our discrete result of the analytical Voronoi of segments shown to the left. The distance metric used is the distance from a point to the

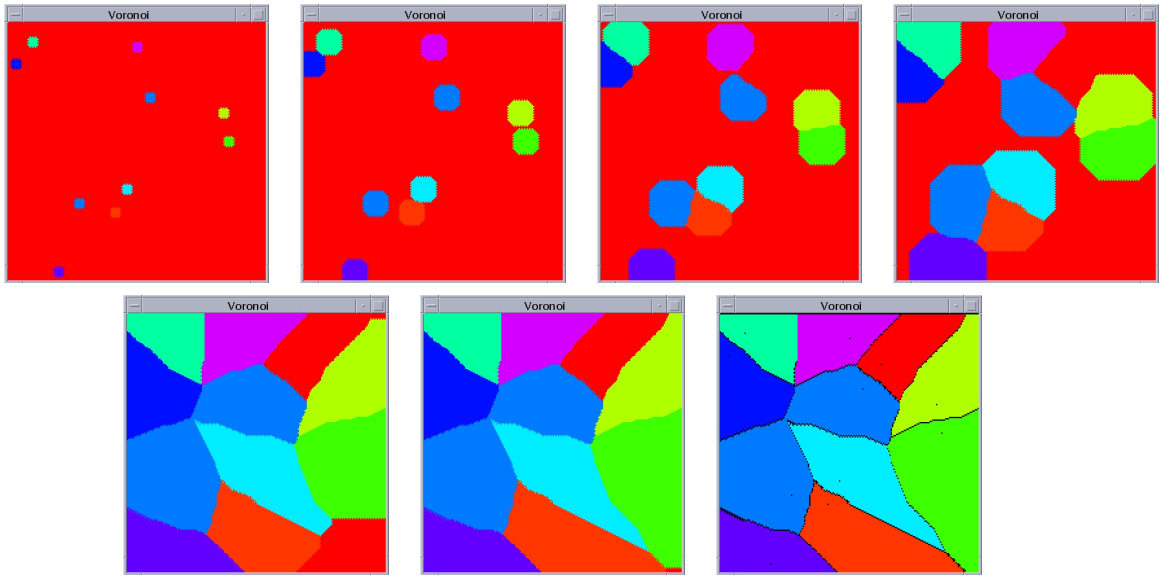

Figure 3.4 The discrete computation of an ordinary Euclidean Voronoi diagram: The generators are initially placed at discrete positions. Each is a source for flooding the pixels around it. Their neighborhood is progressively flooded in parallel (attributing the corresponding colors to each flooded pixel) until the formed regions meet.

closest point on the segment. We can observe that the regions formed are similar to those obtained with the continuous method. It should be noted that this technique will not necessarily lead to closed regions since, depending on the metric used, some regions will flood very fast and could potentially enclose other regions or generators. Some generators can thus stay outside of their own regions, and they are called orphans. In this case, additional rules have to be added to the process in order to get a satisfactory result.

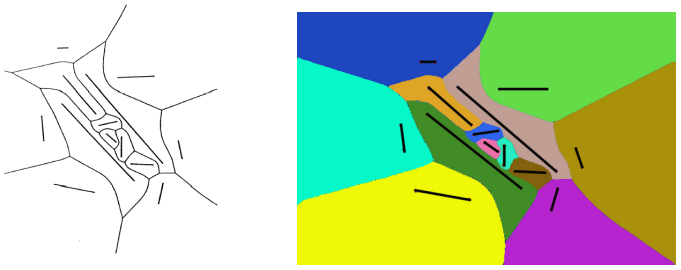

Figure 3.5 A Voronoi Diagram of line segments computed analytically (left) and discretely using our implementation (right).

It should be noted that Hoff *et al.* [KEHKL+99] proposed a method to compute discrete Voronoi Diagrams by taking advantage of the graphics hardware to rapidly compute the intersection ot a set of cones (one per generator) with a plane. We think it should be possible to generalize this method to fit our purposes and rapidly compute a discrete Generalized Voronoi Diagram.

Centroidal Voronoi diagrams

Let us define the energy function for a region as the sum of all the distances squared from sampling points to their corresponding generators. The centroid (or center of mass) of a region is the site which minimizes the *energy function* of that region. It can be influenced by an additional *density*

function. A Centroidal Voronoi Diagram (CVD) is a specific class of Voronoi tesselations where the generators of each region are also their centroids, thus it corresponds to a minimum-energy configuration of the diagram generators. Du *et al.* [DFG99] describe the usefulness of such diagrams for data compression, mesh generations, optimal distributions of resources, cellular biology, territorial behaviors of animals and many more.

The iterative minimizing method known as *Lloyd's algorithm* [Llo82] can be used to compute a centroidal Voronoi tesselation given a density function, a distance metric and an initial number n of generators (which is also the number of desired centroids), placed in any given initial locations. The algorithm finds a locally optimal position for the n generators so that they are also the centroids of the corresponding regions, with respect to the given density function. Lloyd's algorithm alternates two phases until a satisfactory convergence is achieved:

1. Compute the Voronoi Diagram corresponding to the n points.
2. Compute the centroids of the n Voronoi regions issued from step 1. Move the generators to the computed centroids.
3. Repeat step 1 and 2 if convergence is not satisfactory.

See figure 3.6 for an illustration of one iteration of the algorithm.

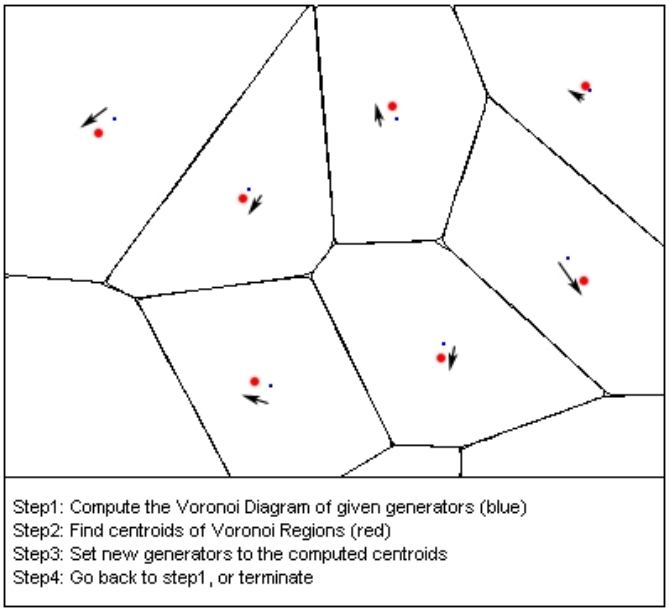

Figure 3.6 Illustration of the Lloyd's algorithm.

The Lloyd's algorithm is a clustering technique, similar to the *k-means* clustering algorithm, which leads to a local minimum solution and whose convergence can be proven in the one-dimensional case, under certain conditions, although higher-dimensions seem to act similarly [DFG99][2]. The convergence criteria we are using is the average distance moved by all the generators. When it is less than a given threshold, the Centroidal Voronoi regions area (and shape) no longer changes significantly, and the algorithm can terminate. Figure 3.7 illustrates the difference between an ordinary Voronoi Diagram and a CVD. The set of generators are placed in the initial

[2]Very recent work of Du *et al.* [QDJ04] establishes new convergence theorems for the Lloyd's algorithm applied to the computations of CVDs.

configuration shown to the left, leading to the ordinary Voronoi Diagram drawn. After some iterations of the Lloyds' algorithm, the generators progressively moved to give the configuration of the right image.

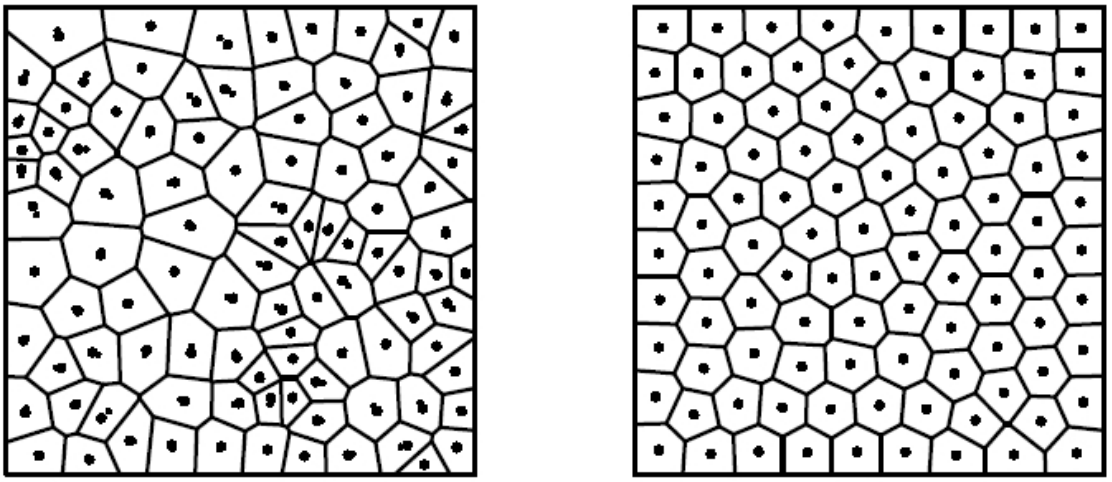

Figure 3.7 Left: an ordinary Voronoi Diagram. The generators are indicated by large dots, whereas region centroids are marked by small dots. Right: The CVD obtained out of the same set of generators. (Image from Secord [Sec02a])

In order to compute a CVD discretely, on a fixed-size grid of pixels, we need to alternate between computing the Voronoi discretely using the method described above, assigning a unique color to each region flooded, and reading back the grid of pixels in order to *scan-fill* the obtained results and determine the shape of the obtained regions so as to be able to compute their centroid.

Computing the CVD of a set of points discretely has a big disadvantage, though: the computation of the centroids of each region (represented by a set of pixels of the same color) is highly dependent on the diagram resolution. The error on the computation of the centroid of a Voronoi region will increase relatively to the decrease of the number of pixels in this region. If the resolution is very low, two generators can merge and a region will disappear. Hoff *et al.* [KEHKL$^+$99] present a solution to this problem by separating the diagram into sub-diagrams computed at full resolution and stitched back together. In order to speed-up our computations in our implementation, we allow the CVDs to be computed at a lower resolution.

Some properties of the Lloyd's heuristic make it a very interesting tool for our purposes. As described in section 3.7, it allows interactive modifications and supports constraints, and the algorithm will globally update the diagram to adapt to the new configuration. It was first introduced into computer graphics by McCool and Fiume [MF92] for the generation of sampling point sets.

3.3 Generating structures using a tiling approach: Overview

Recall that our input is a subdivision described in terms of its vertices and edges. We want as output a subdivision having similar visual properties and subject to a set of constraints, such as the topology of the output which can be significantly different from the input, etc. This notion of similarity can be both objective and subjective, we thus have to characterize the input subdivision in some way.

We first start with an analysis step in order to extract meaningful structure information. This information is a mixture of region information (local information) and structure information (global information), mainly under the form of histograms which allow us to derive statistics on the input structure. We chose to characterize each input region with an ellipse, which is a simple structuring primitive (in the number of parameters necessary for its description).

Once we have collected the data needed to characterize our input sample well enough, we can synthesize similar subdivisions. For this, we proceed in two stages. The first stage aims to generate and distribute regions on the target plane having properties similar to those of the input sample. Special care has to be taken for subdivisions having some regularity or meta-structure. The second stage applies a correction, modifying the borders of the generated regions so that they resemble the input regions more closely.

In the end, we get a subdivision which constitutes a partition of the plane, where the regions are disjoint and whose shapes visually correspond to the shape of the input regions. The meta-structure, if any, should also be preserved as much as possible. This subdivision can then be used in a texturing module for example, or any module demanding a structure as input.

Figure 3.8 and figure 3.9 respectively illustrates our analysis and synthesis process. The next two sections describe them in more details.

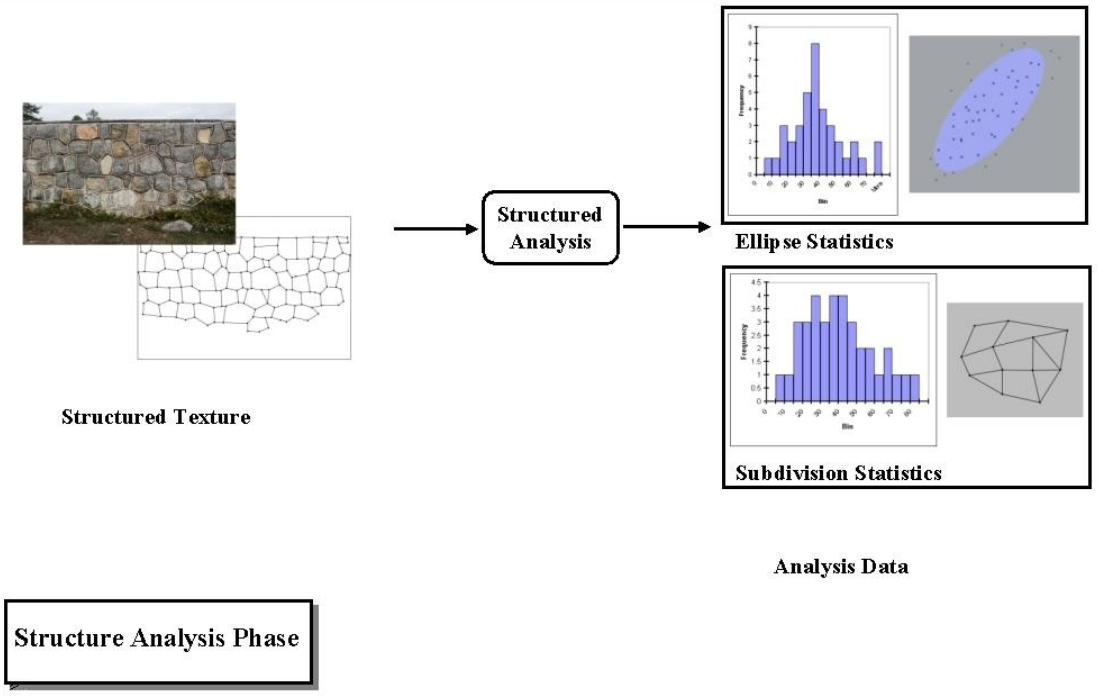

Figure 3.8 Structure Analysis Process: The input structure is analyzed in order to extract statistical properties characterizing its geometry.

3.4 Analysis phase

Our input subdivisions are constituted of edges and vertices with no information about its regions. Analyzing such a subdivision to collect this information can be done in many ways. We chose to use a traditional computational geometry sweep-line method to collect the data we need. A

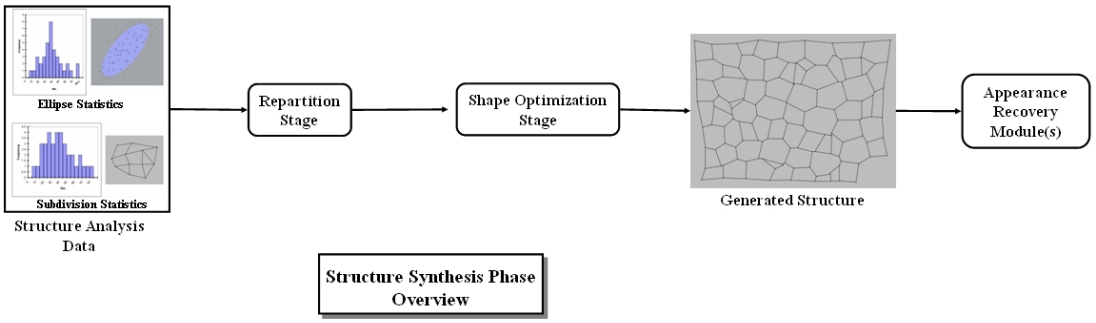

Figure 3.9 Structure Synthesis Process.

sweep-line algorithm simulates the "sweeping" of a line over the plane while maintaining a data structure along the process. The line stops at discrete events where some processing occurs and the data structure is updated [O'R98]. In our case, the discrete events are the subdivision vertices where three different kinds of event can occur (see figure 3.10) for the case of convex regions:

1. Region Closing: two edges end at this vertex.
2. Region Opening: two edges begin at this vertex.
3. Region Continuation: one edge ends and one edges start and the above two conditions are not satisfied.

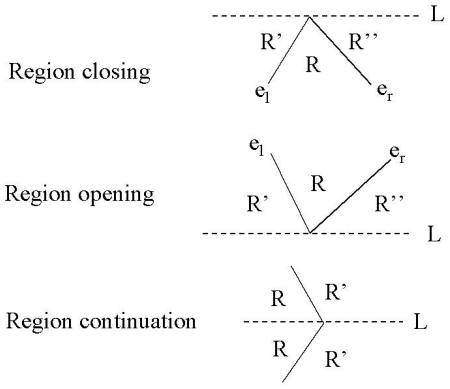

Figure 3.10 The three types of events in a subdivision.

By sorting all our input points by their y-coordinates (using an optimized tree such as a priority-queue), we can traverse the entire subdivision in $O(n \log n)$ and build a *mesh* of regions that we can exploit in the next stages.

Algorithm 1 describes our method for analyzing a given 2D subdivision and constructing its regions. Single variables are denoted by a lowercase letter, while sets of variables use uppercase. An edge e is an oriented structure containing its two endpoints. It also references the region to its left (*Edge.LeftRegion*) and to its right (*Edge.RightRegion*). An edge on the sweep-line is considered to be below a vertex if it is to the left of the vertex and above if it is on its right.

We allow ourselves to treat concave regions as input. This gives rise to more complex events to handle as two regions in construction might have to be merged in the process if they are effectively the same region. For the sake of clarity, the details are not given here.

Approximating region shape

We chose to approximate each region by an *ellipse* which is a simple way to characterize the eccentricity, or anisotropy[3], of a polygonal region. An ellipse is characterized by three parameters (see figure 3.11)

1. A major axis (a)
2. A minor axis (b)
3. A rotation angle (θ)

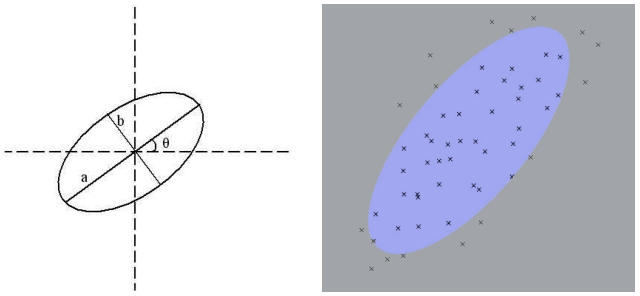

Figure 3.11 Left: The parameters of an ellipse. Right: An ellipse fitted to a set of data points.

In statistics, an ellipse can be fitted to a set of data points using principal component analysis (PCA) [PTVF92], by computing the *covariance matrix* of the set of points. This ellipse can serve as a simpler way to summarize/visualize the data (figure 3.11, right). In our (discrete) case, the data points consist in all the pixels $p_i = (x_i, y_i)$ contained in the region we want to approximate, which we can accumulate by scanfilling this region. A covariance matrix describes how a certain mass is distributed around its center. The center of mass, or centroid, C of a region is the (possibly weighted) mean of all the points contributing to the mass, in our case all the pixels,

$$C = \frac{1}{\sum w_i} \sum (w_i * p_i),$$

where w_i is the weight associated to pixel p_i. If $p'_i = p_i - C$ is the position of a region pixel relative to its centroid, the covariance matrix is the outer product

$$\sum (p'_i p'^T_i),$$

which, if we expand it, gives the 2×2 symmetric matrix:

$$\sum \begin{bmatrix} x'^2_i & x'_i y'_i \\ x'_i y'_i & y'^2_i \end{bmatrix},$$

where $p'_i = (x'_i, y'_i)$, with coordinates in pixel units.

[3] Eccentricity/anisotropy implies that the region can be stretched, i.e., have a principal direction that significantly differs from the other one.

Algorithm 1 Analysis of a 2D subdivision

```
procedure SUBDIVANALYSIS(S)                          ▷ S(E,V): input subdivision
    StatModule.Reset()                               ▷ Reset the statistics module

    ReadSubdivision(S)   ▷ Fill the data structures (order the vertices) and the StatModule

    for each vertex v in S do                        ▷ "Sweep" the subdivision with a virtual sweep-line L
        Separate edges incident to v into two sets: A (edges above L) and B (edges below L).

        // Region Closing
        if #B > 1 then
            for each Pair of edges (e_l, e_r) in B do        ▷ e_l = Left edge, e_r = Right edge
                r ← e_l.RightRegion
                r.Close()                                    ▷ Compute region statistics
            end for
        end if

        // Region Opening
        if #A > 1 then
            for each Pair of edges (e_l, e_r) in A do
                r ← CreateRegion()
                r.AddEdge(e_l)
                r.AddEdge(e_r)
                e_l.RightRegion ← r
                e_r.LeftRegion ← r
            end for
        end if

        // Region Continuation
        if #A > 0 and #B > 0 then
            r_l ← B.LeftmostEdge.GetLeftRegion()
            A.LeftmostEdge.LeftRegion ← r_l
            r_l.AddEdge(A.LeftmostEdge)

            r_r ← B.RightmostEdge.GetRightRegion()
            A.RightmostEdge.RightRegion ← r_r
            r_r.AddEdge(A.RightmostEdge)
        end if
    end for
end procedure
```

To find the ellipse parameters corresponding to this covariance matrix, we solve for its *eigenvalues* and *eigenvectors*. The eigenvalues are used to compute the length of the ellipse axes, while the eigenvectors define the direction of each eigenvalue, and thus provide the ellipse orientation. We center the resulting ellipse on the computed centroid. Figure 3.13 (top row, middle image) shows the ellipses resulting from the analysis of the subdivision shown to the left.

Statistics

The sweeping step allows us to build our network (mesh) of regions with all the connectivity information we need and also to collect the associated data, which compose our statistics. More specifically we collect some topology information:

- valency of a vertex (i.e., the number of edges meeting at this vertex),
- angles of edge pairs around a vertex,
- length of edges,

and region information:

- number of edges,
- interior angles between each pair of consecutive edges,
- fitted ellipse: center position, axis lengths, angle, as described above.

We store the samples for each variable in an histogram (1D or 2D depending on the variable) as a way to compute its distribution. Histogram-based methods are the most practical for us as the other methods (chi square, hypothesis tests, etc.) involve too much computational overhead to be useful for our particular problem. Determining the number of histogram bins which should be used to estimate each distribution the best is a problem in non-parametric statistics [Ize91]. We chose to use a robust result from Diaconis and Freedman [DF81] who showed that the optimal histogram bin size, which provides the most efficient, unbiased estimation of the probability density function, is achieved when

$$W = 2 * IQR * N^{-\frac{1}{3}},$$

where W is the optimal bin width, N is the number of samples and IQR is the interquartile range (the 75th percentile minus the 25th percentile).

The histograms are used in the generation step to sample values from the computed distributions. This is done through the inverse transform method which is straightforward when one uses histograms. We can thus easily obtain samples with the desired probability distribution. The statistical operations are performed with the Gnu Scientific Library [Bee01]. Once we have collected the data, we aim to generate a similar subdivision. The next section presents our generation approach.

3.5 Synthesis phase

We aim to generate a Voronoi subdivision with properties derived from the analysis of the original subdivision. We extracted a set of ellipses fitted to the original regions and we can use these ellipses as an anisotropic metric to describe the eccentricity of the corresponding Voronoi regions. We will thus first describe how an ellipse can be used as a metric to compute a kind of Voronoi

Diagram and how a centroidal (discrete) version of this diagram can also be derived. Computing a discrete Voronoi allows us to obtain a tiling of the plane into a set of regions which we can segment in order to get a geometric 2D mesh of regions. This mesh constitutes our *generated mesh* and we can evaluate its similarity to our original mesh. It is a modifiable structure which we can potentially transform to increase the similarity.

3.5.1 Computation of Voronoi diagrams using an elliptic metric

Given an ellipse e, we define the following metric d to evaluate the distance[4] of any point p to a particular punctual generator c, according to an elliptic metric centered on this generator:

$$d(p,c) = \frac{\|p-c\|}{\|p_r-c\|},$$

where $\|\cdot\|$ is the Euclidean distance operator and p_r is the projection of p on the ellipse (see figure 3.12).

The denominator $\|p_r - c\|$ corresponds to the ellipse radius r_p at point p_r. To compute this radius, we use the ellipse polar coordinates and apply the following formula:

$$r^2 = \frac{b^2}{1-e^2(\cos\alpha)^2}$$

where e^2 is the ellipse squared eccentricity, defined as

$$e^2 = 1 - \frac{b^2}{a^2},$$

a and b are respectively the ellipse major and minor axes, and α is the angle between the line passing through c and p and the horizontal.

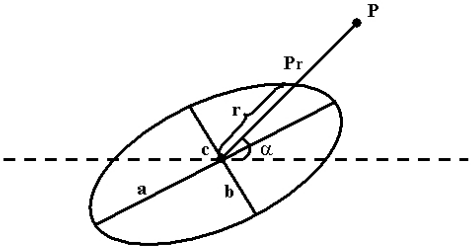

Figure 3.12 Illustration of the metric used to compute the distance of a point to any generator (center of ellipse).

Given this function, any point on the ellipse contour has a distance of 1 from the center of the ellipse ($d(c,P_r) = 1$), while all points inside the ellipse have a distance between 0 and 1, and points outside the ellipse have a distance of more than 1. Given a set of ellipses (whose centers correspond to generators) and using each ellipse as an anisotropic metric for the associated generator, we can compute an associated Voronoi diagram. Due to the nature of the metric and the way we construct them, all regions have the same importance. There is one generator per

[4]Our metric does not respect all the conditions (identity, symmetry, triangular inequality) of a distance metric but, for our purposes, this is not important as we are more interested in the properties of the resulting Diagram.

region/ellipse and it is always contained in its own region. In summary, our diagrams can be computed out of a set of punctual generators, where each generator carries its own metric (given by the ellipse).

The top row of figure 3.13 shows the Anisotropic diagram (right) resulting from computing the Voronoi of the ellipses fitted to the regions shown in the center image, corresponding to the structured texture of the left image. This Voronoi is computed using the discrete method described in section 3.2.1. Each region has a unique color and every pixel inside a region is closer to its generator (the ellipse center) than to any other generator, according to the distance function we just defined. We can observe that the resulting regions (bottom right) correspond fairly well to the original ones (bottom left). Regions on the contour have a different shape since they are not bounded by other regions and thus take all the space available.

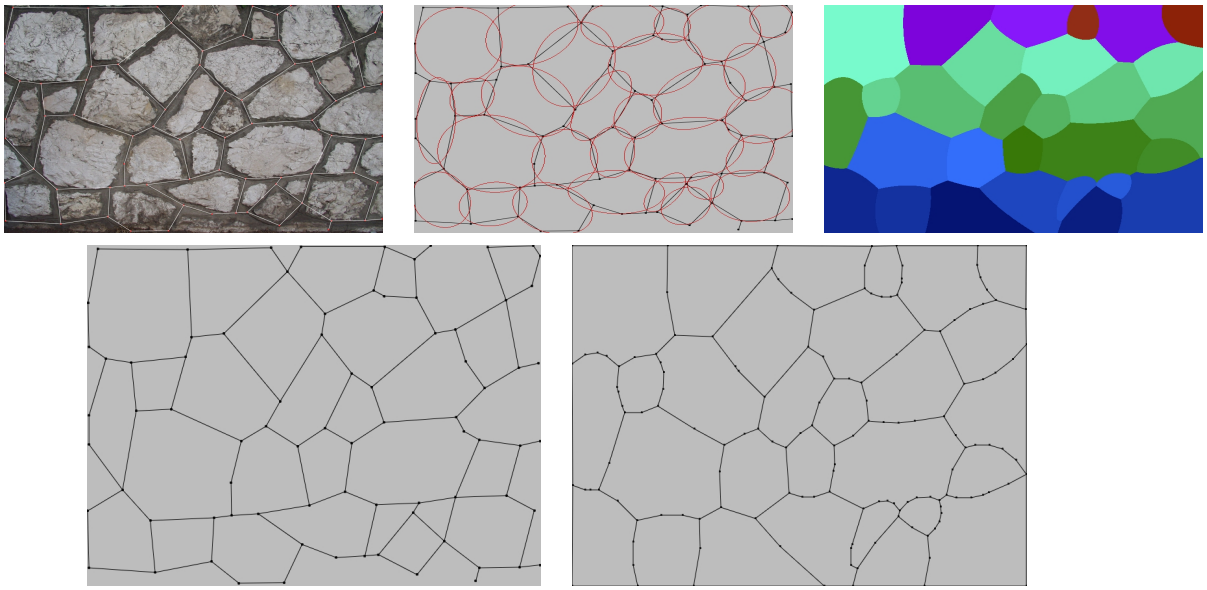

Figure 3.13 Top row: Left, structured texture with associated geometric subdivision. Center, fitted Ellipses. Right, re-synthesized with anisotropic metric. Bottom row. Left, the original subdivision (drawn by hand). Right, the subdivision generated by using the ellipses fitted on the original subdivision. Visually, the regions correspond quite well.

3.5.2 Anisotropic Centroidal Voronoi diagram

We can call the Diagram obtained using the metric defined in the preceding section an *Anisotropic Voronoi Diagram*[5]. We can compute an approximation to an *Anisotropic Centroidal Voronoi diagram* using the discrete version of the Lloyd's algorithm described in section 3.2.1: Given an initial set of generators with their associated ellipses, we alternate between computing the anisotropic discrete Voronoi diagram of these generators as described in the preceding subsection, computing the centroid of the generated Voronoi regions and moving the generators (i.e., ellipse centers) to these computed centroids. The computation of the centroid is done in the same manner as for an ordinary discrete Centroidal Voronoi Diagram. After a few iterations, we end up with an Anisotropic Centroidal Voronoi Diagram composed of anisotropic regions. Like for its isotropic

[5]Labelle and Shewchuk [LS03] and Du and Wang [QD05] have recently proposed other definitions of continuous anisotropic Voronoi Diagrams.

version, the regions reach a minimum-energy configuration after a few iterations.

It is possible, as an option to the algorithm, to let each ellipse rotate around its center in addition to aligning its center to the computed centroid. For this, once we have computed a new region, we can compute the ellipse that would fit inside it and rotate the original ellipse by the angle of this new ellipse. Step 2 of the Lloyd's (section 3.2.1) thus becomes:

2 Compute the centroids and the fitting ellipses of the n Voronoi regions issued from step 1. Move the generators to the computed centroids. Rotate the original ellipses to match the computed ellipse angle.

We do not modify the original ellipse axis lengths so as to preserve the region eccentricity.

The two rows of figure 3.14 show two examples of the computation of an anisotropic Voronoi diagram of a set of ellipses. On the left, we can see the original subdivision, superimposed on the corresponding anisotropic discrete Voronoi diagram, computed by using the ellipses fitted to the original regions. We can observe that the Voronoi regions approximate the original subdivision regions well, except that no T-junctions seems to be reproduced, and there are no more (or less) than three regions meeting at a same point. In the second column, we can see the resulting diagram (and ellipses) after a few iterations of the anisotropic Lloyd. The ellipses have moved in order to evenly occupy all the space available in the plane. This is particularly noticeable in the top example, as two thirds of the space was initially covered by the subdivision regions, leading to a larger displacement of the ellipses and a growth of each corresponding region. The third column shows the same diagram as the second but by visualizing the distance function for all the regions. The image is colored with respect to the distance to the closest Voronoi generator. The closer a pixel is to its generator, the darker it is. The rightmost column is the result of using the anisotropic Lloyd on the original subdivision and letting the original ellipses rotate.

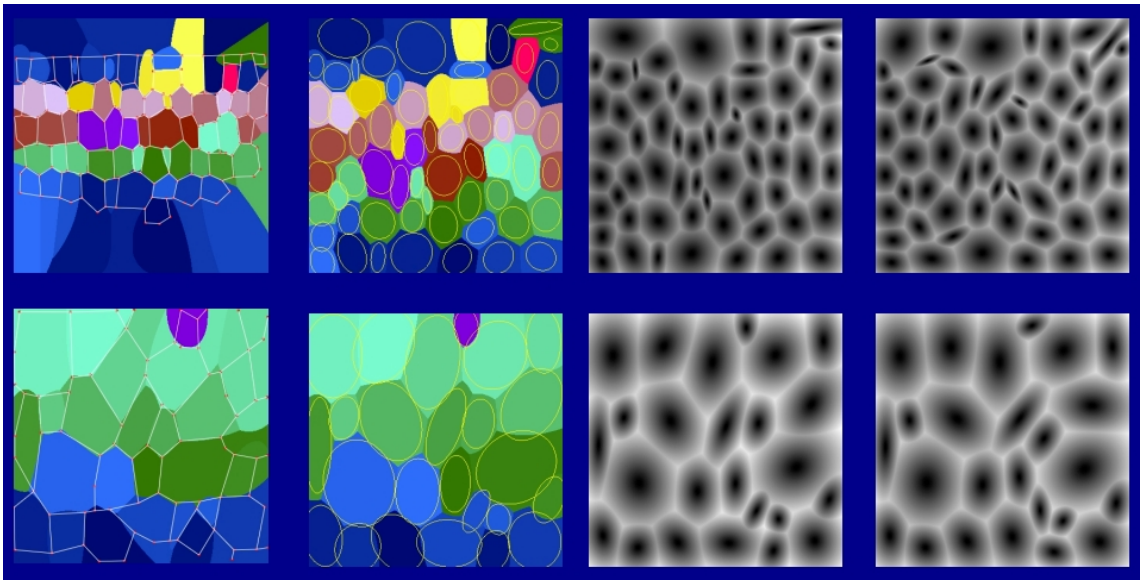

Figure 3.14 Left column: Original subdivisions superimposed on top of the corresponding anisotropic Voronoi Diagram. Second column: Resulting Voronoi Diagrams and elliptic generators after a few iterations of the Lloyd's algorithm. Third column: Same as second, visualized with the distance function. Right column: Result after the Lloyd's algorithm, but allowing the ellipses to rotate.

3.5.3 Trivial synthesis

Now that we know how to generate a Voronoi diagram using an anisotropic metric defined by an ellipse, we can synthesize some subdivisions using the information extracted from the analysis of the original subdivision. A trivial way to do this is to modify the input data and run the Lloyd again to re-stabilize the subdivision. This can be done by adding some noise to the original ellipse data, slightly modifying the center locations or angles of the ellipses. It can also be done by moving regions around, exchanging regions randomly. Figure 3.15 shows two examples of original subdivisions (left column) and some similar subdivisions generated by exchanging regions in this manner. We can observe that indeed the synthesized diagrams look similar to the original.

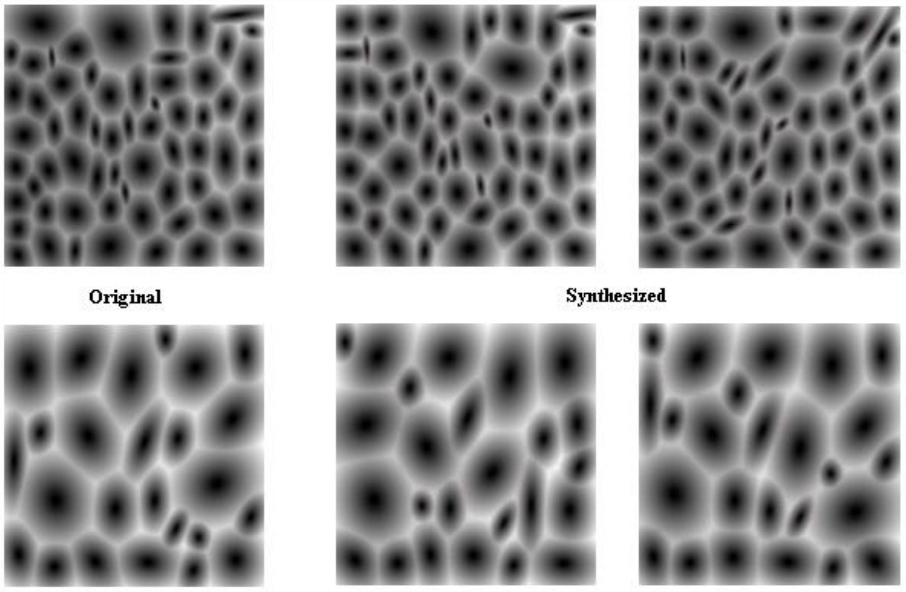

Figure 3.15 Trivial synthesis: exchanging original regions randomly, followed by some iterations of the Lloyd's heuristic.

However, this kind of synthesis is limited. We are only moving around the original regions and thus cannot apply it to a differently sized target area. Tiling the original sample would also be easily noticed. Moreover, if there is some kind of meta-structure, it could be damaged by these blind operations. In order to synthesize a subdivision inspired by an original, we use statistics gathered during the analysis, as described next.

3.5.4 Statistical synthesis

During the analysis phase, we collected information about the region shapes and the topology of the subdivision in the form of histograms. We can now sample these histograms to draw values in order to generate similar regions. This statistical synthesis is a two-stage process: We first sample a number of regions and distribute them on the plane using the Lloyd's algorithm (global, macro-positioning) while in a second stage, we correct the region shapes locally (micro-positioning) according to a shape criterion.

First stage: Sampling regions and populating the plane

To synthesize a new subdivision, we first start by uniformly sampling data from the ellipse parameters distributions. Our data is stored in the form of histograms, where each of the histogram bins counts the number of events falling into a given range of a continuous variable. Assuming that the distribution of events in a bin is uniform, we can convert an histogram into a probability distribution function and obtain a cumulative probability distribution function. This allows us to generate samples with the desired probability distribution via the inverse transform and using a random number generator (for details, refer to the GSL manual [Bee01], which is the library we are using for statistical operations).

The density of ellipses to generate should approximately correspond to the initial ellipse density. For this, we use the following rule: we compute the mean region area value

$$\bar{r} = \frac{1}{N} \sum_{N} r_i,$$

where N is the total number of regions and r_i is the area of region i. Given a new area A' to fill, the number of ellipses N' to generate is

$$N' = \frac{A'}{\bar{r}}.$$

In addition, we offer the possibility for the user to input a *scaling parameter*. This parameter will directly influence the number of regions to generate and thus their final size. A smaller scaling value implies a larger number of regions to generate, and thus a smaller average region size as more regions will compete for populating the same area.

The initial ellipse center positions and the two axis lengths are stored in 2D histograms. The ellipse angles are stored in a 1D histogram. We assume for now that these data are uncorrelated. We sample the values necessary to generate the N' ellipses needed to fill the new area A' from these histograms. We obtain a set of ellipses as can be seen in figure 3.16, left. These ellipses overlap so we need to spread them out on the plane. A few iterations of the anisotropic Lloyd's allow us to obtain the result shown on the right, where each ellipse has found its place and the target plane (subset of $\Re^2$) is entirely tiled (by construction of the discrete anisotropic Voronoi).

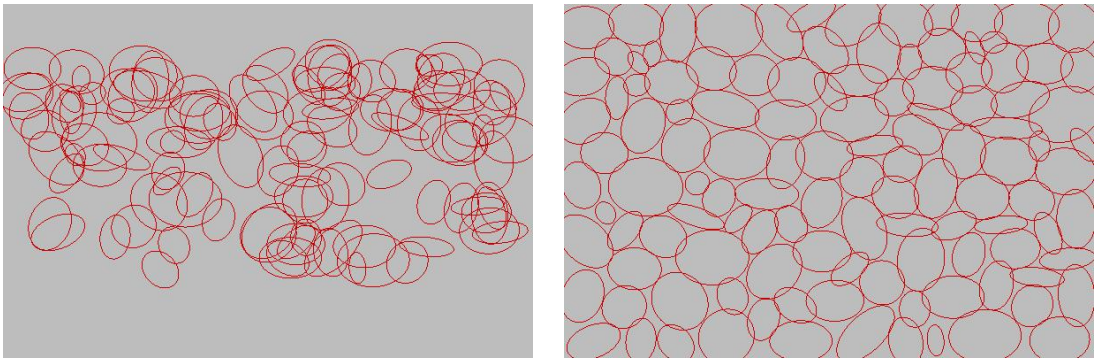

Figure 3.16 Left: Sampled ellipses. Centers have been sampled from their distribution. However, this is not a requirement. Right: Same ellipses, distributed on the plane with Lloyd's algorithm.

Using this technique, we are able to generate subdivisions where the regions eccentricity and orientation average to those of the original regions. Figure 3.17 shows some of these results. In the left column, we have an original subdivision from which similar subdivisions were generated

(right-hand side columns). In the lower right corner, we show a result where we have augmented the region density. As we sample the eccentricity, the ratio of region sizes in the entire subdivision is respected.

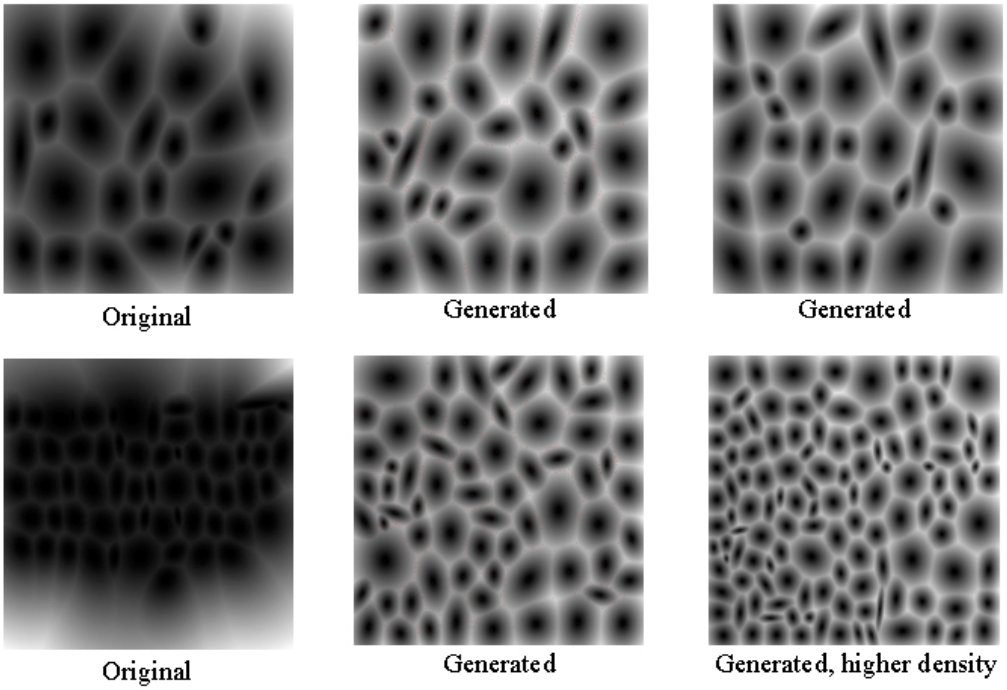

Figure 3.17 Results of statistically generating subdivisions.

Using density maps The technique we just described works well for subdivisions where the original regions are randomly placed on the plane. However, some original subdivisions exhibit some meta-structure, as can be seen in figure 3.19 left (in this case, a layered meta-structure), which is not preserved by the basic Lloyd's algorithm. To take this type of meta-structure into account, we use *density maps*, such as the one pictured in figure 3.18 left, which corresponds to the structured texture shown to its right. We use the map to constrain the generated ellipses to stay as close as possible to the areas of higher density (darker areas). This is done by influencing the computation of the centroid during the Lloyd's iterations. Pixels whose corresponding value in the density map are dark have a bigger weight than brighter pixels. This has the effect of making the regions "slide" towards the darker areas, until convergence of the algorithm. Figure 3.19 (right) shows the effect of using a density map to control the placement of the regions during the Lloyd's iterations whereas the regions shown in the center image were unconstrained. We can clearly see that the layered structure is preserved.

To obtain a density map from a given original subdivision, one could draw or compute it according to the subdivision regions center position (the highest density will pass near the centers). It could also be generated automatically using a combination of image analysis techniques. To generate subdivisions on areas of larger sizes, the corresponding density maps will have to be drawn again, or extended using texture synthesis techniques. It also has to be scaled according to the scaling parameter used.

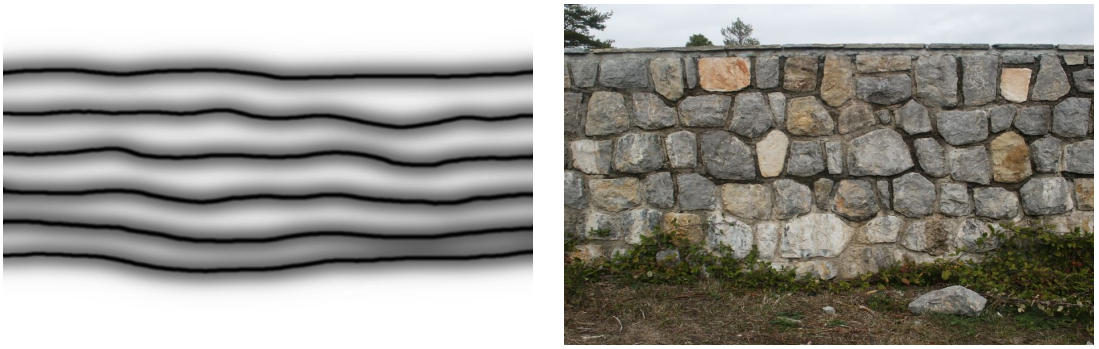

Figure 3.18 A density map used to control meta-structure.

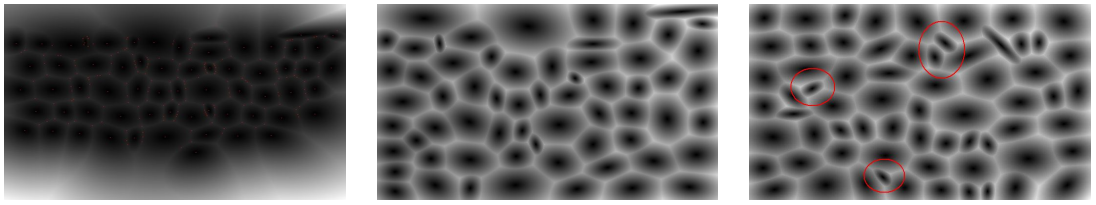

Figure 3.19 Left: Original subdivision. Center: Synthesis without density map. Right: Synthesis with density map; the layered structure is preserved. Some annoying regions are indicated in red.

At the end of this first "populating" stage, we end up with a series of regions partitioning the planes and whose global arrangement and eccentricity should correspond to the input subdivision. However, if we look at results (many of which can be observed in section 3.6), we clearly see that in some cases the obtained subdivisions are too "cellular-like" because of the Voronoi process: many regions look like elongated hexagons and no more than three regions meet at the same point (3-valency). This is restrictive as many of our subdivisions do not follow this local pattern. In order to correct this, we apply a corrective stage whose purpose is to locally correct the region shapes.

Second stage: Correcting region shapes

The region shapes we obtain from the first stage have an eccentricity consistent with those of the original regions but the shape of their contours may not totally correspond (i.e., they are not visually similar), and thus the parameters considered in our representation do not completely encode the shape. We thus have to perform some local corrections in order to optimize the generated shape and make them resemble more the originals. For this, we use a shape criterion to characterize the similarity between two polygonal shapes.

In order to rectify the region shapes, we must convert the colored regions which compose the discrete anisotropic Voronoi Diagram computed by the last iteration of the discrete Lloyd's into a modifiable geometric mesh where all the regions are connected. For this, we use a traditional image analysis segmentation algorithm to detect the region contours followed by a custom polygonization technique of the segmented regions, along with a snapping heuristic to connect the polygons together and get the required mesh.

To evaluate if a given shape is geometrically "close" to another shape, we use the comparison metric described by Arkin *et al.* [ACH+91] to compare polygonal shapes. This metric evaluates the *turning functions* of the polygons to compare. A turning function maps a fraction of arc length

in a polygon to the angle of the polygon at that point. A turning function is thus translation and scale independent (the polygon perimeters being normalized to 1). If we translate the function itself, it corresponds to rotating the polygon and moving the point where the measurement of the arc length begins. To evaluate the distance of a polygon to another, they compute the minimal L^2 distance between all translations of the two polygons turning functions. This number constitutes the similarity measure of the two polygons. If it is less than 1, the polygons are very similar. Their algorithm is efficient, running in $O(n^2 \log n)$, where n is the total number of vertices of the two polygons and the measure corresponds well to how we would subjectively evaluate the similarity of two shapes.

To improve the similarity of the generated subdivision to the original one, we want to optimize the region shapes in order to make generated shapes closer to the original, analyzed, shapes. For this, we associate to each generated shape the original shape that is the most similar according to the shape measure described above. This closest shape must also not be more than 20% larger or smaller than the shape it is being associated to. We do not want to scale the regions as size might be an important feature of the original subdivision. Figure 3.20 left, shows a generated subdivision (in black) with, for each generated region, the superimposed closest original region (in red). We can see that the shape criteria works well. If region orientation is an important characteristic of the original subdivision, we provide the option to prevent the rotation of the associated closest region as can be seen in figure 3.20, right. The process is summarized in algorithm 2.

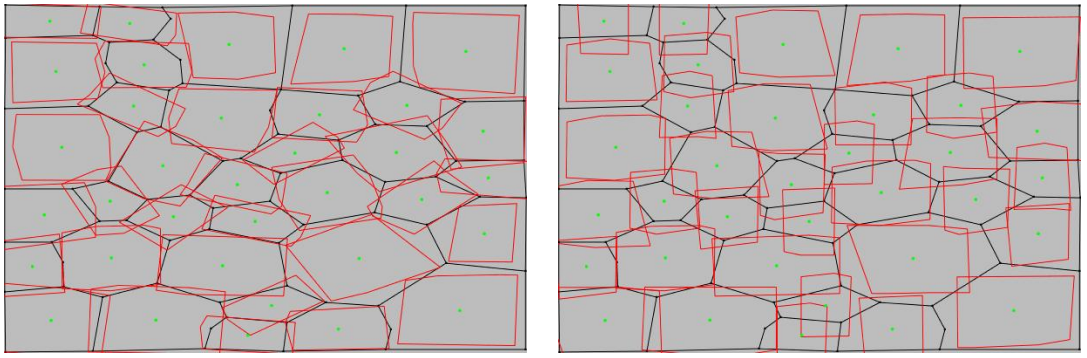

Figure 3.20 Generated subdivision (in black, after first phase) with closest original regions (in red). Rotation of similar regions has been forbidden in right image, whereas it was allowed in left image. This subdivision comes from the *Valbonne Wall* example, cf figure 3.26.

Algorithm 2 Association of closest shapes

procedure ASSOCIATESHAPES(S) ▷ $S(E, V, R)$: generated subdivision
  for each generated region R_g **do**
    Compute similarity measure with all original regions Ro_i
    Keep the region R_l having the lowest measure
    Align the centroid of R_l with the centroid of R_g
    for each vertex v of R_g **do**
      Compute closest vertex v_l in R_l
      Associate v to v_l ▷ Put v in a list of vertices associated to v_l
    end for
  end for
end procedure

Once we have associated an original shape to each generated shape (aligned on their centroid), we are able to compute a similarity measure for the entire subdivision and a mean shape deviation error. To lower this error, we improve the shapes using the process described in algorithm 3. For each vertex, we compute the direction vectors to the closest vertices in each of the original shapes associated to the vertex adjacent regions. We then calculate the mean vector of these vectors to get the "optimal" direction to improve the shapes of all the vertex adjacent regions. We displace the vertex 20% (user-defined value) in this direction. The new subdivision error can be recomputed and the process iterates until the error falls under a certain threshold. From time to time, updating the associated most similar shapes can improve the results. We also optimize on the topology and region information by verifying the edge lengths or angles are part of the original distribution and correcting them if necessary.

Figure 3.21 summarizes this shape optimization process. On the left, we can see the regions as they were output from the first stage. In the center, we can observe the result of applying the second stage. The regions shrunk as the hand-drawn original regions did not cover the entire plane, thus the generated regions grew larger to fill-up the available space during the repartition stage. On the right, we recomputed the closest shapes and performed more shape optimization. The results of both stages can be compared with the original subdivision in figure 3.24.

Algorithm 3 Optimization of the generated region shapes

procedure OPTIMIZESHAPES(S) ▷ $S(E, V, R)$: generated subdivision
  for each generated vertex v_g **do**
    Get its list of associated vertices V_a
    for each vertex v in V_a **do**
      Compute the vector $\overrightarrow{D}$ from V_g to v
      Compute the mean vector $\overrightarrow{D_m}$ of these vectors
      Displace V_g along $\overrightarrow{D_m}$ by 20%
    end for
  end for
end procedure

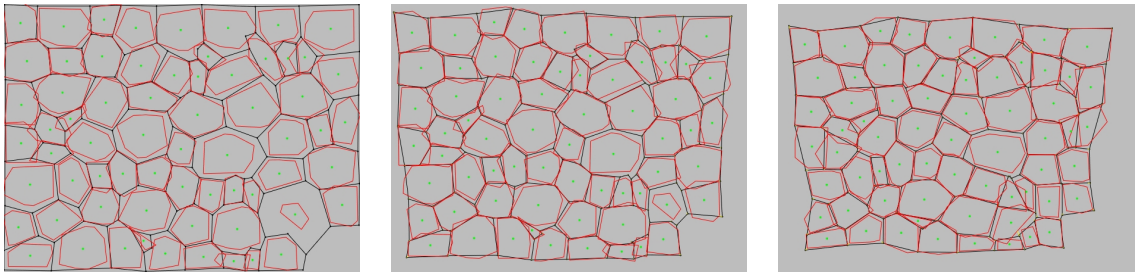

Figure 3.21 Illustration of second synthesis stage. Left: Closest regions before optimization. Center: After optimization. Right: After recomputing closest regions and some optimization.

Image 3.22 summarizes our two-stage synthesis process. We applied it to a variety of input subdivisions and the next section presents the results obtained.

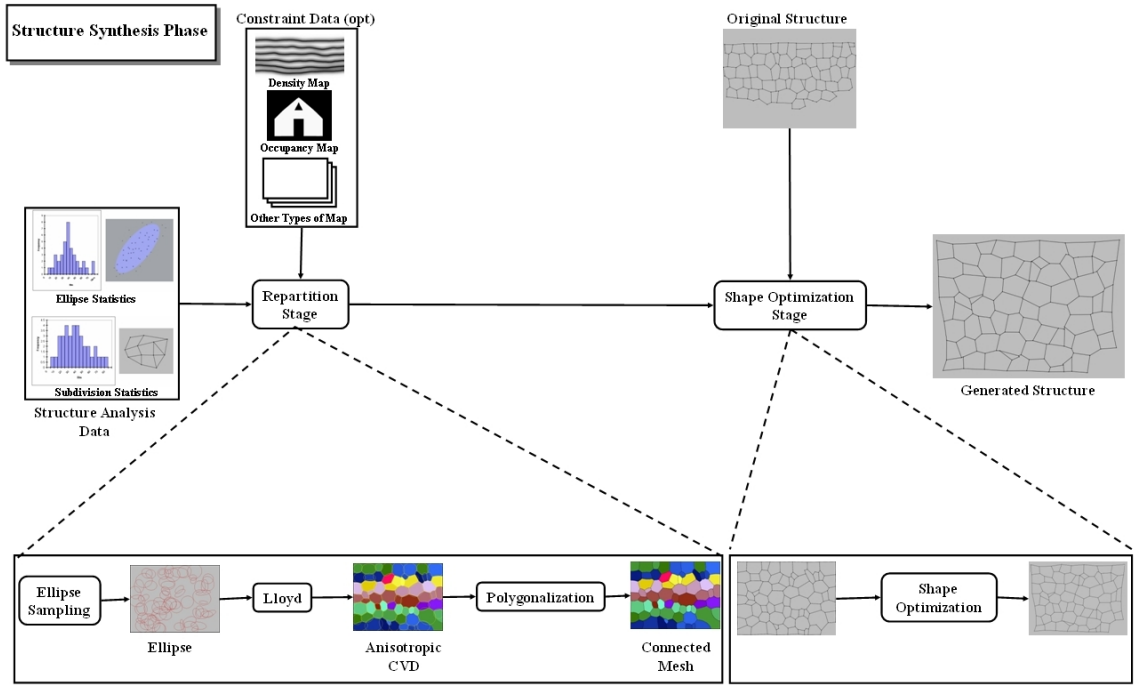

Figure 3.22 Structure Synthesis Process.

3.6 Results

We tested our algorithm on various structured textures such as stone walls, snake skins and cellular structures. The geometric subdivision corresponding to each input image is drawn by hand since the image analysis techniques we tried are not yet robust enough to automatically extract such a graph from an image. In this section, we present the results we obtained and analyze them.

3.6.1 Wall 1: INRIA wall

The first subdivision we handle comes from an image of a wall (figure 3.23, top left) with no particular meta-structure. It is constituted of a mixture of large and small rocks. The corresponding geometric version can be seen in figure 3.23, top right. The results obtained after the first and second stage can be observed in the bottom row images. We can see that the ratio of small regions has been preserved and that the eccentricity of the regions looks correct.

3.6.2 Wall 2: Levens wall

The second subdivision is also a wall which exhibits a layered meta-structure, as can be seen in the original image and drawn subdivision (figure 3.24, top row). Since there is an underlying meta-structure, we use a corresponding hand-drawn density image (figure 3.18) in the first stage of the Lloyd's algorithm. This results in the subdivision of figure 3.24, bottom left. The image on the right shows the result after the shape optimization stage. We can see that the similarity of

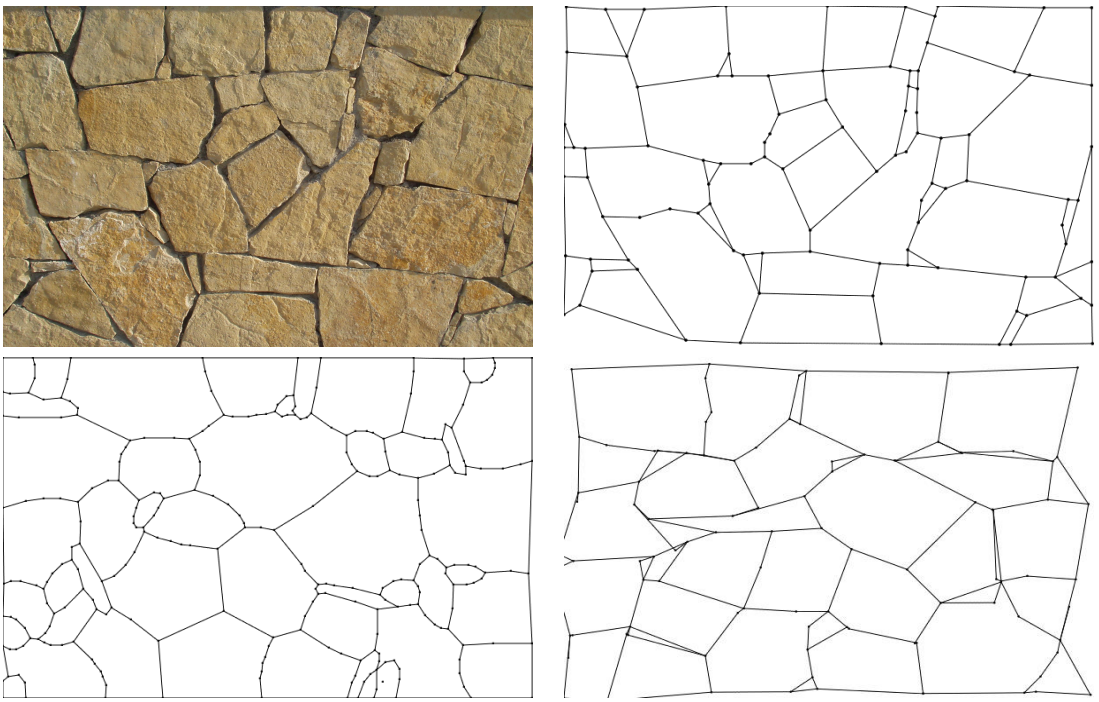

Figure 3.23 Inria wall example. Top row: Input image and corresponding subdivision. Bottom row: Left, the generated subdivision after the first stage. Right, the generated subdivision after the second stage.

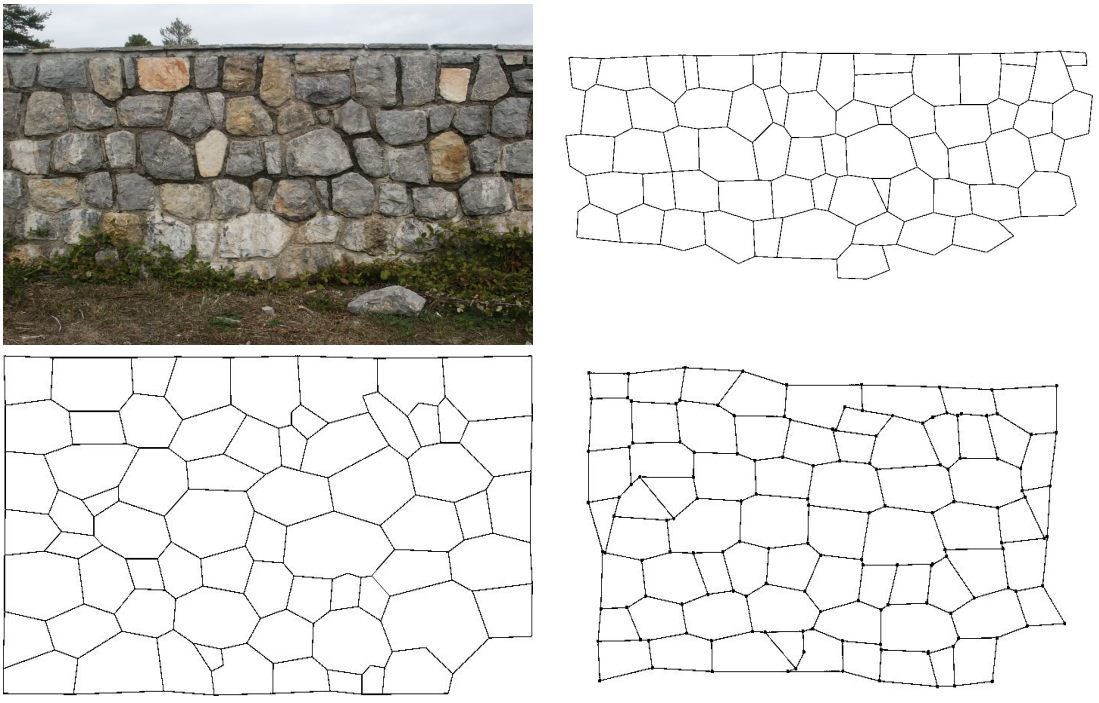

Figure 3.24 Levens wall example. Top row: Input image and corresponding subdivision. Bottom row: Left, the generated subdivision after the first stage (density map used). Right, the generated subdivision after the second stage.

the shapes has improved but there are still some artefacts. The density map is not used during the second stage as the regions do not usually move too much. It is mainly their contour which is affected. However, in some cases, it might be interesting to consider the density map by, for example, penalizing vertex displacements which move them away from the higher density zones.

3.6.3 Wall 3: Valbonne wall

The third example (figure 3.26) is again a wall which exhibits an even more pronounced layered meta-structure. Another particularity is that the shapes of the regions are quite regular. The density map used is very simple, as shown in figure 3.25.

Figure 3.25 The density map used for the Valbonne wall example.

The results of the two stages are shown in figure 3.26. As the first stage does not include any constraint on the shape of the regions, the generated regions are not very similar to the input ones. The second stage helps to correct the shapes, which become less hexagonal. The association of closest shapes does not allow the rotations of the associated regions as the original orientation information is important in this example (as shown in the right-hand side of figure 3.20).

Some problems can be observed in this result. The main issue (meta-structure not preserved) is mainly due to the fact that, in our statistics, we supposed that the angle and the axes data are not correlated. For regular structures such as this one (or traditional brick walls), angle and axes lengths are heavily correlated. Generated regions such as the ones circled in red, correspond to small input bricks and should have been placed upward. We propose some solutions in the last section of this chapter.

It should be noted that if the result of the first stage is not satisfying, like in this case, there is not much that can be done to correct the problems in the second stage if it relates to the global positioning of regions. The second stage only performs shape optimization but the region locations will not change much. Some suggestions for improvements are given in the conclusion of this chapter.

3.6.4 Biological tissues: Meristem

The fourth example is an image of a meristem (figure 3.27, top row), a plant tissue from which new biological cells are formed. It does not have any particular meta-structure. The results after each one of the stages are shown in the bottom row of image 3.27. As the structure to imitate is already cellular, the second stage does not perform significant changes but still allows to recover triangle-shaped cells for example.

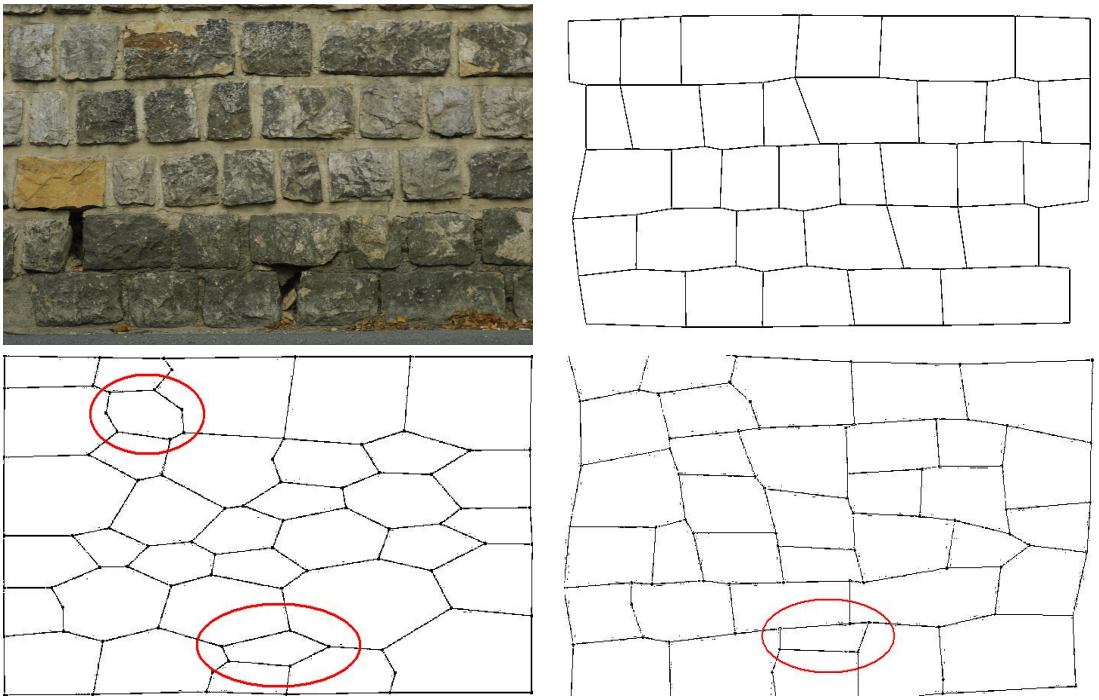

Figure 3.26 Valbonne wall example. Top row: Input image and corresponding subdivision. Bottom row: Left, the generated subdivision after the first stage (density map used). Right, the generated subdivision after the second stage.

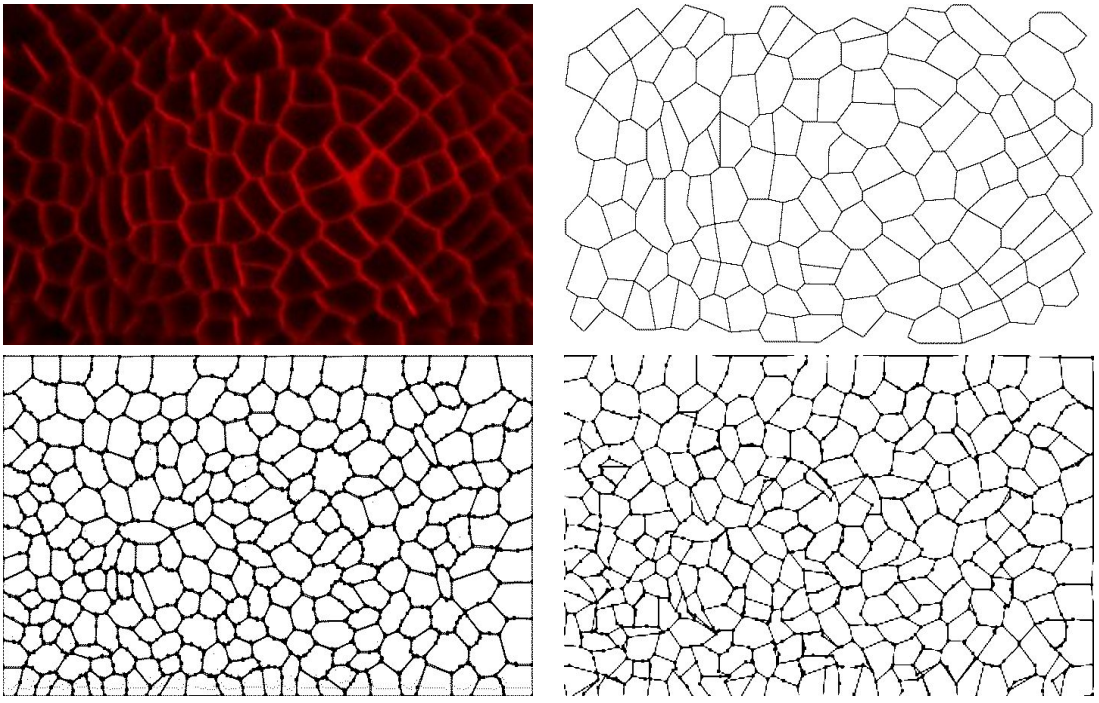

Figure 3.27 Meristem example. Top row: Input image and corresponding subdivision. Bottom row: Left, the generated subdivision after the first stage. Right, the generated subdivision after the second stage.

3.6.5 Organic structure: Snake skin

The fifth example (figure 3.28) is a natural structured texture: a snake skin. It is similar to the Valbonne wall example since it has an underlying meta-structure and regularly-shaped regions, but it also has small little regions, like the INRIA wall. Results are shown in the bottom row of image 3.28. We can note the presence of the small triangles as in the original subdivision. We updated the closest shape association a few times during the second stage to achieve the best fit possible between the regions.

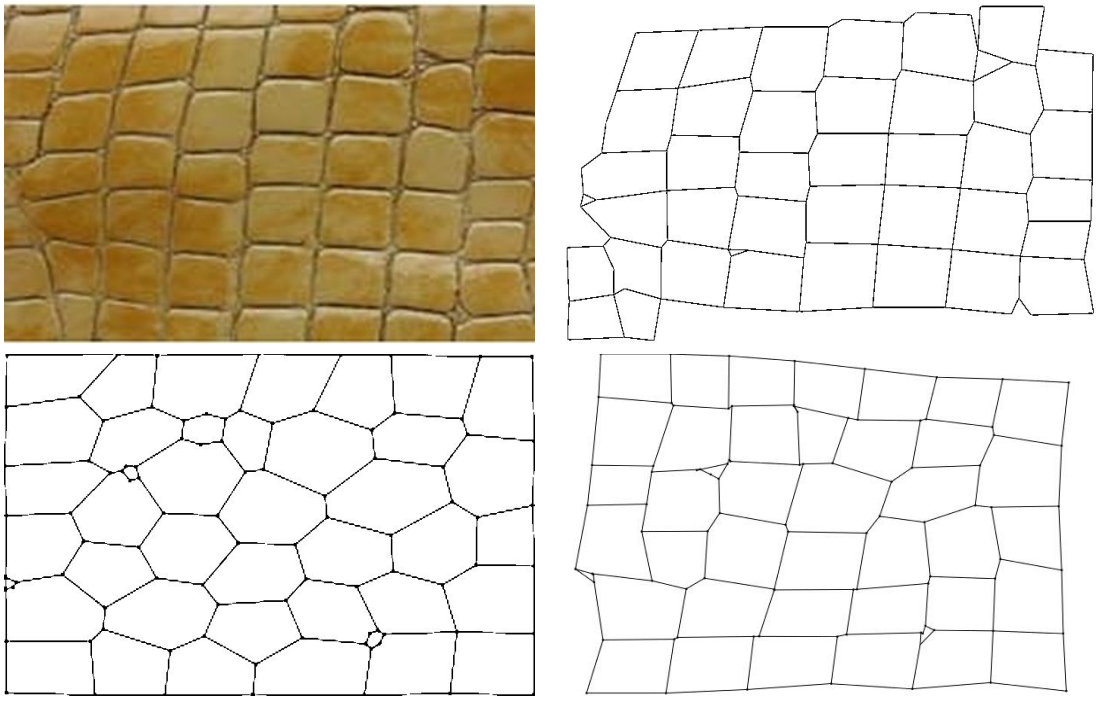

Figure 3.28 Snake skin example. Top row: Input image and corresponding subdivision. Bottom row: Left, the generated subdivision after the first stage. Right, the generated subdivision after the second stage.

3.7 Structure manipulations

The Lloyd's heuristic possesses some nice properties that we can take advantage of. As a global positioning process, it allows a variety of modification operations such as moving, adding or deleting a region. It can also handle some constraints, as we discuss in this section.

3.7.1 Structure modifications

Our generated structures can nicely handle modifications. The following operations are automatically taken into account by running the Lloyd's after performing them. This has the effect of re-arranging the regions globally and stabilizing the mesh again.

Deleting a region We can choose to remove any region from the set of generated regions, for example if we are not satisfied with a certain region shape and/or position. In this case, one ellipse disappears and running the Lloyd's algorithm on the remaining ellipses progressively fills the gap. However, we must be aware that since we have the same amount of space to

cover with fewer ellipses, the size of the generated regions will increase and their shape will be slightly modified (especially in the neighboring regions). The sizes increase not only for the neigboring regions but globally throughout the whole structure. Szakas *et al.* [ST04] recently suggested an algorithm to rapidly update a discrete Voronoi diagram when one of the generators is removed. We could adapt his technique to our particular anisotropic diagrams to make these changes interactive.

Inserting a region We can insert a region anywhere and the regions around are progressively "pushed away" from it. As above, the density relationship described in 3.5.4 implies that the overall region sizes shrink to make space for the new one. Figure 3.29 shows the results of inserting a region in the middle of the subdivision. It can be useful if we want to insert a particular region shape in the set of generated regions at a desired position. It is also possible to freeze a region at a certain position. The other regions will then move around it.

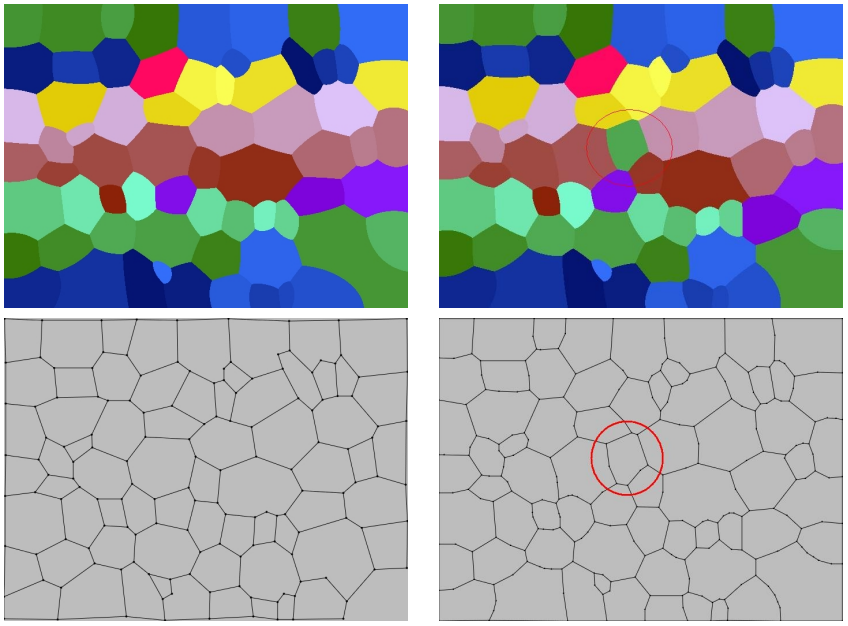

Figure 3.29 Insertion of a region. Top row: The Anisotropic regions before (left) and after (right) the insertion. The inserted region is indicated by a red circle. Bottom row: The corresponding subdivisions.

Moving a region We can displace a region from one place to another. It basically corresponds to removing it from one place and inserting it elsewhere. The region sizes are thus not modified, as can be seen in the examples shown in section 3.5.3 on trivial synthesis.

Changing the size and/or orientation of a region More local region operations can also be performed and the structure updated along the same lines as the other operations.

After the graph has stabilized, we must run the optimization stage to correct the contours. Interactive refinement operations on the borders themselves are also possible as the obtained mesh is fully editable. All these changes would be very difficult to handle with another kind of generation method, such as a particle system or the systems described in section 2.3.2. The Lloyd's algorithm allows us local control on our result. It should be noted that the changes are local but the effect is global, being distributed on the whole mesh. However, regions far away from the changed area will be much less affected than closer regions, which is a desired property for predictable results. Even more control can be achieved by the insertion of constraints, as we show in the next section.

3.7.2 Controlling the synthesis

The Lloyd's heuristic alternates two steps to cluster the pixels of our images into separate regions:

1. Computation of the Voronoi diagram using the given **metric**.
2. Computation of the region **centroids**. **Displacement** of the generators to the computed centroids.

In this work, we have modified the first step to take into account an elliptic metric and produce anisotropic regions covering the plane. We have also shown that it is possible to influence the computation of the centroids, and thus the position of the ellipse centers, through the use of density maps, giving more importance (weight) to some pixels. If required, the displacement of the generators (i.e., the ellipses) can also include a rotation of the ellipse so that it fits the latest Voronoi regions computed.

Other parameters of this process can be influenced or directly controlled (such as the ones in bold above), mainly through the use of *constraint maps*. We describe in this section some of the maps which can be used to offer some control on the results.

Occupancy map

One of the most important types of control is the ability to define the shape of the area to populate, i.e., the areas of the 2D plane where the ellipses are allowed (or conversely forbidden) to evolve. We call such a constraint an occupancy constraint.

We have experimented with two ways to define this constraint: Either by considering interactive, user-drawn constraints or by using an occupancy mask such as the ones in figure 3.30 whose black pixels define forbidden regions.

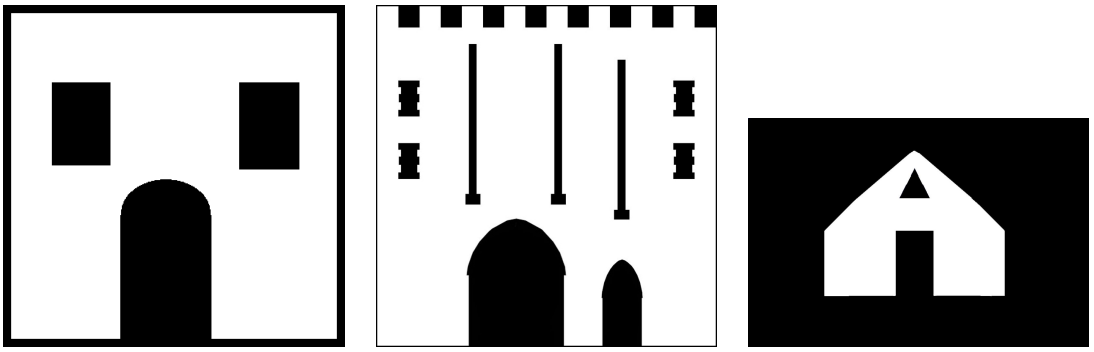

Figure 3.30 Three occupancy constraint maps: there should not be any regions in the black areas.

To take such a constraint into account, we must proceed to the following operations:

1. Remove the ellipses intersecting the forbidden areas (black) of the input occupancy map.
2. Define all the pixels in the constraint areas as invalid pixels: they will not be inserted into the priority queue used to simulate the flooding process. Thus the generators do not enter the constraint areas.
3. The constraint area pixels will not count as region pixels for any surrounding ellipses (i.e., they will not be accounted for in the centroid computation).
4. Fix the mesh vertices on the border of the constraint areas to avoid their displacement during the optimization stage.

Some results can be observed in figure 3.31 while the original subdivision is shown in image 3.32 for comparison purposes.

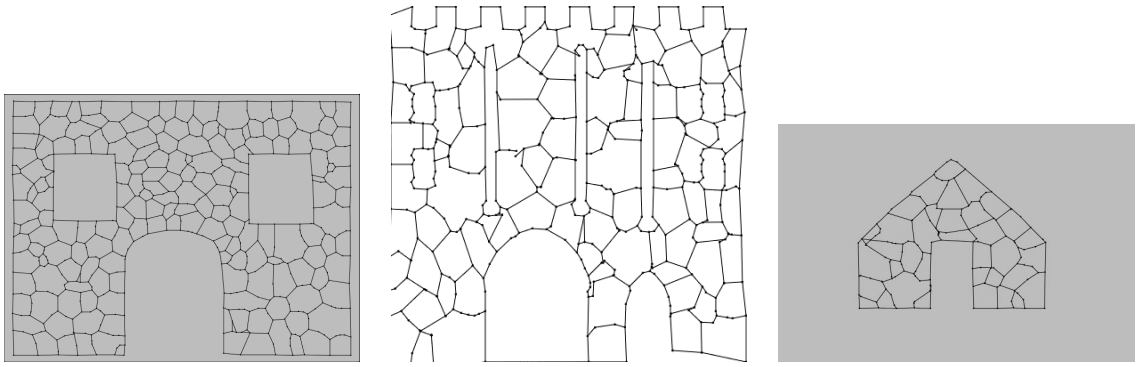

Figure 3.31 Occupancy constraint results

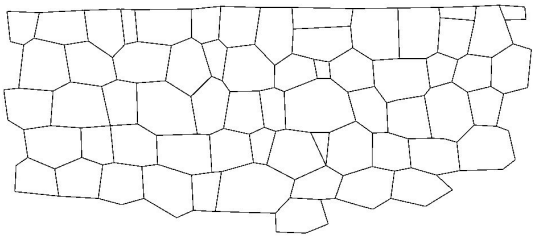

Figure 3.32 The geometric subdivision of the Levens wall structured texture.

The configuration of the ellipses before and after the constraint has been applied can be observed in figure 3.33. For now, the occupancy constraint is only defined on a 2D plane but we

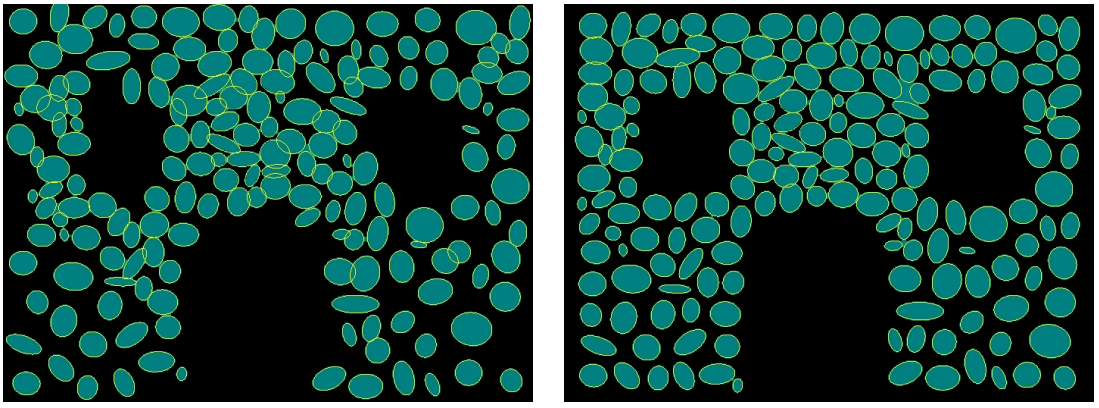

Figure 3.33 Configuration of the ellipses before and after the constraint.

discuss in the last section the applications to a 3D domain. Other types of constraint maps can also be considered and the next section presents some of them.

Other maps

Some other parameters of the Lloyd's can be affected by the usage of control maps. Some possible constraint maps include:

Size map This (grayscale) map controls the size of the ellipses. The more an ellipse is within a dark area, the more it reduces its size (up to a minimum value). Alternatively, we can directly impose a fixed size to ellipses within the defined areas. This can prove useful for architectural applications where smaller stones are required around features like windows or doors. More advanced rules could control the size of the regions.

Orientation map This map influences the ellipse angles. Each pixel in the map contains a direction vector (issued from a direction field or from another source) which can be converted into an angle to assign to the ellipse containing this pixel. As an example, we can imagine the stones around a door to be forced to be aligned perpendicularly to the door frame.

Metric map This map could influence the computation of the metric locally to control the shape of the resulting regions. The metric could vary gradually or change instantaneously in certain regions. For example, exchanging our anisotropic metric for a Manhattan metric will generate square tiles, as done by Hausner [Hau01].

To reproduce the architectural design around the door shown in image 3.34, one would need an occupancy map to avoid having any stone within the window area, a metric map to influence the shape of the stones around the window, a size map to constrain the sizes of the stones (two sizes) and an orientation map to force the stones to be perpendicular to the window frame.

Figure 3.34 A window of Seton Castle, Scotland illustrating the need for controlling stone arrangements.

The procedural generators of Legakis [LDG01] can also control the size and orientation of its blocks (e.g., bricks). However, he has to program the generator for each specific model and he cannot edit anything after the system has placed the blocks. Our approach is more generic and fits the artists way of working as they are used to draw and lay several maps (appearance layers) on top of their models to define their final appearance. We can imagine several other kinds of maps, up to complex maps containing rules or procedures to apply.

3.8 Conclusion

3.8.1 Improvements

Looking at our result structures, we can detect some artefacts for which we present here possible solutions and improvements that we plan to integrate in the future.

Region placement problems

Depending on the original sampling, some regions can "get stuck" in an incorrect location (such as those circled in red in the right result image of figure 3.19) and not be able to move to a better position with the Lloyd's algorithm having attained a local minimum. The resulting arrangement is thus not completely satisfactory. In the same manner as Bossen and Heckbert [BH96] or Cohen *et al.* [CSAD04], we can use an operation Cohen *et al.* name a *region teleportation* to recover from such a configuration. This operation allows a region to be "transported" from its current location to a "better" location by removing it and inserting it somewhere else. In order to determine which region should be teleported, we propose to use an heuristic attributing a score to each region according to various critera. The score for each region would be a function of its number of edges, its area or angles (verifying if it is part of the corresponding statistical distributions), its distance to the closest high-density region, its deviation from the original shapes, etc. The region with the lowest score would be removed, and it could be reinserted in a location where the scores are high, running a Lloyd to restablize the configuration.

We are confident that this operation would improve the results on the Levens wall, as well on the Valbonne wall. For this last example, the regularity of the shapes and the fact that we did not correlate the axis lengths and the ellipse angles led to the observable defects. Performing an auto-correlation analysis on the input data would permit the detection of the correlation and appropriate statistics could be generated. Generating the region centers along the higher density lines would also be an efficient way to improve the results and should be offered as an option when handling a regular structure such as this one. Region centered in the low density areas should be forbidden.

Occupancy map constraint problems

From the observation of the occupancy map constraint results, we can infer the following improvements:

- The teleport operation would be also very useful. For instance, if we detect a big ellipse in a small allowed area, the shape of the resulting region might be wrong as it is mostly determined by the constraint shape. This ellipse should be removed and replaced by a smaller one, more coherent with the available space.

- If an ellipse covers too many pixels of a constraint area (its center is not allowed into it, but its axes are not constrained), split the ellipse or teleport it.

- The possibility to modify the region sizes during the process (while trying to stay coherent with the statistics) according to the available space.

In summary, there is still a lot we can do and experiment to improve our results. However, the appearance layers will help reduce (or hide) any small artefact.

3.8.2 Summary

We have described a technique to synthesize subdivisions from the analysis of a given subdivision sample. The analysis phase extracts statistical properties of the input mesh and characterizes the regions by computing a set of fitting ellipses. The synthesis is then performed by statistically sampling new ellipse parameters to generate similar regions. The generated ellipses are distributed onto the target surface (2D plane) using the Lloyd's heuristic, which can be influenced by the use of a density map if the original subdivision possesses an underlying meta-structure or other kinds of constraint maps. In a second stage, the region contours are adjusted using a shape optimization algorithm to better match the original input regions.

This two-stage process is necessary as it is a hard problem to generate the correct similar shapes first and try to pack them on the target plane. Moreover, there would not be any control or edition possible.

User control is allowed at various stages, going from controlling the overall sizes of regions through the scaling parameters up to using complex maps during the population phases to influence the regions locations, sizes, orientation and shapes. Local editing operations such as adding, removing or modifying a region are also possible. Most of this flexibility is due to the Lloyd's algorithm used in the first phase. The second phase is still necessary to adjust the region shapes after the modifications, and the final mesh is editable (as it consists in a real 2D mesh of polygons, edge and vertices).

In the end, we obtain a 2D editable mesh to which we can add appearance details to match the original textured structure or create a new one. This is done by filling the regions and contours with some detail information. The next chapter describes how we can add such 2D details to a generated subdivision, while the subsequent addresses the 3D detail aspect.

Chapter 4

Generating 2D Appearance From Example

"Realism demands complexity, or at least the appearance of complexity"

— Paul Heckbert, *Survey of texture mapping*

As discussed previously, our goal is to generate cellular textures based on examples. For all the reasons developed earlier (see chapter 1), we have chosen a "divide-and-conquer" strategy, thus separating the analysis and synthesis of the *geometry* (structure) from the *appearance* of these textures. In the previous chapter, we presented our method which produces a "bare" 2D structure, built of regions whose shapes are similar to those of the input examples. In this chapter, we will present our approach to analyze and generate the 2D appearance (or texture) associated with these regions. The untextured mesh defines the shape, orientation and location of the generated regions and is thus of prime importance in the final visual aspect of our structured texture. It constitutes the "backbone" of the final result. However, this geometric mesh is not usable "as is" in our target applications; we need to dress it with 2D textures. Its 3D visual appearance will be addressed in the next chapter.

In this chapter, we follow a similar strategy as that adopted for the *separate* analysis/synthesis of geometry and appearance. In particular, we will separate the problem of appearance analysis and generation into that of the interior of the regions and the materials separating them. This choice has a natural justification, since, in the real (textured) world, the structured textures we are interested in (rock walls, animal skins, etc.) are often composed of two different materials: one for the interior of the regions and another surrounding these regions. For example, a rock wall is made of rocks which have been excavated from some quarry and which hold together thanks to mortar or clever alignment. In the latter case, the outside material is simply empty space, but it still makes a visible separation between the regions. Thus, it makes sense to decouple the texturing of these two components as they are different, independent entities.

There is a great amount of detail information in textures, especially photo-realistic textures like the ones we can extract from our photographs. It thus makes sense to take advantage as much as possible of the richness of this available information. Texture gives us the geometric, color and reflection details we need to make our scene look realistic.

In order to texture our generated structures, we set the following goals and requirements:

Rendering speed Since our target applications can be interactive such as video games or interactive visualization of environments, preprocessing should be done off-line as much as possible, allowing real-time display of the end result.

Memory For some applications, memory can also be an issue (for examples, for game consoles). Ideally, we would like our solution to use as little memory as possible, especially since our structure synthesis model already does not require a lot of space.

Information reuse Our original image contains a lot of detail information and we should capitalize on this as much as possible to enrich our results.

User input The user is important and should be kept in the loop for controlling and editing of results.

Future reuse Detail information extracted from the original image should be put in a form that could make it usable later, for other different or similar operations.

As we have seen, a structure is the union of a finite set of closed regions. Each region is formed by a sequence of straight edges. An edge separates at most two regions and, in the following, we thus refer to it as an *interstice* between two regions to illustrate the fact that its width is mainly non-uniform. Similarly, a set of regions (and edges) meet at a structure vertex and we use the term *junction* in this case. The complexity of these contour elements can be used to simulate the 2D silhouette complexity of the individual regions. For instance, if our regions are rocks, the subtle geometric details from the rock contours can be visually approximated by the complex mortar silhouette instead of the rock contour itself. We aim to simulate these geometric details using the information contained in the images of the detailed contours.

In this chapter, we first describe how we separately address the texturing of these three elements: the region interiors, interstices and junctions. We then explain how we ultimately combine them to give a resulting, entirely generated textured structure exhibiting realistic details which are similar to the example input. We finish by showing some results and discussing the method. In this chapter, we speak of "structure" instead of "subdivision" since our focus is on the graphics aspect and the word "appearance" will be used to describe the 2D appearance (i.e., 2D colored textures) of our geometric structures. This final appearance will result from the combination of the extracted region details with the recovered contour details.

Figure 4.1 summarizes the analysis phase of our appearance recovery method. We have as input an original structured texture image and its corresponding 2D mesh. The appearance analysis phase produces appearance data which is used to synthesize the appearance of any given structure. This data is composed of *unified textures* and atlases. We create such textures for the material of the region interiors (see for example figure 4.9) and also for their contours (see figure 4.13 and figure 4.17). These unified textures allow us to generate similar, *unrepetitive* cellular textures over large amounts of geometry, with negligeable texture memory consumption, avoiding the pitfalls of texture generation techniques (see section 2.2.2 for a more detailed discussion).

Figure 4.2 summarizes the synthesis step. We have as input a 2D mesh and the appearance data of a previously analyzed textured structure. The synthesis phase will create one polygon for each region, edge and vertex present in the input mesh. It will also output textures coordinates which index the textures contained in the appearance data for all theses polygon vertices.

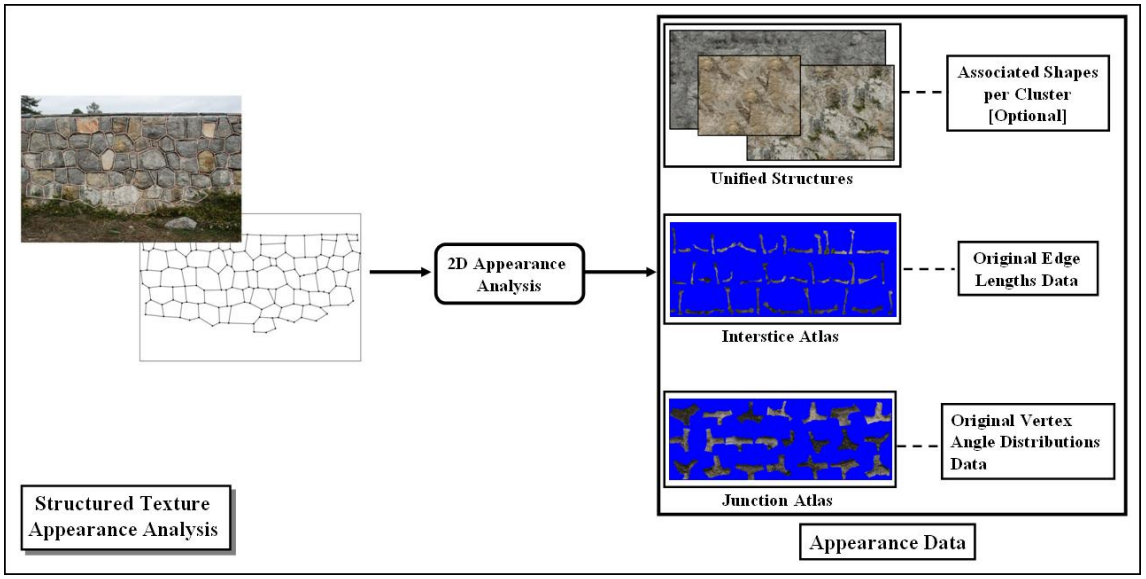

Figure 4.1 Analysis of the appearance of a structured texture.

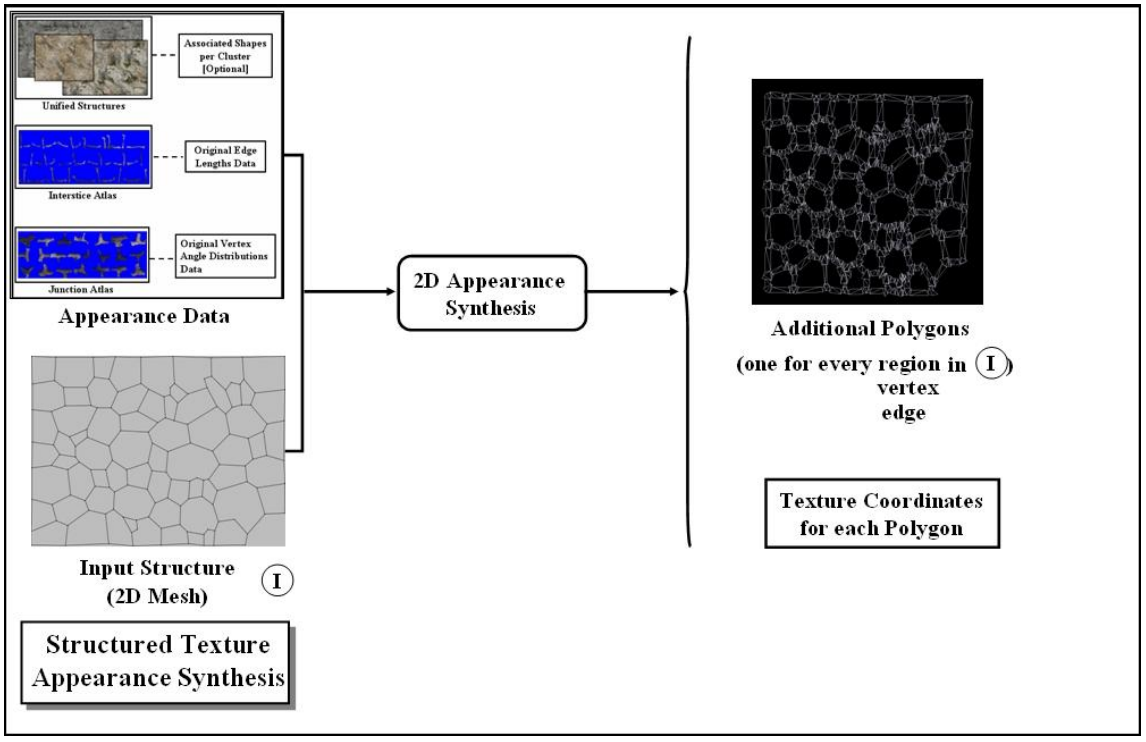

Figure 4.2 Synthesis of the appearance of a structured texture.

4.1 Generating region details

The details which compose a region can be due to geometry (silhouette of the primitive or 3D relief information), color and/or lighting. This information can be partly recovered from the original image, processed and used to synthesize similar details. In this section, we describe the method we adopt to collect the regions 2D appearance information from the input image and how we reuse it to texture the interior of any set of generated regions. The recovery of other types of details is addressed in the chapter 5.

4.1.1 Method description

We want to transfer 2D appearance details present in the original image to texture the generated polygonal geometric structure. We first concentrate on the details *inside* the regions (which constitutes the primitives of the structured texture analyzed), the contour details being handled by the recovery method described in section 4.2. We want to extract the color information from the original image using the corresponding user drawn 2D mesh indicating the location of the regions. Figure 4.3 shows an original structured texture photograph with its regions highlighted in white. The image to its right is a structure synthesis result (as described in chapter 3), based on this input, which remains to be textured. This information, along with the region correspondence information of the previous step, constitutes our input data, the output data being a fully textured structure.

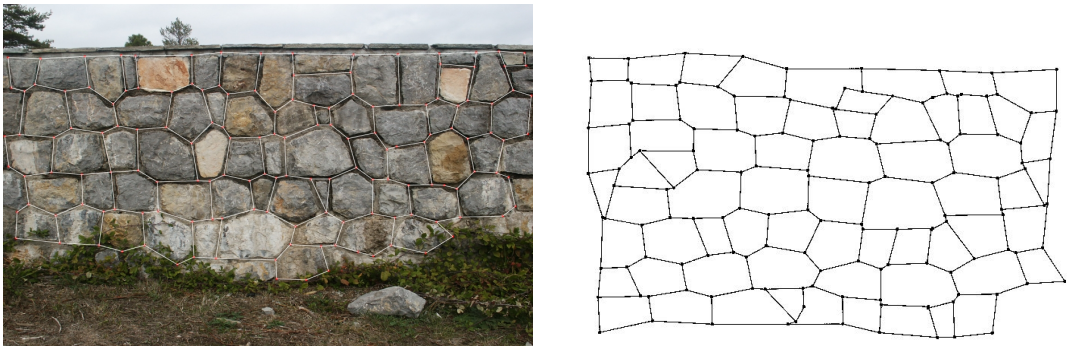

Figure 4.3 Left: The regions from an original image highlighted by the white geometric structure drawn onto the image by a user. Right: The generated structure (2D mesh of connected regions).

In order to transfer the color information from the original image to the result structure, various methods could be employed. We describe briefly here some possible approaches and their drawbacks.

- For each generated region, we could have cut the color information from inside a similar region and pasted it onto the target generated region (*texture transfer*). However, this approach implies some additional processing if the original region is smaller than the target, e.g., some pixel-based texture synthesis to fill the extra target pixels with some color information. The search for a larger region (where the generated region entirely fits in) is also not guaranteed to be successful.

- We could also have employed texture synthesis techniques to completely texture all the generated regions interiors. This approach poses a number of problems. First, a pixel-based synthesis technique (as described in section 2.2.2) might alter or even remove the small details that are important in the texture, such as marble veins for example. On the other side, a patch-based approach is complicated to implement as blocks (or patches) have to

be extracted from all the original regions. Special care has to be taken regarding their size as each block should originate from only one region at a time to avoid falsely including an interstice texture in the process. Another drawback is that texturing a large number of regions using texture synthesis techniques would not be interactive and, most of all, the memory consumption would be far too high to be usable in practice. In addition, texture synthesis has to be finely tuned to each example and can easily produce garbage, as seen in the images of figure 2.9.

- An interesting approach could be to compute the parametrization between a generated region and its closest original region (*closest* in the same sense as defined in subsection 3.5.4) and directly transfer the texture from the input region to the target one. A way to do this could be to map from the original textured region onto a square (using the closest vertex on the contour for instance) and then map again from the square to the generated region. However, if the shape difference between the regions is too important, the resulting texture would be deformed, resulting in distorted features.

Approach chosen We have as input a photograph and the corresponding geometrical structure indicating the approximate placement of the regions in this image. We have to texture a synthesized structure, which can be of any size, with the color information coming from the input photograph. In order to meet our speed and space requirements, we cannot realistically use texture transfer or texture synthesis at rendering time. Ideally, to achieve a fast texture mapping pass during rendering, we should already have texture coordinates on our mesh vertices indexing in a small number of texture images loaded in the graphics card and not handle any additional image operations. Considering this, we decided to synthesize texture coordinates in a preprocessing step. These coordinates could index in the original image but this would pose the problems described earlier regarding deformations due to re-parametrization, or computation incurred by the verification of region sizes.

We thus chose to collect and put together the original texture information in order to build a minimal set of *unified textures* into which we can index randomly in order to fill each generated region with color information. Intuitively, we can think of this as trying to recover the raw material that was originally used to construct the structure regions, e.g., the rock in the quarry used to carve out the wall stones. Figure refimg:RegionAppearance$_{A}nalysisPhaseillustratesthisprocess.$

The main advantages of this kind of approach are the following:

- All the costly computation can be done as a preprocess.
- We create a small set of relatively small textures per analyzed structure, thus limiting memory requirements.
- Using solely a small texture set, only texture coordinates for each of the generated structure vertices have to be computed.
- Since they represent the underlying material (e.g., the original stone used to create the wall, etc.), these textures are potentially reusable in other contexts.

We must however try not to alter or loose precious detail information while building these textures. In the following subsections, we describe how we build this unified sharable texture information and use it to rapidly texture generated structures of any size.

4.1.2 Region data collection

We want to construct unified common textures using texture information coming from the original image. To this end, we divide the set of original colored regions into clusters of similarly-tinted regions which are used to create the desired unified textures using texture synthesis techniques. In order to achieve this goal, we go through the main steps pictured in figure 4.4.

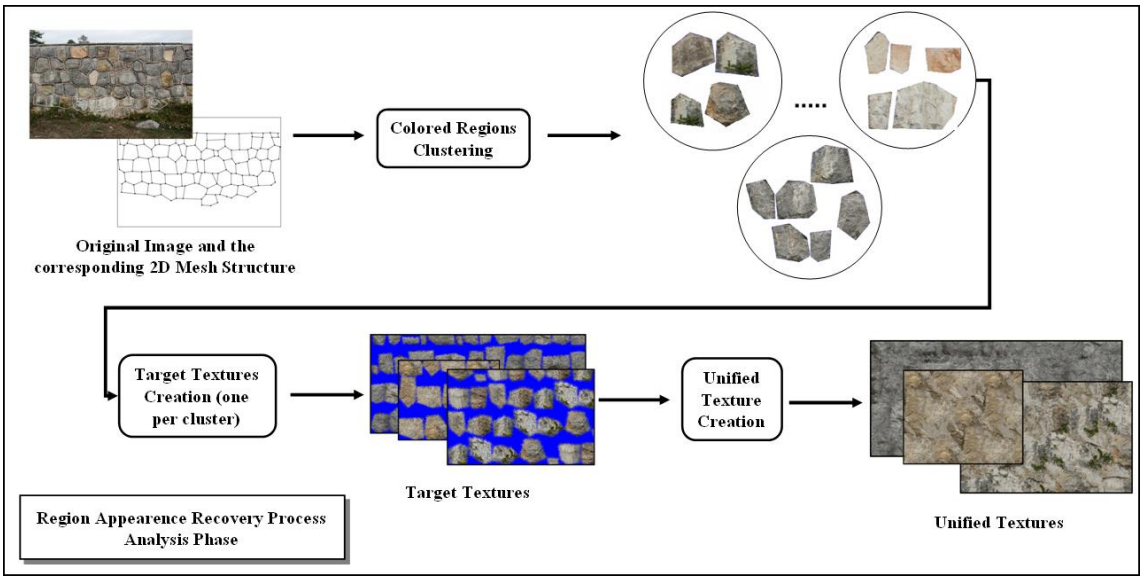

Figure 4.4 Analysis of the regions appearance.

Original texture extraction Using the structure drawn by the user on the original image, we extract a set of *textured regions* from this image. This is done by taking each original polygon in turn, shrinking it slightly to make sure it only covers the material we want to extract and not the interstice, and scan-filling it to gather all its pixels. Figures 4.5 and 4.6 left column shows the result of extracting all the original colored regions from the Levens wall, the Valbonne wall and the SnakeSkin sample images.

Clustering To build a small set of *clusters* of *similarly* colored regions, we compute a *color metric* to be used in the clustering process for each extracted colored region. Various color metrics can be used. We chose a metric that categorizes a region according to its perceptual tint in order to unite together colors that are perceived alike.

For this, we convert the original image from the RGB color space to the non-linear three-dimensional CIELAB colorspace. For this, we have to first convert to the XYZ colorspace and then to the CIELAB space [WS82]. We chose this color space since it is considered approximatively perceptually uniform, meaning that the Euclidean distance between two colors in this space corresponds to their perceptual difference. We thus define the color metric for a region to be the average of all the three-dimensional points in the 3D CIELab color space which corresponds to the region pixel colors. Figure 4.7 shows an individual region (left) for which a color metric in the CIELAB color space (right) is computed.

In order to cluster the regions using their respective computed metrics, we chose to use an adaptive k-means clustering algorithm [KP99]. This method determines the optimal number of

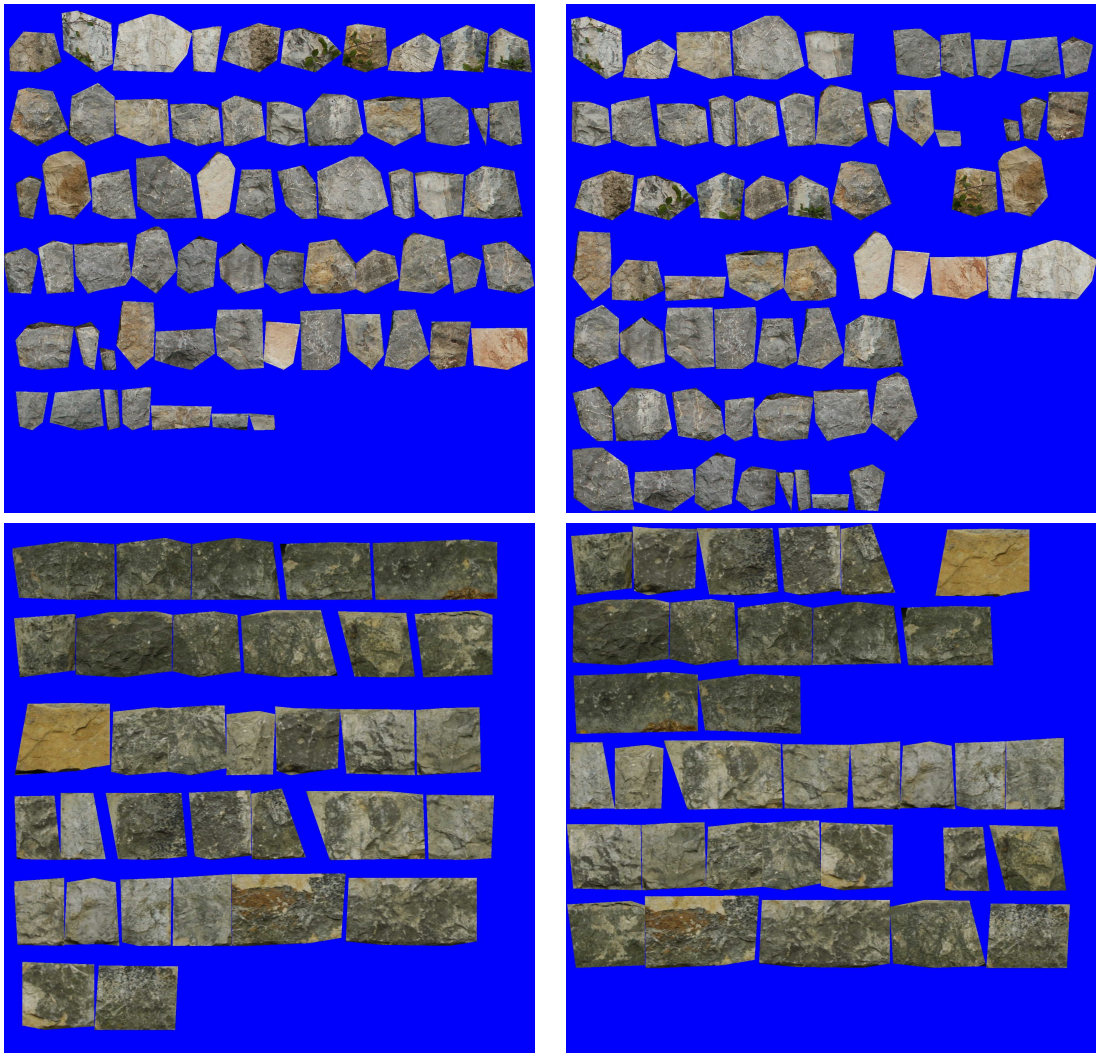

Figure 4.5 Left column: Original extracted colored regions from (top to bottom) the Levens wall and the Valbonne wall examples. Right column: The same regions, perceptually clustered according to their colors.

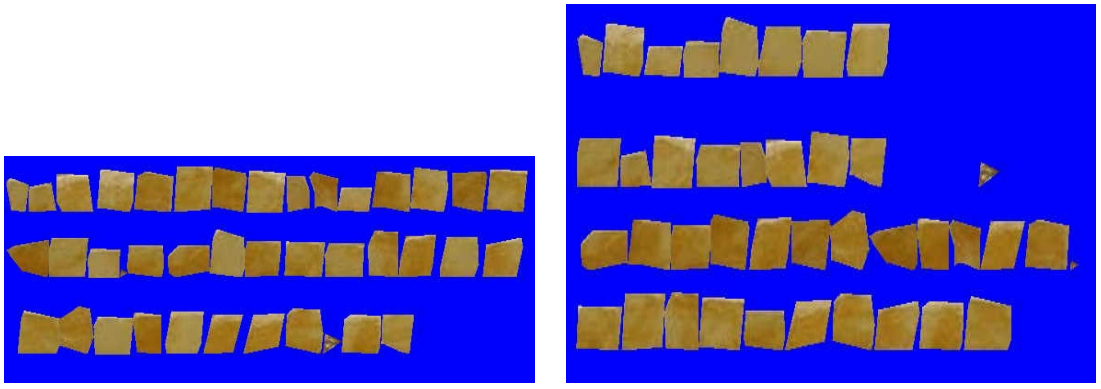

Figure 4.6 Left column: Original extracted colored regions from the SnakeSkin example. Right column: The same regions, perceptually clustered according to their colors.

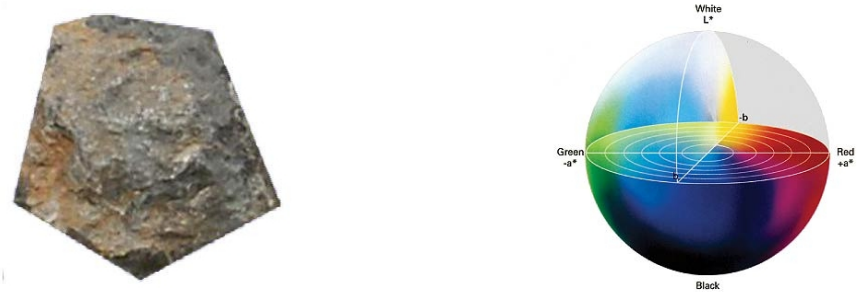

Figure 4.7 A color metric in the CIELAB color space (right) is computed for each individual region (left).

clusters for a certain data set and groups together the "closest" colors. Using the CIELAB colorspace in which Euclidean distances are meaningful allows the use of such a clustering technique and obtain clusters of perceptually similar colors. Typically, in our examples, we found less than 10 clusters. Figure 4.5, right column, shows the clusters of regions obtained for the Levens and Valbonne wall and the SnakeSkin examples. We can see that the groups formed effectively contain regions of similar tints.

Preservation of material detail In order to avoid damaging small material/texture features present in the original colored regions (e.g., small rock veins), we build a set of *guiding textures* to be used by the texture synthesis algorithm employed (recall the description of constrained texture synthesis in section 2.2.2). For this, we populate an empty image with the - possibly repeated - images of the original colored regions pertaining to the same cluster. These regions are "aerographed"[1] on their contour to avoid detecting their shape in the final resulting image output by the texture synthesis algorithm. This is an automatic Photoshop script which does not add any manual overhead. We end up with one guiding texture per colored region cluster. Figure 4.8 shows one of the guiding images built with the regions contained in one of the Levens colored region clusters and a guiding image from the SnakeSkin example.

Synthesis of unified texture One final unified texture is created per region cluster. It is constructed by running a texture synthesis algorithm guided by the guiding textures created in the previous stage. In our case, we use the Efros-Freeman quilting method [EF01], described in more detail in section 5.3. Small texture blocks are extracted from each colored region in each cluster to compose the "block pallet" which the synthesis algorithm uses to create the result image by placing one little block after the other, each block being chosen according to some measure, as described in more details in section 5.3. The guiding texture is employed to influence the choice of the blocks in the parts where there is some color information (i.e., other than blue background pixels). This guiding texture is necessary as most texture synthesis algorithms may alter the small features of original regions when the choice of blocks is "blind".

Figure 4.9 shows two of the result textures for the Levens wall examples. The first one (a) has been created with the aid of the guiding texture seen in figure 4.8. The second one (b) exhibits some interesting features such as leaves, allowing them to be reproduced in the generated outputs. Image (c) is the unified texture resulting from the the SnakeSkin guiding texture of figure 4.8, bottom.

[1] Photoshop filter simulating the painting technique known as "aerograph" which consists in spraying finely pulverized paint onto a surface.

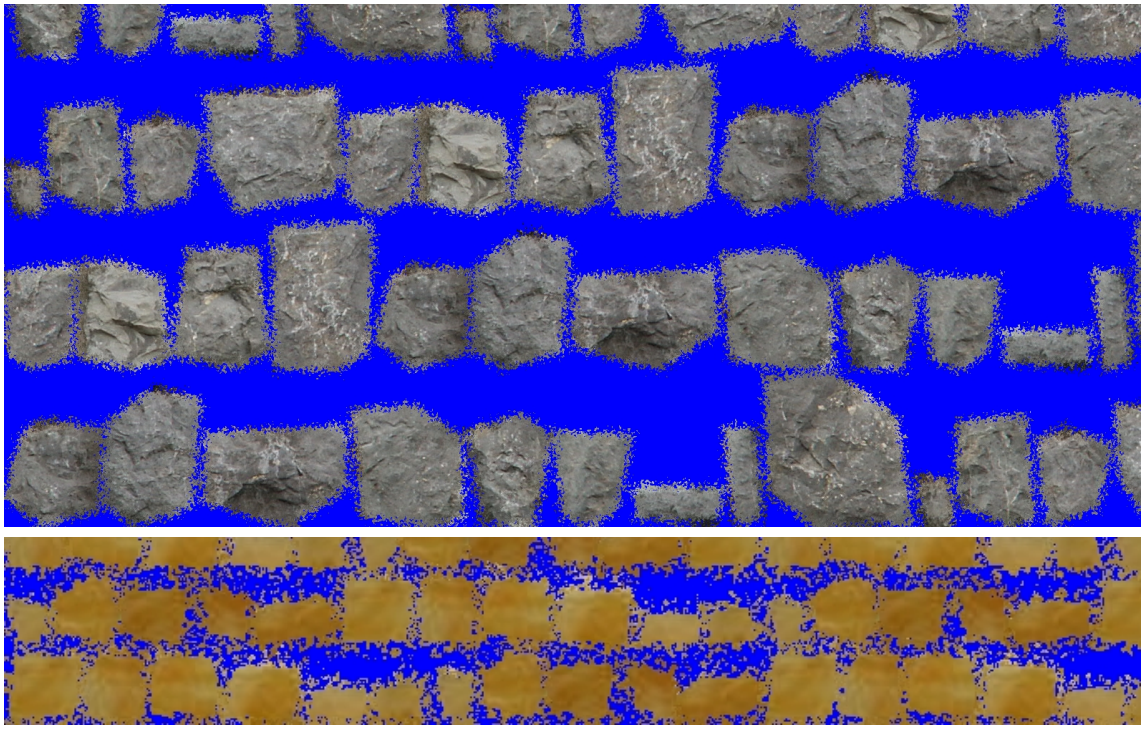

Figure 4.8 Two examples of guiding textures containing multiple samples of the regions composing a cluster. Top: Levens wall example. Bottom: SnakeSkin example.

We end up with a small set of texture images representing the material constituting the original regions, one for each cluster of regions. These images can then be used to texture the generated regions as described in the next section. One important condition to respect, though, is that the interior of each extracted region must be composed of an approximatively uniform material (without any noticeable macrostructure). If this is not the case, the resulting unified texture might be of poor quality, depending on the texture synthesis algorithm employed to create it.

4.1.3 Region data Synthesis

At the end of the structure synthesis process, we obtain a 2D mesh of polygonal regions. To texture each individual generated region, we find its closest original region (as described in section 3.5.4) and its corresponding cluster. We retrieve the unified texture associated to this cluster and project the generated polygon at a random position onto the texture. We scale the generated polygon appropriately to avoid stretching or compressing the texture which would lead to resolution artefacts in our results. To do this, we must ensure that the number of pixels in the generated region is the same as the number of pixels covered by the projection of the generated polygon. Figure 4.10 shows the projection of a generated polygon into a unified texture. If we normalize the projected coordinates of the generated polygon to be between 0 and 1, we directly obtain texture coordinates that we store along with the generated mesh. At rendering time, these coordinates (UVs) are used to index directly into the unified texture which is loaded into the graphics card, thus achieving real-time rendering of the generated structured textures. Some final renderings can be seen in the results section (4.3).

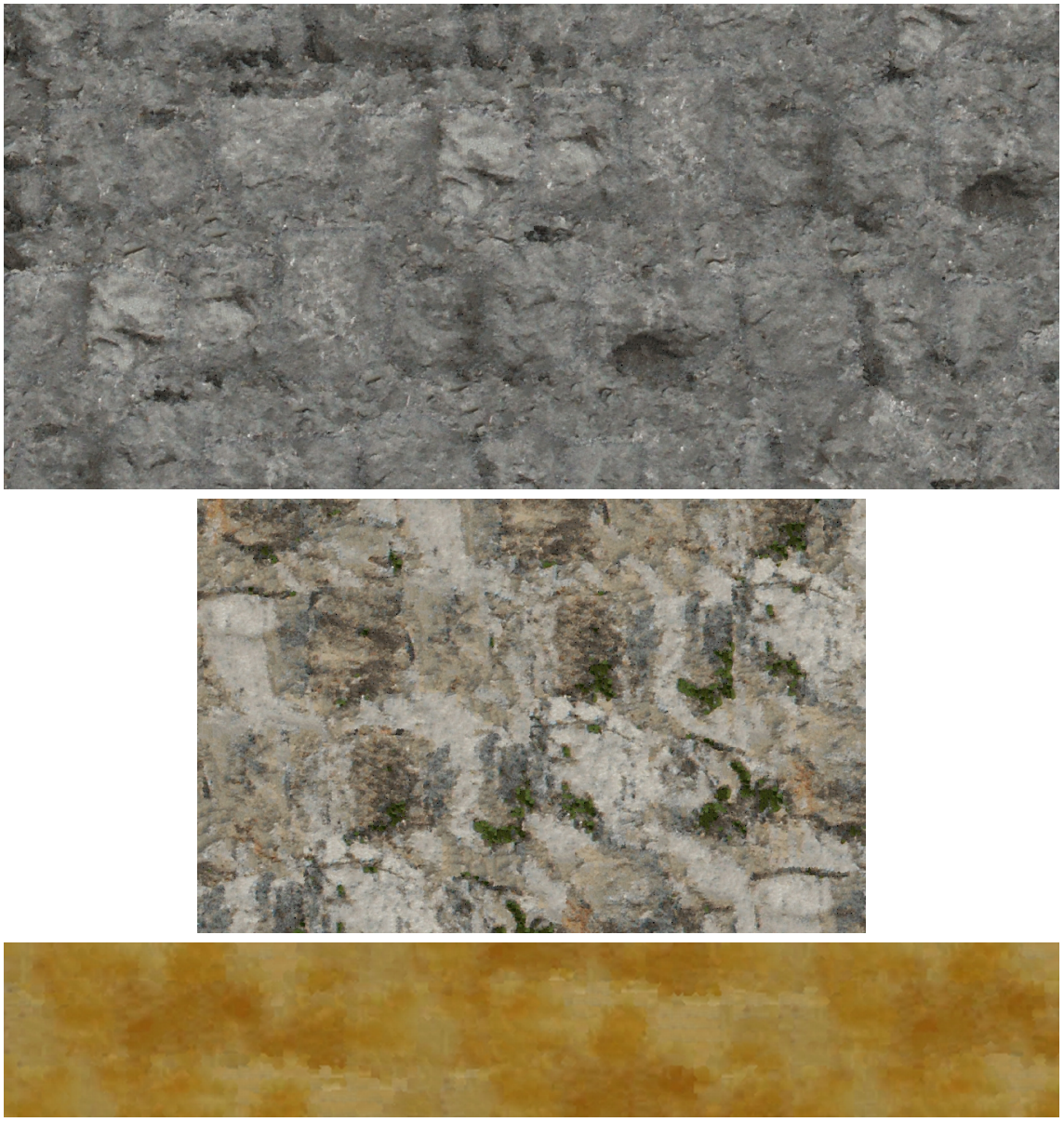

Figure 4.9 (rom top to bottom: two of the Levens wall unified textures. Note the leaves present in the second image. One of the SnakeSkin unified textures.

4.2 Generating contour details

The process described in the previous section allows the synthesis of some detail for the interior of the generated regions. However, we still have regions whose shape is a straight-edged polygon whereas our original regions usually exhibit a much more geometrically complex contour (see figure 4.11 for an example of texturing only the region interiors). We could try to find a method to reproduce this complexity and add the geometric contour edges required to achieve it. However, this would add a lot of unnecessary geometry. Instead, we decided to simulate this complexity by using the *interstice* and *junction* textures (which together we call "separators") which separate two regions and whose own intricate contours are determined by the complexity of their adjacent regions. These separators can be empty space, mortar, roads, skin,... depending on the structured texture analyzed.

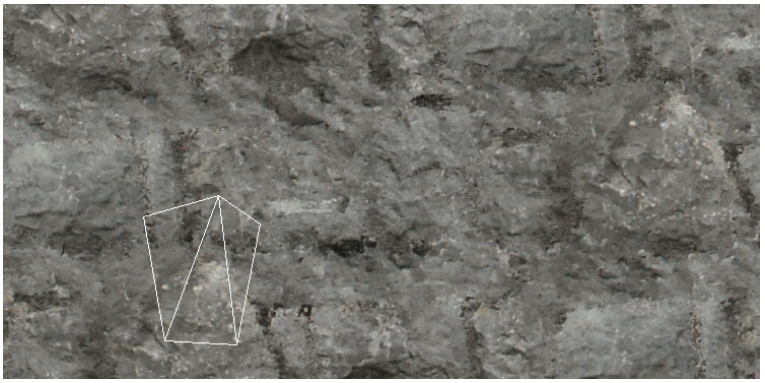

Figure 4.10 A generated polygon projected into one of the unified textures

Figure 4.11 Result after texturing the interior of the regions of the Levens wall original structure.

In this section we present the method we developed in order to extract these separators from the original image and use them to enrich the generated regions contour. A region contour geometry is constituted of vertices and edges, corresponding to junctions and interstices in their textured counterpart. Each edge separates at most two regions whereas a vertex is the intersection of several edges, and thus regions. Due to this difference, we will decouple the texturing of these two elements in a first pass, in order to combine them in the final rendering.

4.2.1 Texturing interstices

We want to recover information about the interstices present in the original image in order to synthesize similar interstices. We considered various ways to address this problem, which are briefly discussed later in the section. As for the region interiors, we want to do as much preprocessing as possible beforehand to speed-up the rendering. The main idea of the approach we chose is to build a texture atlas with the original interstices and use it to texture the generated edges.

Interstice Data collection

We extract texture bands containing only interstice pixels from the original image. Segmenting the original image to retrieve these pixels automatically is difficult. There is often too much noise, too many similar colors and the image analysis algorithms have to be fine-tuned to each particular example, incurring too much manual tuning in practice. In our case, we have an important clue on the approximate location of these interstices as we have a hand-drawn 2D mesh on the original image. Automating this interstice extraction could also benefit from this information, as we briefly see at the end of this section.

Here are the main steps of our method to extract the interstice pixels from the original image:

1. Collect all the edges from the 2D mesh drawn by the user.

2. Create a 2D polygon by "expanding" each edge in the original image space as shown in figure 4.12, left. This polygon is slightly longer than the edge.

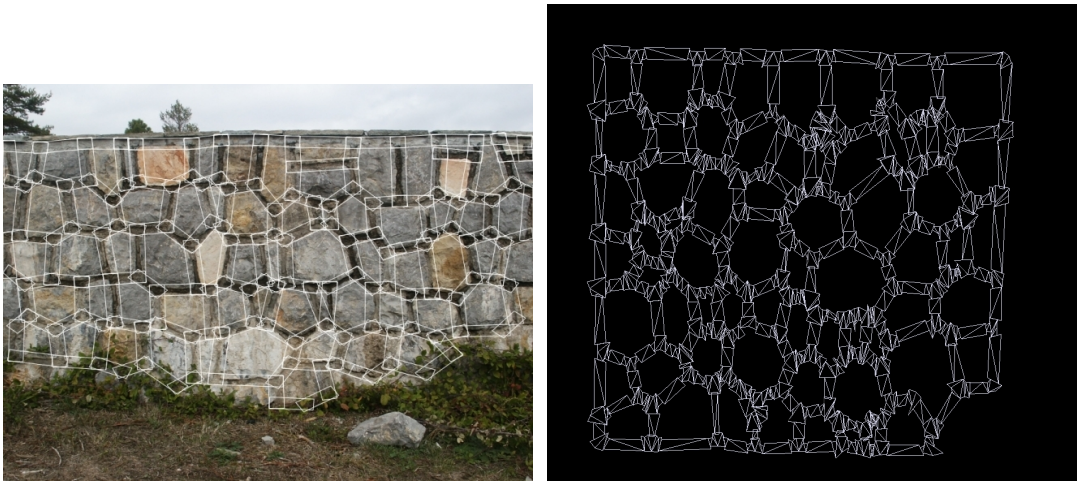

Figure 4.12 Left: 2D polygons created around the edges of the original image to extract the interstice textures. Right: 2D polygons created around every edge and vertex of the generated subdivision; they will be textured with texture extracted from the original image.

3. Scan-fill this polygon to copy the original image pixels.

4. Paste these pixels into a result image (the same for all interstices) which constitutes our interstice atlas. See figure 4.13, top, for an example of a section of interstice atlas.

5. Associate the location of the interstice texture extract in the atlas with the original edge length and store this information.

6. When all edges have been processed, remove any region interior pixels from the interstice images and replace them by a transparent alpha value. For now, this stage is done by hand using Photoshop but we discuss later how this process can be automated. Figure 4.13, bottom, shows the interstice atlas after editing.

In the end, we obtain one texture containing all the interstices from the original images, and a data structure to quickly reference them according to the corresponding edge length. We can use this atlas to synthesize new interstices for generated structures.

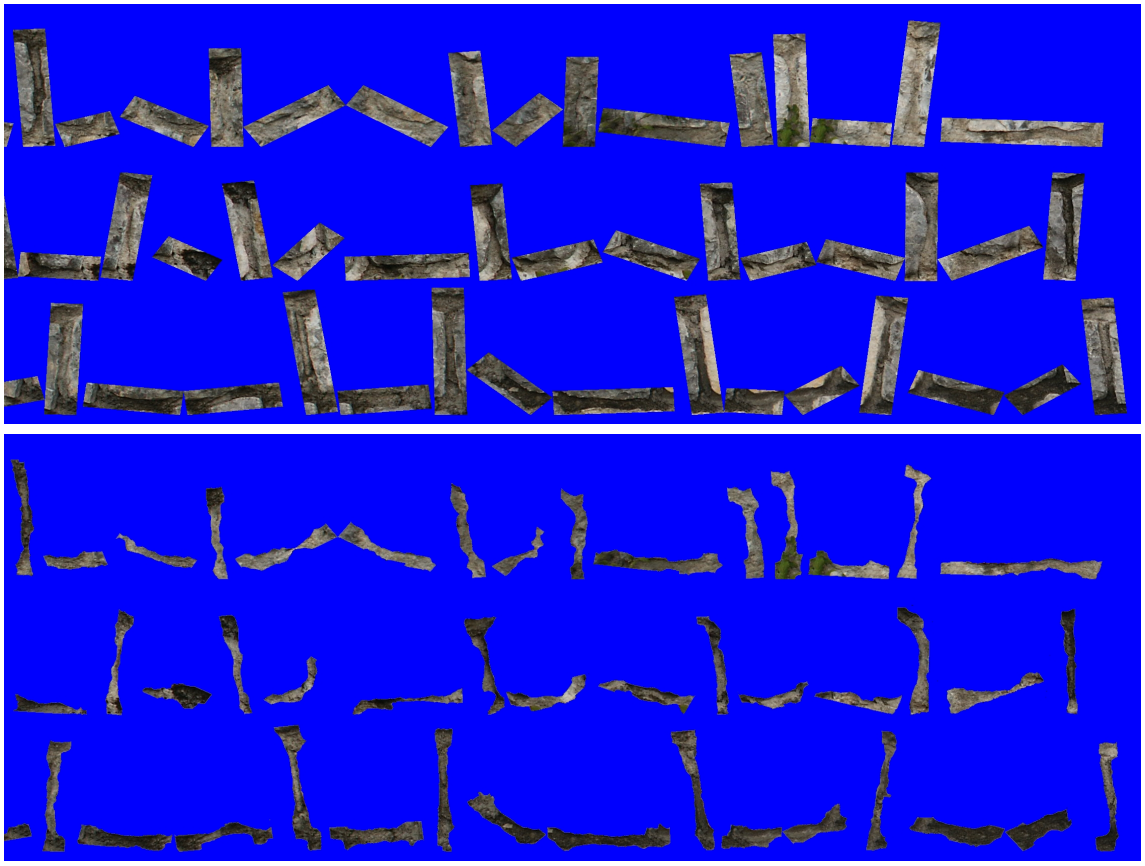

Figure 4.13 Top: The interstices extracted around all the original edges. Bottom: Same interstices, edited by hand to remove the rock pixels, replaced by transparent alpha values.

Interstice Data synthesis

A generated structured mesh is composed of a series of straight edges of various lengths. We will "garnish" them with the interstice textures coming from a previously created interstice atlas. This atlas does not necessarily have to originate from the same structured texture as the generated 2D mesh.

We use the following technique: For each of the generated edges, we find the original edge having the closest length. This original edge is linked to an interstice texture in the interstice atlas, which we associate to the generated edge. This association is performed by creating an additional 2D polygon around the generated edge (as can be seen in the right image of figure 4.12), and assigning to each of its vertices the textures coordinates corresponding to the chosen interstice in the atlas. Figure 4.14, left, illustrates this association process. When the length of the generated edge is different than its closest original length - which is often the case - the mapped texture will be either stretched or compressed. However, this does not cause important visual artefacts as the interstices are often small compared to the rest of the structured texture. Moreover, for structures generated out of the same image from which the interstices have been extracted, the edge lengths are similar.

Once again this approach has the advantage of allowing to texture a generated structure of any size using only one texture, which can thus totally fit in memory. Only textures coordinates have to be computed and this is a very fast process.

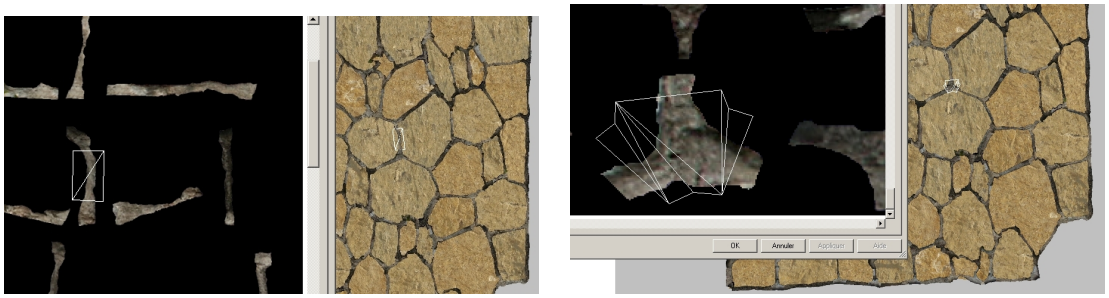

Figure 4.14 Left: Original interstice chosen to texture the indicated generated edge. Right: Same but for a junction.

Figure 4.15 left shows an image of a generated structure where the interiors have been textured using the method described in the preceding section and the edges have been textured using the approach we just described. The close-up on the right shows that there is an evident need for treating what happens at the meeting point of the interstice bands, which we address in the next subsection.

Figure 4.15 Left: Generated structure with interiors and interstices synthesized. Right: Close-up on a few regions, interstices are overlapping.

4.2.2 Texturing junctions

A junction is defined as the intersection of several interstices. This corresponds to the mesh vertices in the 2D structure. To extract information on these junctions from the original image, we use a similar technique as for the interstices: we create an atlas of junctions which is then directly referenced during rendering. The main difference lies in the way the generated junctions are selected during synthesis.

Junction Data collection

As for the two previous steps, we have as input the original image and the 2D mesh structure which the user has drawn. The junctions are extracted in the following steps:

1. Create a 2D polygon centered at each of the mesh vertices. This polygon is star-shaped and has one branch per edge joining at this vertex. See image 4.16 for an illustration of the junction extration process.

Figure 4.16 Junction polygons created on the original image to extract junction textures.

2. Scan-fill this polygon to copy the original image pixels.

3. Paste these pixels into a result image (the same for all junctions). See figure 4.17 for an example of a section of a junction atlas.

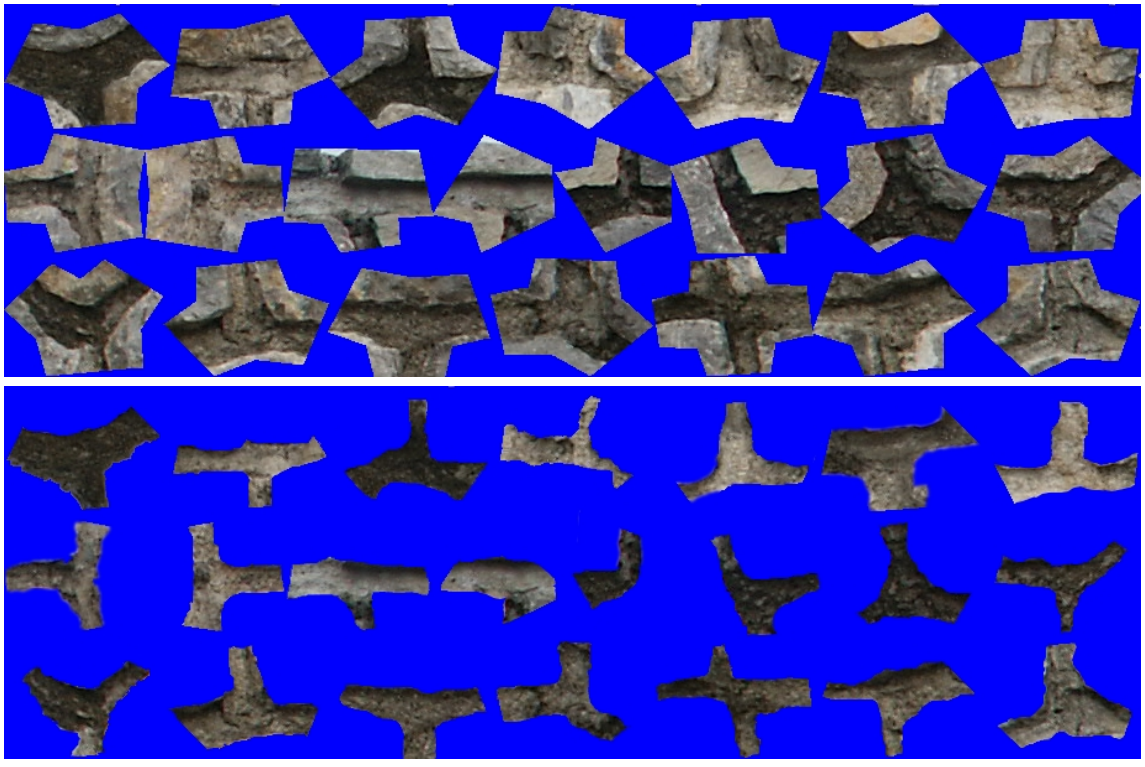

Figure 4.17 Top: The junctions extracted around all the original vertices. Bottom: The same junctions, edited to remove the rock pixels, replaced by transparent alpha values.

4. Associate the location of the texture extract in the result image with some information which allows us to characterize a given junction. We chose this information to be the distribution of angles around each vertex. We thus construct one search tree per valency present in the structure. For each vertex with valency v, we insert in the tree corresponding to v, a

multidimensional data point composed of the ordered list of angles around it, along with the location of the corresponding junction polygon in the junction texture. To make sure we cover all the possible combinations of angles possible, we also insert rotated and symmetrized versions of each initial data point. For example, if we collect for one vertex the initial set of angles {175^o, 85^o, 100^o}, we will also insert the following sets: {100^o,85^o,175^o}, {85^o,100^o,175^o}, {175^o,100^o,85^o}, {100^o,175^o,85^o} and {85^o,175^o,100^o}. We store the angle data in a data structure that is designed for fast search: approximate-nearest neighbor search (ANN) [AMN+98].

5. Edit the result image to remove the rock pixels in the same way as for the interstices, leaving only the junction pixels. Replace them with a transparent alpha value. See figure 4.17, bottom, for an image of some extracted edited junctions.

In the end, we have one texture where all the junctions in the original image have been placed side-by-side, along with a search structure which we will exploit during the junction data synthesis.

Junction Data synthesis

Our generated 2D mesh structure is composed of vertices around which additional junction textured polygons will be placed (see figure 4.12, right, for an image of some of the created for the Levens wall generated structure.). Each generated vertex has valency v' corresponding to the number of generated edges around it. To find the most appropriate junction to attach to this vertex, we collect the angles around it, creating a v'-dimensional data point that we use to query the corresponding tree for its nearest neighbor. The metric used for this standard tree-ordered search is Euclidean distance. This search leads us to the closest junction in terms of its angle distribution. The point found can be a rotated/symmetrized version of an original angle set. We thus need to mirror and/or rotate appropriately the corresponding junction texture in order to align it correctly on the generated junction polygon. Figure 4.14, right, illustrates this association process.

In order to also consider the similarity of the junction color with the colors of its surrounding interstices, we use the following heuristic:

1. Instead of returning the closest data point, we return the k closest ones (where $k < 10$ in practice). ANN [AMN+98] allows us to efficiently return the k closest neighbors.

2. Compute a color metric for each junction. We use the same color metric in CIELAB space as for the colored region clustering process described in section 4.1.2.

3. Compute a color metric for each surrounding interstice. Compute the mean of these color metrics.

4. Keep the junction whose color metric is closest to this mean.

More advanced heuristics such as precisely matching the colors of the corresponding texture endings could be designed. We could also consider touching up the original junction colors (or only the luminance) in order to get a better color fit.

The search for the most appropriate junction is done for every vertex in the generated 2D mesh. The selected junction polygon coordinates in the junction atlas are taken as texture coordinates for the associated generated junction polygon. An example rendering can be observed in figure 4.18. Looking at the close-up on the right, we can notice the visual artefacts where the junction and interstice polygons meet. The junction and interstice polygons

are not facing each other and there is thus a visible continuity break. We must now address this problem and devise a method to achieve a seamless blend between the different parts extracted.

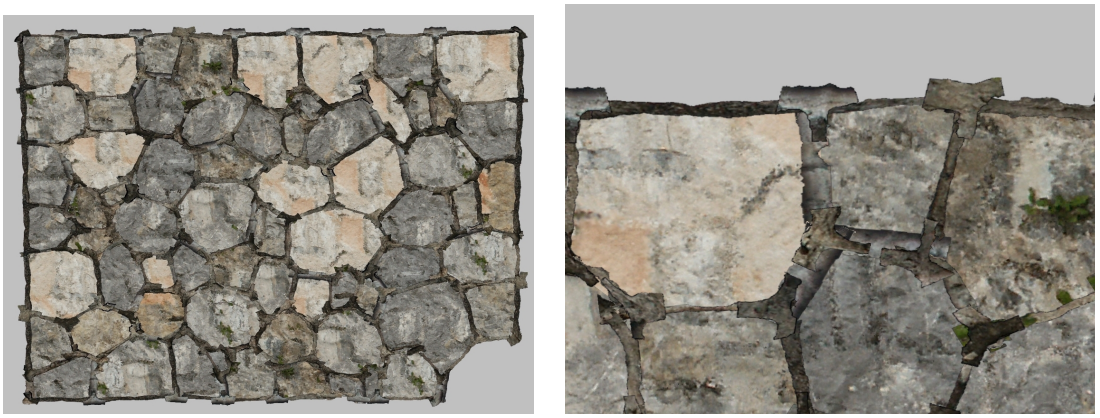

Figure 4.18 Left: Generated structure with synthetic interiors, interstices and junctions. Right: Close-up on a few regions, interstice and junction textures do not match..

4.2.3 Combining interstices and junctions

The region, interstice and junction synthesis steps result in the creation of an equivalent number of textured polygons which completely cover the target surface. The junction and interstice polygons are placed slightly in front of the region polygons. This has the effect of hiding the region straight edges, thereby providing a complex silhouette to the region, due to the contour (interstices and junctions) texture silhouette complexity. However, the junction and interstice polygons overlap as a result of their construction process. In addition, the textures they support often do not match properly at the polygon borders, as can be seen in image 4.20 (a). This is due to the fact that the textures are not necessarily centered on their supporting polygon and their width can be different.

To correct this problem and correctly match the texture endings of the junctions and the associated interstices by modifying the supporting polygon vertices position, we use the following method:

1. For each branch J_i of the junction polygon, find its intersection with the corresponding interstice polygon. This gives a 2D segment (object space) for each junction branch. Figure 4.20 (e) and (d) show a junction polygon and its interstices polygons.

2. Find the corresponding segments in the junction and interstice texture spaces. Traverse these segments to determine the extent of the texture colors. Figure 4.21 shows the resulting texture limits for a junction and its surrounding interstices, indicated by green and red lines.

3. Compute the scaling and translation values to apply to J_i endpoint vertices to align the texture endings of the junction branch and its facing interstice. This corresponds to match the green and red lines together. If the scaling is too important, transform the interstice polygon instead.

4. Apply these values and cut the polygons at their intersection to remove unwanted texture parts. Figure 4.20 (b) and (e) show the result after transforming the polygons.

5. Since only the branch endpoints have moved, some unwanted distortions can appear in the junction texture. To correct this problem, we compute the translation to apply to the intermediate vertices (highlighted in red in the images) in order to place them in the same position relative to their two adjacent vertices as they were before the matching process as illustrated in figure 4.19.

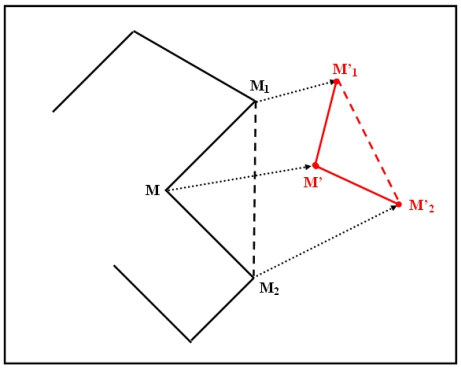

Figure 4.19 Translation of the intermediate points (between two junction branches) in order to reduce the texture stretching artefact. The junction branch endpoints M_1 and M_2 have moved to position M'_1 and M'_2. Junction vertex M must keep the same position relative to its adjacent vertices and is thus moved to position M'.

The final result is shown in figure 4.20 (c) and (g).

We apply this process to correct all the generated junctions. A comparison of a generated structure before and after the match can be observed in figure 4.22. The structure has been generated (after the repartition stage), and the appearance is completely generated from the information extracted from the original photograph (shown below). Figure 4.23 shows the interstices generated using the original structure of the Levens wall.

4.2.4 Automation of interstice and junction extraction

It should be possible to automatically extract the pixels composing the interstice and the junctions, i.e. to automatically separate the contour pixels from the region interior pixels. Three additional sources of information could be used for this purpose:

1. The image color histogram could serve to get the range of colors of the interstice in the case it is easily recognizable from the range of colors of the rocks. In figure 4.24, the histogram corresponding to the image (INRIA wall) on the left is shown on the right. It is composed of two peaks. Selecting one of them has the effect of selecting all the interstices pixels.

2. The gradient image of the original image, telling us where the highest contrast changes are situated. This could be a clue to where the regions borders are.

3. A user-given patch (similar to the ones used in chapter 5) discriminating the interstice colors from the rock colors.

To discriminate the interstice pixels, we could start from the drawn $2D$ edge and collect the pixels around it that could potentially be interstice pixels. To accept or not a pixel as an interstice pixel, we could attribute a score to each pixel, computed from the available information sources (is its color inside the interstice color range? Is the contrast change very high, in which case we might have attained the border and we should penalize the pixel score?) and accept the pixel if its score is above a given threshold.

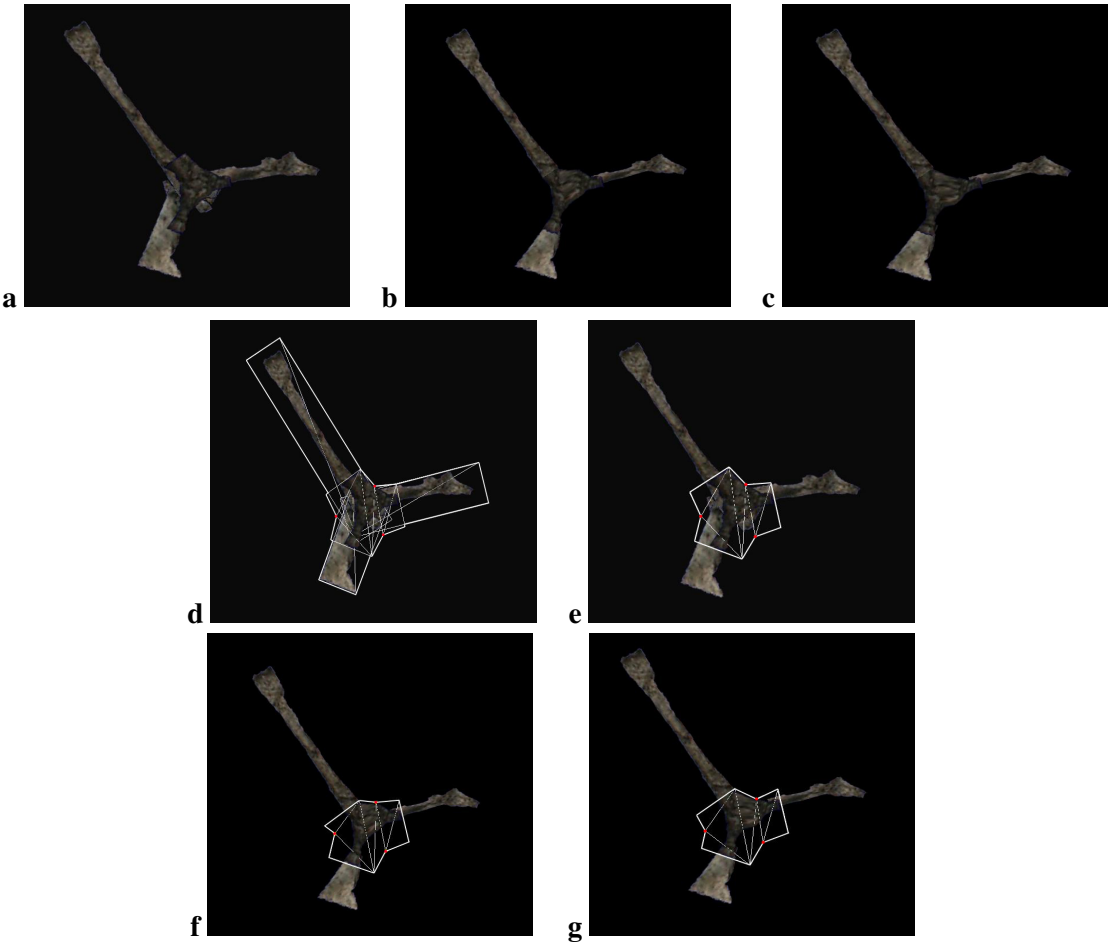

Figure 4.20 Modification of the junction vertices to match the facing interstice pixels. (a) Junction and Interstices before the matching phase. (b) After the matching phase with no correction of the intermediate points. (c) Correction of the intermediate points position. (d) The junction and interstices of underlying polygons. The junction polygon, (e) before the matching, (f) before the correction of the intermediate points and (g) at the end of the process. Notice the reduction of the stretching artefact.

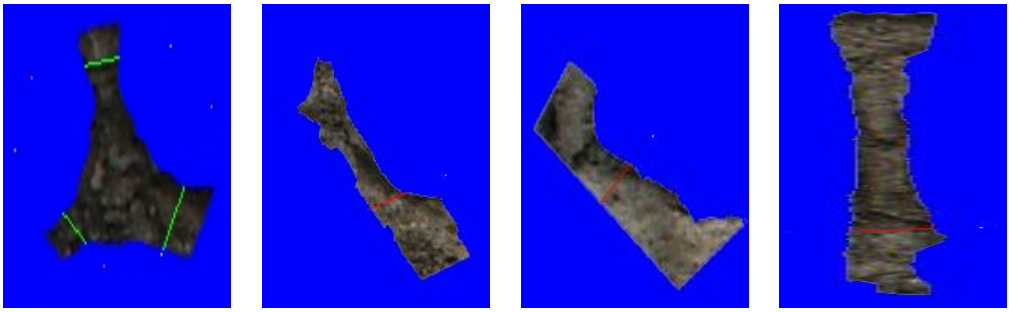

Figure 4.21 A junction and the three interstices connected around it (texture space). The intersection of the supporting polygons and their corresponding texture elements are indicated by green and red lines.

Figure 4.22 Left: A generated structure before the junction correction process. Right: The same structure after the correction process.

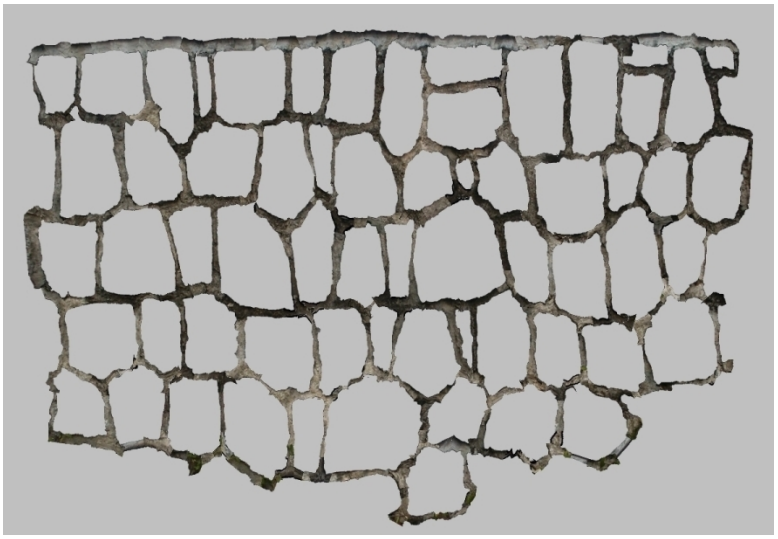

Figure 4.23 Result after texturing the contours of the Levens wall original structure.

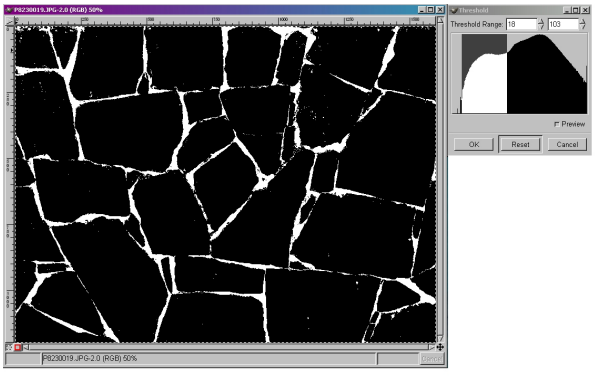

Figure 4.24 The INRIA wall histogram with the interstice color peak selected.

4.3 Results

We present in this section some of the results obtained. We can directly transfer the polygons created in 3D. Texture coordinates have been assigned to all of their vertices and index directly in the small set of textures loaded (unified textures, interstice and junction atlas).

Structure and appearance data from the same source

Levens wall Image 4.25 shows a synthetic view of the Levens wall, obtained by overlaying the contour texturing results (image 4.23) and the region texturing results (image 4.11). The structure is the original one, whereas the appearance is completely generated from the information extracted from the original photograph (shown below). The generated polygons index in the unified texture associated to the closest original shape. In this case, we use the original structure and we can observe that we effectively recover the original colored region tints.
Figure 4.26 shows a generated version of the Levens wall. The structure is the result of the shape optimization stage and the appearance is synthetic.

INRIA wall Figure 4.27 shows two of the unified textures created out of the INRIA wall structured texture (which can be seen in image 4.28, left). We effectively get the impression of a uniform rock material, where the small important original details have been preserved.
Image 4.28, right, shows the recovery of the appearance of the INRIA wall.

Snake Skin One of the Snake skin unified textures is displayed in figure 4.29 and the result of the appearance recovery is shown in image 4.30. The poor quality of the textures is due to the small dimensions (400×270) of the original input photograph.

Modifying the underlying structure

Image 4.31 shows the result of the appearance synthesis before and after the insertion of the ellipse from figure 3.29. We can observe that the textures changes are local, only the regions around the modified area are affected[2].

Modifying the appearance

Since we have a series of unified texture, we can perform some changes on the textures and these modifications will be automatically reflected in the textured models. In image 4.32, we show on the top row two unified textures from the INRIA wall which have been modified to add some moss and crack information using Photoshop. Two examples results are shown on the bottom row.

Mixing structure and appearance data

Image 4.33 shows the result of mixing texture sources. The region interiors are textured with the INRIA wall while the interstices and junctions come from the Levens Wall example.
Another example can be observed in image 4.34 where the Valbonne wall structure has been textured with the appearance from the Levens wall. Some wrong color matches on the region contours show the need to improve our contour synthesis method. Possible options could be to give more weight to the color difference evaluation during the interstice/junction choice or even to modify

[2]Since the shape of the regions changed, their corresponding original region might have changed as well and this explains the change in appearance. This can be corrected by removing the constraint to take the same appearance as to closest original region.

Figure 4.25 Top: A final result using the original structure of the Levens wall with a completely synthetic appearance. Bottom: The original photograph from which the textures were extracted for comparison purposes.

Figure 4.26 Generated structure and appearance for the Levens wall.

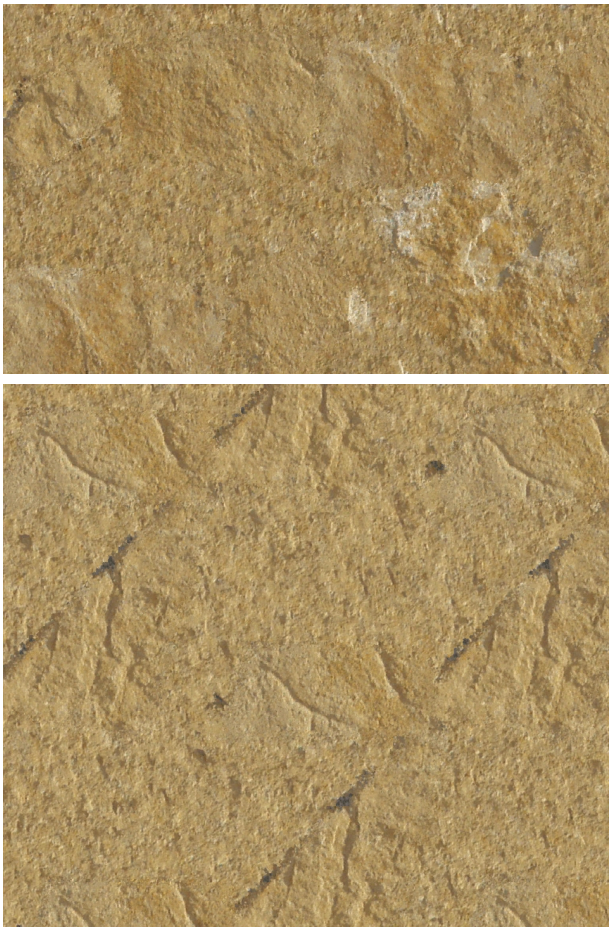

Figure 4.27 Two unified textures built with the regions extracted from the INRIA wall example.

Figure 4.28 Appearance recovered for the INRIA wall example. Left: Original structured texture. Right: Synthetic appearance of the original structure.

Figure 4.29 A unified texture extracted from the Snake skin photograph.

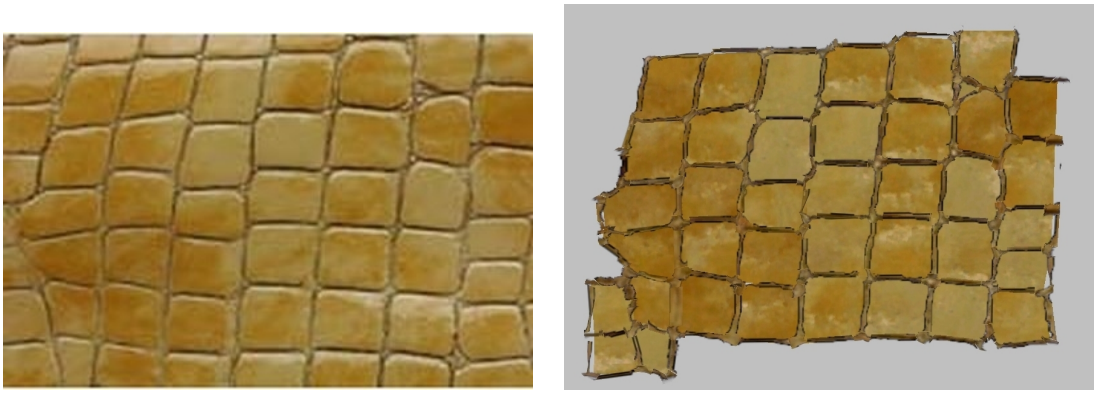

Figure 4.30 Appearance recovered (right) from the original image (left) of a snake skin.

slightly the colors to blend in more seamlessly between interstices and junctions. For some cases, the lighting can be imprinted into the texture and thus a correlation with the interstice/junction orientation should be added during the synthesis process to take this into account.

Applying to a different topology

Figure 4.35 shows two results of using structure and appearance data to texture a target shape of a different topology. The problems which may be seen on the region contours are mainly due to the fact that a lot of small edges have been generated by the polygonization used to transform the Voronoi regions into a connected mesh, and these edges cause problems to our matching algorithm. This is a current shortcoming of our method and its improvement is part of the future work planned (see section 6.2).

Figure 4.31 Appearance synthesis before and after the insertion of a region.

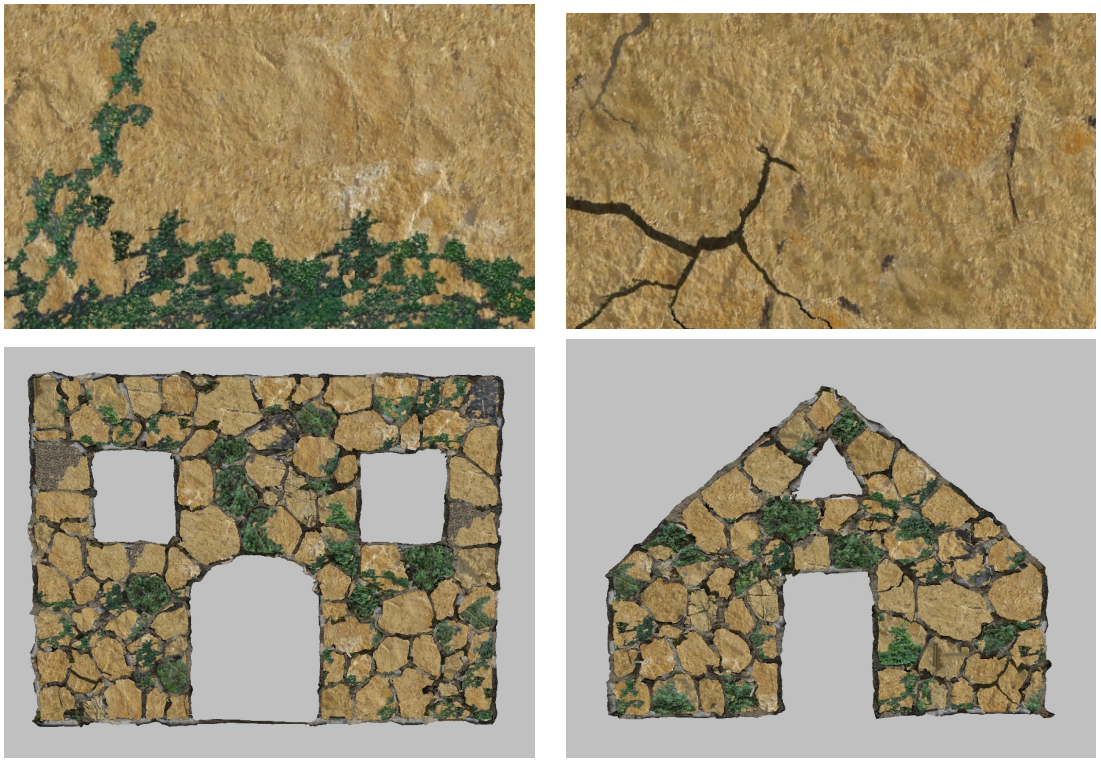

Figure 4.32 Modification of appearance analysis data (top) is reflected in the textured structures (bottom).

Figure 4.33 The Levens wall structure has been textured with the region appearance data of the INRIA wall.

Figure 4.34 The Valbonne wall structure is textured with the appearance of the Levens wall.

Figure 4.35 Two views of constrained structure and appearance synthesis.

Some more examples can be observed in figure 4.36, along with their corresponding constraint maps. On the first row, the house facade structure has been synthesized from the Levens wall structure, the left one using the appearance information from the Levens wall while the right one uses the appearance information from the INRIA wall example. On the second row, the castle facade structure and appearance comes from the Levens wall example, while the house facade structure and appearance comes from the INRIA wall example.

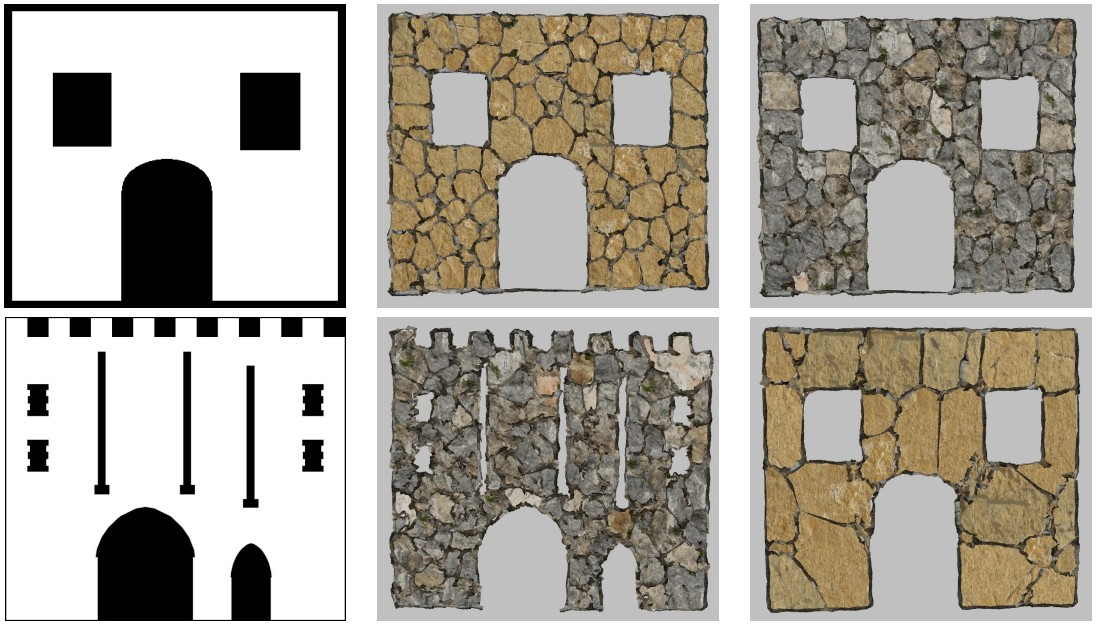

Figure 4.36 Results of constrained structure and appearance synthesis, with the constraint maps shown in the left column. On the last line,

Building a village

Due to time limits, the results concerning the texturing of a village using a limited set of unified textures and other results could not be published in time. They will be posted on a web site related to this thesis: http://www-sop.inria.fr/reves/Marie-Claude.Frasson/ThesisResults/.

4.4 Conclusion

One of the goals of this thesis is to synthesize structured textures from the analysis of one or more samples. To this aim, we chose to decouple the structure generation from the appearance generation. In the preceding chapter, we presented our approach to generate geometric structures (subdivisions) from the analysis of an input subdivision. At the end of the structure synthesis process, we obtain 2D cellular meshes composed of connected regions. In this chapter, we exposed our approach to texture these meshes with the detail information found in the original images. This 2D texture information allows the simulation of the geometric complexity which we cannot afford to create.

Our approach had to fulfill the following requirements (described in the introduction of this chapter):

1. Real-time rendering speed.
2. Low texture memory usage.
3. Exploitation of detail information contained in the original image.
4. Ability for a user to influence or edit the results.
5. Potential to reuse extracted information.

In order to texture our generated meshes, we decided to decouple the region detail information from that of its contour. Often, in real-world structured textures, these two elements are composed of two very different materials. They **are** two distinct objects, it thus makes sense to analyze and synthesize them separately and this is exactly what we did.

To synthesize the appearance of the region interiors, we construct a small set of unified textures. Each texture is built out of a number of original regions, which have been perceptually grouped together according to their color in the LAB space, using a 3D clustering technique. These textures are randomly sampled to texture the generated region polygons (created for that purpose) according to, but not necessarily, their shape similarity to the regions composing each unified texture.

The generated regions are straight-edges polygons, filled with texture information. The geometric complexity of their contours is simulated by the recovery of the contour detail information, namely the texture parts around the interstices between two regions and around the junctions at the intersection of several regions. These two kinds of texture elements are extracted from the original image using the original mesh information. They are united in texture atlases along with some information (edge length, angle distribution) which is used to associate a texture element to each of the generated edges and vertices. One polygon is created for every generated edge and vertex and textures coordinates indexing into the atlases are assigned to its vertices according to the association established. Junctions and interstices are matched at their borders using only vertex movements and they can be rendered on top of the generated textured regions using alpha values.

Our approach complies with the requirements enumerated above:

Rendering speed Our rendering can be totally interactive as there is no synthesis of texture information during rendering. All the computation is done off-line and rendering uses the generated texture coordinates which index directly in the small set of unified textures/atlases

created beforehand. These textures should totally fit in the graphics card memory, thus avoiding any texture swapping delays. The interstice and junctions are combined through polygon deformations whose computation is also done in real-time.

Memory usage Texture memory usage is limited. In the examples shown we only use less than ten unified textures in addition to two texture atlases (interstices and junctions). Their resolution depends on the original image resolution and their size is, of course, smaller than the original image as they are extracted from it. Only these textures are used, no additional texture information is created.

On the other side, one polygon is created for every region, vertex and edge of the generated mesh which adds a lot of polygons to our scene. We made this compromise in order to achieve the memory gains just described and knowing that the actual graphics card can render in real-time millions of polygons.

Exploitation of detail information We take advantage of most of the detail information present in the original image. The region interiors are used to recover material textures. The extraction of the separators and their silhouettes allow to simulate the complexity of the region contours at a low cost. No additional detail is created and the original texture details are preserved.

User control Since the user has access to the unified material texture and junctions/interstices atlases, he is free to modify this texture information at will, adding or removing details (for example, moss on a wall), modifying some contours, replacing some material textures by others,... These changes will immediately impact the resulting structured texture.

The decoupling of structure from appearance also enables the user to perform local changes on the underlying structure with little impact on the appearance synthesis: it has to be regenerated only locally. In the same manner, the decoupling of the region interior and contours allows the modification of one without affecting the other, for example, enlarging all the contours or attributing different rendering parameters. Annoying shadows or highlights can also be removed by hand or using the kind of techniques described in the next chapter.

In addition, the user can create various results out of the analysis of more samples. Material textures coming from one structured texture can be used with junction/interstice atlases originating from another.

The possibilities for experimentations are thus very interesting and constitute an additional strength of our approach.

Reusability The region and contour information are computed once for every structured texture analyzed. It can be used for as many structure generations as we want, not necessarily linked to the original structured texture they were extracted from. The material textures can even be used for other purposes as they constitute entire images of a real-world material.

We now have the possibility to generate endless versions of structured textures examples. However, they are still inherently flat which limits the amount of realism. In the next chapter, we address how we could recover additional 3D detail information to make our structured textures more believable.

Chapter 5

Generating 3D Details from Example

"Les détails font la perfection, et la perfection n'est pas un détail."

—Leonard de Vinci

The techniques we described up to now allow the synthesis of new textured structures at will and without any size restriction, by analyzing an example input photograph of a real-world structured texture. However, even though the textures which are used to synthesize their 2D appearance have been extracted from photographs of the 3D world, the resulting objects are inherently flat. They lack the richness of 3D details that can make them look realistic (figure 5.1). As an example, the rock material typically has a complex surface geometry that we completely lose. We define 3D detail as being any information which enhances the 3D impression given by a viewed scene. It can be geometry, silhouette, color, reflectance (highlights, shadows,etc.) or any additional information given to the scene models.

Figure 5.1 The Levens wall is not flat in real life!

5.1 Motivation

3D detail is a very important cue to realism. It gives the viewer extra information about the scene being viewed, and enhances realism. Unfortunately, as we have seen in section 2, detail modelling is tedious, detail recovery from real images is hard and detail rendering is long. Real-world details (in contrast to artistically created details) are the most realistic we can get. Artists try to use them by applying textures from real-world photographs onto their models, a process often referred to as "Image-based texturing" [POF98] [BMR01] [IBG03]. These photo-realistic textures contain the result of projecting the details onto a plane at a fixed instant: geometry information is flattened, lighting information is "printed" into the texture and perceived surface color is not necessarily the real one. As a result, the texture-mapped object looks very realistic if seen from the same position as the camera which took the photograph and under the same illumination conditions. However, if one of these parameters changes, the object suddenly appears strange. Moreover, if the photograph was taken from a far viewpoint, with a direction very different from a direction orthogonal to the surface or with a low-resolution lens, the texture can appear blurry, stretched and the visual quality degrades drastically. These models are thus not ideal for interactive visualization. Representing details more precisely than with a simple texture becomes crucial when viewing a surface from a close or different viewpoint. This is illustrated in figure 5.2 where a simple 3D model (top right and bottom row) has been reconstructed out of some photographs (top left) of a complex rock block and textured using these photographs. The model looks fine when viewed from a similar angle as the original photograph (top right) but its simplicity (and flatness) is evident when we look at it from a grazing angle as we do not see any relief information (bottom right).

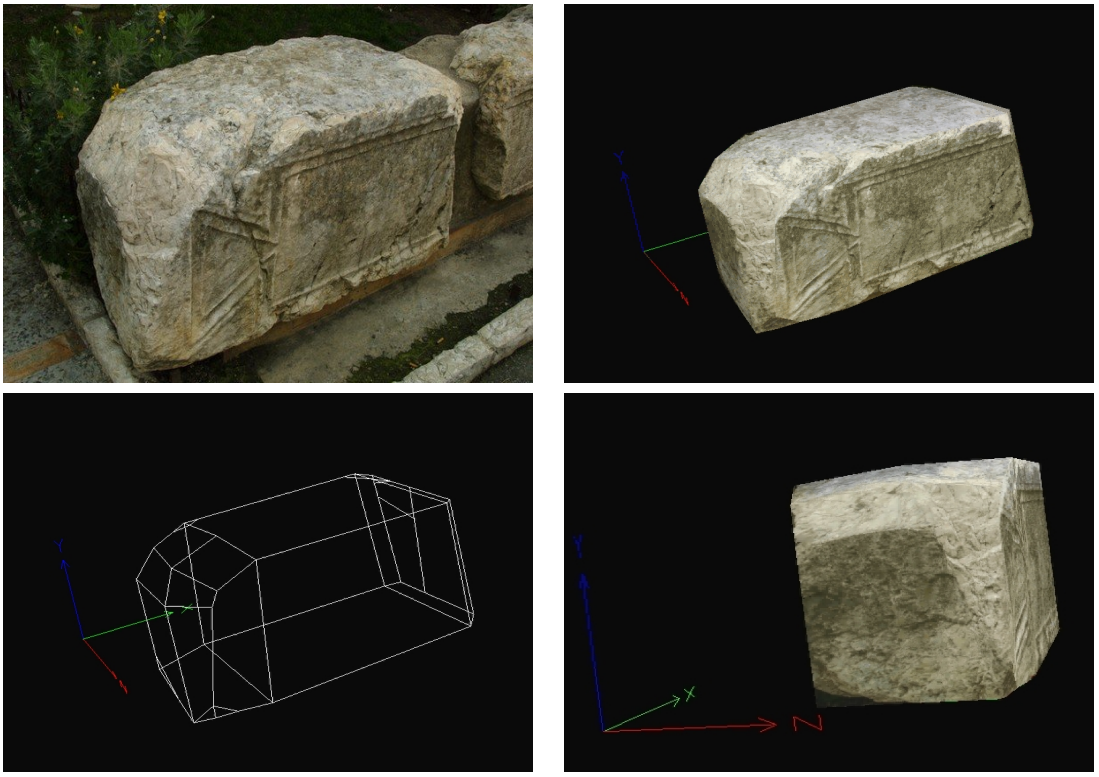

Figure 5.2 Block Example. Top left: A picture from a real-world complex rock block. Top right: A view of a virtual reconstruction of the block, obtained using photographs. Bottom Left: Wireframe view of the reconstructed model. Bottom right: Reconstructed block seen at a grazing angle: Silhouettes are straight, there are no bumps on the supposedly-rocky surface.

Another example can be seen in the images of figure 5.3. Here we can see an image-based modeled scene whose geometry is very coarse (top right). The photorealistic textures add a lot of realism to the entire scene (left column). However, it is not suited for interactive visualization as approaching the wall reveals its flatness (bottom right). In addition, the repetition of structured textures on the wall is noticeable.

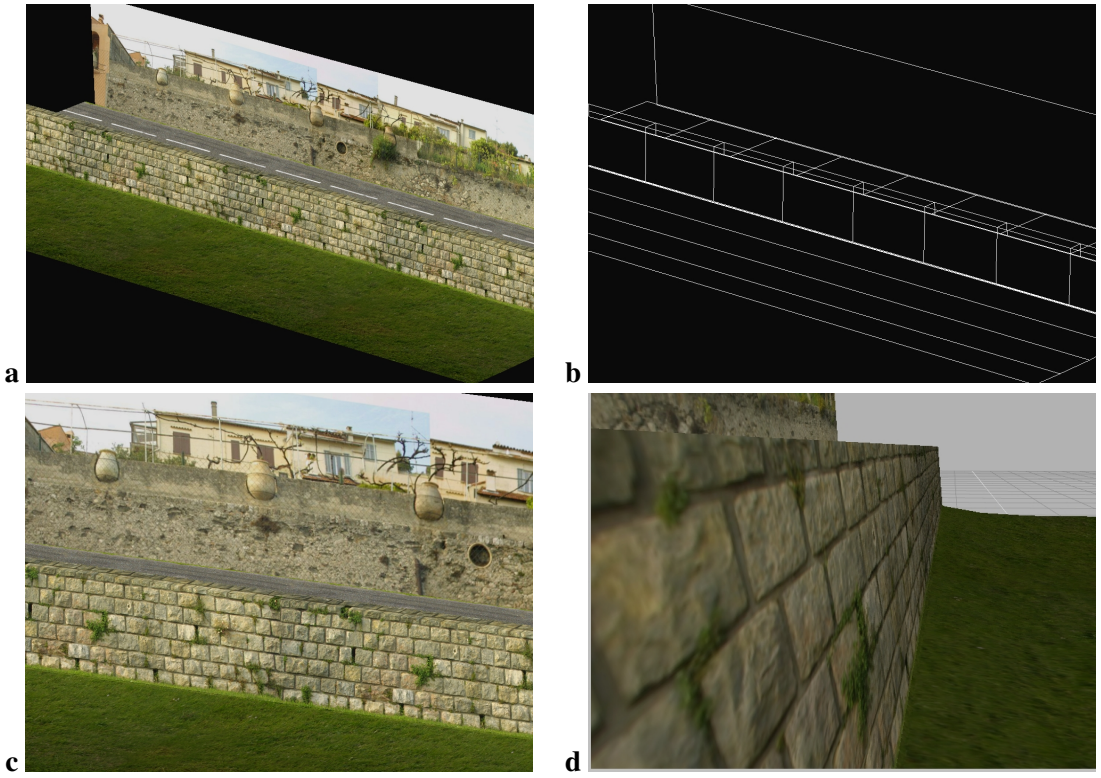

Figure 5.3 Biot wall Example. We simulate a lot of detail on a coarse model (b), using photo-realistic textures (a,c). However, it is much less realistic when seen from a close viewpoint (d).

To overcome this common problem, we propose to exploit available texture detail and user-provided information in order to recover and reintroduce details that were lost because of the imaging process (i.e., photography) performed. It should be noted that we do not aim to exactly *reproduce* the details as they were in their original state, but rather to produce *plausible*, similar-looking details. Including the user in the detail recovery process is crucial as there are currently no techniques robust enough to recover satisfying details automatically and the user can make use of his high-level understanding of the visual information to guide the process or provide additional information. In addition, artists are not fond of automatic tools over which they do not have control. Therefore, offering a workflow in which they can take an active part is an important goal.

To achieve these goals, we propose a generic interactive system designed to improve the visual quality of simple models - whose 2D appearance is created with image-based texturing techniques - by enhancing, generating and propagating details using user-provided information. The detail synthesis process results in additional appearance layers and/or enhanced models which increase the realism of the scene. We want the resulting scenes to be as believable as possible. By adding details here and there, realism improves drastically. Moreover, we do not have to generate details everywhere, only in places where the viewer inevitably notices the coarse approximations to the real scene in geometry and/or texture.

In the case of structured textures, we decoupled the 2D appearance from the underlying structure as well as the region interiors from their contours as they were clearly separate entities. The structure organizes the overall appearance but does not have any influence on the appearance of each of its individual primitives. Similarly, the primitive appearance is distinct from the appearance of what holds the primitives together. These observations also applies to 3D detail. It is being organized by the structure but does not depend on it, thus it makes sense to decouple structure and 3D detail as well. We will therefore take advantage of the structure information to enhance structured textures with 3D detail. The memory gains obtained for the 2D appearance part using shared data can also benefit the 3D detail synthesis phase. This final step results in a complete pipeline, allowing the generation of visually rich cellular textures, based on the analysis of examples. The generated structures are compact geometric representations, with associated 2D and 3D texture/detail information which produce results which are visually similar to the user-provided examples.

In this chapter, we will first present our framework for detail recovery. An application for the synthesis of displacement maps is described and some results will be shown. We will see that the structure information is required to improve these results for structured textures, which applies to details other than height information. Some possible additional layers are finally described before concluding.

5.2 User-assisted example-based detail generation: System overview

The framework we propose is interactive. Given a model, the user has to indicate which kind of details he wants to recover, where and, partly, how. The user needs to supply missing detail information as a set of small multi-layered objects (called *patches*) that the system uses to enrich a given scene, either at the geometry level, at the texture level or a combination of both. Depending on the type of detail considered, the synthesis is guided by the available underlying texture or model, which we also call the *guiding texture* and *guiding model* in the following text. The synthesis process itself also depends on the type of detail considered. In its current state and for our particular needs, it presently consists in extensions of an existing recent texture synthesis techniques [EF01]. In the future, this synthesis will also be tailored to the details and associated patches considered. In summary, the user supplies detail examples that the system analyzes, along with the available data to enhance (model, texture), in order to resynthesize this detail (or a satisfying approximation of it) where it was lost (cf figure 5.19 for a result of enhancing the flat wall of image 5.3).

The patch object

We define a *patch* P as a description of a certain kind of detail to be identified on a texture or model. It is a flexible structure composed of various *layers* $(L_1, L_2, ..., L_n)$, whose role is to provide a correlation between scene features (e.g., color and depth) in order to exploit available scene information and provide additional user knowledge. The patch is associated with a method to recreate/transfer/propagate the identified detail by exploiting the scene data and the given information.

For now, layers are small 2D images (typically on the order of a 100x100 pixels) where one pixel in a layer corresponds to the same pixel in the other layers. One objective is to extend this structure to be able to contain 3D information or any other kind of information, as described in section 5.5.1. As they provide additional data needed to recover details of any (imaginable) type, there can be numerous layers designed. They are most often edited by hand as a preprocess. For

example, 2D layers can be created by an artist using an image-editing program like Photoshop. This concept completely fits in the way artists usually work.

Detail recovery generic algorithm

Here is a brief description of the generic algorithm at the core of our proposed framework for detail recovery. A more detailed description of its use with a specific patch type can be found in the next section.

Input:

- Coarse model with its image-based guiding textures.
- User-provided patches containing missing detail information (potentially containing different types of layers).

Output:

- Enhanced model (geometry and/or texture), which can be rendered.

For a model to enhance:

1. *Initialization* phase: Assign one or more patches to each polygon or to the entire model.
2. *Detail synthesis* phase: For each polygon (or the entire model), apply the detail recovery process corresponding to the supplied patch.
3. *Rendering* phase: Apply the rendering technique corresponding to the generated details (e.g., displacement mapping to render generated displacement maps).

We have designed and implemented a flexible system to put our proposed framework into practice (figure 5.4 presents a screenshot of our system). We first present one specific type of patch, which allows the addition of some height information to our generated structured textures. However, it can be generalized to other types of textures as we will show in our results. The next section describes how our system handles this patch.

5.3 First application: Displacement map synthesis

Amongst the various types of details we might want to recover, depth is one of the most important as it is the main thing lost when one photographs a scene, due to the projection from 3D to 2D. Recovering depth from images is a well-studied problem and there is no easy robust solution available. As seen in section 2.1.2, artists use hand-made displacement maps to provide depth information for a given color texture. Considering this, we present a first patch in our detail synthesis framework with the goal of easing the creation of displacement maps.

Displacement mapping is a commonly used technique to add geometric details to CG scenes. A displacement map, where height differences are indicated by differences in grayscale color values, is first created, usually by hand, and applied to the geometry. When an artist uses a large texture and wants to displace the associated model to create some relief, he typically has to draw a large corresponding displacement map. This is a tedious and long task since all the details must be sketched by hand on the entire map. This is true even if similar details were already drawn in another part of the texture but in another position/orientation/shape or are only slightly

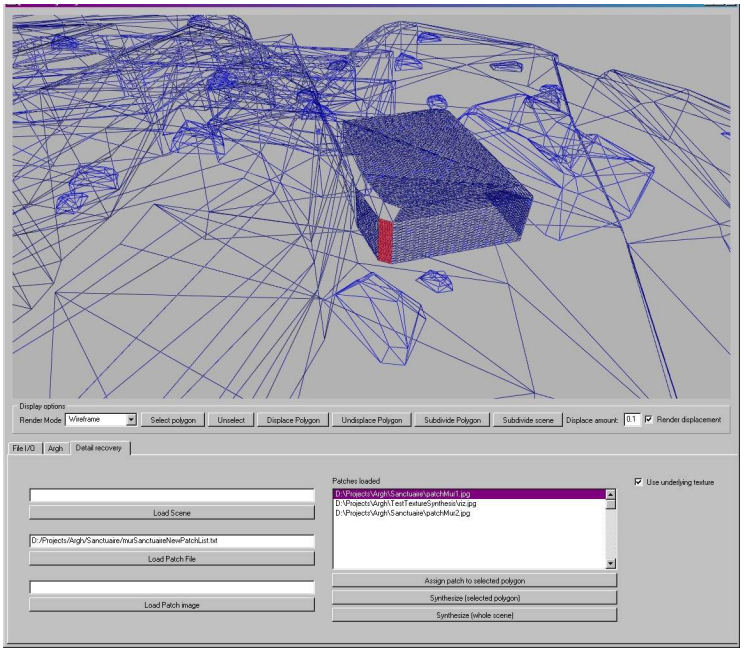

Figure 5.4 A screenshot of our system.

different (see images 5.16 or 5.21 for such examples). Often, directly using a grayscale version of the texture present on the surface as a shortcut to drawing the map does not give good results as, in most cases, the grayscale values of the colors cannot be directly linked to the surface depth variations. In figure 5.5 top, we can see a texture coming from a photograph of a real-world wall. Transforming this texture to grayscale values and applying it as a displacement map leads to the result of figure 5.5 bottom. The interstice between bricks looks correct as it was darker than the rest but the surface of the bricks is too noisy and the foliage is completely incorrect. To correct this, the artist needs to draw the complete displacement map by hand (possibly using the grayscale version as a base pattern to edit) and this can take a lot of time. Thus, a method to help generate these maps would be more than welcome.

For our method to allow the recovery of such detail, we introduce three types of layers:

- *Color layer*: This is a small color image. It can be drawn, extracted from one of the original image-based textures or from another instance of an original texture (e.g., with better resolution, photographed from closer).

- *Height layer*: This is a 2D map containing height information, in the form of grayscale values, for the corresponding pixels in the color layer.

- *Mask layer* (optional): This is a layer which specifies pixels to ignore from another layer. For example, pixels whose color is the result of an illumination effect (shadows, highlights) and whose value should be ignored.

The user must supply patches containing at least the two first layers. Figure 5.6 contains two example patches used for the wall example where the depth layer was edited by hand using Photoshop. We made sure to capture two main different details of the wall: the bricks and the foliage, where the correlation between color and depth is clearly different. For the bricks, the interstice is darker and its depth should be negative, whereas the foliage is dark as well (in tint)

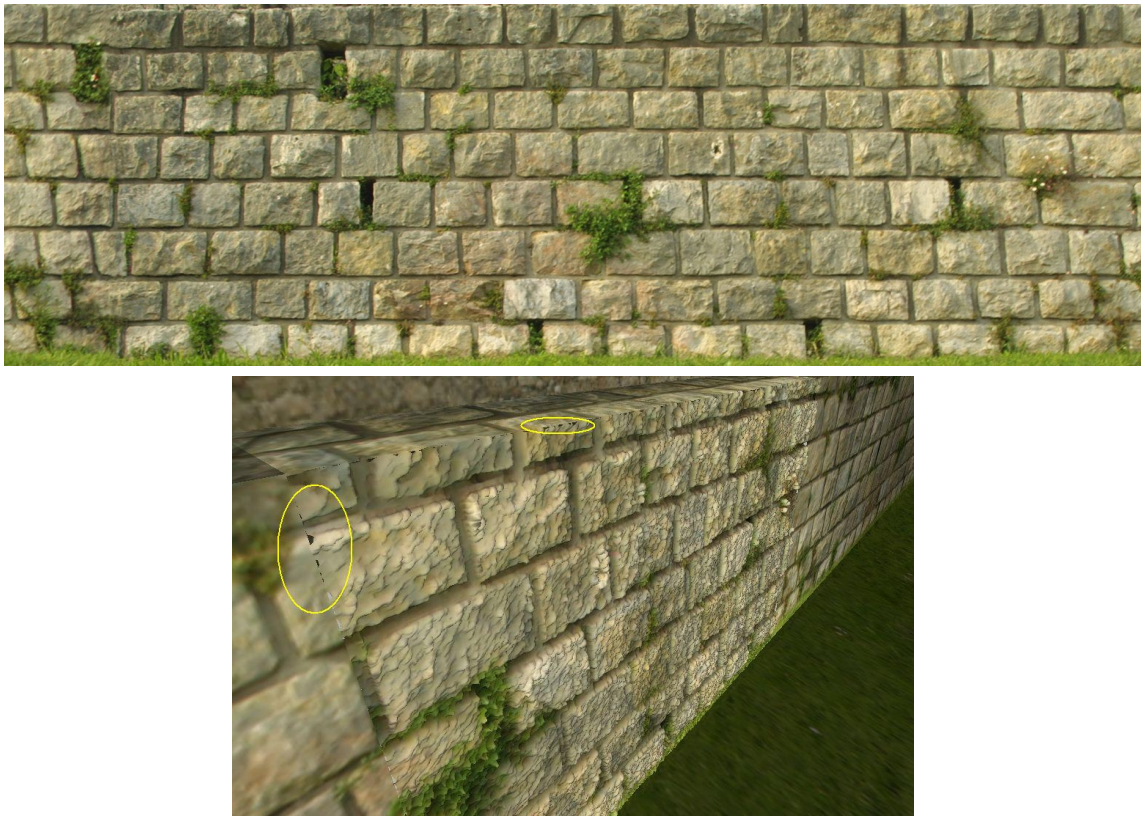

Figure 5.5 Top: Texture map coming from a real photograph. Bottom: Unsatisfactory displacement resulting from using a grayscale version of the texture as a displacement map. There are too many high-frequency details which are not plausible. Only the front polygon has been displaced, to be able to compare with the undisplaced wall in the back/front. Some gap problems are circled in yellow.

but it is rather protruding. With only one patch, there would inevitably be depth mistakes as the dark tint would be correlated with only one depth information. Since the synthesis algorithm uses color distances (as described below), if one color in the guiding texture is not part of any patch, it chooses the closest color and thus potentially the wrong depth.

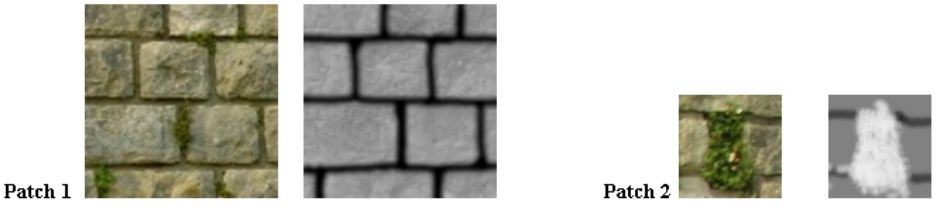

Figure 5.6 Two patches used for the wall example, containing a color and a depth layer.

Once the system has been initialized by assigning one or more such patches to the polygons where depth is requested, the detail synthesis process can start. It will generate one *result color image* and one *result depth image* per designated polygon by using our extended version of Efros and Freeman's image quilting technique [EF01] which we describe in the following.

Image quilting technique This texture synthesis method (depicted in figure 5.7) creates a final result texture by choosing a set of blocks of pixels from an input texture (image 5.7, left). The block size is fixed and user-defined. The final texture is generated block by block in raster-scan order. Each block overlaps the block at its left and/or above it by a certain percentage (user-defined). For each new block to place on the result texture, a list of randomly chosen *candidate* blocks coming from the input sample texture is created. Each candidate block is evaluated in turn by computing an error metric on the pixel colors in the overlapping region(s). The one with the lowest error is chosen as the next block in the texture being synthesized. Once all the blocks have been selected, the color error between adjacent blocks (top,left) is minimized using a minimum boundary cut (in color space) in their overlap region (image 5.7, right).

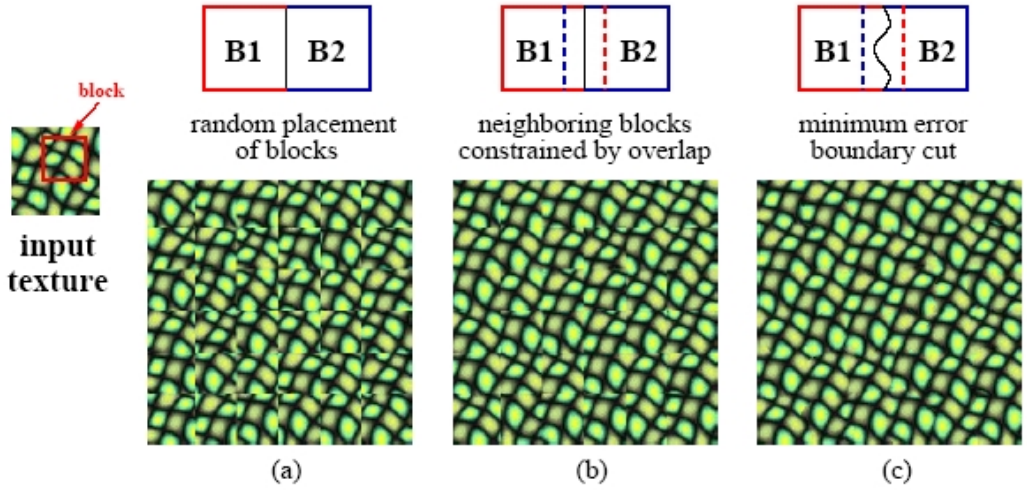

Figure 5.7 The Image Quilting technique. From an input texture (left), the algorithm extracts blocks which are constrained to overlap (b) (rather than placed randomly (a)). The visible seams in the overlap regions can be reduced using a minimum boundary cut algorithm computed on the color errors. (Image from Efros and Freeman [EF01])

The technique can also use a *guiding image* (whose size is the same as the result texture) to influence its choice of blocks during the synthesis (as described in section 2.2.2 and illustrated in figure 2.8). This adds the requirement that each candidate block be also as close as possible (in color) to the corresponding block in the guiding image, by adding a term to the computation of the error metric. This metric can thus contain one or two terms (depending if a guiding image is used or not):

- *Overlap term O*: This is the error in the overlap regions (left and/or top) of the candidate block and the already chosen blocks in the resulting color image.

  We define B_c as the candidate block (B_2 in figure 5.7) and B^{ov_h} and B^{ov_v} as the horizontal and vertical overlap regions of a block B_i with its adjacent top (B_{top}) and left (B_{left}) block. If there is no such block, the corresponding term is null. Refer to image 5.8 for an illustration of these various terms.

  Using this notation, the overlap error can be written as

$$O(B_c) = (B_{top}^{ov_h} - B_c^{ov_h})^2 + (B_{left}^{ov_v} - B_c^{ov_v})^2. \tag{5.1}$$

- *Guiding term G*: This is the error between the candidate block pixel values and the pixel values of the block at the current *guiding image* position B_{guide} (which is equal to the

position in the result texture being synthesized). The guiding error term can be written as:

$$G(B_c) = (B_c - B_{guide})^2. \tag{5.2}$$

Each error term is computed as the distance in the RGB color space of the pixel values using a L_2 Norm (sum of squared Euclidean distance between pixel intensity values)[1]. The final error term $E(B_c)$ can be written as:

$$E(B_c) = \alpha O(B_c) + (1 - \alpha) G(B_c). \tag{5.3}$$

where α is a tuning parameter indicating the tradeoff between the color coherence for the texture synthesis and the fidelity to the guiding image.

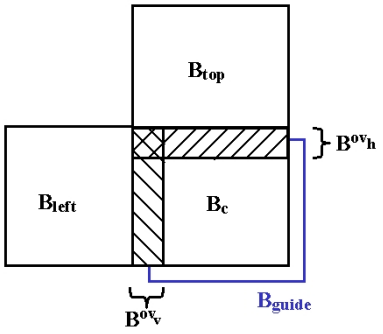

Figure 5.8 The candidate block (B_c) overlaps with the blocks to its left and/or above it. Its color error with the corresponding block B_{guide} in the guiding texture can also be integrated in the error metric.

5.3.1 Multi-patch depth synthesis

Our approach can use multiple input patches at the same time. This multi-patch synthesis algorithm, pictured in figure 5.9, proceeds as follows:

1. The texture already mapped onto the polygon serves as a *guiding image* to influence the synthesis process.

2. For each input patch, we compute a color metric for the color layer. This metric is a simple color average in the CIELAB colorspace but could be replaced by any other more advanced similarity metric.

3. We synthesize the result color image block by block, following the guiding image in a raster-scan order. For each block to synthesize, we compute a color metric based on the guiding image corresponding block and compare it to the pre-computed metrics of each of the input patches. This permits us to choose an input patch from which random candidate blocks are extracted [2]. In order to rapidly build the final result images at the end, an index to each input patch is kept along with the chosen blocks during synthesis.

[1] We use the same colorspace as in Efros' algorithm. However, we feel the CIELAB space would be more appropriate for such distance computations.

[2] Using this metric for choosing the input patch is a speed-up technique for the texture synthesis process to limit its search space for candidate blocks, but we could evaluate candidates from all the patches simultaneously.

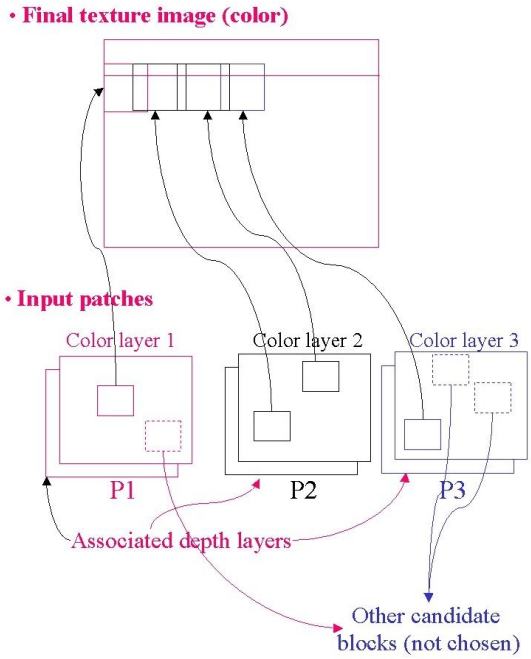

Figure 5.9 Multi-patch texture synthesis.

4. Once an input patch has been selected, we use a technique similar to image quilting to pick a block amongst the candidate blocks: An error metric is computed for each candidate block and the block with the smallest error is selected as the next block in the result color image being synthesized.

   We added a term to the quilting method error metric to consider the depth layer when choosing a block. This is the *depth term* D. It represents the error in the overlap regions (left and/or top) of the depth block corresponding to the candidate color block and the already chosen depth blocks in the result depth image. This term is important in order to maintain the spatial coherence as much as possible in the generated depth map and can be written as:

   $$D(B_c) = (B^{ov_h}_{top_d} - B^{ov_h}_{c_d})^2 + (B^{ov_v}_{left_d} - B^{ov_v}_{c_d})^2, \tag{5.4}$$

   where B_{i_d} is the depth block in the depth image corresponding (same position) to the block B_i in the original image. See image 5.10 for an illustration of these terms.

   Therefore, the error metric E is now the sum of the three terms G (Guiding error), O (Overlap error) and D (Depth error) (equations 5.2, 5.1 and 5.4):

   $$E_{new}(B_c) = \alpha_1 O(B_c) + \alpha_2 G(B_c) + \alpha_3 D(B_c),$$

   where $\alpha_1 + \alpha_2 + \alpha_3 = 1$.

   We can tune the influence of each term through the parameters α_1, α_2 and α_3.

5. The result color image is built by copying each chosen block from the appropriate patch color layer using the index stored during synthesis. The depth result image is built by copying the corresponding depth blocks from the associated depth layer. The chosen blocks overlap at a certain percentage, as defined.

6. Stitch all the blocks together to reduce the seams visibility as is done in Image Quilting.

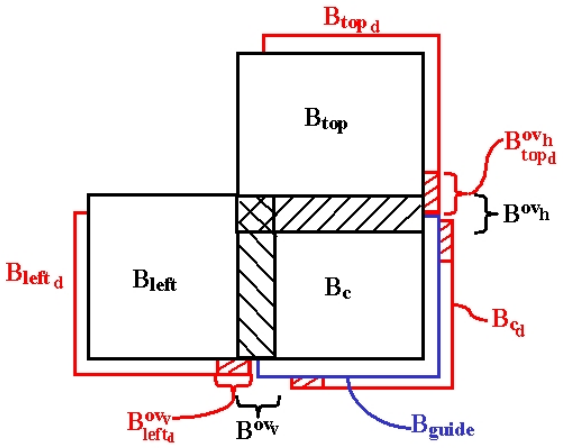

Figure 5.10 The coherence in depth space is preserved by adding an overlap term similar to the one in color space.

We thus obtain one depth map per designated polygon which we can use as its displacement map. We can blur the maps, if needed, so the changes in height are not too sharp. To render the scene with this new information, we apply a typical displacement mapping technique, subdividing the polygon to an appropriate resolution and using the generated depth map to displace the polygon subdivision vertices. To correct blurriness resulting from the stretching of the color texture by the displacement, we can re-synthesize colors in areas where we detect a large displacement. However, while producing renders of our scenes, we faced a problem that has been known for years in the film production industry but which has not been addressed in research. We describe it and propose a possible solution in the next section.

5.3.2 Rendering issues

We chose to apply the displacement maps separately on each face as the subdivision vertices are not shared across edges. This is important when a different polygon subdivision level is needed on each face. We call a *seam* the limit between two adjacent coarse (non-subdivided) polygons which form a *sharp* edge. Initially, this space is null since the polygons are joined together. Once they have been displaced, this space can become non-null and we call it a *seam gap*. This is a typical problem with displacement mapping, faced by CG production people [DKS01]. We thus preprocess the scene and construct one seam structure per pair of adjacent polygons, containing information on these neighboring polygons (which will consist in several polygons, once subdivided). These seam gaps pose a problem both for the geometry and the texture.

Seam gap stitching

At the geometry level, since we apply the displacement map separately on each face, two adjacent faces can become disconnected once their vertices have been displaced. We use a simple zipping algorithm to reconnect the two edges:
Let E_1 and E_2 be the edges of the seam gap we wish to repair and N_1 and N_2 their number of vertices (N_1 is not necessarily equal to N_2). We want to stitch E_1 and E_2. For this, we use a two-pass process:

1. We follow each edge in parallel and for every vertex of E_1, we find the closest vertex in E_2

so that no other vertex further along in E_2 is already connected to a vertex in E_1. This is to avoid overlapping new edges.

2. We join the unconnected vertices together and build new triangles where needed.

The result is a strip of triangles joining the two separated faces (see figure 5.11). We fill the seam structure with the newly created triangles since it is a separate, editable structure, it is easier and faster to process a seam separately after displacement mapping.

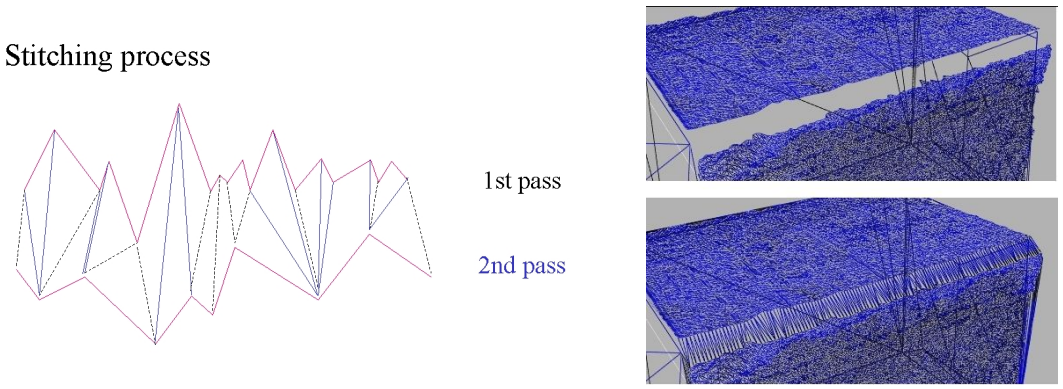

Figure 5.11 Two-pass stitching process (left) and result on an exaggerated seam gap.

Seam texturing

The texture should be consistent across faces, even in the presence of sharp edges. We need to build a texture image for the strip of triangles resulting from the above-described stitching. We developed a simple efficient technique: we extract a sufficiently large strip of texture from each of the faces adjacent to the seam. We reverse (unfold) them so as to preserve continuity from the faces into the seam. We make them overlap and we blend them together using a minimum boundary-cut method so as to minimize the color error in the region they overlap. This method is described in more details in the following algorithm:

Let P_1 and P_2 be the coarse polygons around the seam. They each have a texture (original or generated) and thus some texture coordinates defined for their vertices. See figure 5.12 for an illustration of the process. The following steps are performed:

1. Compute d, the maximum distance between the seam extremal vertices (AB and CD if ABCD are the vertices of the seam between P_1 and P_2).

2. Let $d' = 0.75 \times d$ (0.75 = user-specified parameter). For P_1 and P_2, we can find four 3D points at this distance from the seam: P_1, P_2, P_3,P_4.

3. Find their equivalent 2D points in the texture space of each polygon: P'_1, P'_2, P'_3, P'_4.

4. Extract the texture sample in the 2D polygons. $A'P'_4P'_3D'$ and $B'P'_1C'P'_2$ in each of the two textures.

5. Build a texture image where these two texture extracts overlap at a user-defined percentage (50% in practice).

6. Use the minimum boundary cut algorithm to minimize the color blend error in the overlap region.

7. Apply this generated texture to the seam geometry (the stitch). To get the seam 3D vertex texture coordinates, we simply interpolate in texture space according to the 3D position of the stitch vertices relative to ABCD [3].

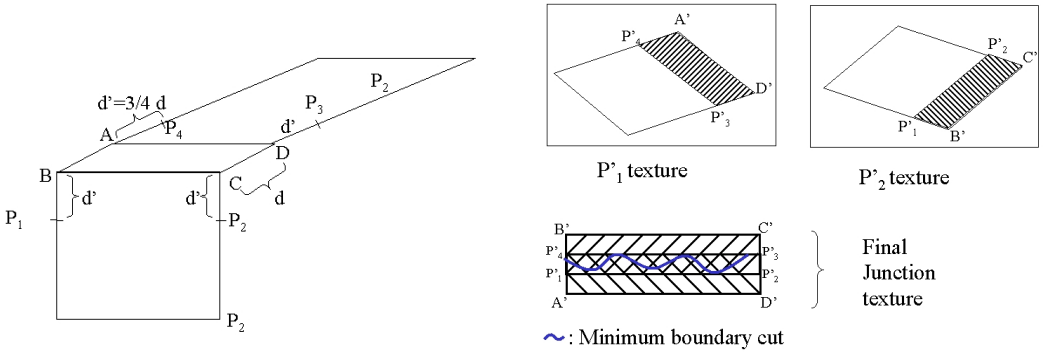

Figure 5.12 Texturing a seam by blending textures coming from two adjacent polygons. Two texture portions are extracted from the adjacent polygons, then merged seamlessly using a minimum boundary cut.

For very small seams (a few pixels), we simply blend the colors coming from the two thin strips of extracted texture. A seam texturing result for the wall can be seen in figure 5.13: we can see the two adjacent polygon texture extracts and the blended result. Figure 5.14 shows another example of a successful seam reconstruction.

Figure 5.13 Texture extracts from adjacent polygons (top and bottom) merged into one single texture (result). A close-up of the result is also shown.

The results obtained on various models are presented in the next section.

[3] Since the seam is typically small, errors due to the fact that the seam geometry is not entirely flat (as our texture reconstruction process implies) are generally not noticeable.

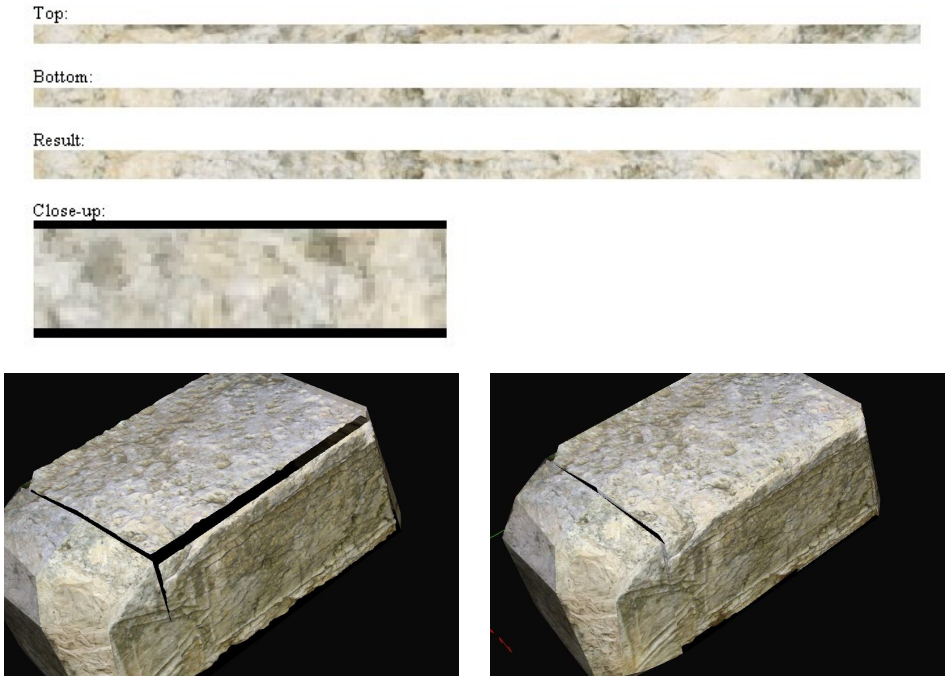

Figure 5.14 Solving the texture gap problem. Top: The stitch used to texture the recreated seam. Bottom: Exaggerating a seam gap (left) to appreciate the stitching and texturing results (right). Only the two righthand polygons have been stitched.

5.3.3 Results

We obtain satisfying results for displacement map recovery and seam stitching on a variety of input models, presented in figures 5.15 to 5.20.

Rock Figure 5.15 presents results obtained for an image-based reconstructed model of an old stone. In the left column, we show one of the patches used and the displacement map obtained for the front polygon. The right column contains two renderings obtained after displacement mapping. Note that only the front facing and the top polygons were treated. The displacement map leads to a consistent stone displacement but not necessarily that of the original stone. Notice particularly the results on the stitch where the two adjacent polygon textures have been merged seamlessly across the new strip of triangles.

Door Figure 5.16 shows results obtained for a wood door. The result using a grayscale version of the texture is shown in the top left, next to the two patches used to get the results displayed in the bottom row. We can clearly see the improvement in realism, obtained semi-automatically, since only two small patches were required.

Chip Same for the results on a chip (Figure 5.17). Building the displacement map automatically is not possible due to the many variations in color.

Brick Wall Figures 5.18 to 5.20 present the results for the brick wall model. The generated displacement map is of satisfactory quality. Only one big polygon has been displaced to show the increase in realism in comparison to the flat polygon on the right. Notice that the foliage is correctly displaced. However, some borders have not been correctly handled as indicated by the yellow square. This is due to clearer mortar parts in the input texture (image 5.5 top) whose color has been mismatched with a similar brick color.

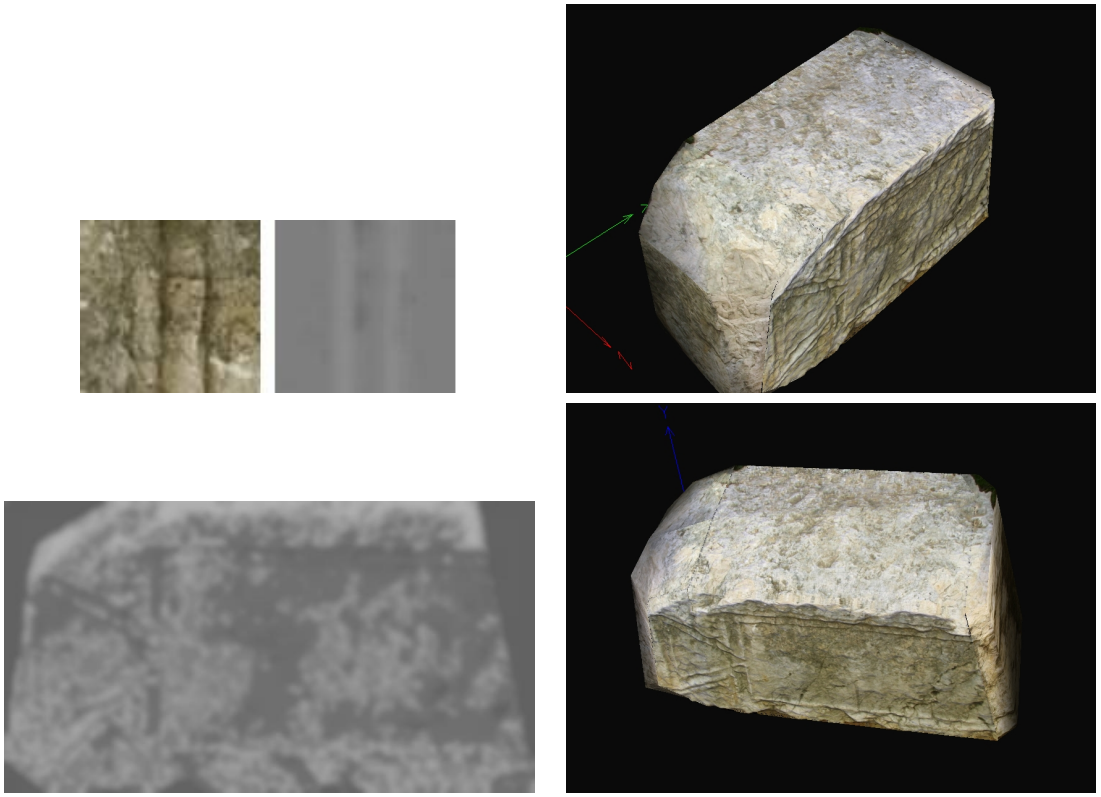

Figure 5.15 Results for an old stone. Left column: One of the patches used (top) and one of the displacement maps obtained (bottom). Right column: Two views of the displaced model.

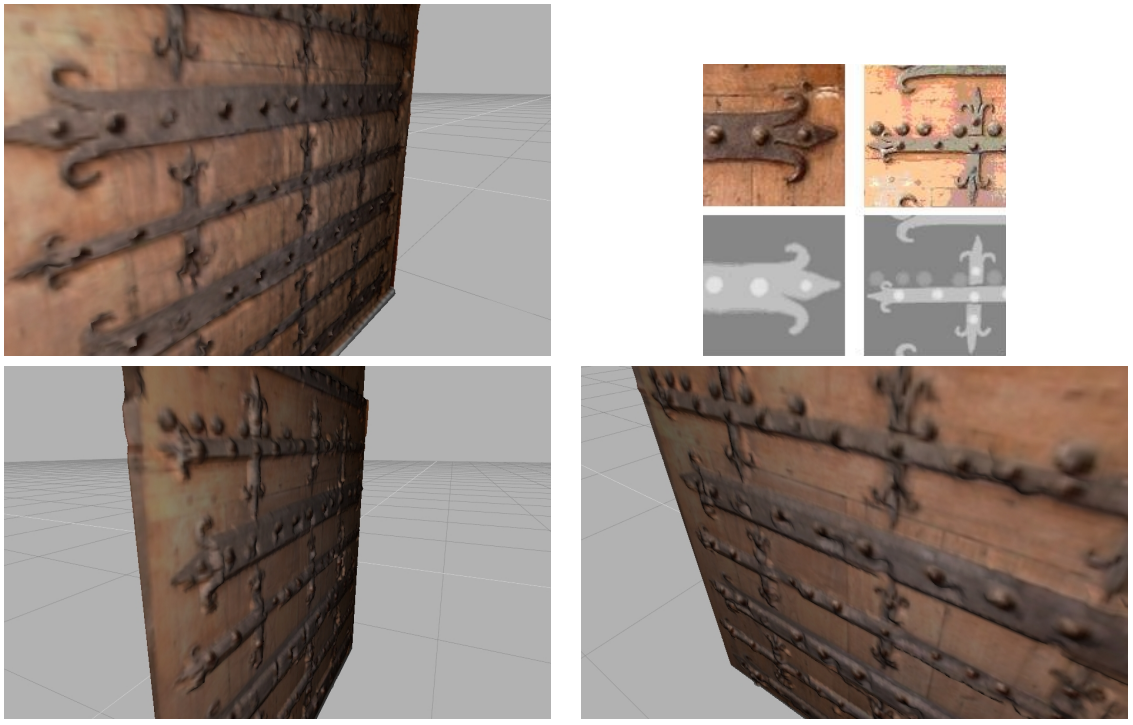

Figure 5.16 Door example.

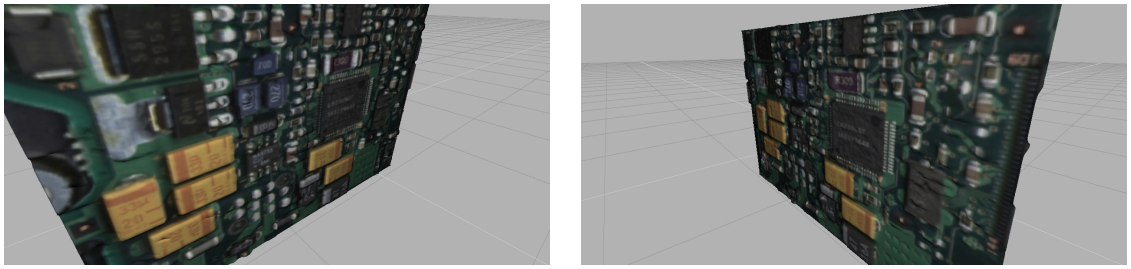

Figure 5.17 Chip example.

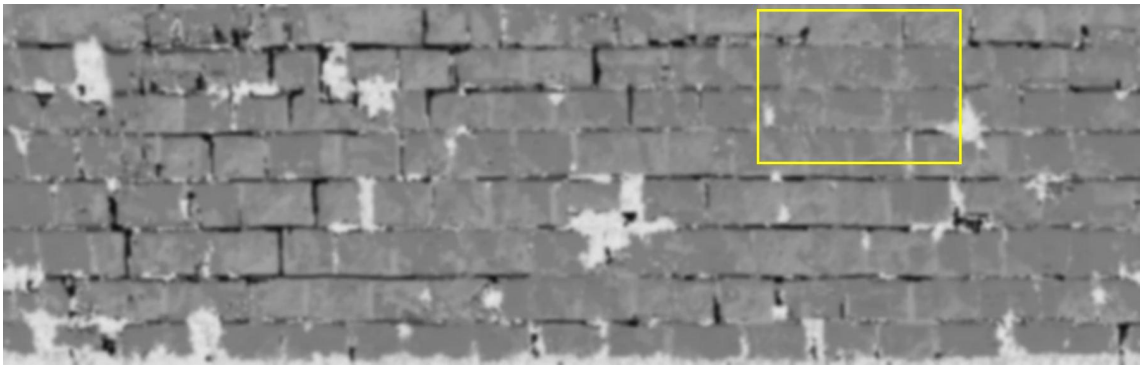

Figure 5.18 Wall Example: The generated displacement map.

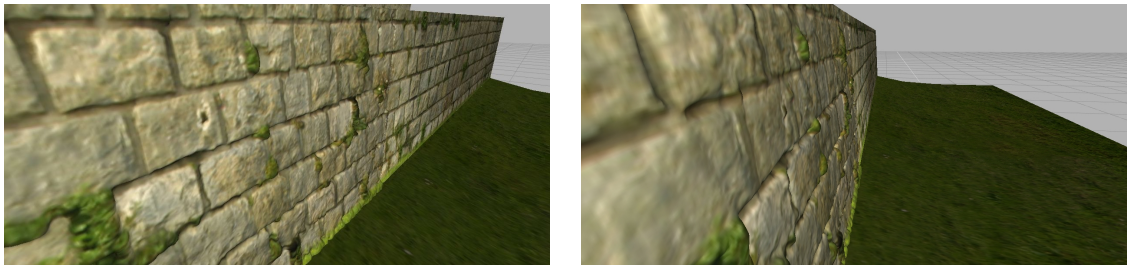

Figure 5.19 Wall Example: Two views of the displaced textured wall.

The rendering is quite coarse as we use a custom basic renderer. For the same reason, the wall has not been subdivided enough, resulting in the smooth aspect of the surface, as seen in image 5.20. This aspect is also due to the fact that no difference has been made between the mortar and the stones. These problems are evidently overcome when using the approaches described in the two previous chapters, decoupling structure and appearance, as we shall discuss in the next section.

5.4 Enriching generated structured textures with 3D detail information

As seen above, a direct application of the displacement map generation achieves satisfactory results for unstructured textures, such as that of the stone (figure 5.15). However, for cellular structures, such as that of the wall in figure 5.19, the technique suffers from all the problems

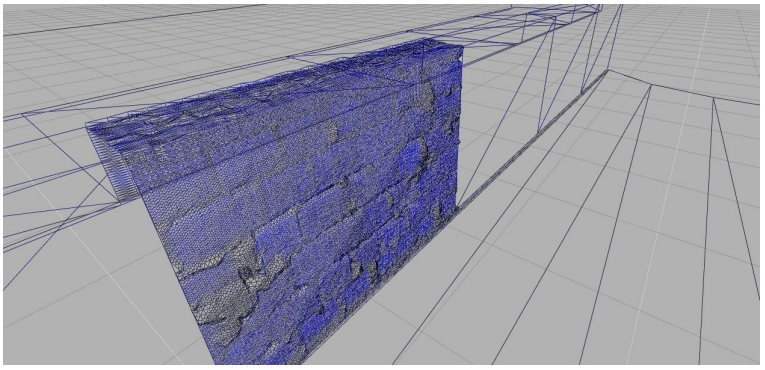

Figure 5.20 A wireframe rendering of the Biot wall example, showing the smooth subdivision.

faced by texture synthesis techniques as discussed earlier (section 2.2.2). The obvious solution to this problem is the combination of our techniques for structure and 2D appearance synthesis, developed in the two previous chapters.

To see the problem in more detail, consider the case of the wall example of figures 5.18 and 5.19. Since the notion of structure is not taken into account during the displacement map synthesis, no distinction is made between a pixel from a region interior and a pixel from the contour. Therefore, the synthesis will blindly pass through borders, potentially damaging them. This can be seen in image 5.18 where the part of the texture inside the yellow square clearly has border problems. The height pixels in the border area are clearer than the ones of the surrounding regions, which is not what we want. In addition, if we observe the close-up in image 5.19, right, we can see that the interstices are not as clearly marked as we might want them to. The transitions from region to region is quite smooth when it should not as rocks and interstices are two different objects and should be clearly seen as such.

Two main approaches can be employed to add 3D information to structured textures generated with the techniques described in the two preceding chapters.

Creating one displacement map per unified material textures As described in section 4.1, texturing of the generated region interiors is performed by first building a set of unified material textures and then sampling them to texture the generated regions. This texture instancing allows a fast and memory efficient rendering of a structure of (almost) any size. To continue to take advantage of this instancing, we can build one displacement map per unified texture using the techniques described in this section. At least one patch per unified texture has to be drawn, more if the texture contains some features, like moss growth, for instance. Since the color range spanned by one unified texture is narrow, due to its construction process, there are fewer risks of mismatches between color values when attributing a depth value.

By building unified displacement maps, we can also take advantage of instancing for the depth channel. When a texture for a region is sampled from a unified texture, the corresponding area from the associated displacement map is used as the depth information for this region interior. The depth synthesis is also a preprocessing step and thus does not incur a penalty in computation time. The same approach can be used for interstices and junctions, with a specific patch for each. This patch will be used to treat the interstice atlas and the junction atlas, creating corresponding depth atlases.

One advantage of this technique is that additional rendering parameters can be applied sepa-

rately to the region interiors and their contours, by building additional common reflectance textures to enhance the unified color and depth textures. Setting different reflectivity factors for the contour and the regions material will increase the contrast between the two and consequently enhance the realism of the whole structure.

Using the structure information during synthesis The structure information can be directly used during synthesis or for editing purposes. Once the structure and appearance have been generated for a target surface, each pixel can be straightforwardly categorized as pertaining to a region or a contour. Thus, they can be handled separately and a variety of patches and effects (filters) can be applied to them in a post-process step. It is possible to modify at will and independently the two parts of the generated appearance: the interior or the contours, without risk of affecting the other element. For example, expanding all the contours by a certain amount by specifying a filter that would increase the contour pixel span, modifying the reflectance properties of the region interiors, etc. Some example of possible patches are described in the last section of this chapter.

In summary, it is possible to use the structure information as a pre- or post-appearance process. In the first case, it allows the generation of depth information beforehand for interactive rendering. In the second case, it allows the independent modification of the region interiors or their contours.

The integration of the detail recovery module with the structure and appearance recovery modules is not yet completed. In the future, we will experiment to determine which parts of this process work well, and develop solutions for the case where further algorithmic enhancements are required.

5.5 Conclusion

Displacement is one type of detail we can recover using our system with simple patches; semi-automatically producing displacement maps is a valuable tool for artists. However, one can argue that this technique does not work on all examples, especially on textures where there is no correlation whatsoever between the color values and the height values. We discuss next some proposals on how to inject more knowledge (in the form of additional patch layers) in order to recover other types of detail. These extensions are described in the following section.

5.5.1 Extensions: Recovering other type of details

As we have seen, adding details to a scene can have various significations. In our approach, we mostly target detail recovery whose manual specification constitutes a repetitive, time-consuming task, as we have seen with the creation of displacement maps. In the following, we propose additional patch layers to fulfill this objective. We describe the considered layers, how the user has to specify them, how they have to or can be combined with others, etc. They should allow to recover various 3D information attributes.

Shape layer The displacement map synthesis technique fails on example such as the metal plate in figure 5.21 (left). Here, color and depth clearly cannot be associated because the almond-shaped extrusion is of the same color as the metal base. Indeed, using the color and height layers shown in the middle image leads to the wrong resulting depth, as can be seen in the right image. However, simply adding a criterion of shape to constrain the depth recovery would avoid this problem. By

defining a shape criterion, we could test if the pixels (or blocks) we are currently synthesizing are *inside* or *outside* the shape defined in the shape layer and constrain the search-space for candidate pixels (or blocks) to the corresponding part of the color, and thus depth, layers.

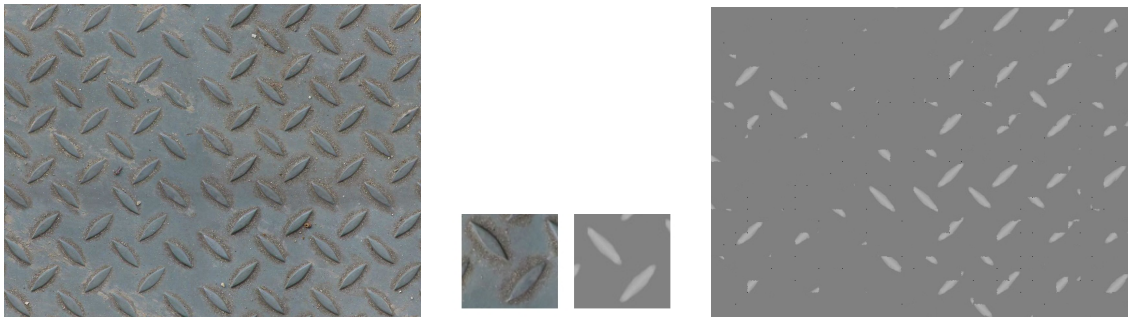

Figure 5.21 Example where displacement map synthesis fails. Left: The guiding texture. Center: The color and depth layer of the patch used. Right: The generated depth.

The shape criterion we must identify should have the following properties: it should be invariant to translation, rotation and possibly, for some cases, scaling. The criterion used in chapter 3.5.4 would be a good choice. It should be associated with a robust segmentation technique, keeping in mind that the shapes we want to handle should be simple and thus not pose any problem to these algorithms. The identification process should also be flexible enough to allow the specification of an approximate shape and a tolerance criterion (deviation parameter from the specified shape) for its identification. This criterion could also improve the results obtained on the door example of the previous section.

With this approach, we are in fact separating the search space for candidates into two regions. Multiple shape layers can be defined for the same patch, dividing the search space into as many regions. One can argue that this resembles the Texture-by-Number application of Hertzmann *et al.* [HJO+01] (see section 2.4). Our criterion is different as we consider the shape of objects as the constraint and the regions are automatically formed by localizing these shapes into the guiding texture, even if they have been scaled, translated or rotated, we do not have to draw them in advance.

Geometry edition layer Building on ideas from Brooks and Dogson [BD02] (see section 2.4), we propose to extend this approach to 3D. In the original 2D approach, interactive specification of texture-editing operations performed on a small part of a texture are replicated throughout the entire image using a measure of similarity exploiting the pixel neighborhoods. We propose similar 3D model operations and additional ones (edge erosion, surface deformations, etc.) on a small part of a model and replicate them on the entire model or on other models. Similarity measures in 3D to estimate a "3D local neighborhood" have to be defined using, for example, surface local curvature, along with texture similarity, if needed. In addition, we must define a simple way to specify the 3D editing operations, for example by storing the displacements between the original and the modified surfaces. In short, this layer uses 3D geometric attributes of the underlying guiding shape to control modifications to apply to the geometry (a sort of "model analogies" approach). Recently, a similar idea was presented by Zelinka and Garland [ZG04b] using "geodesic fans", a new framework for local surface comparison, and Bhat *et al.* [BIT04] who extended the image analogies framework in 3D.

Geometry template layer This layer aims to fit simple 3D template models to detected projection of these models in the guiding texture. Rather than having 2.5D information in the layer like

in the depth layer, we could have true 3D (for example, models of small rocks, facade features like protruding balconies, windows, ornaments). Using a combination of a color and a shape layer, we could detect these objects in the guiding texture and enhance the underlying coarser model with the 3D template information contained in the layer (see figure 5.22). This template will have to be transformed (scaled, rotated, morphed) to fit as much as possible to its detected projection. Efficient results from the shape recognition research area will have to be used here to recognize the potential projections of the given template models in the image.

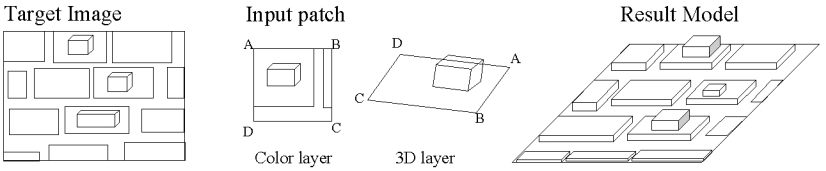

Figure 5.22 The geometry template layer. Left: a guiding texture. Center: two layers of the patch. Right: Augmented texture with 3D information.

Models such as the ones pictured in figure 5.23 would greatly benefit from the use of this layer. Remember that we only aim to add plausible details so an exact solution is not required. It should be noted that techniques aiming to add geometry to existing models will have to address problems of texture seams and texture stretch.

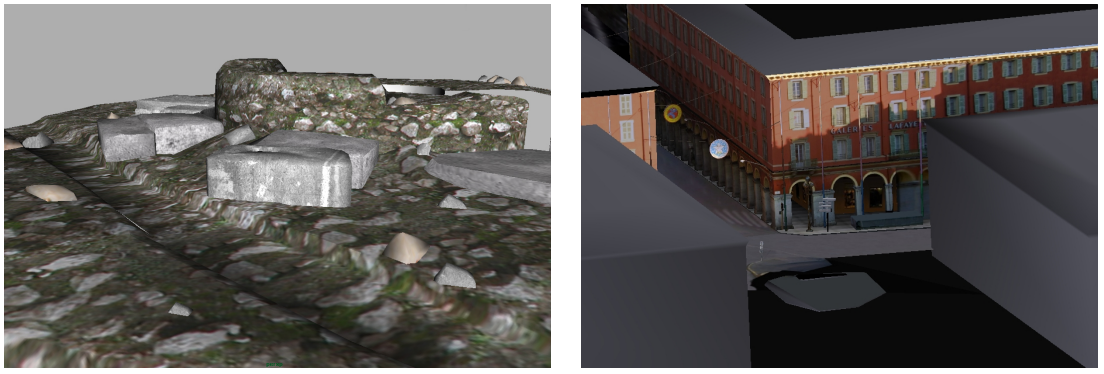

Figure 5.23 Adding 3D template models for the rocks in the left image or for the windows and other facade attributes in the right image should increase the realism of these models.

Reflectance (shading) layer When a texture coming from a photograph is applied on a surface with some reflectance properties, all the objects "printed" in the texture get these properties, after having lost their own properties due to the imaging process (i.e., photography). This constitutes a big loss of detail as intrinsic reflectance properties constitute important cues on the objects surface geometry and adds a lot of richness and realism.

In figure 5.24, we can see two images of a scratch with different reflectance behavior depending on the viewpoint. Using either of these images on a coarse model of the box produces a very different effect. We will not get the impression of a scratch in the surface but rather a drawn line of fixed color on the surface of the wooden box. For such a small effect, it is not computationally efficient to affect the geometry of the surface. Merillou *et al.* [MDG01] offer a solution to simulate this geometry. However, a *scratch map* has to be drawn by hand and our framework could help automate them from this task.

Figure 5.24 Different light behaviors for a scratch. (Image from Merillou *et al.* [MDG01])

We thus propose to augment the original color texture with reflectance information. This reflectance information can either be an additional texture containing reflectance parameters to be applied on top of the base color texture, or some lighting model parameters or approximations used during the lighting computation for the considered surface/part of surface (e.g., shading information like BRDFs [War92] or BTFs [DvNK99]). We offer the user the ability to design a patch reflectance layer (corresponding to a given patch color layer) with specific lighting information for various parts of the color layer. The synthesis process would propagate this reflectance information, along with the other layer informations, to produce a reflectance texture. This additional appearance layer would then be used at rendering time to produce different effects for each material. In addition, having knowledge of the structure is very useful for this kind of layer since the structure region interiors often have reflectivity parameters which are different from those of their contours, and we can independently modify them as described earlier.

We can imagine various other types of layers. We can even think of layers containing rules to apply to the sample depending on the guiding texture or underlying model. For example, varying the shape of texture elements depending on the variation in shape of the object (e.g., to vary the spot sizes on a leopard skin, which are different on the tail and on the back). Basically, if we can identify a correlation between variations in the guiding texture and/or model and some changes we want to apply, we could design a layer to specify it. Additional layers containing parameters to modulate the information found in other layers should also be easily definable. In order for all these layers to be easily described, used and combined, the layer structure needs to be extended, especially related to the layers involving 3D.

5.5.2 Discussion

We propose a novel framework to allow an artist to semi-automatically enhance a scene by identifying correlations between scene features which can be exploited by a synthesis algorithm. By letting him provide a description of the kind of details he wants to recover using the available coarse geometry and textures and by introducing an automation in the repetitive "intelligent" recovery of this detail, we aim to relieve him from tediously creating all these details by hand. Indeed, the underlying image-based textures and geometry already contain *a lot* of information on the original scene, and a human user is much more reliable to evaluate its relevance than an automatic analysis method. Exploiting this information in combination with user-provided knowledge thus constitutes a new, promising approach to the problem of detail recovery. Moreover, using a

set of layers to supply the additional information fits the artists usual way of working (i.e., using Photoshop to add layered rendering information on top of their base textures).

We implemented the depth recovery part of the system as it was the natural step to take to enrich the structures generated with the techniques previously described. We are able to synthesize a depth image using *multiple input patches* containing a color and a depth layer. We extended the image quilting technique (which is mono-patch) in order to consider the depth factor when choosing the blocks to paste into the result, so that both color and depth are coherent in the results. In addition, our technique uses multiple patches and the best color patch is chosen for each block using a pre-computed color metric. This allows us to avoid wrong color matches and to handle textures with various different features contrarily to current existing methods.

When rendering, geometric seams appear in the model due to the displacement of adjacent surfaces. This is a well-known problem in production. We developed a solution to this problem by stitching the two polygons and introducing a novel texturing technique to seamlessly cover the resulting stitch.

The generic system proposed is flexible as the patch-supplied information can be extended at will. The detail augmented representations obtained can be directly rendered using various existing/extended techniques and the resulting scenes have a more realistic look.

Our framework resembles the Image Analogies framework [HJO+01] described in section 2.4 as we use training data to propagate a global change, however our approach is more general for various reasons. First, we do not aim at handling only images but also geometric models. In their approach, a training image pair (A,A') is provided for the system to "learn" the transformation between A and A'. A new image B is submitted and the system computes a new image B', according to the analogy found in the training pair. To compute the new image and the analogies between pixels, they use a combination of two existing texture synthesis algorithms [WL00] [Ash01] and a perceptual similarity metric computing the sum of squared differences between two "feature vectors", as described in section 2.4.

Their training pairs corresponds to our patch object. However, in our approach, a patch object can have many different layers (while in theirs, there are only two in order to build the analogy). We also allow more than one training pair per example. In addition, their "learning" is done by searching for similar pixel neighborhoods in the various images using an L^2 distance. The learning we propose is much more involved as it is a procedure taking into account the various kinds of information present in the patch layers. For example, using the shape layer to constrain the depth synthesis implies using a shape detection algorithm to locate the given shape. We thus have extended the feature vectors to much more complex elements. In summary, our approach offers a generalization which allows to apply the detail recovery techniques to a wider class of images (and also models) using a generic training set rather than operating on images that are very similar to the training images. We plan to integrate more advanced detail analysis/synthesis algorithms.

In parallel to this work, Bhat *et al.* [BIT04] have developed an extension to the Image Analogies approach for 3D geometry (described in the "geometry" paragraph of section 2.2.1). The interest in extending Hertzmann *et al.* work is an indication of the promising potential of this approach. In addition, the combination of this framework for detail recovery with the techniques for cellular structure and appearance recovery presented in the first two chapters will allow the creation of 3D detailed cellular structures from the analysis of a sample photographs and some user-interaction. The generated models are suitable for interactive visualization as they only require a small set of appearance layers. The use of our patch-based approach is thus even more appropriate in this context.

Chapter 6

Conclusion

"Science never solves a problem without creating ten more."

—George Bernard Shaw

In this thesis, we have investigated how to define and create structure and detail (2D/3D) generators from the analysis of examples. Our work was motivated by the fact that there is no existing working solution to the problem of generating structured textures from example. Furthermore, by taking advantage of all the information contained in real-world photographs, we can increase the realism of computer-generated images by adding details at a reduced cost, while avoiding unpleasant repetitions.

In order to generate structured textures (mostly cellular textures with anisotropic cells), we have decoupled the geometric structure from the appearance (2D/3D) in the analysis/synthesis process. This novel idea offers a lot of flexibility for the generation and modifications of the results. Following this key principle, we have proposed three different procedural generators for three different purposes, which can be used separately or in combination.

We first proposed a method to analyze a given geometric structure (subdivision) in order to extract a *compact representation* that allows us to *statistically synthesize* similar subdivisions. This geometric structure generator is *controllable* through the use of constraints maps or by way of interactive edition. Our results show that we correctly recover the shape and region statistics of the input samples.

In order to texture the resulting 2D meshes, we presented a method to extract from the original image some *appearance data*, associated with the original mesh data. This appearance data includes the creation of *unified textures* which can be seen as a representation of the original material composing the cellular regions (e.g., the quarry for the stones of a wall) and contour texture atlases. The synthesis process creates polygons around the mesh elements (regions and their contours), and assign to their vertices texture coordinates indexing into the small set of unified textures contained in the appearance data. Our results show the flexibility to modify the appearance without affecting the structure and vice-versa.

Finally, to give even more realism to the generated structures, we also proposed solutions for the 3D detail recovery. We presented a framework to exploit information available in the models and textures in order to recover and propagate details on the whole scene, aided by user

intervention. Procedural generators have to be defined for each kind of detail and we presented our implementation of displacement map synthesis.

We next discuss the contributions of this thesis.

6.1 Contributions

Some possible applications for the generators we propose include, but are not limited to, the virtual reconstitution of edifices (such as the Castle in image 1.3), the synthesis of existing walls and pavements, the recovery of cellular animal skins (such as snake or giraffe skins), the design of new textures from the analysis of more than one sample, etc. In short, our novel method allows the *recovery of realistic detailed cellular textures* with shape-varying cells *from the analysis of a real-world sample*.

A lot of different texture kinds enter in the class of textures that we can handle. The requirements for these textures being that the individual regions are formed of a material which is almost uniform so that the texture extraction process leads to unified textures of good quality. The regions must also be closed and surrounded by a clearly-defined separating element so that we can robustly extract the contour information to build our contour atlases.

This core contribution can be decomposed into the following contributions, which can separately be used in other contexts.

- **Decoupling structure and appearance** We believe that one of the main causes of poor performance of textures synthesis algorithms on the structured textures we address is that they treat geometric structure and texture appearance together. We decoupled these two entities and presented our approaches to create *structure generators* and *appearance generators* from the analysis of real-world photographs. Their results can be combined to produce similar structured textures, or totally new ones, by mixing generators coming from different sources. Another advantage is the possibility to perform local changes independently without the need to recompute everything from scratch.

- **Creation of structure generators**

  We proposed a method to synthesize geometric subdivisions from the statistical analysis of an input subdivision. Our solution builds on and extends computational geometry techniques to handle *anisotropic primitives* and *user-defined constraints*. The generated textures are resolution independent and can adapt to a variety of target domains. The results show that the synthesized subdivisions exhibits the same shape statistics as the original.

- **Creation of appearance generators**

  We presented our techniques to generate a 2D appearance for a geometric subdivision using a small set of reusable texture components extracted from a structured texture, where each region contains approximately uniform material. We decoupled the texturing of the structure cells from that of their contour (or separating element) which allows us to combine region and contour materials coming from two different sources. Our approach offers a *significant gain of texture space* and interactive rendering speeds as there is no new information created during rendering, only textures coordinates indexing into the small set of common unified appearance textures created in a preprocess. These unified textures are reusable in other contexts as they are plain material textures and the separation of the texturing of the region interiors from their contours allows not to mix unrelated textures during synthesis.

- **Leaving control to the user**

  Our goal is not a totally automatic approach, since we want to permit some control by the artist at various stages. Our structure generation techniques allows the user to influence the synthesis process through the use of constraint maps and permits local modifications of the results. This lets the artist concentrate on the artistic aspect of the work, rather than on the more tedious shape creation and placement. In addition, modifications of the appearance data allow to vary the final visual results.

- **Texture synthesis**

  Current texture synthesis methods fail on structured textures, especially irregular structured textures such as the ones we are handling. Our method provides a solution in this area using our two-step synthesis process described previously. We plan to test our approach on the specific examples where texture synthesis methods currently fail.

- **Creation of unified textures**

  We proposed a pipeline for constructing uniform material textures out of the collection of a set of colored regions. Their color in the CIELAB space serves as a metric for a clustering into tint groups which allows to build a corresponding set of textures. See image 4.4.

- **Framework for detail generation**

  With the goal of increasing the realism of a scene, we present a detail recovery framework which should alleviate the tedious work of generating similar details throughout a scene. It mainly consists in defining a procedural method to exploit given texture or model information in order to recover and reintroduce a particular kind of detail, most often by creating an additional appearance layer. We show results for the synthesis of displacement maps. We also propose other possible appearance layer generators.

- **Displacement map synthesis**

  Our results on displacement map synthesis are satisfying, demonstrating that putting more advanced control metrics (or constraints) into texture synthesis algorithms can be a promising avenue of research in order to bring a solution to their common pitfalls.

Figure 6.1 shows an input textured structure along with a generated similar geometric subdivision and the final result after the appearance recovery process.

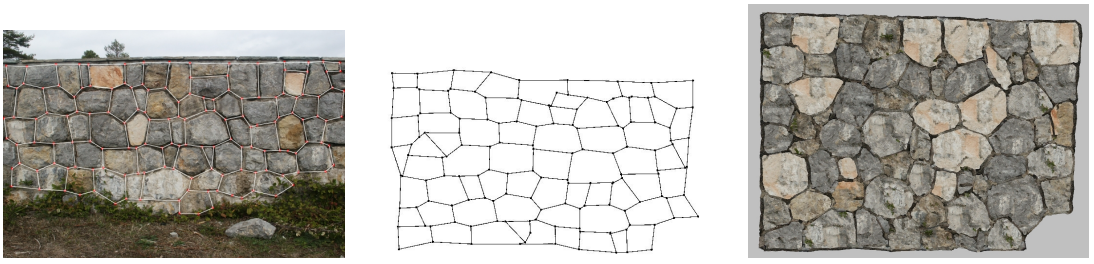

Figure 6.1 From an original structured texture photograph to a synthetic similar structured texture.

Some secondary contributions of our work which are worth mentioning include:

- A solution to the common problem of "cracks" in a model on which displacement mapping is applied. After stitching back the separated polygons, a new idea to seamlessly cover the seam has been presented.

- The computation of discrete Anisotropic Voronoi Diagrams and VDs of line segments, which, to our knowledge, has not been previously presented.
- The extension of Efros' [EF01] texture synthesis technique to use multiple input patches. A work parallel to ours presented a similar idea recently [TA04].

6.2 Improvements and future work

Our thesis opens a number of interesting avenues for improvements and future work. We first describe more immediate future work related to each of the generators we proposed and then expose some more global future research directions.

Structure generation from example

We could have devoted our entire thesis to the problem of generating geometric structures from example since the problem is far from trivial. We exposed a possible solution but there is a lot of potential for improvements and experimentations, the first ones being treating more examples to validate the generality of the method (such as those illustrated in figure 3.1).

Synthesis improvements Some improvements related to the synthesis process have been described at the end of chapter 3 such as the teleport operation. They should definitely be given a try.

Analysis data It would be interesting to explore the possibilities provided by the results of our analysis step.

Our distributions could be used for classification purposes. Using the analysis results of an input structure (e.g., a stone wall), we could compare them with the distributions issued from previous analysis, which have been classified, and determine if it pertains to a certain class of structures (e.g., walls/animal skins/others) or even subclasses (e.g., medieval walls, Provencal walls, etc.). This could prove useful for automatic texture database indexation and retrieval.

Another attractive idea we would like to try out is to allow the user to interact with the analysis statistical parameters (ellipse data histograms, mesh topology information) which would allow further experimentation. By restricting the samples to a certain distribution range or by modifying directly the parameters (e.g., forcing the almost vertical edges to become vertical), the user could generate a totally new set of structures, inspired more or less from the original. Mixing/blending distributions could also lead to interesting results. In the same line of thoughts, interpolating between distributions could allow to morph from one structure style to another.

The parameters for region contour texturing can also be exposed to allow for interactive tuning. The width of the created polygons along with the location of the intersection between interstice and junction polygons can make a visible difference on the result.

Constraints Our model allows constraints to be defined either in the form of maps, or interactively input by the user. We have tried a few constraints and already suggested other ones in section 3.7.2 such as size, orientation or metric map which could directly influence the two main steps of the Lloyd's algorithm: Either the computation of the Anisotropic Voronoi Diagram, either the modification of the generators.

Interactive system We wish we would have time to implement an interactive system to allow for modification and edition of results. This would require the optimization of our current region population/shape optimization stages. Many papers [KEHKL+99] [ST04] mention optimization possibilities for discrete Voronoi computations, even taking advantage of the hardware programming possibilities of current graphics cards. An user-friendly interface would have to be designed, and preferably integrated into a modeling system (e.g., XSI [Sof]). We discuss a more general proposal in the general future work section below.

Meta-structure We did not have the time to consider more advanced structured patterns, i.e., containing significant or artistic placement rules between the primitives. We focussed more on their shape and distributions than their placement relative to each other. However, in many structured textures, these local rules are an important feature which, if not considered, will not permit the generation of similar subdivisions (for an example of a structured texture with a meta-structure, see figure 6.2 (also called, in this case, a bi-modal texture)). Solving this would require a higher-level analysis of the meta-structure of our input subdivisions, for example analyzing correlations between similarly-shaped regions, in the spirit of Dischler's computation of co-occurences between his texture particles [JMKBD02].

Figure 6.2 Structured texture with an underlying meta-structure.

Appearance generation from example

To generate the appearance of our structured textures, we decoupled the treatment of the region interiors and their borders. The interstices and junctions edition is a manual process for which we have suggested some automation procedures in 4.2.4. Removing all the region color pixels by hand can be a tedious process, thus automatizing this part would ease the appearance analysis part.

We faced problems when generating appearance for geometric structures containing a lot of small edges (due to the polygonization process of the Discrete Voronoi Diagram regions obtained at the end of the region repartition step). In the case of a very small edge, the adjacent junction polygons might intersect and completely cover the edge associated interstice polygon and thus lead to problems during the matching phase (illustrated in figure 4.20). In addition, the same original interstice is always associated to the small edges, thus leading to annoying repetitions in case it is visible. Some more work should be done to correct this, potentially by directly matching two adjacent junctions and ignore the middle interstice.

More generally, we think there is room for improvement regarding the texturing of the region contours. We plan to investigate other ways to recover the contour appearance. A clear advantage of our method is that since we decoupled the texturing of the region interiors with that of their contours, we can easily try out new methods for one without affecting the other.

3D detail generation from example

We believe that extending our proposed framework for detail recovery using the layers proposed in section 5.5.1 can be very promising. We think it could alleviate the sometimes tedious process of detail creation and propagation.

An interesting approach to try out could be to connect the detail recovery to a learning process, giving examples of detail recovery until the process is acquired and repeated by the system. In general, we think introducing learning techniques in computer graphics approaches can offer many interesting directions for further research. The position paper of Hertzmann [Her03] gives interesting insights on the subject.

Future Research Directions

Each of the generators we presented in this thesis (structure, appearance, detail) can be used as a stand-alone or combined together to lead to the synthesis of detailed structured textures from example. For example, our structure generation techniques can be used as input for Miyata's [Miy90] [IMS03] [MIS01] or Legakis' [LDG01] 3D relief synthesis techniques where they randomly generate 3D displacements using noise functions to displace a fine mesh representing each generated cell. The structure would then come from the analysis of an example but the appearance would be completely synthetic. Conversely, we could plug our appearance synthesis process into the output of their structure generation techniques, or any other structure synthesis techniques (see section 2.3.2 for a review).

In the case generators are used in combination, we would like to experiment with other types of cellular structures such as aerial city views, where regions are blocks of buildings separated by roads (see image 6.3 for an example). This case is special as the underlying constraints (such as the macro-structure) are more complex than those previously seen. Indeed, the road network contains higher-level constraints, such as the continuity of roads and the coherence of the network, that should be preserved in order to generate plausible new versions of the city. A solution for this would be to input a constraint map containing the major roads and let the system fill in the rest of the space with regions from the input. Some authors [PM01] tried to generate cities procedurally and we can investigate the constraint maps they propose (population density map, elevation map, road map).

Application to 3D models

Our resulting structured textures (with or without 3D details) are, for the moment, generated in 2D planes. In order to apply them on 3D models, we either have to define a suitable parametrization so that the regions do not look distorted when mapped to an object or generate them directly onto the 3D models. In this last case, a good starting point would be the approaches used in texture synthesis or by Kunze *et al.* [RK97] who constructs geodesic Voronoi diagrams on parametric surfaces.

Another aspect to consider is that, for certain types of structures such as stone walls, the individual cells are inherently 3D. Thus, special care has to be taken when applying them to

Figure 6.3 Aerial city view of Paris.

3D models, especially around sharp corners. We need to avoid the "gift wrap" aspect most CG walls exhibit. Legakis *et al.* [LDG01] offer such possibilities, however for each model to texture, special pattern generators have to be written. We could extend our detail recovery technique by adding a layer specifying how to apply our generated cells to corner or edges, which could prove to be more flexible than Legakis *et al.* 's.

On the rendering side, in the same paper [LDG01], they propose an optimization method for rendering large volumes of data such as the ones we are faced with if we synthesize additional geometry for each individual cell. The approach takes advantage of geometric similarities between generated cells to use geometry instancing. Since we already have shape correspondences between generated and original regions, we could employ a similar approach.

Interactive framework

We would like to integrate the techniques presented in this thesis in an interactive structured texture analysis and synthesis framework. Since our goals are to relieve the artists from tedious work while providing as much control as possible on the artistic side of the work, we have to provide interactive tools incorporating our techniques. An artist should be able to interactively scale, rotate, delete, move the structure regions around in real-time, input constraints and get an interactive feedback, etc. This would require optimizing the discrete Lloyd's algorithm or try to implement an analytical solution. Most importantly, the system should facilitate any experimentation and correction. Ultimately, the development of the appropriate interaction paradigms and toolboxes, incorporating the techniques introduced in this thesis, will fully exploit the novel capabilities offered, and demonstrate the true extent of their utility.

Appendix A

Glossary

This glossary contains the words which are used throughout this thesis and whose definition can help understand the concepts exposed. It is not meant to be a precise dictionary but is rather meant to provide an intuitive explanation to avoid any possible confusion.

Alpha An extra Color channel to hold transparency information. An alpha value for a pixel represent the percent of the pixel covered by a given structure.

Appearance Manifestation of the nature of objects and materials through visual attributes such as size, shape, color, texture, glossiness, transparency, opacity, etc.

Artifacts/Artefact A classifiable visual error. This term does not originally have this meaning but it has been appropriated by computer graphics researchers.

Atlas See Texture atlas.

CG Computer generated.

CIElab A color space which uses three values to describe the precise three-dimensional location of a color inside a visible, almost perceptually uniform color space. In this space, Euclidean distances can be measured between color points. CIELAB color is a color space which encompasses RGB, CMYK, and describes generally the visible spectrum that the human eye can see.

CPU Central Processing Unit.

Generative models A generative model is a model for randomly generating observed data. It can be used to model observed draws from a probability density function.

GPU Graphics Processing Unit, more commonly called the graphics card where most of the computations related to graphics are made.

Mathematical morphology Mathematical Morphology is a tool for extracting image components that are useful for representation and description. Morphology can provide boundaries of objects, their skeletons, and their convex hulls. It is also useful for many pre- and post-processing techniques, especially in edge thinning and pruning. Generally speaking most morphological operations are based on simple expanding and shrinking operations. The primary application of morphology occurs in binary images, though it is also used on grey level images. (notes from Bob Fisher's CVOnline (http://homepages.inf.ed.ac.uk/rbf/CVonline/))

Mesh Net of interconnected polygons.

Parametrization Mapping of a texture texels onto a 3D model. The parametrization type (cylindrical, front, ...) specifies how texture coordinates will be derived. Texture coordinates determine how a texture is mapped onto a primitive.

Scan-fill An algorithm that handles an image one row at a time, e.g. performs its computation for pixels left-to-right as it scans across the image. After one row is generated, the algorithm proceeds to the next row.

Scene We call a *scene*, any computer-generated 3D scene, containing 3D virtual objects (also called *models*) represented using one of the various existing surface representation (polygons, points, implicit functions, image-based representations, ...).

T-junction A junction where three edges meet and form a "T".

Texture atlas A Texture Atlas is an efficient color representation for 3D Paint Systems. The model textures are packed onto one single textures and the rendering system indexes into this one texture at rendering.

Valency The number of edges meeting at a vertex.

Appendix B

Sweeping approach: Topological front

The tiling approach seen in chapter 3 to analyze and synthesize a structure from an example may not work well for structures exhibiting a layered organization of regular shapes, such as the one in image 3.26. For such structures, a *sweeping* technique may be more appropriate. It consists in sweeping the original plane to collect information on the subdivision (such as events happening) and sweeping the target plane to regenerate similar events.

We first examine the topology of a subdivision and the regions it is made of in order to extract characteristics which will allow us to characterize a given subdivision. Then, we address the geometrical aspects of the subdivision and particularly of its regions in order to define important variables to sample statistically. This collected topological and geometrical information constitutes analysis data which should allow us to generate a similar-looking subdivision during a synthesis phase.

The work described here reports on an exploratory study which we undertook during the thesis. Even though such an approach could be beneficial in some cases, we believe that the methodology we actually adopted (chapter 3), is better for several reasons, such as ease of implementation and robustness, ease of adaptation to different geometric forms, etc.

B.1 Topological aspects

B.1.1 On the topology of a region

A region R is a simple closed polygon composed of a certain number of edges E and vertices V. Given a unit vector $\vec{u}$, a vertex v_i is called a *critical point* along the direction $\vec{u}$ if the topology of the intersection between a sweep-line orthogonal to $\vec{u}$ (and R) changes. If the topology changes from the empty set to the vertex v_i and then to one or more line segments, the vertex v_i is said to be a *minimum*. It the topology changes from one or more line segments to the vertex v_i and then the empty set, the point is a *maximum*.

A vertex which is not a critical point is called a *regular point*. A regular point is always situated between a minimum and a maximum and can be encountered in two configurations that we term *left regular* and *right regular*. A regular point is left regular if the segment following the point is situated *to the left* of the line passing through the segment preceding the point (in the sweep direction). A point is right regular in the opposite case (see figure B.1 for an illustration of these cases).

It is known from Morse theory that any region bounded by a simple curve has at least two critical points, namely one maximum and one minimum whatever the direction $\vec{u}$. If it has more,

then the polygon is not convex (i.e., it is concave).

Figure B.1 shows the three types of points.

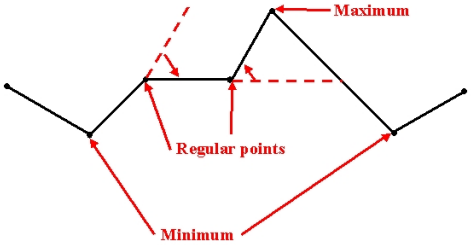

Figure B.1 Critical and regular points.

In the following, we analyze the topology of a region in terms of critical points for selected directions, namely horizontal and vertical directions.

We can decompose a region into a lower envelope and an upper envelope (see figure B.2). For convex polygons, the lower envelope is bounded by the minimum in the vertical direction v_{minv}, and the minimum and maximum in the horizontal direction v_{minh} and v_{maxh}, i.e., it contains all the edges between v_{minh} and v_{maxh} and passing through v_{minv}. The upper envelope is its complement. For concave polygons, we can use a similar definition but instead arbitrarily use the *first* minimum encountered in the vertical direction along with the *first* minimum and the *last* maximum in the horizontal direction.

The following relation holds: $u + l = n$, where n is the number of edges in a region, u is the number of edges of its upper envelope and l is the number of edges of its lower envelope.

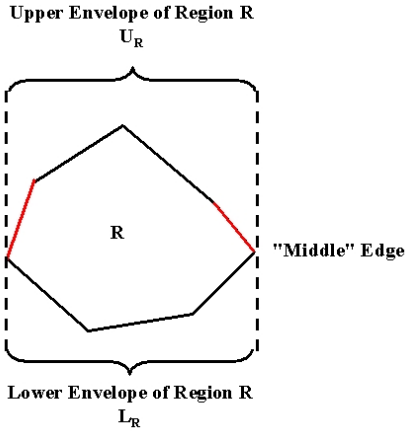

Figure B.2 A region decomposed into its upper and lower envelope.

B.1.2 On the topology of a subdivision

A subdivision is composed of three different types of elements: regions, vertices and edges. As for any planar complex, the Euler relation holds: $N_R + N_V = N_E + 1$, where N_R is the number of regions, N_V the number of vertices and N_E the number of edges. If some regions are not

closed (as can happen on the boundaries), $N_R + N_V <= N_E + 1$.

For a large number of subdivisions and especially for those we want to handle, we can traverse them along a particular direction by following a *polyline*. A polyline can be defined as a list of control points p_i connected by straight line segments (edges). Two successive polylines (going in the same direction) of a subdivision define a *layer* composed of a number of regions. The sequence of all successive layers in a subdivision defines the entire subdivision.

We can traverse a polyline by sweeping across the plane and collecting its edges and vertices. Without loss of generality, a polyline can be defined as running from left to right. It is *x-monotone*, meaning that any vertical line cross the polyline only once. When determining a polyline, we define the following rule: Starting from an edge to the left, follow the edges encountered at a vertex by always choosing as a next edge the one which is the farthest from the vertical direction (and oriented to the right) and does not cross or touch the previous polyline. Some backtracking rules must be established when walking along a polyline. A polyline main direction can be defined to be the general (or average) direction followed by the polyline edges.

As our subdivisions can be composed of irregular regions, we distinguish the three cases shown in figure B.3:

1. A region "protrudes" from the layer. We tag it as being a *cavity*. It pertains to the layer but the polyline circumvents the cavity. The cavity can be above or below the polyline main direction. See figure B.3 (a). To detect such a "cusp", we can follow each edge originating from a vertex along the polyline, tracing "paths" from this vertex. If the path reaches the preceding polyline, we abandon it. If the path reaches below the polyline we are building, we have a cusp. Moreover, defined in term of right turns R and left turns L, a cusp has the form RL^*R or LR^*L, where $*$ denotes a finite number of turns.

2. A region spans two layers. We consider it is part of both layers and can tag it as being a *double* region. Such a region can be identified by comparing the size of each new region encountered. This also happens when tracing the edge path and directly reaching a neighboring polyline (i.e., the neighboring layer). See figure B.3 (c).

3. The polyline has a sharp, step-like change. In this case, we tag the point as being a *junction point*. If the change makes the polyline go below its main direction, the next layer will start at this point. If it makes the polyline go above its main direction, the next layer will end at this point. See figure B.3(b).

Following these guidelines, we can separate a subdivision into its composing layers (and possibly cavities, double regions and junction points). Removing a layer reveals the layer below it and adding a layer is possible above the "interface" (i.e., the upper polyline) of the preceding one.

B.2 Geometrical aspects

B.2.1 On the geometry of a region

A region is defined by its topology but also by its geometry. Some important variables for a region are:

- Its number of vertices N_V.
- Its number of edges ($N_V - 1$).

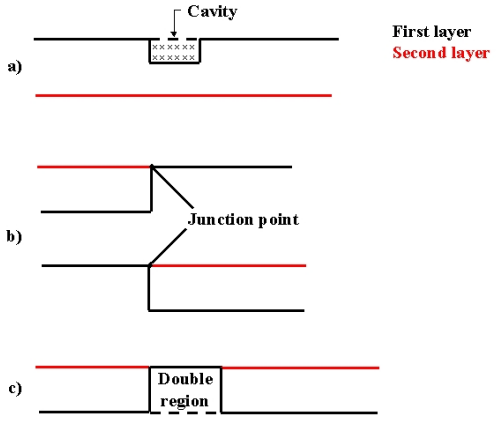

Figure B.3 Various polyline events.

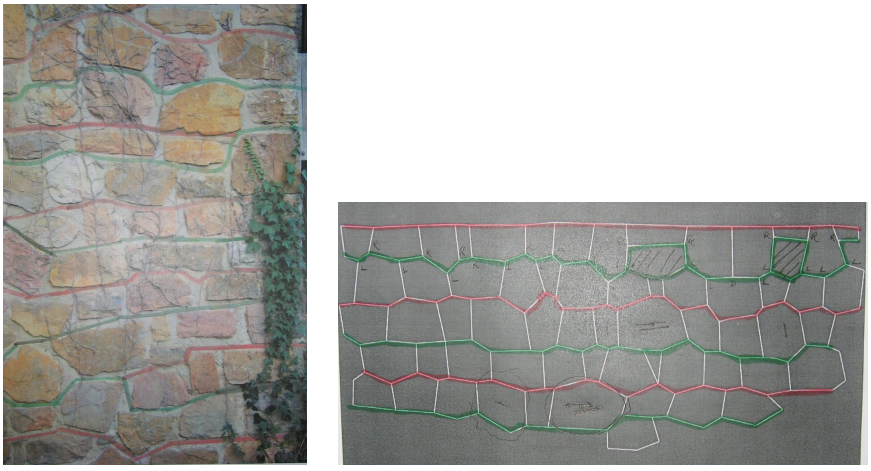

Figure B.4 Some polyline events on subdivisions extracted from a real-life examples.

- The length of its edges.
- The angle between its edges.
- The number of edges in each of its envelopes.

B.2.2 On the geometry of a subdivision

The subdivision itself also has a number of geometrical properties. The valency of each vertex (i.e., the number of edges incident to the vertex) is a very important property of the subdivision. For example, the valency of all Voronoi diagram vertices is always 3.

Along a polyline, the number of critical and regular points along with their frequency and regularity can be sampled. A *concavity* is defined by a right regular point whose valency is 2. Concavities along a polyline can also be counted.

B.3 Subdivision analysis

In order to analyze a subdivision, we have to cut the subdivision into polylines, and thus layers. As a first step, to create these polylines, we can identify them by hand. We can also manually initialize the process at the beginning of each identified polyline and let the system determine the rest of the polyline. As a rule, we can set the first layer to contain all the lowest opened regions (in the case there are opened regions) and then stack the other layers on top of it.

Two consecutive polylines frame a layer of regions. A polyline goes through parts of lower envelopes (for regions above) or parts of upper envelopes (for regions below). We call these parts the *upper* (resp. *lower*) *sectors*. Each are contained in the corresponding upper (resp. lower) envelope of the considered region. We call the edges that remain between two polylines the edges of the *middle sectors*. The sum of the edges of the three sectors is the number of edges in this region. While traversing the polylines, we detect the various cases enumerated in section B.1.2.

We collect the following variable statistics while traversing the subdivision (and thus the polylines):

- The length of edges on the polyline.
- The angle between edges. [1]
- The number of edges for the upper envelopes (and sectors) and their orientations.
- The number of edges for the lower envelopes (and sectors) and their orientations.
- The valency of points along a polyline.
- The number of cavities, concavities, double regions, along a polyline and their regularity.
- The number of edges per middle sector and their orientations.
- The orientation of the middle sectors relative to each other. This corresponds to the continuation event as described earlier.
- The diameter (or length) of the sectors, i.e., the distances between their two endpoints. This measure can be more complex, such as a shape criteria for sectors.

The shape criteria (such as the one used in section 3.5.4) for each region of the subdivision is also computed and stored in a distribution.

Analysis of a polyline A more thorough analysis of a polyline can be made to try to determine a regularity (if any) in the succession of vertices along the polyline (local minima, local maxima and regular points). This regularity can be exploited when generating a new polyline. By analyzing and reproducing the sequence of polyline events, we hope to get similar polylines. This analysis can help us determine a set of rules that the polylines should follow (for example, the fact that there should not be two concavities in a row in the same sector).

We can also imagine building a grammar with the different events which can occur along a polyline. Each token being a sector of the polyline, and a polyline constituting a phrase. Once we have determined a grammar, we hope to build new phrases with it. This idea has to be further explored.

[1] This might not be sufficient and we might have to consider the neighborhood of an edge to generate the following edges. See Curve Analogies [HOCS02]

B.4 Subdivision synthesis

The synthesis of a subdivision could be initialized by generating a polyline according to the polyline variables and event distributions: A sequence of edges whose relative angles follow the angle distributions. The rules determined during the analysis of the polyline (sequence of minima, maxima and regular points) should also be used. This polyline is then cut into sectors which correspond to lower sectors: we walk along the polyline, collecting edges until an acceptable lower sector is created (acceptable according to the lower sector distribution and/or the rules determined during the polyline analysis). This constitutes the beginning of a new region. We continue until the end of the polyline, thus leading to a cutting of the polyline into a certain numbers of regions to be completed. Figure B.5 shows such a generated first polyline. A cavity has been generated according to the cavity distribution.

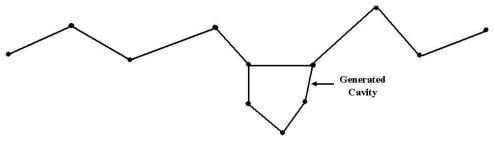

Figure B.5 First generated polyline: lower sector generation. A cavity has been generated in the process.

Then a series of phases alternate until the generation algorithm ends. We traverse the newly generated sectors:

1. For each sector endpoint, we draw a valency from the valency distribution.
2. We generate middle sectors starting from this endpoint. Each middle sector is generated by taking into account the generated middle sector to its left and the lower sectors it separates. The angle constraints between successive middle sectors must be respected. Most of the time, there will be 0 or 1 middle sector generated. See figure B.6.
3. We now have a front of opened regions that we must close. For each opened region, we generate an upper sector according to already generated sectors. The region must be coherent with those of the original subdivision. We use the region shape criteria and the distribution of upper sectors in order to complete the region. See figure B.7.
4. When all regions are closed, we have a new polyline. We can iterate again to generate another stripe of regions using this polyline or stop here.

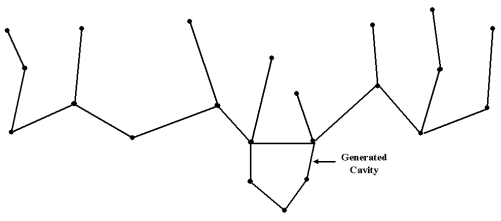

Figure B.6 Middle sector generation.

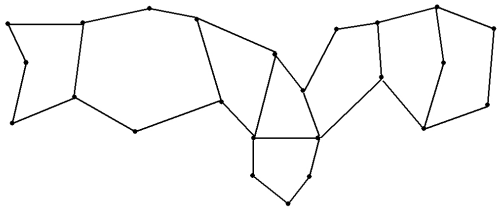

Figure B.7 Upper sector generation.

Cavities, double regions and junction points must be also generated in the process according to their distribution.

This method has not been implemented and further study should be made but we hope it could give an interesting research direction for the analysis and synthesis of layered structures.

Appendix C

Introduction (Version française)

"Lorsque tu ne sais pas où tu vas, regarde d'où tu viens."

—Proverbe africain

Pour la plupart d'entre nous, il y a 30 ans de cela, l'informatique graphique (infographie) n'avait pas grand chose à voir avec le réalisme et les détails. C'était plutôt la possibilité d'avoir autre chose que des lettres et des chiffres sur un écran et la faculté de pouvoir interagir graphiquement avec l'ordinateur. Nous ne nous lassions pas de jouer avec un des tout premiers jeux vidéo: *Pong* (cf. image C.1), alors qu'il ne consistait simplement qu'en deux barres se déplaçant de haut en bas et d'un disque traversant l'écran. De nos jours, la puissance informatique a tellement évolué que des jeux de tennis hautement réalistes mettant en scène des joueurs aux mouvements crédibles sont maintenant disponibles (voir image C.1). Plus important encore, nous *nous attendons* à de tels degrés de réalisme dans les jeux que nous utilisons, et voudrions toujours en avoir plus.

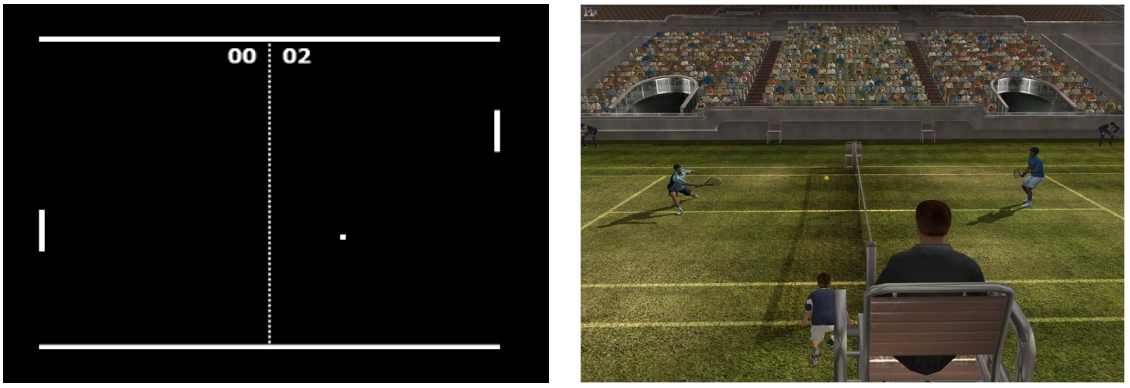

Figure C.1 Gauche: Le premier jeu vidéo de "tennis": Pong. Droite: Un des plus récents jeu vidéo de tennis (Image de TopSpin [Stu]).

Cette demande incessante de réalisme ne concerne pas seulement le marché des jeux vidéos mais aussi tous les secteurs dans lesquels les techniques d'infographie sont de plus en plus utilisées, tels que le cinéma ou la visualisation scientifique. Nous sommes de plus en plus habitués à visualiser des mondes 3D virtuels et sommes donc devenus très critiques par rapport à la qualité visuelle de ces mondes. Nous les voulons aussi réalistes, ou crédibles, que possible. Les chercheurs en infographie sont donc constamment en train d'investiguer de nouveaux algorithmes pour rendre les images générées par ordinateur presque indiscernables de photographies réelles. Pour certains types de scènes, cet objectif a été atteint. Par contre, pour d'autres, plusieurs

artefacts sont encore présents. La plupart du temps, la simplicité des modèles, le manque de détails et la répétition des structures, motifs, personnages et couleurs sont les défauts des scènes synthétiques qui nous dérangent le plus.

Nous considérons les *details* comme étant tout ce qui ajoute au réalisme d'une scène. Cela peut être de la couleur, de la géométrie complexe, des propriétés de réflectance des surfaces, des silhouettes, de l'information diversifiée, etc. Toute la complexité qui caractérise le monde réel. Dans cette thèse, nous voulons offrir des solutions pour augmenter le réalisme d'une scène en procurant certains aspects de cette complexité de détail par l'utilisation d'exemples provenant du monde réel, tout en sacrifiant aussi peu que possible l'espace mémoire disponible ou le temps d'exécution.

C.1 Motivation

Afin de répondre à l'augmentation permanente de besoins en réalisme, beaucoup d'efforts visent à l'amélioration des composants matériels et logiciels de la pipeline traditionnelle utilisée en infographie (modélisation, animation et rendu). Dû aux récentes avancées technologiques dans le domaine des unités centrales et des cartes graphiques, des augmentations de vitesse impressionnantes ont pu être atteintes pour la phase de rendu. Par contre, ce n'est pas le cas des outils de modélisation et d'animation. En effet, étant donné que la plupart des scènes graphiques sont faites à la main, les résultats finaux dépendent des habilités des artistes impliqués et des outils qu'ils utilisent. Selon le temps et le budget disponibles, ils utilisent des raccourcis lors de la création de ces scènes: Copiant et collant de modèles et de textures afin de les répéter, donnant l'illusion de détails en utilisant des textures sur des modèles simples, calculant à l'avance l'information d'ombrage dans des textures additionnelles, etc. En outre, l'utilisation escomptée de la scène ainsi que les limitations du matériel disponible peuvent aussi placer une restriction sur la taille de la scène (i.e., nombre de polygones, nombre de lumières, rendu des textures, nombre de propriétés de réflectance, quantité de calcul impliquée dans la création de chaque image) et ainsi sur les détails qu'elle peut contenir. Par exemple, une scène de jeu vidéo doit être beaucoup plus "légère" (en terme de demande en temps de calcul) qu'une scène de production cinématographique afin de pouvoir être affichée en un délai acceptable. Le temps de production et d'édition de scènes détaillées reste donc un goulot d'étranglement majeur.

Afin de simuler des détails géométriques manquants, les artistes utilisent une pléthore de méthodes. La manière traditionnelle adoptée consiste à appliquer à un modèle géométrique une série de *couches d'apparence*[1] dont la combinaison définit l'aspect final de l'objet. Chaque couche décrit des propriétés de surface (matériel) de l'objet, telles que la couleur, la réflectivité, le déplacement, la translucence, etc. Elle peut contenir des paramètres variés qui peuvent être ajustés indépendamment des autres couches. Cette notion d'indépendance est très importante pour les techniques traditionnelles de modélisation car elle permet l'ajustement individuel de chaque couche, évitant ainsi un possible impact sur les autres couches. Cependant, la création de ces couches n'est pas toujours tâche facile. Dans ce qui suit, nous décrivons trois problèmes relatifs à ces processus, que nous avons identifié et qui ont guidé le travail réalisé lors de cette thèse.

[1] En production, cette pile de couches est appelée un "shader" pour des raisons historiques, mais elle ne s'apparente maintenant plus exclusivement à l'ombrage.

Problème#1: Synthèse de texture structurée

La première couche et la plus commune ajoutée sur un modèle est la texture couleur. Une immense variété de modèles de texture existe, allant des texture générées de manière procédurale à celles provenant de photographies, que l'on appelle photoréalistes. Les textures sont la plupart du temps utilisées pour ajouter de la richesse visuelle sur un maillage géométrique plus simple. Cependant, elles souffrent de plusieurs problèmes en raison de leur taille fixe (pour les textures non procédurales) qui entraîne une résolution limitée, en plus des problèmes possibles de paramétrisation.

Deux grandes classes de textures sont particulièrement d'intérêt: les textures stochastiques et les textures structurées. Les textures stochastiques sont vues comme une réalisation d'un processus stochastique 2D tandis que les textures structurées correspondent à un arrangement de primitives respectant une règle de placement donnée. Des expériences ont montré[Mar82] que la vision humaine est très sensible aux caractéristiques de haut niveau (en particulier, leurs discontinuités) et plus tolérante à de petites déformations. Ainsi, des artefacts sur des textures structurées seront plus facilement discernables que sur des textures stochastiques. Les textures structurées sont souvent utilisées pour des objets d'arrière-plan (environnement) mais, malgré cela, l'observateur remarquera immédiatement d'éventuelles répétitions d'une texture de base, des problèmes de frontière à la jonction des morceaux de textures, des défauts de paramétrisation, etc. L'image C.2 illustre quelques-uns de ces problèmes communs.

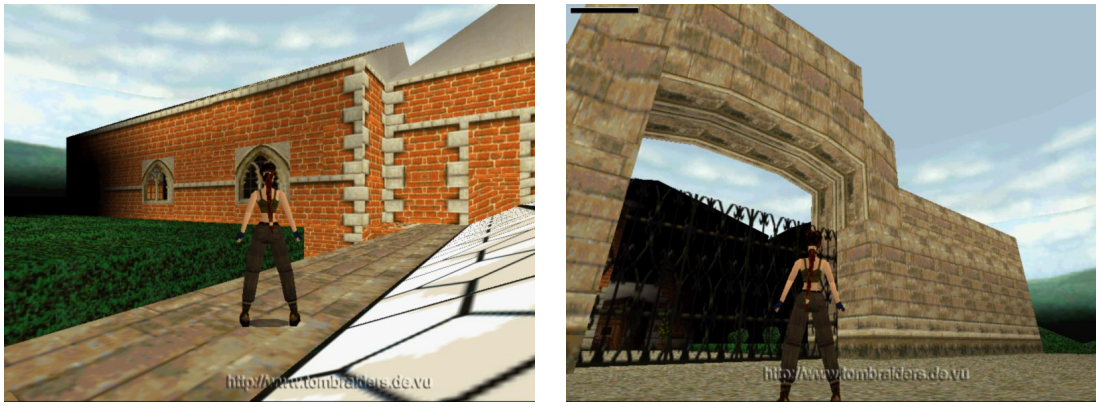

Figure C.2 L'apposition de texture structurées provoque souvent des artefacts gênants: texture mal appliquée, répétitions de structure, résolution limitée. (Image de [tom])

Idéalement, nous voudrions pouvoir générer de telles textures structurées sans leurs problèmes inhérents de jonction et de résolution. Ces textures peuvent aussi être appelées des "textures cellulaires" puisqu'elles sont composées de régions fermées, telles que les roches d'un mur de pierre ou les cellules d'une peau de serpent. Nous utiliserons le terme "structure" ou "texture structurée" dans la majeure partie de ce mémoire, à l'exception des cas pour lesquels l'appellation "texture cellulaire" facilite la compréhension. Pour reproduire de telles textures, un artiste modélisateur peut les dessiner et les agencer de manière répétitive mais ce processus peut rapidement devenir long et fastidieux. Il existe des générateurs de structures mais leurs résultats sont souvent imprévisibles et leur manque de contrôle les rends peu attractifs pour les artistes, qui ont besoin de pouvoir ajuster leur travail à volonté.

Une solution pratique serait de pouvoir prendre en entrée un exemple du résultat désiré et obtenir une procédure paramétrable capable de produire un résultat similaire. En effet, les textures extraites du monde réel constituent une excellente source d'inspiration pour la conception de textures virtuelles et nous devrions prendre avantage le plus possible de la qualité de leurs détails. La texture des murs du château de Pickering (image C.3) est un exemple d'une texture

structurée que nous pourrions vouloir reproduire afin d'obtenir une reconstitution virtuelle fidèle du château complété. Afin de synthétiser une texture similaire, une possibilité serait d'utiliser des algorithmes de synthèse de texture dont le but est de répliquer une texture à partir de l'analyse d'un échantillon. Par contre, même si les résultats de ces algorithmes sur des textures stochastiques sont satisfaisants, il n'existe pas encore de bonne solution pour les textures structurées.

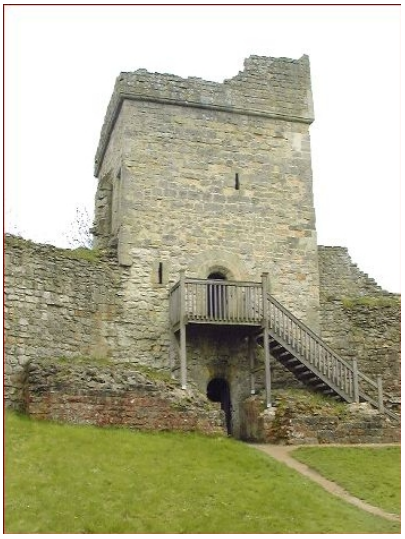

Figure C.3 Le château Pickering. Comment pouvons-nous le compléter virtuellement et répliquer la texture de mur existante de manière convaincante et non répétitive?

Ainsi, le premier problème que nous considérons dans cette thèse est le suivant: Comment pouvons-nous analyser une structure donnée (provenant dans la plupart des cas d'un exemple du monde réel) dans le but d'être capable de la propager sur diverses surfaces sans se confronter aux problèmes intrinsèques aux techniques de placage de textures? Comment pouvons-nous définir un tel "générateur de structure"? De plus, nous croyons fermement que les solutions complètement automatiques ôtent le travail artistique des mains des artistes. Nous devons les soulager de la partie fastidieuse de leur travail tout en les laissant influencer la génération de texture structurée de manière à la fois globale et locale. Les techniques de modifications locales permettent d'éviter d'avoir à régénérer une structure entière du début et facilitent la conception rapide et itérative.

Problème #2: Couplage de la Structure et de l'Apparence

Observez le village de la figure C.4 (droite), composé de plusieurs instances de la même maison, que l'on peut voir à gauche. L'utilisation de la même texture pour chaque maison donnera un aspect artificiel à la scène entière. Par contre, si nous arrivons à synthétiser pour chaque maison individiduelle une texture structurée de murs de pierre différente des autres, à l'aide de techniques de synthèse de texture telles que [EF01] ou [WL00] par exemple, la consommation de mémoire texture sera très élevée puisque nous ne pourrons pas dans ce cas partager des textures entre plusieurs objets. Les approches procédurales pourraient être une alternative à considérer mais elle sont réputées pour être difficiles à contrôler et à prédire.

Une observation importante est que pour des texture structurées telles que ces murs de pierre, la *structure*, c'est-à-dire la forme des pierres et leur placement, et l'*apparence*, c'est-à-dire la texture et les détails 3D (par exemple le relief) de chaque pierre et du mortier qui les unit entre elles peuvent (et doivent) être traitées indépendamment. Nous croyons qu'une des raisons pour laquelle les méthodes de synthèse de texture échouent sur de telles textures est parce qu'elles tentent de les traiter ensemble.

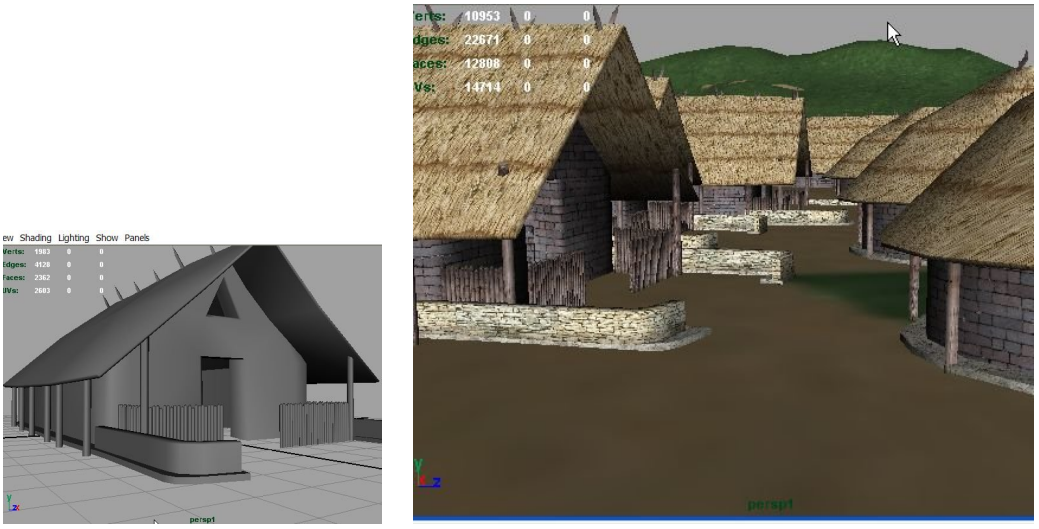

Figure C.4 Gauche: Une maison générée par ordinateur qu'il reste à texturer. Droite: Une village de maisons similaires dont les murs sont recouverts de textures structurées.

Problème #3: Synthèse de Détail 3D

Afin d'accroître la richesse visuelle des scènes 3D, des couches d'apparence plus avancées doivent être utilisées afin d'appliquer sur la surface de l'information concernant, par exemple, les normales de surface, la hauteur ou même l'éclairage, ceci afin d'éviter les problèmes de planarité de surface et de silhouettes rectilignes illustrés à la figure C.5. Ils permettent l'ajout d'information 3D discrète aux données de couleur et de géométrie sous-jacentes. Par contre, la construction de couches d'apparence comprenant des motifs similaires mais arrangés en divers endroits et orientations peut rapidement devenir très fastidieuse pour l'artiste, qui ne peut alors s'aider du traditionnel copier-coller. Il n'existe actuellement aucune méthode (semi) automatique pour le soulager de cette tâche.

Figure C.5 Gauche: Scène dans laquelle la planarité des textures structurées est évidente. Droite: Des couches d'apparence additionnelles ont été appliquées et le réalisme a augmenté.(Images de Oliveira [dON00])

Plus génériquement, la création de détail est un processus fastidieux. L'utilisation d'outils de modélisation traditionnels pour la construction de détails géométriques fins représente une quantité énorme de travail répétitif. Une solution peut être de prendre un échantillon de détail (par exemple, sous la forme d'une photo ou d'une mesure) et de l'utiliser pour améliorer un modèle approximatif bâti à la main en y ajoutant, générant ou propageant ce détail. Par contre, chaque

type de détail possède ses caractéristiques et ses particularités qu'il est nécessaire de comprendre si l'on veut pouvoir le régénérer De plus, l'information reliée au détail doit pouvoir être éditable et manipulable pour qu'une telle solution soit utilisable en pratique.

Ainsi, le troisième problème auquel nous nous attaquons consiste à déterminer des manières d'alléger le travail de l'usager pour la génération de détails petits mais néanmoins cruciaux pour le réalisme, soit en générant des couches d'apparence correspondant aux textures structurées synthétisées, soit, plus génériquement, en créant des générateurs d'information 3D par l'exemple.

En résumé, dans cette thèse, nous investiguons trois thèmes autour de l'idée d'exploiter l'information du monde réel afin d'accroître le réalisme dans les scènes 3D.

1. Comment synthétiser une structure à partir de l'analyse d'un échantillon de sorte à éviter les répétitions gênantes.
2. Comment générer une apparence réaliste pour cette structure en utilisant une description compacte de manière à pouvoir permettre un rendu temps-réel, tout en évitant l'utilisation de textures répétitives.
3. Comment propager l'information 3D à partir d'exemple pour soulager les artistes des tâches fastidieuses de création de détail.

Dans la section suivante, nous énoncerons les objectifs que nous nous sommes fixés pour attaquer les problèmes que nous venons de décrire et les approches que nous proposons pour les atteindre.

C.2 Objectifs et Approches Proposés

Dans ce travail, nous nous fixons le but principal d'améliorer le réalisme des scènes générées par ordinateur en procurant des solutions pour analyser l'information provenant du monde réel (structure, apparence, information 3D) et l'utiliser pour créer des "générateurs" procéduraux de couches d'apparence, ajustables par l'utilisateur, qui permettront d'enrichir efficacement les modèles des scènes. Ceci constitue un but à long terme pour lequel cette thèse apporte les premiers éléments.

Les deux premières parties de cette thèse se concentrent sur la synthèse de textures structurées basée sur l'analyse d'exemples du monde réel qui sont difficiles à reproduire en utilisant les techniques actuelles. Nous avons choisi de traiter ce problème particulier en découplant la structure de l'apparence, ce qui nous donnera beaucoup plus de flexibilité. La troisième partie décrit un système pour la récupération de l'information 3D par l'exemple et propose une application de cette approche aux textures structurées. Les techniques exposées sont appliquées à l'information de hauteur et généralisées à d'autres types de détails.

1. Génération de structure par l'exemple

La synthèse de structure à partir d'exemple est un problème difficile. Nous créons des générateurs de structure qui apportent une solution à ce problème, nous permettant de propager une texture structurée donnée de manière procédurale tout en permettant à l'usager d'influencer les résultats. Ce problème est géométrique de manière inhérente et nous le traiterons comme tel. Nous essayons de reproduire analytiquement une structure géométrique donnée représentée par un maillage 2D composé d'arêtes et de sommets. Notre objectif consiste à extraire une représentation compacte

de la structure à partir de l'analyse d'un exemple (échantillon de la structure). Cette représentation nous permet de répliquer la structure sur un domaine cible de forme et topologie diverses. L'usager est capable d'influencer la synthèse et de modifier le résultat sans avoir à tout régénérer.

L'approche que nous proposons consiste à évaluer l'anisotropie de chacune des régions composant la structure en l'approximant à l'aide d'une forme géométrique simple. Ceci nous permet de bâtir des statistiques simples auxquelles nous ajoutons de l'information reliée à la topologie de la structure. Ces données statistiques sont exploitées lors de la phase de synthèse afin de générer des nouvelles régions de formes similaires aux originales lors d'un processus en deux étapes. Dans un premier temps, les régions sont échantillonnées à partir des statistiques et vont progressivement peupler un domaine cible en utilisant une méthode de pavage. Plusieurs sortes de contraintes, entrées à l'aide de cartes de données par exemple, peuvent être prises en compte au cours de cette étape. Dans un second temps, la forme des régions est optimisée de façon à ressembler le plus possible aux formes originales. La structure générée est éditable, les régions peuvent être modifiées localement et la structure se met à jour globalement sans nécessiter une re-synthèse totale. Le processus est flexible; plusieurs extensions sont possibles et décrites dans les chapitres 3 et 6.

2. Génération d'apparence 2D par l'exemple

Comme énoncé plus tôt, lorsque nous analysons une texture structurée donnée, nous découplons l'analyse de la structure et celle de l'analyse de l'apparence étant donné que nous les considérons comme des entités indépendantes. Notre but principal lorsque nous analysons l'apparence d'une texture structurée est de créer un générateur d'apparence qui soit léger, c'est-à-dire qui requiert un petit nombre de textures et qui calculera le rendu du résultat de manière efficace. Il est à noter que nous nous occupons en premier lieu seulement de la texture couleur 2D, qui correspond à "l'apparence 2D", la première couche d'apparence qui est habituellement apposée sur un modèle 3D.

Puisque nous voulons produire un générateur léger, nous devons créer des composants de texture réutilisables qui peuvent être accédés plusieurs fois, de manière à utiliser aussi peu d'espace que possible en mémoire texture. Afin d'obtenir une apparence similaire à celle de l'exemple fourni, ces composants doivent être extraits de l'information de texture originale dans l'image, tout en faisant attention à ne pas dégrader les petits détails qui sont essentiels à la richesse d'une texture. Dans le but d'obtenir un rendu rapide, aucune nouvelle information de texture ne peut être créée au cours du rendu, même si la géométrie cible peut être différente de l'originale. Notre choix est donc de faire autant de pré-calculs que possible hors-ligne. De manière similaire à la génération de structure, l'utilisateur devrait être capable d'influencer le processus jusqu'à un certain point.

L'approche que nous proposons est inspirée par le principe de l'*instanciation* qui consiste à "recycler" les textures en indexant à plusieurs reprises dans la même image afin de texturer plusieurs modèles. Nous avons en entrée une image d'une texture structurée ainsi que la structure géométrique 2D correspondante, composée d'un ensemble de régions fermées. Au cours de la phase d'analyse, nous extrayons des éléments de texture de l'intérieur des régions et les utilisons afin de construire un ensemble de *textures unifiées*. Il est à noter que contrairement à l'instanciation traditionnelle de texture, chaque région (par exemple une pierre dans un mur) indexe dans une partie différente de cette texture unifiée, évitant ainsi les répétitions. Nous traitons différemment les contours des régions (par exemple le mortier dans un mur) en découpant des bouts de textures autour des sommets et des arêtes de la structure géométrique 2D et en

les rassemblant dans un atlas de textures. Nous conservons également les données de structure géométrique et les associons à l'information de texture extraite.

Les structures géométriques résultant du processus de synthèse précédemment décrit sont composées de sommets et d'arêtes autour desquels nous créons des polygones additionnels. Au cours de la phase de synthèse, nous associons à chacun d'eux une texture provenant d'un des atlas et échantillonons les textures unifiées pour attribuer une information de couleur pour texturer l'intérieur des régions. Une attention spéciale doit être portée à la jonction entre les morceaux de texture de contours qui doit être le plus indétectable possible. Nous avons décidé d'établir un compromis entre le nombre de polygones et le nombre de textures utilisées afin d'atteindre les requis d'espace et de rapidité que nous nous sommes fixés. Nous pouvons beaucoup plus nous permettre un nombre élevé de polygones qu'un nombre important de textures, particulièrement si ces textures sont de haute qualité. L'utilisateur est autorisé à modifier les textures partagées afin de créer de nouveaux résultats d'apparence. Les générateurs de structure peuvent aussi être combinés avec des générateurs d'apparence provenant d'exemples différents afin de synthétiser de toutes nouvelles textures structurées.

3. Génération de détails 3D par l'exemple

La dernière partie de cette thèse vise à récupérer de l'information 3D à partir d'images. Les détails géométriques fins, les propriétés de réflectance, etc... sont imprimés dans les textures et nous devrions profiter de cette information, guidés par les yeux experts d'un utilisateur. Ces données peuvent nous permettre de construire des couches d'apparence additionnelles afin d'ajouter plus de réalisme à nos objets. Notre but est de générer des détails *plausibles*, pas nécessairement des détails exacts, en utilisant l'information de texture et de modèle disponible ainsi que le guidage de l'utilisateur, dans le but ultime de rendre nos scènes plus crédibles. Nous aspirons à soulager l'utilisateur de tâches répétitives et fastidieuses telles que la création de couches d'apparence pour les texture structurées, tout en lui laissant le travail intentionnel, créatif.

Nous proposons donc un système générique de récupération de détail dans lequel l'information sur un détail désiré (géométrie, réflectance, etc.) peut être fourni par l'usager. Le système utilise cette information additionnelle en analysant le modèle à améliorer et génère les détails correspondants de manière appropriée. Des résultats spécifiques sur la récupération de cartes de déplacement sont décrits, et nous pouvons prendre avantage de toute information de structure ou de texture récupérée dans une étape précédente pour améliorer les résultats. Des idées et suggestions pour la récupération d'autres sortes de couches d'apparence sont exposés. En un sens, nous créons des générateurs de détails assistés par l'utilisateur. L'avantage principal de tels générateurs est qu'ils sont procéduraux, et donc peu coûteux en mémoire. Ils contiennent seulement une description de l'information manquante et la manière de la reconstruire selon les modèles/textures sous-jacents. L'interactivité et les entrées utilisateurs sont des fonctionnalités clefs de notre approche et nous sommes convaincus que seul un utilisateur humain peut procurer l'information requise pour une récupération de détail robuste, puisque les méthodes automatiques actuelles ne sont pas encore assez matures, et seul un utilisateur sait vraiment l'information qu'il désire récupérer.

C.3 Contributions

Nous résumons dans cette section les contributions principales de notre travail. Ces contributions, ainsi que quelques autres, sont décrites plus en détail dans les conclusions.

- **Synthèse de textures cellulaires détaillées à partir de l'analyse d'exemples**

  Il n'existe aucune méthode pour générer des textures structurées détaillées à partir de l'analyse d'exemples, notamment les textures cellulaires contenant des cellules de formes variées. Nous proposons des méthodes pour récupérer leur géométrie ainsi que leur apparence, en 2D et en 3D.

- **Découplage de la structure et de l'apparence**

  La structure et l'apparence sont souvent des entités indépendantes et devraient être traitées comme tel lors de l'analyse de textures structurées en vue de la création de générateurs qui permettent leur reproduction. Nous explorons cette nouvelle idée en décrivant nos techniques pour créer des *générateurs de structure* en parallèle à la création de *générateurs d'apparence*.

- **Création de générateurs de structure**

  Nous présentons une nouvelle technique procédurale pour générer des structures géométriques en utilisant des statistiques extraites de l'analyse d'une structure géométrique donnée en entrée.

- **Création de générateurs d'apparence**

  Nous avons conçu des techniques pour extraire des composants d'apparence réutilisables à partir d'une image de texture structurée, réalisant ainsi des gains mémoire significatifs. La texture composant l'intérieur des régions de la structure est découplée de la texture de leurs contours (ou matériau séparateur) de façon à ne pas mélanger des matériaux non reliés. Ceci offre la possibilité additionnelle de pouvoir les combiner avec d'autres provenant d'une texture structurée différente.

- **Système pour la génération de détail**

  Nous présentons un système conçu pour alléger le travail de création répétitive de détails en utilisant l'utilisateur et les informations de structure, modèle et texture disponibles.

- **Contrôle de l'utilisateur**

  Dans toutes les solutions présentées, nous avons laissé à l'utilisateur du contrôle sur les paramètres et les parties importantes du processus.

- **Synthèse de texture**

  Notre approche apporte une contribution à la recherche sur la synthèse de texture puisque les méthodes actuelles échouent sur les textures structurées. Nous ouvrons également des voies de réflexion sur les manières d'ajouter du contrôle utilisateur dans les techniques de synthèse de texture, car ce contrôle est primordial à l'obtention de résultats satisfaisants.

C.4 Organisation de la thèse

Ce document (en anglais, sauf pour l'introduction et la conclusion) est organisé comme suit:

Chapitre 2: Travaux reliés Ce chapitre décrit les techniques utilisées pour enrichir des modèles géométriques avec de l'information de détail, soit réelle, soit simulée, en utilisant des couches d'apparence. Il présente des techniques de l'état de l'art sur la synthèse de textures structurées, allant des approches géométriques aux approches cellulaires. Plusieurs méthodes pour propager des détails à partir d'exemple, telle que les techniques de synthèse de texture, sont passées en revue avant de donner un survol de travaux antérieurs se concentrant sur l'intégration de l'utilisateur dans le processus de création de détails.

Chapitre 3 Ce chapitre présente notre nouvelle approche pour l'analyse d'une structure géométrique 2D de façon à en extraire un ensemble de statistiques simples qui sont ensuite utilisées lors d'une phase de synthèse en deux étapes afin de générer des structures similaires. Nous décrivons la technique de pavage que nous utilisons, montrant ses avantages quant au contrôle qu'elle alloue durant le processus de synthèse, à travers l'utilisation de cartes de contraintes ou par manipulation interactive.

Chapitre 4 Ce chapitre décrit notre solution pour extraire des informations sur l'apparence d'une texture structurée de manière à générer et afficher rapidement l'apparence d'une autre structure donnée. La création de textures unifiées réutilisables ainsi que celle d'atlas de textures est expliquée en détail, ainsi que le mécanisme pour mélanger de manière uniforme tous les extraits de textures utilisés.

Chapitre 5 Ce chapitre présente notre système pour la propagation semi-automatique d'information de détail donnée par l'usager, à l'aide de l'information de structure, géométrie et texture disponible. Il montre des résultats pour la synthèse de cartes de déplacements et décrit des couches d'apparence additionnelles qui nous permettraient de récupérer d'autres sortes d'information 3D.

Conclusion Nous concluons la thèse en résumant nos contributions et discutons d'avenues de recherche possibles reliées à la synthèse de structure, d'apparence et de détail.

Annexe A: Glossaire (anglais) Les termes dont la définition est à préciser sont expliqués.

Annexe B: Approche par balayage Description succincte d'une approche géométrique pour la synthèse de structure ayant été explorée mais ne satisfaisant pas nos requis d'interactivité utilisateur. Néanmoins, le concept est intéressant.

Annexe C Ce chapitre.

Annexe D Version française du chapitre 6 (conclusion).

Appendix D

Conclusion (Version française)

"Science never solves a problem without creating ten more."

—George Bernard Shaw

Dans cette thèse, nous avons investigué la manière de définir et de créer des générateurs de structure et de détail (2D/3D) à partir de l'analyse d'exemples. Notre travail était motivé par le fait qu'il n'existe aujourd'hui aucune solution satisfaisante au problème de génération de textures structurées à partir d'exemple. De plus, en prenant avantage de toute l'information contenue dans des photographies du monde réel, nous pouvons augmenter le réalisme des images générées par ordinateur en ajoutant des détails à un coût réduit, tout en évitant les répétitions déplaisantes.

Afin de générer des textures structurées (pour la plupart des textures cellulaires comportant des cellules anisotropes), nous avons découplé la structure géométrique de l'apparence (2D/3D) lors du processus d'analyse/synthèse. Cette idée novatrice offre beaucoup de flexibilité pour la génération et la modification subséquente des résultats. Tout en suivant ce principe-clef, nous avons proposé trois générateurs procéduraux ayant trois finalités différentes et qui peuvent être utilisés séparément ou de manière combinée.

Nous avons tout d'abord proposé une méthode pour analyser une structure géométrique donnée (subdivision) de manière à en extraire une *représentation compacte* nous permettant de *générer de manière statistique* des subdivisions similaires. Ce générateur de structures géométriques est *contrôlable* à travers l'utilisation de cartes de contraintes ou par modifications interactives. Nos résultats montrent que nous récupérons correctement la forme et les statistiques des régions provenant des échantillons d'entrée.

Afin de texturer les maillages 2D résultants, nous avons présenté une méthode pour extraire de l'image originale de l'*information d'apparence* associée avec l'information du maillage original. Cette information d'apparence est constituée de *textures unifiées* que l'on peut voir comme une représentation du matériau original qui compose les régions cellulaire (par exemple, la carrière de laquelle ont été extraites les pierres d'un mur) ainsi que des atlas de texture pour les contours des régions. Le processus de synthèse crée des polygones autour des éléments du maillage (les régions et leurs contours) et leur assigne des coordonnées de textures indexant dans le petit ensemble de textures unifiées contenue dans l'information d'apparence. Nos résultats montrent le flexibilité de modifier l'apparence sans affecter la structure et vice-versa.

Enfin, afin d'ajouter encore plus de réalisme aux structures générées, nous avons aussi proposé des solutions pour la récupération de détail 3D. Nous avons présenté un système dont le but est d'exploiter l'information disponible dans les modèles et les textures d'une scène afin de récupérer et de propager des détails dans la scène entière, à l'aide d'information procurée par l'utilisateur. Des générateurs procéduraux doivent être définis pour chaque type de détail et nous avons présenté notre implémentation pour la synthèse de cartes de hauteurs.

Dans ce qui suit, nous discutons des contributions de notre thèse.

D.1 Contributions

Les applications possibles pour les générateurs que nous proposons incluent, mais ne sont pas limitées à, la reconstitution virtuelle d'édifices (tels que le château de l'image C.3), la synthèse de murs et pavages existants, le récupération de textures biologiques (telle que des peaux d'animaux ou des ensembles de cellules), la conception de nouvelles textures à partir de l'analyse de plusieurs échantillons, etc. En bref, notre nouvelle méthode permet la *récupération de textures cellulaire détaillées réalistes* avec des cellules de formes variées *à partir de l'analyse d'exemples du monde réel*.

Cette contribution principale peut être décomposée en plusieurs contributions que nous détaillons ici et qui peuvent être exploitées séparément dans d'autres contextes.

- **Découplage de la structure et de l'apparence** Nous croyons que l'une des causes majeures des piètres performances des algorithmes de synthèse de textures sur les textures structurées est dûe au fait qu'ils traitent la structure et la texture conjointement. Nous avons découplé ces deux entités et présenté notre approche pour créer des générateurs de structure et des *générateurs d'apparence* à partir de l'analyse de photographies du monde réel. Leurs résultats peuvent être combinés afin de produire des textures structurées similaires, ou bien des textures totalement nouvelles en mélangeant les générateurs provenant de l'analyse de sources différentes. Un autre avantage est la possibilité d'effectuer des changements locaux indépendamment sans avoir besoin de tout recalculer depuis le début.

- **Création de générateurs de structure**

  Nous avons proposé une méthode pour synthétiser des subdivisions géométriques à partir de l'analyse statistique d'une subdivision d'entrée. Notre solution se base et étend des techniques de géométrie algorithmique afin de traiter des *primitives anisotropes* et des *contraintes utilisateur*. Les textures générées sont indépendantes de la résolution et peuvent s'adapter à une variété de domaines cibles. Les résultats montrent que les subdivisions synthétisées exhibent les même statistiques de forme que les originales.

- **Création de générateurs d'apparence**

  Nous avons présenté nos techniques pour générer une apparence 2D pour une subdivision géométrique en utilisant un petit ensemble de composants de texture réutilisables extraits d'une texture structurée. Nous avons découplé la génération de texture pour les cellules de la structure de celle de leurs contours (ou élément séparateur) ce qui nous permet de combiner des matériaux de régions et de contours provenant de deux sources différentes. Notre approche offre un *gain significatif en espace texture* et permet des vitesses de rendu interactives puisqu'il n'y a pas de nouvelle information calculée au cours du rendu, seulement des coordonnées de texture indexant dans le petit ensemble de textures d'apparence partagées

qui ont été créées en précalcul. Ces textures unifiées sont réutilisables dans d'autres contextes puisqu'elles constituent des images de matériau uniforme et la séparation de la texturation des intérieurs de région de celle de leurs contours permet de ne pas mélanger des textures non reliées au cours de la synthèse.

- **Laisser du contrôle à l'utilisateur**

  Notre but n'est pas de créer une méthode complètement automatique puisque nous voulons permettre à l'artiste d'avoir du contrôle à plusieurs étapes du processus. Nos techniques de génération de structure permettent à l'usager d'influencer le processus de synthèse à travers l'utilisation de cartes de contraintes et d'effectuer des modifications locales sur les résultats obtenus. Ceci permet à l'usager de se concentrer sur l'aspect artistique de son travail plutôt que sur la tâche plus fastidieuse de création et de placement de formes. De plus, toute modification de l'information d'apparence permet de varier les résultats visuels finaux.

- **Synthèse de texture**

  Les méthodes de synthèse de texture actuelles échouent sur les textures structurées, en particulier les textures structurées irrégulières telles que celles que nous traitons. Notre méthode amène une solution dans ce domaine en utilisant notre processus de synthèse en deux étapes, décrit précédemment. Nous planifions tester notre approche sur les exemples pour lesquels les méthodes de synthèse de texture ont actuellement des difficultés.

- **Création de textures unifiées**

  Nous avons proposé une pipeline pour la construction de textures de matériau uniforme à partir d'une collection de régions colorées. La couleur moyenne de chaque région dans l'espace de couleur CIELAB sert de métrique pour un regroupement des régions par classes de teintes, ce qui nous permet ensuite de construire un ensemble correspondant d'images de textures. Voir l'image 4.4 pour un résumé du processus.

- **Système pour la génération de détail**

  Dans le but d'augmenter le réalisme d'une scène, nous présentons un système de récupération de détails qui permettrait d'alléger le travail fastidieux de génération de détails similaires à travers une scène. Cela consiste principalement à définir une méthode procédurale afin d'exploiter une texture ou un modèle donné de manière à récupérer et réintroduire un type de détail particulier, le plus souvent en créant une couche d'apparence additionnelle. Nous montrons des résultats pour la synthèse de cartes de hauteurs. Nous proposons également d'autres générateurs de couches d'apparence potentiels.

- **Synthèse de cartes de hauteurs**

  Nos résultats sur la synthèse de carte de hauteurs sont satisfaisants et démontrent qu'injecter des métriques de contrôle plus avancées (ou des contraintes) dans les algorithmes de synthèse de texture peut être une avenue de recherche prometteuse à l'apport d'une solution à leurs problèmes habituels.

La figure D.1 montre une texture structurée d'entrée avec une subdivision géométrique générée et le résultat final après la récupération de l'apparence.

Quelques contributions secondaires de notre travail qu'il est utile de mentionner incluent :

- Une solution au problème connu des "fissures" dans un modèle sur lequel une carte de déplacement est appliquée. Après avoir raccommodé ensemble les deux polygones séparés, une nouvelle idée pour texturer le raccord de manière invisible a été présentée.

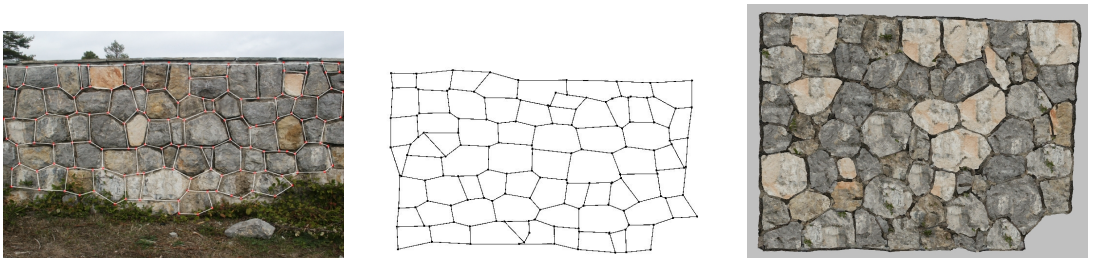

Figure D.1 D'une photographie de texture structurée à une texture structurée synthétique similaire.

- Le calcul des diagrammes de Voronoi Anisotropes et de DVs de segments qui, à notre connaissance, n'avait pas été précédemment présenté.
- Une extension à la technique de synthèse de texture d'Efros [EF01] afin d'utiliser plusieurs échantillons en entrée. Un travail parallèle au nôtre a présenté une idée similaire récemment [TA04].

D.2 Améliorations et Travaux Futurs

Notre thèse ouvre des perspectives intéressantes pour des améliorations ou des travaux futurs. Nous décrivons tout d'abord des travaux futurs à court terme par rapport à chacun des générateurs que nous avons proposé et exposons ensuite des directions de recherches futures plus globales.

Génération de structure à partir d'exemple

Nous aurions pu dédier notre thèse en entier au problème de générer des structures géométriques à partir d'exemple car le problème est loin d'être trivial. Nous avons exposé une solution possible mais il y a beaucoup de possibilités d'améliorations et d'expérimentations, les premières étant de traiter plus d'exemples afin de valider la généricité de notre approche (tels que ceux illustrés à la figure 3.1).

Amélioration de la synthèse Quelques améliorations possibles relatives au processus de synthèse ont été décrites à la fin du chapitre 3 tel que l'opération de téléportation par exemple. Il faudrait certainement les essayer.

Utilisation des données de l'analyse Il serait intéressant d'explorer les possibilités procurées par les résultats de notre étape d'analyse.

Nos distributions pourraient être utilisées dans des buts de classification. À l'aide des résultats de l'analyse d'une structure d'entrée (par exemple un mur de pierre), nous pourrions les comparer avec les distributions provenant d'analyses antérieures, ayant été classifiées, et déterminer si elle appartient à une certaine classes de structures (par exemple des murs/tissus biologiques/autres) ou encore à des sous-classes (par exemple des murs médiévaux, provençaux, etc.). Ceci pourrait s'avérer utile pour l'indexation et le recherche automatique dans une base de données de textures.

Une autre idée séduisante serait de permettre à l'utilisateur d'interagir avec les paramètres de l'analyse statistique (histogramme des données des ellipses, information de topologie du maillage) ce qui permettrait plus de possibilités d'expérimentations. En restreignant les échantillonnages à un certain intervalle dans les distributions ou en modifiant directement les paramètres (par exemple, en forçant les arêtes presque verticales à devenir complètement verticales), l'utilisateur

pourrait générer un tout nouvel ensemble de résultats de structures, plus ou moins inspirées de l'original. Effectuer un mélange/transition au niveau des distributions pourrait aussi mener à des résultats intéressants. Dans la même ligne d'idées, interpoler entre des distributions pourrait nous permettre de transformer progressivement (morphing) une structure en une autre.

Les paramètres pour la texturation des contours de régions pourraient également être exposés pour permettre un ajustement interactif. La modification de la largeur des polygones additionnels créés de même que l'emplacement de l'intersection entre les polygones des interstices et des jonctions peut produire une différence visible sur le résultat final.

Contraintes Notre modèle permet de définir des contraintes soit sous la formes de cartes, soit entrées directement par l'utilisateur. Nous avons essayé quelques contraintes et déjà suggéré d'autres types dans la section 3.7.2 telles que des cartes de contraintes sur la taille, l'orientation ou la métrique qui pourraient directement influencer les deux étapes principales de l'algorithme de Lloyd : Soit le calcul du Diagramme de Voronoi Anisotrope, soit la modification des générateurs.

Système interactif Nous aurions aimé avoir le temps d'implémenter un système interactif pour permettre la modification et l'édition des résultats. Ceci nécessiterait l'optimisation des étapes actuelles de répartition de région et d'optimisation de formes. Plusieurs articles [KEHKL+99] [ST04] mentionnent des possibilités d'optimisation pour les calculs de Voronoi discrets, qui peuvent même parfois profiter des possibilités des cartes graphiques actuelles. Une interface utilisateur efficace serait à concevoir, et de préférence intégrée dans un système de modélisation existant (par exemple XSI [Sof]). Nous exposons une proposition plus générale à ce sujet dans la section de travaux futurs plus globaux plus bas.

Méta-structure Nous n'avons pas eu le temps de considérer des motifs de structures plus complexes, i.e. qui contiennent des règles de placement entre les primitives plus établies ou plus artistiques. Nous nous sommes plutôt concentrés sur leur forme et leurs distributions que sur leur placement les unes par rapport aux autres. Néanmoins, dans beaucoup de textures structurées, ces règles locales peuvent être un attribut important qui, s'il n'est pas considéré, ne pourra permettre de générer des subdivisions similaires (pour un exemple de texture structurée avec une méta-structure, voir la figure D.2 (aussi appelée, dans ce cas particulier, une texture bi-modale)). La résolution de ce problème impliquerait une analyse de plus haut niveau de la méta-structure de nos subdivisions d'entrée, par exemple en analysant les corrélations entre des régions de formes similaires, dans le même esprit que le calcul des co-occurrences de Dischler entre ses particules de textures [JMKBD02].

Figure D.2 Texture structure avec une méta-structure sous-jacente.

Génération d'apparence à partir d'exemple

Afin de générer l'apparence de nos textures structurées, nous avons découplé le traitement des intérieurs des régions de celui de leurs contours. L'édition des interstices et des jonctions est un processus manuel pour lequel nous avons suggéré des méthodes d'automatisation dans la section 4.2.4. L'enlèvement à la main de tous les pixels de couleur appartenant à l'intérieur des régions peut être un processus très fastidieux, ainsi automatiser cette partie devrait faciliter le processus d'analyse d'apparence.

Nous avons rencontré des problèmes lors de la génération de structures géométriques contenant beaucoup de petites arêtes (dues au processus de polygonisation des régions du Diagramme de Voronoi Discret provenant de la phase de répartition de régions). Dans le cas d'une arête très courte, les polygones des jonctions adjacentes peuvent s'intersecter et recouvrir complètement le polygone associé à l'arête en question et ainsi mener à des problèmes au cours de la phase de mise en correspondance des polygones (telle qu'illustrée à la figure 4.20). De plus, le même interstice original est toujours associé aux petites arêtes, menant ainsi à d'ennuyeuses répétitions. Du travail additionnel devrait être effectué afin de corriger ce problème, éventuellement en mettant directement en correspondance deux jonctions adjacentes et en ignorant l'interstice entre les deux.

Plus généralement, nous pensons qu'il y a place à l'amélioration en ce qui concerne la texturation des contours de région. Nous prévoyons investiguer d'autres méthodes pour récupérer l'apparence des contours. Un avantage certain de notre méthode est que puisque nous avons découplé la texturation des intérieurs de région de celle de leurs contours, nous pouvons aisément essayer de nouvelles méthodes sans affecter les résultats de l'autre.

Génération de détails 3D par l'exemple

Nous pensons que l'extension de notre système de récupération de détail en utilisant les couches proposées dans la section 5.5.1 peut être une avenue très prometteuse. Nous considérons que cela pourrait alléger le processus souvent fastidieux de création et de propagation de détail.

Une approche intéressante à essayer pourrait être de relier la récupération de détail à un processus d'apprentissage en donnant des exemples de récupération de détail jusqu'à ce que le processus soit acquis et répété par le système. En général, nous pensons que l'introduction de techniques d'apprentissage dans des approches de synthèse d'images peut offrir beaucoup de directions de recherche futures intéressantes. Le papier sur le sujet de Hertzmann [Her03] contient des réflexions intéressantes sur le sujet.

Futures directions de recherche

Chacun des générateurs présentés dans cette thèse (structure, apparence, détail) peut être utilisé de manière autonome ou combiné avec un autre pour mener à la synthèse de textures structurées détaillées à partir d'exemple. Par exemple, nos techniques de génération de structure peuvent être utilisées comme donnée d'entrée pour les méthodes de synthèse de relief 3D de Miyata [Miy90] [IMS03] [MIS01] ou de Legakis [LDG01] dans lesquelles ils génèrent aléatoirement des hauteurs 3D en utilisant des fonctions de bruit afin de déplacer un maillage très fin représentant chaque cellule. La structure proviendrait donc de l'analyse d'un exemple mais l'apparence serait complètement synthétique. Inversement, nous pourrions connecter notre processus de synthèse d'apparence à la sortie de leurs techniques de génération de structure, ou d'autres techniques de synthèse de structure (voir la section 2.3.2 pour une revue de ces techniques).

Dans le cas où les générateurs sont combinés, nous aimerions expérimenter avec d'autres types de structures cellulaires comme des vues aériennes de villes par exemple, dans lesquelles les régions sont les blocs de bâtimens séparés par des routes (voir l'image D.3 pour un exemple). Ce cas est spécial puisque les contraintes sous-jacentes (telle que la macro-structure) sont plus complexes que dans les exemples traités précédemment. En effet, le réseau routier contient des contraintes de haut niveau, telles que la continuité des routes et la cohérence du réseau, qui se doit d'être préservé si l'on veut pouvoir générer de nouvelles versions plausibles de la ville. Une solution à ceci pourrait être de fournir une carte de contraintes contenant les routes principales et laisser le système remplir le reste de l'espace avec des régions de l'entrée. Certains auteurs [PM01] ont essayé de générer des villes de manière procédurale et nous pourrions investiguer les cartes de contraintes qu'ils proposent (carte de densité de population, carte d'élévation, carte de routes).

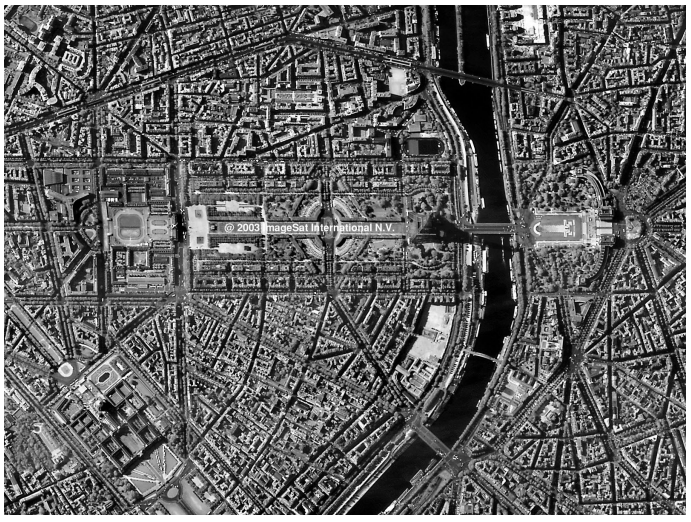

Figure D.3 Vue aérienne de la ville de Paris.

Application à des modèles 3D

Nos textures structurées résultantes (avec ou sans détails 3D) sont, pour le moment, générées sur des plans 2D. De façon à pouvoir les apposer sur des modèles 3D, nous devons soit définir une paramétrisation appropriée de manière à ce que les régions n'aient pas l'air trop distordues lorsque appliquées sur un objet, soit les générer directement sur des modèles 3D. Dans ce dernier cas de figure, un bon point de départ serait les approches utilisées en synthèse de texture ou par Kunze *et al.* [RK97] qui construisent des diagrammes de Voronoi géodésiques sur des surfaces paramétriques.

Un autre aspect à considérer est que, pour certains types de structures telles que des murs de pierre, les cellules individuelles sont 3D de manière inhérentes. Ainsi, une attention spéciale doit être portée lorsqu'on les applique sur des modèles 3D, particulièrement autour des arêtes vives du modèle. Nous devons éviter l'aspect "emballage cadeau" que la plupart des murs en image de synthèse exhibent. Legakis *et al.* [LDG01] offrent de telles possibilités mais il faut écrire un générateur de motifs distinct pour chaque modèle à texturer. Nous pourrions étendre notre méthode de récupération de détail en ajoutant une couche spécifiant la manière d'appliquer nos cellules générées aux coins et arêtes du modèles, ce qui permettrait d'être plus flexible que Legakis *et al.* .

Au niveau du rendu, dans ce même article [LDG01], les auteurs proposent une idée d'optimisation pour le rendu de grandes quantités de données telles que celles que nous aurions à considérer si nous synthétisions une géométrie additionnelle pour chaque cellule individuelle. L'approche profite de similarités géométriques entre les cellules générées afin de pouvoir utiliser de l'instanciation géométrique. Puisque nous connaissons déjà les correspondances de forme entre les régions générées et les originales, nous pourrions employer une approche similaire.

Système interactif

Nous aimerions intégrer les techniques présentées dans cette thèse au sein d'un système interactif d'analyse et de synthèse de textures structurées. Puisque nos buts sont de soulager les artistes des tâches fastidieuses tout en leur laissant autant de contrôle que possible sur l'aspect artistique de leur travail, nous devons leur procurer des outils interactifs incorporant nos techniques. Un artiste devrait être capable de tourner, changer d'échelle, supprimer, déplacer les régions de la structure en temps réel, entrer des contraintes et obtenir un retour interactif, etc. Ceci impliquerait l'optimisation de l'algorithme de Lloyd discret ou la conception d'une solution analytique. Encore plus important, le système devrait faciliter toute expérimentation et correction désirée. Ultimement, le développement des paradigmes d'interactions et boîtes à outils appropriés, incorporant les techniques introduites dans cette thèse, devraient permettre d'exploiter pleinement les capacités novatrices offertes et de démontrer l'étendue de leur utilité.

Bibliography

[ACH+91] Esther M. Arkin, L. Paul Chew, Huttenlocher Huttenlocher, Klara Kedem, and Mitchell Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(3):209–216, March 1991.

[all] Allegorithmic sa. http://www.allegorithmic.com.

[All04] Pierre Alliez. Personal communication, 2004.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[Ash01] Michael Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[BCD03] Stephen Brooks, Marc Cardle, and Neil A. Dodgson. Enhanced texture editing using self similarity. In *Vision Video and Graphics*, pages 231–238, 2003.

[BD02] Stephen Brooks and Neil Dodgson. Self-similarity based texture editing. In Stephen Spencer, editor, *Proceedings of the 29th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-02)*, volume 21, 3 of *ACM Transactions on Graphics (TOG)*, pages 653–656, New York, July 21–25 2002. ACM Press.

[Bee01] Nelson H. F. Beebe. GNU Scientific Library (GSL). http://www.math.utah.edu/software/gsl.html, 2001.

[BH96] Frank Bossen and Paul Heckbert. A pliant method for anisotropic mesh generation. In *5th International Meshing Roundtable*, October 1996.

[BIT04] Pravin Bhat, Stephen Ingram, and Greg Turk. Geometric texture synthesis by example. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 41–44, New York, NY, USA, 2004. ACM Press.

[Bli78a] J. F. Blinn. A scan line algorithm for displaying parametrically defined surfaces. *Computer Graphics*, 12(3):27–27, August 1978.

[Bli78b] James F. Blinn. Simulation of wrinkled surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 286–292. ACM Press, 1978.

[BMR01] Fausto Bernardini, Ioana M. Martin, and Holly Rushmeier. High-quality texture reconstruction from multiple scans. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(4), pages 318–332. IEEE Computer Society, 2001.

[BN76] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, 1976.

[Bon97] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Bro89] Michael J. Brooks. *Shape from shading*. MIT Press, Cambridge, MA, USA, 1989.

[Cat74] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974.

[CCC87] Robert L. Cook, Loren Carpenter, and Edwin Catmull. The reyes image rendering architecture. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 95–102. ACM Press, 1987.

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[Coo84] Robert L. Cook. Shade trees. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 223–231, July 1984. Published as Computer Graphics (SIGGRAPH '84 Proceedings), volume 18, number 3.

[CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Transactions on Graphics (TOG)*, 23(3):905–914, 2004.

[DDSD03] Xavier Décoret, Frédo Durand, François Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In *Proceedings of the ACM Siggraph*. ACM Press, 2003.

[DF81] Persi Diaconis and David Freedman. On the histogram as a density estimator: L_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandt Gebiete*, 57:453–476, 1981.

[DFG99] Qiang Du, Vance Faber, and Max Gunzburger. Centroidal voronoi tessellations: Applications and algorithms. *SIAM Review*, 41(4):637–676, December 1999.

[DG99] Jean-Michel Dischler and Djamchid Ghazanfarpour. Interactive image-based modeling of macrostructured textures. *IEEE Computer Graphics and Applications*, 19(1):66–74, 1999.

[DKS01] Michael Doggett, Anders Kugler, and Wolfgang Straßer. Displacement mapping using scan conversion hardware architectures. *Computer Graphics Forum*, 20(1), March 2001.

[DMG02] Jean-Michel Dischler, Karl Maritaud, and Djamchid Ghazanfarpour. Coherent Bump Map Recovery from a Single Texture Image. In *Proc. Graphics Interface*, pages 201–208, May 2002.

[dON00] Manuel Menezes de Oliveira Neto. *Relief texture mapping*. PhD thesis, University of North Carolina at Chapel Hill, 2000.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[DvGNK99] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics (TOG)*, 18(1):1–34, January 1999.

[DvNK99] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar, and Jan J. Koenderink. Reflectance and texture of real-world surfaces. *ACM Transactions on Graphics (TOG)*, 18(1):1–34, 1999.

[DWS+88] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 21–30, New York, NY, USA, 1988. ACM Press.

[DYB98] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 105–116. Springer-Verlag Wien New York, 1998.

[EF01] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.

[eGZW01] Cheng en Guo, Song Chun Zhu, and Ying-Nian Wu. Visual learning by integrating descriptive and generative methods. In *Proceedings of the Eighth International Conference On Computer Vision (ICCV-01)*, pages 370–377, Los Alamitos, CA, July 9–12 2001. IEEE Computer Society.

[EL99] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *ICCV (2)*, pages 1033–1038, 1999.

[EMP+94] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Worley. *Texturing and Modeling: A Procedural Approach*. Academic Press, October 1994. ISBN 0-12-228760-6.

[FJP02] William T. Freeman, Thouis R. Jones, and Egon C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, March/April 2002.

[FLCB95] Kurt Fleischer, David Laidlaw, Bena Currin, and Alan Barr. Cellular texture generation. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 239–248. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[FPT81] Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981.

[FS93] Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 247–254, New York, NY, USA, 1993. ACM Press.

[geo] Geometry lab. http://www.geometrylab.de/ConvexDistFkt/ConvexVoronoi.html.en.

[GSX00] Baining Guo, Harry Shum, and Ying-Qing Xu. Chaos mosaic: Fast and memory efficient texture synthesis. Technical report, 2000.

[Hae90] Paul Haeberli. Paint by numbers: abstract image representations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 207–214, New York, NY, USA, 1990. ACM Press.

[Hau01] Alejo Hausner. Simulating decorative mosaics. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 573–580. ACM Press, 2001.

[HB95] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 229–238, New York, NY, USA, 1995. ACM Press.

[Hec86] Paul S Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, 1986.

[Hec89] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, 1989.

[Her03] Aaron Hertzmann. Machine learning for computer graphics: A manifesto and tutorial. In *Proceedings of Pacific Graphics 2003. Invited Paper*, pages 22–36, October 2003.

[HH90] Pat Hanrahan and Paul Haeberli. Direct wysiwyg painting and texturing on 3d shapes. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 215–223. ACM Press, 1990.

[HJO+01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In ACM, editor, *SIGGRAPH 2001 Conference Proceedings, August 12–17, 2001, Los Angeles, CA*, pages 327–340, New York, NY 10036, USA, 2001. ACM Press.

[HL90] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1990. ACM Press.

[HOCS02] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In Simon Gibson and Paul Debevec, editors, *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 233–246, Aire-la-Ville, Switzerland, June 26–28 2002. Eurographics Association.

[Hop96] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108, August 1996.

[HS93] Paul Haeberli and Mark Segal. Texture mapping as a fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering*, pages 259–266. Eurographics, June 1993.

[IBG03] Ryan M. Ismert, Kavita Bala, and Donald P. Greenberg. Detail synthesis for image-based texturing. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 171–175. ACM Press, 2003.

[IMS03] Takayuki Itoh, Kazunori Miyata, and Kenji Shimada. Generating organic textures with controlled anisotropy and directionality. *IEEE Computer Graphics and Applications*, 23(3):38–45, May/June 2003.

[Ize91] Alan Julian Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413), March 1991.

[JMKBD02] Dischler Jean-Michel, Maritaud Karl, Levy Bruno, and Chazanfarpour Djamchid. Texture particles. In *Eurographics 2002 - EG 2002, Saarbrucken, Germany*, Sep 2002.

[KEHKL+99] III Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. pages 277–286, 1999.

[KK89] J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 271–280. ACM Press, 1989.

[KP99] R. Kothari and D. Pitts. On finding the number of clusters. *Pattern Recognition Letters*, 20(4):405–416, April 1999.

[KSE+03] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics (TOG)*, 22(3):277–286, 2003.

[LDG01] Justin Legakis, Julie Dorsey, and Steven Gortler. Feature-based cellular texturing for architectural models. pages 309–316, 2001.

[LLH04] Yanxi Liu, Wen-Chieh Lin, and James Hays. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics (TOG)*, 23(3):368–376, 2004.

[Llo82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-28(2):129–137, March 1982.

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Gu, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. In Jessica Hodgins, editor, *ACM Transactions on Graphics (TOG)*, volume 20(3), pages 127–150. ACM Press, 2001.

[LN02] Sylvain Lefebvre and Fabrice Neyret. Synthesizing bark. In *13th Eurographics Workshop on Rendering*, 2002.

[LP00a] Laurent Lefebvre and Pierre Poulin. Analysis and synthesis of structural textures. In *Proceedings of the Graphics Interface 2000*, pages 77–86, Toronto, Ontario, May 15–17 2000. Canadian Information Processing Society.

[LP00b] Laurent Lefebvre and Pierre Poulin. Extraction and synthesis of bump maps from photograph. In *Graphics Interface Posters Proceedings*, pages 12–13, May 2000.

[LS03] Francois Labelle and Jonathan Richard Shewchuk. Anisotropic voronoi diagrams and guaranteed-quality anisotropic mesh generation. In *Proceedings of the nineteenth Conference on Computational Geometry (SCG-03)*, pages 191–200, New York, June 8–10 2003. ACM Press.

[Ma00] Lihong Ma. Bisectors and voronoi diagrams for convex distance functions. Master's thesis, FernUniversitĺat Hagen, 2000.

[Mar82] David Marr. *Vision. A Computational Investigation into the Human Representation and Processing of Visual Information*. W.H. Freeman Company, 1982. MAR d 82:1 1.Ex.

[MDG01] S. Merillou, Jean-Michel Dischler, and D. Ghazanfarpour. Surface scratches: measuring, modeling and rendering. In *The Visual Computer*, volume 17(1), pages 30–45. Springer, 2001.

[MF92] Michael McCool and Eugene Fiume. Hierarchical poisson disk sampling distributions. In *Proceedings of the conference on Graphics interface '92*, pages 94–105, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[MGW01] Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial texture maps. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 519–528, New York, NY, USA, 2001. ACM Press.

[MIS01] Kenji Miyata, T. Itoh, and K. Shimada. A method for generating pavement textures using the square packing technique. *The Visual Computer*, 17(8):475–490, 2001.

[Miy90] Kazunori Miyata. A method of generating stone wall patterns. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 387–394, August 1990.

[MS95] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 95–ff., New York, NY, USA, 1995. ACM Press.

[NA03] Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 97–105, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[NC99] Fabrice Neyret and Marie-Paule Cani. Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings*, pages 235–242. ACM SIGGRAPH, Addison Wesley, August 1999.

[OBSC00] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[O'R98] Joseph O'Rourke. *Computational geometry in C (2nd ed.)*. Cambridge University Press, 1998.

[Pea85] Darwyn R. Peachey. Solid texturing of complex surfaces. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 279–286, New York, NY, USA, 1985. ACM Press.

[Per85] Ken Perlin. An image synthesizer. pages 287–296, 1985.

[PFH00] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[PH89] Ken Perlin and Eric M. Hoffert. Hypertexture. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 253–262, July 1989.

[PM01] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, New York, NY, USA, 2001. ACM Press.

[POF98] Pierre Poulin, Mathieu Ouimet, and Marie-Claude Frasson. Interactively modeling with photogrammetry. In *Proceedings of Eurographics Workshop on Rendering 98*, pages 93–104, 1998.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 1992.

[QD05] Desheng Wang Qiang Du. Anisotropic centroidal voronoi tessellations and their applications. *SIAM Journal on Scientific Computing*, 26:737–761, 2005.

[QDJ04] Maria Emelianenko Qiang Du and Lili Ju. Convergence properties of the lloyd algorithm for computing the centroidal voronoi tessellations. *SIAM Journal on Numerical Analysis*, Submitted October 2004.

[RK97] T. Rausch R. Kunze, F.-E. Wolter. Geodesic voronoi diagrams on parametric surfaces. In *Proc. of CGI'97*, pages 230–237. IEEE Computer Society Press, June 1997.

[RTG97] Holly Rushmeier, Gabriel Taubin, and André Guéziec. Applying shape from lighting variation to bump map capture. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 35–44, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

[SCA02] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 673–680, New York, NY, USA, 2002. ACM Press.

[Sec02a] Adrian Secord. Random marks on paper: Non-photorealistic rendering with small primitives. Master's thesis, The University of British Columbia, October 2002. Department of Computer Science.

[Sec02b] Adrian Secord. Weighted voronoi stippling. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pages 37–43. ACM Press, 2002.

[Sof] Softimage. Softimagelxsi. http://www.softimage.com.

[ST04] Joseph Szakas and Christian Trefftz. Two parallel computational geometry algorithms: A discrete voronoi diagram with disappearing seeds and convex hull. In IEE, editor, *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page p. 262a, Santa Fe, New Mexico, 2004.

[Stu] Microsoft Game Studios. Top spin. http://www.xbox.com/en-US/topspin/default.htm.

[TA04] Francesca Taponecco and Marc Alexa. Steerable texture synthesis. In *Annual Conference of the European Association for Computer Graphics EUROGRAPHICS 2004*, September 2004.

[tom] Lara croft - tomb raider. http://www.tombraiders.de.vu.

[TS03] Christian Trefftz and Joseph S. Szakas. Parallel algorithms to find the voronoi diagram and the order-k voronoi diagram. In *17th International Parallel and Distributed Processing Symposium (IPDPS-2003)*, pages 270–270, Los Alamitos, CA, April 22–26 2003. IEEE Computer Society.

[Tur91] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298, July 1991.

[TW02] Leandro Tonietto and Marcelo Walter. Towards local control for image-based texture synthesis. In *SIBGRAPI '02: Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing*, page 252. IEEE Computer Society, 2002.

[War92] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 265–272. ACM Press, 1992.

[WFM01] Marcelo Walter, Alain Fournier, and Daniel Menevaux. Integrating shape and pattern in mammalian models. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 317–326. ACM Press / ACM SIGGRAPH, 2001.

[WFR98] Marcelo Walter, Alain Fournier, and Mark Reimers. Clonal mosaic model for the synthesis of mammalian coat patterns. In *Proceedings of Graphics Interface*, pages 82–91, Vancouver, BC, Canada, 18–20 June 1998.

[WK91] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.

[WL00] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Wor96] Steven Worley. A cellular texture basis function. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, New York, NY, USA, 1996. ACM Press.

[WS82] G. Wyszecki and W. S. Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley, 1982.

[WWL+03] Xi Wang, Lifeng Wang, Ligang Liu, Shimin Hu, and Baining Guo. Interactive modeling of tree bark. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 83, Washington, DC, USA, 2003. IEEE Computer Society.

[WY04] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics (TOG)*, 23(3):364–367, 2004.

[Yes79] Chris I. Yessios. Computer drafting of stones, wood, plant and ground materials. In *SIGGRAPH '79: Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 190–198. ACM Press, 1979.

[YHBZ01] Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and shape synthesis on surfaces. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 301–312, London, UK, 2001. Springer-Verlag.

[ZeGWW02] S.-C. Zhu, C. e. Guo, Y. Wu, and Y. Wang. What are textons? *Lecture Notes in Computer Science*, 2353, 2002.

[ZG03] Steve Zelinka and Michael Garland. Mesh modelling with curve analogies. In *SIGGRAPH '03: Proceedings of the SIGGRAPH 2003 conference on Sketches & applications*, pages 1–1, New York, NY, USA, 2003. ACM Press.

[ZG04a] Steve Zelinka and Michael Garland. Jump map-based interactive texture synthesis. *ACM Transactions on Graphics (TOG)*, 23(4):930–962, 2004.

[ZG04b] Steve Zelinka and Michael Garland. Similarity-based surface modelling using geodesic fans. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 204–213, New York, NY, USA, 2004. ACM Press.

[ZTCS99] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.