



**HAL**  
open science

# Modèle d'apparence semi-procédural pour le contrôle et le passage à l'échelle de la synthèse de textures et de matériaux

Pascal Guehl

► **To cite this version:**

Pascal Guehl. Modèle d'apparence semi-procédural pour le contrôle et le passage à l'échelle de la synthèse de textures et de matériaux. Autre [cs.OH]. Université de Strasbourg, 2022. Français. NNT : 2022STRAD039 . tel-04090150

**HAL Id: tel-04090150**

**<https://theses.hal.science/tel-04090150v1>**

Submitted on 5 May 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

*ÉCOLE DOCTORALE Mathématiques, Sciences de l'Information et de l'Ingénieur*

unité de recherche : ICube – UMR 7357

**THÈSE** présentée par :

**Pascal GUEHL**

soutenue le : 14 décembre 2022

pour obtenir le grade de : **Docteur de l'université de Strasbourg**

Discipline/ Spécialité : Informatique

**Modèle d'apparence semi-procédural  
pour le contrôle et le passage à l'échelle  
de la synthèse de textures et de matériaux**

**THÈSE dirigée par :**

**M. DISCHLER Jean-Michel**

Professeur des universités, université de Strasbourg

**RAPPORTEURS :**

**Mme CANI Marie-Paule**

**M. GUÉRIN Éric**

Professeur des universités, Ecole Polytechnique

Maître de conférences, INSA de Lyon

---

**AUTRES MEMBRES DU JURY :**

**M. LEFEBVRE Sylvain**

**M. ALLÈGRE Rémi**

Directeur de recherche, INRIA Nancy Grand Est

Maître de conférences, université de Strasbourg



# Remerciements

Je tiens tout d'abord à remercier les membres de mon jury de thèse pour avoir accepté de rapporter et d'examiner les travaux présentés dans ce manuscrit. Merci pour l'intérêt que vous portez à mes travaux et pour le temps consacré.

Je tiens à remercier mes encadrants de thèse : mon directeur Jean-Michel Dischler et mon co-encadrant Rémi Allègre, pour le temps passé à me former, transmettre leur passion et leur savoir, leur soutien durant des moments difficiles.

Je tiens également à remercier Sylvain Lefebvre et Franck Hétroy-Wheeler pour avoir participé et contribué à mes comités de suivi de thèse.

Je tiens également à remercier tous les membres de l'équipe de recherche IGG du laboratoire de recherche ICube, notamment Dominique Bechmann pour son soutien, et Basile Sauvage pour ses efforts de didactique et l'animation du pôle Texture. Merci aux ingénieurs de recherche Sylvain Thery et Frédéric Larue, et à l'ensemble des doctorants de l'équipe (notamment Geoffrey Guingo, Paul, Nicolas, Florian, Quentin, Mélinda, Katia, etc.), et aux autres doctorants du laboratoire rencontrés à différentes occasions.

Je tiens à remercier mes anciens professeurs et encadrants pour leur qualité d'enseignement, leur passion et leur inspiration : au laboratoire de recherche LIRIS à l'Université Lyon 1 (notamment mon encadrant de stage de Master Jean-Claude Iehl, et les divers professeurs comme Raphaëlle Chaine, Eric Galin, David Coeurjolly, etc.), dans l'équipe Maverick de l'INRIA Grenoble (notamment mon directeur Fabrice Neyret lorsque j'étais ingénieur de recherche, et les membres de l'équipe comme Cyril Soler, etc.).

Je remercie également Pôle Emploi, l'agence française pour l'emploi, pour son allocation financière complémentaire mensuelle. Sans mon allocation, je n'aurais pas pu financer les dépenses autour de mon doctorat. De même pour la CAF (Caisse d'Allocations Familiales), pour sa prime d'activité mensuelle.

# Modèle d'apparence semi-procédural pour le contrôle et le passage à l'échelle de la synthèse de textures et de matériaux

## Résumé

L'automatisation des tâches de création de textures et de matériaux des mondes virtuels 3D est en forte demande dans l'industrie du film et des jeux vidéo.

Dans ce cadre, nous nous intéressons aux textures stochastiques structurées (et par extension aux matériaux), parmi les plus répandues. Caractérisées par des propriétés variant spatialement, elles sont particulièrement difficiles à reproduire par les méthodes de synthèse par l'exemple et ont fait l'objet de peu de travaux de recherche.

Nous proposons une nouvelle méthode de synthèse de textures et de matériaux, dite semi-procédurale, consistant à séparer la synthèse de la structure, de celle des détails de couleur. La structure, procédurale, utilise une nouvelle fonction générique de bruit à convolution parcimonieuse, infinie, sans répétition, cohérente, éditable et contrôlable. Les détails sont générés par l'exemple via une approche d'optimisation parallèle automatique engendrant une similitude visuelle. Cette méthode tire parti du meilleur du monde de la synthèse procédurale et par l'exemple : résultats fidèles aux exemples, représentation à faible coût de stockage, possibilité de synthétiser à la volée, degré élevé de contrôle, et passage à l'échelle.

Mots-clés : informatique graphique, synthèse de textures, génération procédurale, fonctions de bruit, texture basis function, matériau, modélisation procédurale inverse

## Résumé en anglais

Automating the tasks of authoring textures and materials for 3D virtual worlds is in high demand in the film and video game industry.

In this context, we are interested in stochastic structured textures (and by extension to materials), among the most widespread. Characterized by spatially varying properties, they are particularly difficult to reproduce by by-example synthesis methods and have been the subject of little research work.

We propose a new method of textures and materials synthesis, called semi-procedural, consisting in separating the synthesis of the structure, from that of the color details. The structure, procedural, uses a new generic sparse convolutional noise function that is infinite, repeatless, coherent, editable and controllable. The details are generated by-example via an automatic parallel optimization approach resulting in visual similarity. This method takes advantage of the best world of procedural and by-example synthesis: results faithful to examples, low storage cost representation, ability to synthesize on the fly, high degree of control, and scalability.

Keywords: computer graphics, texture synthesis, procedural generation, noise functions, texture basis function, material, inverse procedural modeling

# Table des matières

<b>I</b>	<b>Introduction</b>	<b>9</b>
<b>1</b>	<b>Création de Contenus Numériques 3D et Gestion de l'Apparence</b>	<b>10</b>
1.1	Informatique Graphique, besoins industriels . . . . .	10
1.1.1	Industries créatives . . . . .	12
1.1.2	Convergence du cinéma et des jeux vidéo . . . . .	16
1.2	Création de textures et de matériaux . . . . .	16
1.2.1	Textures et matériaux . . . . .	17
1.2.2	Génération de textures et de matériaux . . . . .	20
1.2.3	Spécificités des industries créatives . . . . .	26
<b>2</b>	<b>Verrous Scientifiques et Techniques, Contributions</b>	<b>28</b>
2.1	Contexte et problématique . . . . .	28
2.2	Objectif de la thèse et contributions . . . . .	29
2.3	Modèle procédural de structure . . . . .	31
2.4	Synthèse de la couleur ou des propriétés des matériaux . . . . .	33
2.5	Publications, collaborations et valorisation . . . . .	34
2.6	Plan de la thèse . . . . .	36
<b>II</b>	<b>État de l'Art</b>	<b>37</b>
<b>3</b>	<b>Modèles de Textures et Différentes Approches de Synthèse</b>	<b>38</b>
3.1	Introduction . . . . .	38
3.1.1	Notion de texture et taxonomie . . . . .	38
3.1.2	Modèles de texture . . . . .	41
3.2	Génération non paramétrique . . . . .	45
3.2.1	Approches statistiques Multi-Échelle . . . . .	45

3.2.2	Approches par pixel et par patch . . . . .	46
3.2.3	Approche par pavage . . . . .	54
3.2.4	Approche par apprentissage profond . . . . .	55
3.3	Synthèse par méthodes procédurales . . . . .	57
3.3.1	Bruits procéduraux . . . . .	57
3.3.2	Fonctions de base de texture ( <i>Texture Basis Functions</i> ) . . . . .	62
3.3.3	Autres approches et motifs procéduraux . . . . .	63
3.4	Représentations et génération des matériaux . . . . .	65
3.4.1	Représentations des matériaux . . . . .	65
3.4.2	Acquisition et synthèse de matériaux . . . . .	66
<b>4</b>	<b>Contrôle de la Synthèse des Textures et Matériaux Structurés et Non Homogènes</b> . . . . .	<b>68</b>
4.1	Contrôle des structures . . . . .	68
4.1.1	Approches géométriques . . . . .	68
4.1.2	Approches procédurales . . . . .	69
4.1.3	Gestion des variations spatiales . . . . .	74
4.2	Reproduction, contrôle et édition des textures non homogènes . . . . .	74
4.2.1	Guidage par champs de vecteurs 2D . . . . .	75
4.2.2	Guidage par cartes de labels ( <i>Texture-by-Numbers</i> ) . . . . .	75
4.2.3	Génération par apprentissage automatique . . . . .	79
4.3	Gestion de l'organisation spatiale . . . . .	81
4.3.1	Synthèse par contraintes sur les relations d'adjacence de blocs . . . . .	81
4.3.2	Arrangements et distributions d'éléments . . . . .	84
4.4	Interpolation de textures . . . . .	86
<b>5</b>	<b>Modélisation Procédurale Inverse de Textures et de Matériaux</b> . . . . .	<b>87</b>
5.1	Approches classiques . . . . .	87
5.1.1	Extraction et recherche de paramètres procéduraux . . . . .	87
5.1.2	Extraction et recherche de modèles procéduraux et de leurs paramètres . . . . .	90
5.2	Décomposition sur une base de BRDFs . . . . .	93
5.3	Approche par apprentissage automatique . . . . .	94
5.3.1	De croquis vers des modèles procéduraux . . . . .	94
5.3.2	De caractéristiques perceptuelles vers des textures procédurales . . . . .	96

5.3.3	Apprentissage de préférences utilisateur et espaces latents . . . . .	99
5.3.4	De textures vers des graphes de textures et matériaux procéduraux . . . . .	99
<b>6</b>	<b>État de l'art : Conclusion</b>	<b>102</b>
<b>III</b>	<b>Modèle d'Apparence Semi-Procédural</b>	<b>104</b>
	<b>- INTRODUCTION -</b>	<b>105</b>
<b>7</b>	<b>Approche <i>Semi-Procédurale</i></b>	<b>106</b>
7.1	Observations sur les textures structurées . . . . .	106
7.1.1	Observations générales . . . . .	106
7.1.2	Les textures stochastiques structurées courantes . . . . .	110
7.2	Vers un modèle générique procédural de structures . . . . .	115
7.2.1	Distribution spatiale des éléments . . . . .	115
7.2.2	Formes visuelles locales des éléments . . . . .	115
7.2.3	Interactions mutuelles entre éléments . . . . .	117
7.2.4	Formulation discrète de la PPTBF . . . . .	117
7.2.5	Transformations spatiales . . . . .	118
7.2.6	Génération des structures visuelles . . . . .	120
7.3	L'Approche <i>semi-procédurale</i> . . . . .	121
	<b>- SYNTHÈSE DE STRUCTURES -</b>	<b>124</b>
<b>8</b>	<b>Modèle de Structures Procédurales <i>PPTBF</i></b>	<b>125</b>
8.1	Instanciation du concept de PPTBF . . . . .	125
8.2	Distributions procédurales de points . . . . .	126
8.2.1	Objectifs . . . . .	126
8.2.2	Différents phénomènes et types de processus ponctuels . . . . .	127
8.2.3	Génération procédurale . . . . .	129
8.2.4	Résumé des paramètres des <i>processus ponctuels P</i> . . . . .	138
8.2.5	Déterminisme et générateur de nombres pseudo-aléatoires <i>PRNG</i> . . . . .	139
8.3	Contraintes spatiales et interactions mutuelles entre éléments . . . . .	140
8.3.1	Fenêtre générique . . . . .	140
8.3.2	Fenêtre sans recouvrement . . . . .	143

8.3.3	Fenêtres avec recouvrement . . . . .	150
8.3.4	Synthèse des paramètres de la fonction <i>window W</i> . . . . .	153
8.4	Formes visuelles locales des éléments . . . . .	154
8.4.1	Objectifs . . . . .	154
8.4.2	Objets de types filiformes ou constants . . . . .	154
8.4.3	Modèle de <i>mixture de noyaux</i> . . . . .	162
8.4.4	Distributions des noyaux de Gabor . . . . .	166
8.5	Variations spatiales et interpolations de structures . . . . .	168
8.5.1	Métamorphose de PPTBFs . . . . .	168
<b>9</b>	<b>Évaluation du Modèle Procédural de Structures <i>PPTBF</i></b>	<b>173</b>
9.1	Approche expérimentale . . . . .	173
9.2	Comparaisons aux méthodes du type <i>bruit par l'exemple</i> . . . . .	187
9.3	Performances . . . . .	191
9.4	Limitations . . . . .	192
<b>10</b>	<b>Estimation des Paramètres des Structures Procédurales <i>PPTBF</i></b>	<b>193</b>
10.1	Introduction . . . . .	193
10.1.1	Contexte . . . . .	193
10.1.2	Problématique . . . . .	194
10.2	Base de données de structures procédurales . . . . .	195
10.3	Descripteurs de structures et métrique de similarité . . . . .	197
10.4	Recherche de plus proches voisins . . . . .	199
10.5	Espace de structures . . . . .	204
10.5.1	t-SNE : Visualisation de l'espace de structure . . . . .	204
10.5.2	GPLVM : Navigation dans un espace de structure . . . . .	206
10.6	Conclusion . . . . .	207
	<b>- SYNTHÈSE DE TEXTURES ET DE MATÉRIAUX -</b>	<b>209</b>
<b>11</b>	<b>Synthèse de Textures et de Matériaux <i>Semi-Procéduraux</i></b>	<b>210</b>
11.1	Ajout de détails par l'exemple . . . . .	210
11.1.1	Objectif . . . . .	210
11.1.2	Solution proposée . . . . .	211
11.1.3	Aperçu de la synthèse de textures et de Matériaux semi-procéduraux . . . . .	216
11.1.4	Algorithme . . . . .	218
11.2	Conclusion . . . . .	219

<b>12 Évaluation du Modèle <i>Semi-Procédural</i></b>	<b>220</b>
12.1 Introduction . . . . .	220
12.2 Synthèse de textures . . . . .	220
12.3 Synthèse de Matériaux . . . . .	225
12.4 Variations Spatiales de Paramètres . . . . .	226
12.5 Transfert d'apparence . . . . .	229
12.6 Comparaisons avec la modélisation de textures procédurales inverse . . . . .	230
12.7 Performances . . . . .	232
12.8 Discussion et limitations . . . . .	233
<b>IV Conclusion</b>	<b>235</b>
<b>13 Bilan</b>	<b>236</b>
<b>14 Perspectives</b>	<b>241</b>
<b>V Annexes</b>	<b>247</b>
<b>A Algorithme de la PPTBF</b>	<b>248</b>
A.1 Pseudo-Code . . . . .	248
<b>B Format de Fichier de Structures PPTBF</b>	<b>252</b>
B.1 Description . . . . .	252
<b>VI Bibliographie</b>	<b>255</b>

# Première partie

## Introduction



# Chapitre 1

## Création de Contenus Numériques 3D et Gestion de l'Apparence

On présente ici une brève introduction à l'Informatique Graphique et aux besoins industriels en matière de création de contenus numériques 3D et de gestion de l'apparence, en restant à un haut niveau d'abstraction. Les détails seront exposés dans les prochains chapitres, notamment l'état de l'art.

### 1.1 Informatique Graphique, besoins industriels

Cette thèse s'inscrit dans les thématiques de l'Informatique Graphique, domaine scientifique qui s'attache à générer des images par ordinateur (on parle de *synthèse* ou de *génération* d'images). Le but est de définir des modèles et algorithmes pour représenter des scènes virtuelles 3D, notamment en termes de géométrie et d'apparence, et les visualiser (on parle de *rendu*). Les communautés de recherche associées sont très actives et les avancées sont suivies de près par l'industrie. Une de ses branches principales s'intéresse à la modélisation et la simulation physiquement réaliste du transport de la lumière dans l'environnement et ses interactions avec et entre les objets 3D en fonction de leurs propriétés optiques, on parle de *Physically-Based Rendering (PBR)* (figure 1.1). Une autre branche est axée sur une approche non photoréaliste, dite artistique, on parle de *Non-Photorealistic Rendering (NPR)* ou de rendu expressif (figure 1.2).



**FIGURE 1.1** – Exemple de rendu physiquement réaliste (PBR, *Physically-Based Rendering* de scènes 3D avec le moteur 3D AMD Radeon ProRender<sup>1</sup>(par exemple, design de produits, architecture, effets spéciaux).



**FIGURE 1.2** – Exemple de rendu stylisé artistique (NPR, *Non-Photorealistic Rendering*) par coups de pinceau sans création de surface 3D dans l’approche [SSGS11], *OverCoat : An Implicit Canvas for 3D Painting*.

1. <https://www.amd.com/en/technologies/radeon-prorender>

On appelle l'*apparence* d'une scène 3D le résultat visuel issu du processus de rendu, qui fait intervenir, outre le transport de la lumière, des *modèles de matériaux* définissant les propriétés optiques de la surface des objets 3D. En ajoutant la gestion de différents niveaux d'échelles, on définit un *modèle d'apparence*. Afin de donner une apparence aux objets 3D, il est nécessaire de les "habiller" de *textures* et de *matériaux* encodant, sous forme d'images, les propriétés optiques en fonction de l'échelle d'observation. Dans le cadre de cette thèse, on s'intéresse plus particulièrement à des algorithmes de *génération* ou *synthèse* de textures et de matériaux, en particulier *procédurale*. La génération procédurale consiste à élaborer des processus de calcul à faible coût de stockage, calculables en parallèle (sur GPU), de manière multi-échelle (en temps constant, quelle que soit l'échelle d'observation), et non bornée dans l'espace. Cette approche algorithmique présente plusieurs difficultés, dont le contrôle de l'apparence des textures et des matériaux en résultant. On cherche ici à répondre à des besoins rencontrés plus particulièrement dans la création de textures et de matériaux pour des applications dans l'industrie créative, couvrant essentiellement le cinéma et les jeux vidéo, détaillés ci-après.

### 1.1.1 Industries créatives

L'industrie du cinéma et du jeu vidéo ont depuis toujours contribué à repousser les limites en termes de réalisme et de complexité des scènes 3D, limitées par les ressources matérielles (processeurs, mémoires, stockage), des contraintes de budget et de temps. La tendance étant de créer des mondes de plus en plus vastes, ultra-détaillés et réalistes.

**Cinéma.** L'industrie du cinéma est caractérisée par la génération d'images de très haute qualité et un très grand réalisme, quels que soient les temps de génération (une image peut nécessiter plusieurs heures calcul). Les tâches sont complexes et requièrent une expertise telle que les métiers sont séparés en départements dédiés comme le "storytelling" (mise en récit), la modélisation 3D (géométrie), le "texturing" (qui sera détaillée plus tard ; il s'agit d'ajouter des propriétés aux surfaces d'une scène 3D, comme la couleur), le "rigging" (création d'os et articulations pour le mouvement de personnages), l'animation, le "lighting" (modèles d'éclairage), les effets spéciaux (par exemple : feu, explosion, simulation physique), le rendu, le "compositing" (par exemple, l'assemblage et l'incrustation d'effets spéciaux et autres éléments dans les scènes 3D). Parmi les acteurs majeurs ayant contribué à des

avancées notables en informatique graphique, on peut citer : Digital Domain, ILM, Weta Digital, Pixar et Disney.

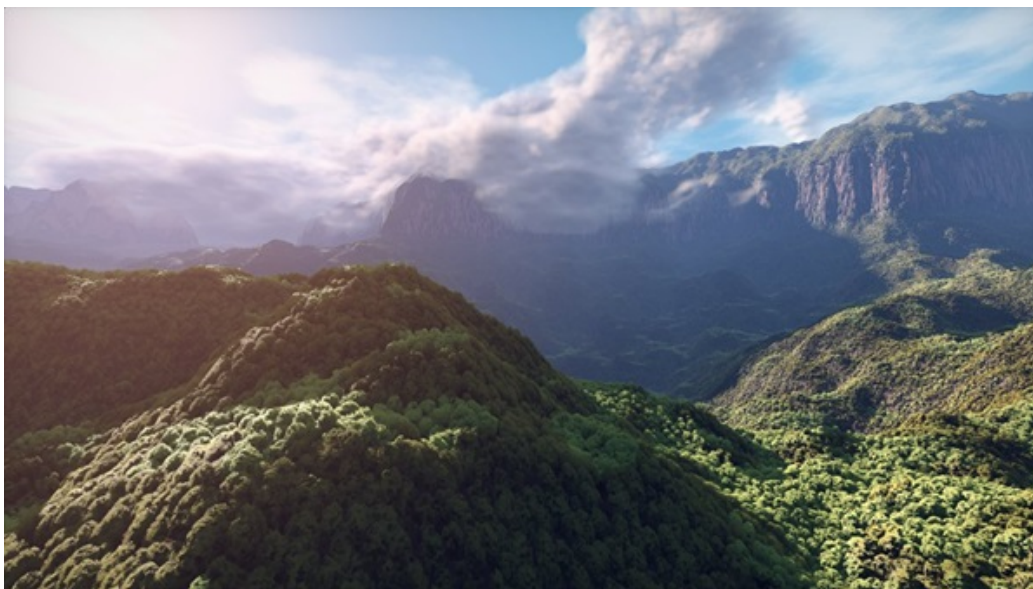
**Jeu vidéo.** L'industrie du jeu vidéo est très proche du cinéma en termes de recours aux outils de l'Informatique Graphique. Sa spécificité est d'être contrainte par des applications temps réel. Elle doit donc s'attacher à gérer un compromis entre qualité et vitesse de rendu afin d'obtenir la meilleure expérience d'immersion et d'interaction avec un ou des utilisateurs, en exploitant au mieux les ressources matérielles. Contrairement au cinéma qui génère des lots d'images de façon *hors-ligne*, le jeu vidéo impose aux infographistes et aux développeurs de s'adapter au matériel des clients, en général des PCs standards équipés de processeurs graphiques (GPU). Le même modèle 3D ultra-détaillé au cinéma aura un équivalent moins résolu dans le jeu vidéo : on parle de version *low poly* pour la géométrie, et de textures moins résolues. Parmi les acteurs majeurs du domaine ayant, par leurs besoins, contribué à faire progresser l'Informatique Graphique, on peut citer : Nintendo (jeux et consoles), Squaresoft, Epic Games et Unity Technologies pour leurs moteurs 3D temps réel Unreal Engine et Unity. Il faut également citer les deux acteurs NVidia et AMD pour leurs apports au développement des processeurs graphiques et des bibliothèques de programmation 3D temps réel bas niveau, avec également leur collaboration pour fournir des standards pour la programmation sur GPU comme CUDA, OpenCL, OpenGL et Vulkan.

**Scène Démo.** Très proche du monde du jeu vidéo, la scène démo (ou *demoscene*) s'attache le plus souvent en des concours de programmation, avec un fort penchant artistique mélangé à de la technique. De nombreux concours et conférences consistent notamment à créer et présenter le programme le plus impressionnant avec une taille finale ne dépassant pas 64 Ko (par exemple, NVScene<sup>2</sup>). Des outils comme Shadertoy<sup>3</sup> créé par Inigo Quilez suscitent de plus en plus d'intérêt, notamment sur la thématique de la génération procédurale (figure 1.3).

---

2. <http://nv.scene.org/2015/about>

3. <https://www.shadertoy.com/>



**FIGURE 1.3** – Exemple de modélisation procédurale et rendu temps réel de mondes virtuels : *Forest*<sup>4</sup> par Inigo Quilez. Modélisation par champs de distances signées et lancer de rayons dans Shadertoy.

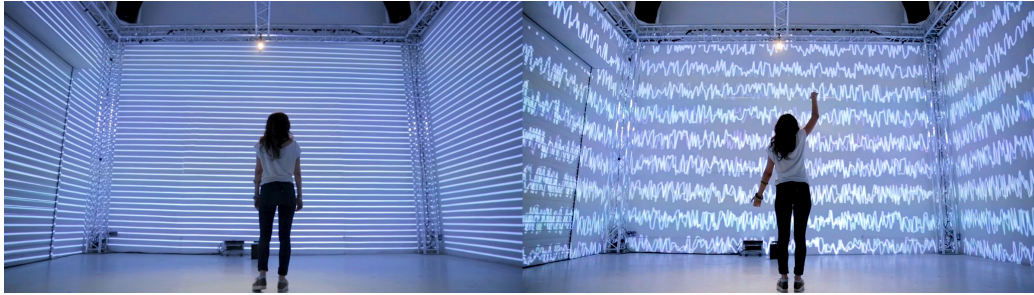
**Installations multimédia interactives.** Les installations multimédia se répandent de plus en plus. Les compagnies artistiques mêlant art et science se développent, ainsi que les installations synchronisant de la génération procédurale visuelle (*visuals*) avec des performances Live de musiciens électroniques. À titre d'exemple, l'installation *Noisy Skeleton*, de la société Theoriz<sup>5</sup>, propose de faire interagir un utilisateur placé dans un dispositif de type *cave* avec un environnement virtuel couplé à de la synthèse sonore et un visuel graphique interactif associé sous forme de signaux aléatoires représentant des ondes. Dans le domaine académique, on peut citer l'équipe de recherche *Images Numériques et Réalité Virtuelle (INREV)*<sup>6</sup>, du laboratoire Arts des Images et Art Contemporain, composant de l'École doctorale Esthétique, Sciences et Technologies des Arts de l'Université Paris 8.

---

4. <https://www.shadertoy.com/view/4ttSWf>

5. (<https://www.theoriz.com/fr/bienvenue/>)

6. <https://inrev.univ-paris8.fr/>



**FIGURE 1.4** – Installation multimédia interactive : *Noisy Skeleton*<sup>7</sup>. L'utilisateur interagit avec un environnement virtuel (visuel et sonore), modélisé sous la forme d'une grille de bruits, dont les amplitudes et fréquences dépendent de la position et gestes.

**Réalité virtuelle.** La technologie associée à la réalité virtuelle (VR) a énormément évolué, les dispositifs matériels se généralisent et sont désormais à la portée des consommateurs (par exemple, des casques immersifs). De plus en plus de demandes se tournent vers les applications mélangeant VR et AR, XR (*extended reality*), en essayant d'être le plus réaliste possible au vu des contraintes temps réel et de communication tels que les transferts réseau des données pour les mondes collaboratifs.

---

7. <https://vimeo.com/theoriz/noisyskeleton>

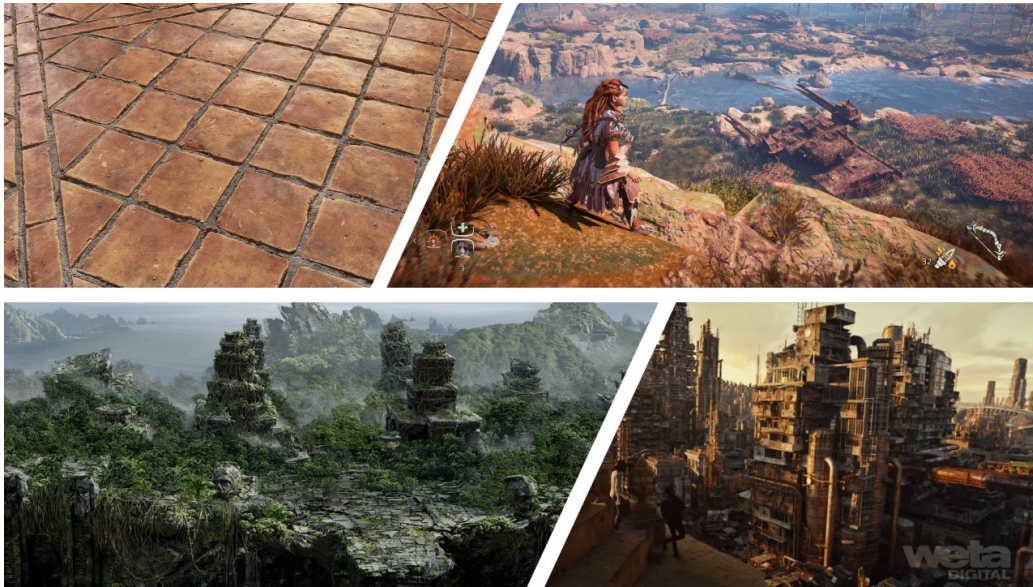


### 1.1.2 Convergence du cinéma et des jeux vidéo

On assiste depuis quelques années à une convergence entre le cinéma et le jeu vidéo : les outils, matériels et librairies se standardisent, les algorithmes se mutualisent. Le cinéma utilise de plus en plus d'outils du jeu vidéo comme les moteurs 3D Unreal Engine et Unity. La production virtuelle (*VP*, *Virtual Production*) s'impose pour la production et la visualisation de films assistés par ordinateur, axée sur une interaction temps réel sur les plateaux de tournages. Il est possible de *prévisualiser* le rendu des scènes 3D virtuelles lors des phases de conception. Ceci afin d'améliorer l'interaction avec le directeur artistique et avoir une meilleure idée du résultat final. À titre d'exemple, on peut acquérir les mouvements d'un comédien humain (*motion capture*) dans une salle vide (sur écran vert) et visualiser le rendu 3D du personnage virtuel associé et incrusté dans le monde virtuel 3D associé. Concernant le jeu vidéo, la société de développement du moteur de jeux Unity Technologies a racheté en 2021 la société d'effets spéciaux Weta Digital (par exemple, *Le Seigneur des Anneaux*, *King Kong*, *Avatar*) afin de pouvoir disposer de leurs technologies et outils de génération de mondes massifs. Le développement économique du cinéma et des jeux vidéo est croissant, de même que leurs besoins en innovation et R&D, ce qui justifie leur intérêt pour les avancées dans la recherche scientifique, notamment les possibilités d'automatisation des processus chronophages et coûteux financièrement.

## 1.2 Création de textures et de matériaux

Les scènes virtuelles 3D du cinéma et du jeu vidéo sont de plus en plus gigantesques, détaillées et réalistes. Elles peuvent s'étendre sur plusieurs kilomètres avec une précision du millimètre (figure 1.5). Les coûts de modélisation géométrique et de création de textures et matériaux représentent une part très importante dans les budgets. À titre d'exemple, chez Weta Digital, dans *Le Seigneur des Anneaux*, les objets 3D étaient très détaillés, notamment même les parties non visibles des objets (par exemple, l'arrière d'un vase) avaient également été modélisées au cas où le directeur artistique aurait souhaité modifier la scène, changer un emplacement, bouger la caméra ou zoomer. On définit ci-après les textures et les matériaux, avant de passer en revue les outils mis à disposition de l'industrie graphique.



**FIGURE 1.5** – Exemples de rendus de scènes d’apparences complexes, à grande échelle, dans les films et jeux vidéo. En haut : scène 3D d’un infographiste (à gauche) et jeu vidéo *Horizon* (à droite). En bas : films *King Kong* (à gauche) et *Alita : Battle Angel* (à droite).

---

[https://fr.wikipedia.org/wiki/Horizon\\_Zero\\_Dawn](https://fr.wikipedia.org/wiki/Horizon_Zero_Dawn)

[https://fr.wikipedia.org/wiki/King\\_Kong\\_\(film,\\_2005\)](https://fr.wikipedia.org/wiki/King_Kong_(film,_2005))

[https://fr.wikipedia.org/wiki/Alita:\\_Battle\\_Angel](https://fr.wikipedia.org/wiki/Alita:_Battle_Angel)

## 1.2.1 Textures et matériaux

### Textures

Dans les scènes 3D, la géométrie des objets est généralement représentée par leur surface, sous formes de maillages (*mesh*) composés majoritairement de triangles. Un maillage est généralement muni d’une paramétrisation 2D, permettant de plaquer des textures représentées par des images 2D (on parle de *texture mapping*). Les textures permettent de simuler une apparence complexe, et sont souvent peintes par des artistes. La figure 1.6 montre une texture plaquée sur un maillage pour lui ajouter de la couleur. Le problème de la paramétrisation de maillages a donné lieu à de nombreux travaux (voir par exemple [YLT19] pour un état de l’art). Les textures sont utilisées pour



augmenter le réalisme des scènes virtuelles sans avoir à complexifier la géométrie. Les données géométriques sont dites *augmentées* ou *amplifiées* par les textures, encodant diverses propriétés allant de la couleur à des cartes de normales, ou des cartes de hauteur permettant d’ajouter du relief (microgéométrie) à une surface.

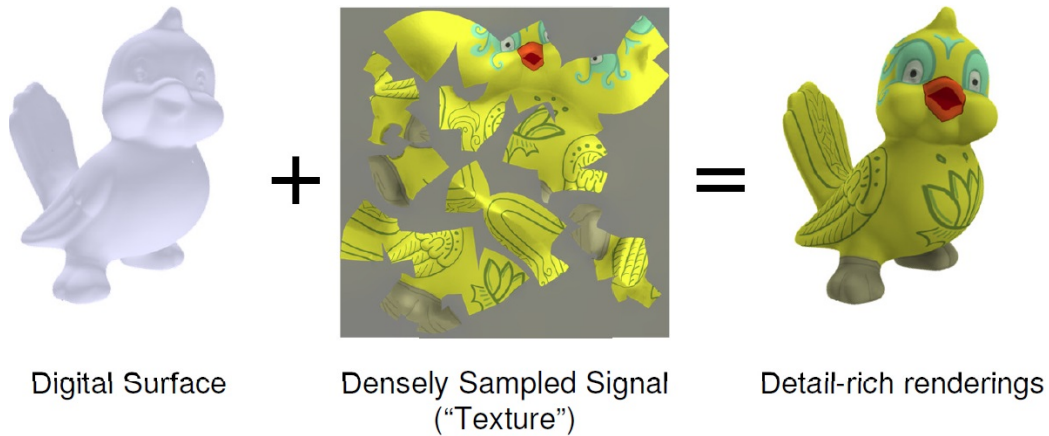
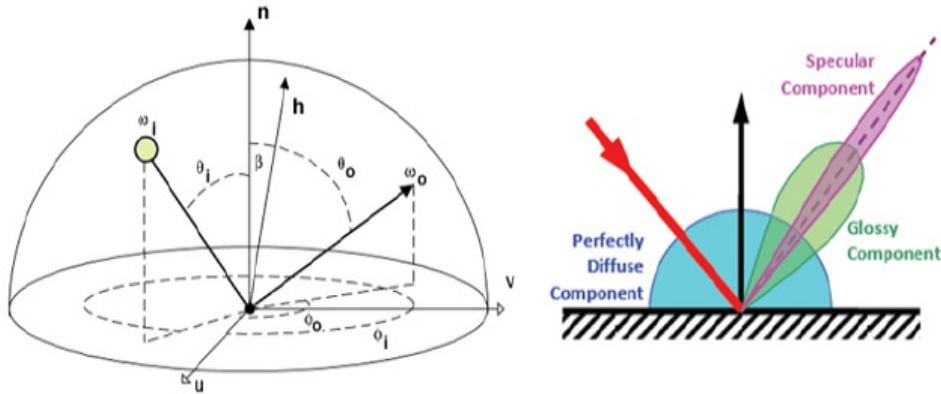


FIGURE 1.6 – Exemple de *texture mapping*

Placage de texture ajoutant des détails à un modèle 3D moyennement résolu [YLT19].

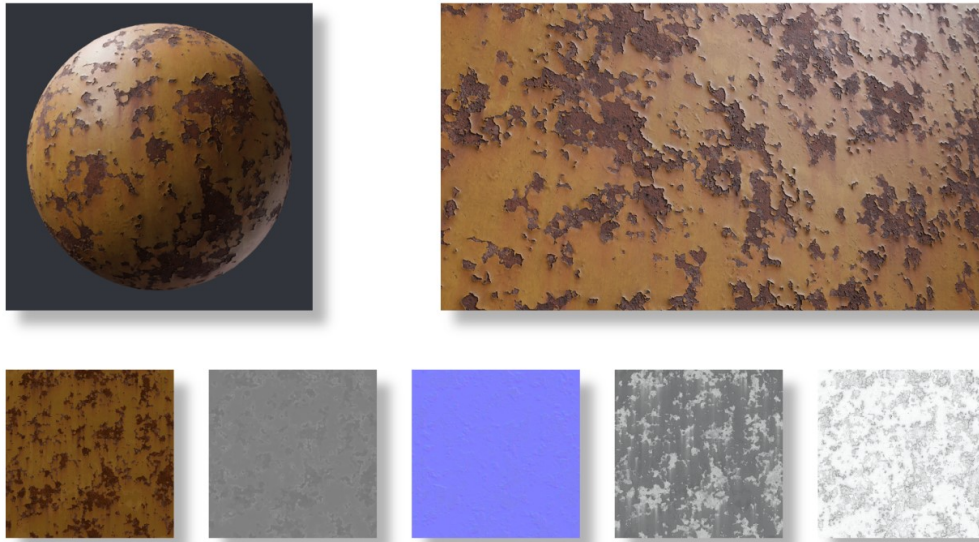
## Matériaux et représentation SV-BRDF

Un matériau encode les propriétés optiques d’un objet utilisées lors du rendu dans le cadre des interactions lumière-matière. Par exemple, un matériau peut être réfléchissant, translucide, avoir une apparence rugueuse (avec des aspérités sous forme de micro-géométrie). Mathématiquement, la fonction de réflectance bidirectionnelle, BRDF (*Bidirectional Reflectance Distribution Function*), est utilisée pour modéliser ces comportements. Elle indique notamment (figure 1.7) pour une direction d’un rayon lumineux en entrée  $\omega_i$  (*in*), quel sera ou seront les directions privilégiées en sortie  $\omega_o$  (*out*), après avoir interagi avec la surface (et éventuellement l’intérieur) de l’objet selon la normale  $\mathbf{n}$  au point d’impact. Le terme générique de la BRDF est plutôt BxDF avec pour  $x$  soit le  $R$  de *réflectance* (pour les objets réfléchissants), soit le  $T$  de *transmittance* (pour les objets transparents). Dans les jeux vidéo et les films, les matériaux créés par les infographistes sont le plus souvent



**FIGURE 1.7** – Modèle de BRDF. Interaction de la lumière avec différents comportements selon le type de matériau.

représentés et stockés sous forme d'un ensemble de textures 2D représentant chacune une de ses propriétés, potentiellement variant spatialement, à savoir : albedo (couleur de l'objet), normal (indiquant l'orientation des surfaces), hauteur, réflectivité, métallicité, éventuellement occlusion ambiante, etc. On parle de modèle en couches (*layer*) et de matériaux PBR (figure 1.8). Il existe d'autres modèles de matériaux, mais beaucoup plus complexes en termes de simulation, temps de calcul et de stockage, difficilement exploitable pour des applications interactives ou temps réel. Par exemple, la *BTF* (*bidirectional texture function*) ([DVGNK99], *Reflectance and texture of real-world surfaces*, Dana *et al.*) prend en compte le comportement du matériau sous tous les angles de visualisation possibles (permettant par exemple les ombrages internes, ou l'auto-ombrage). On peut en faire une acquisition par numérisation 3D, ou la simuler. Une des représentations les plus utilisées actuellement est la représentation SV-BRDF, pour *Spatially Varying BRDF*, qui offre un compromis répondant à plupart des besoins. De nombreux modèles s'appuient sur cette représentation et peuvent être implémentés sur GPU, par exemple GGX [WMLT07]). Dans nos travaux, notre méthode de synthèse peut s'appliquer à différents modèles de matériaux, à partir du moment où leurs propriétés sont encodées dans des couches de textures.



**FIGURE 1.8** – Matériau encodant des propriétés classiques utilisées par les moteurs de rendu actuels. En haut : un matériau plaqué sur un objet 3D et un zoom. En bas : les différentes couches de textures composant le matériau (albedo, hauteur, normale, rugosité, occlusion ambiante).

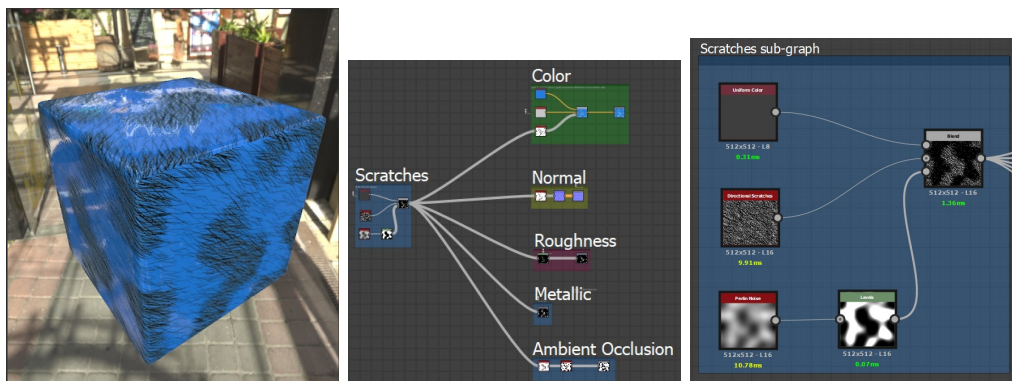
## 1.2.2 Génération de textures et de matériaux

L'industrie dispose de plusieurs outils et techniques pour modéliser les textures et matériaux, comme la suite d'outils d'Allegorithmic/Adobe<sup>8</sup> qui permet de modéliser des textures et matériaux avec Designer (voir ci-après) et de les peindre avec Painter. Alchemist, créé récemment et basé sur de l'IA (intelligence artificielle) permet de transformer une image en matériau et de modifier certains de ces paramètres, à la manière de Substance. Les programmeurs ou *Technical Director* peuvent aussi utiliser des langages de programmation dédiés afin de créer des programmes (ou *shaders*) comme RenderMan de Pixar, et des langages bas niveaux comme OpenGL accélérés sur carte graphique.

---

8. <https://www.adobe.com/fr/products/substance3d/3d-augmented-reality.html>

**Outils : Substance Designer et graphes de textures procédurales.**  
 L'outil *Substance Designer* modélise les textures et matériaux sous forme de graphes de nœuds procéduraux (par exemple, combinaison et composition de fonctions de base, tables de couleurs) (figure 1.9). On peut voir ce type d'outils comme une implémentation logicielle des concepts basés sur l'approche de Cook dans *Shade Trees* [Coo84].

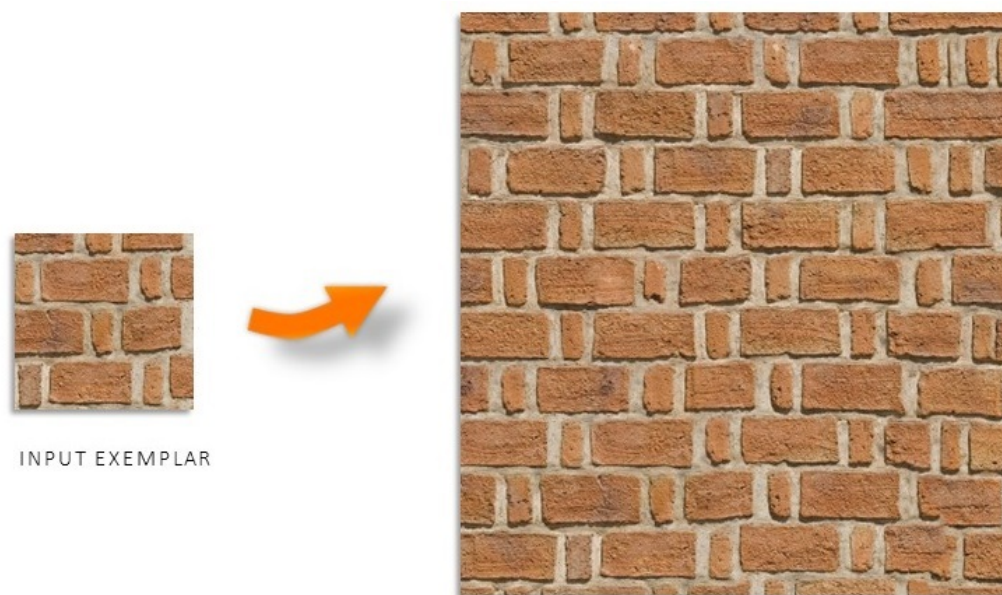


**FIGURE 1.9** – Substance Designer : outil de modélisation de textures et matériaux procéduraux sous forme de graphes. À gauche : rendu physiquement réaliste d'un matériau PBR avec un carte d'environnement. Au milieu : vue haut niveau du graphe associé générant des textures de couleur (albedo), normal, rugosité, métallicité et occlusion ambiante, à partir de noeuds procéduraux. À droite : zoom sur le sous-graphe de noeuds associé pour créer des motifs de bruits procéduraux (rayures stochastiques) et mélangé avec un masque procédural pour remplir le motif dans régions d'intérêts spécifiques.

Grâce à son système de nœuds programmables, ce type d'outil offre beaucoup de flexibilité dans les possibilités d'édition et de contrôle, qui sont très importantes pour les artistes. Cependant, l'usage de ce type d'outil est complexe et requiert un degré élevé d'expertise. À la fin du processus de création, le graphe peut être sauvegardé et les textures sont exportées à une résolution donnée. Le graphe peut également être exporté vers l'outil Substance Painter qui permettra de peindre la texture ou matériau du graphe sur des modèles 3D via des outils de peinture de type brush.

**Algorithmes : Synthèse de textures.** Si les artistes disposent d'outils dédiés comme Substance Designer pour générer ces textures et matériaux, ces processus sont complexes, longs et coûteux. L'industrie essaie de les automatiser pour réduire les coûts et les plannings. La recherche scientifique s'intéresse à la génération automatisée de ces textures selon diverses approches : on parle de *synthèse de textures*. Plusieurs approches coexistent, nous développons ici les deux plus courantes : la *synthèse non paramétrique* et la *génération procédurale*. On verra plus tard que l'on peut combiner la synthèse non paramétrique et la génération procédurale. Nous ne développons pas ici d'autres approches comme les simulations physico-chimiques.

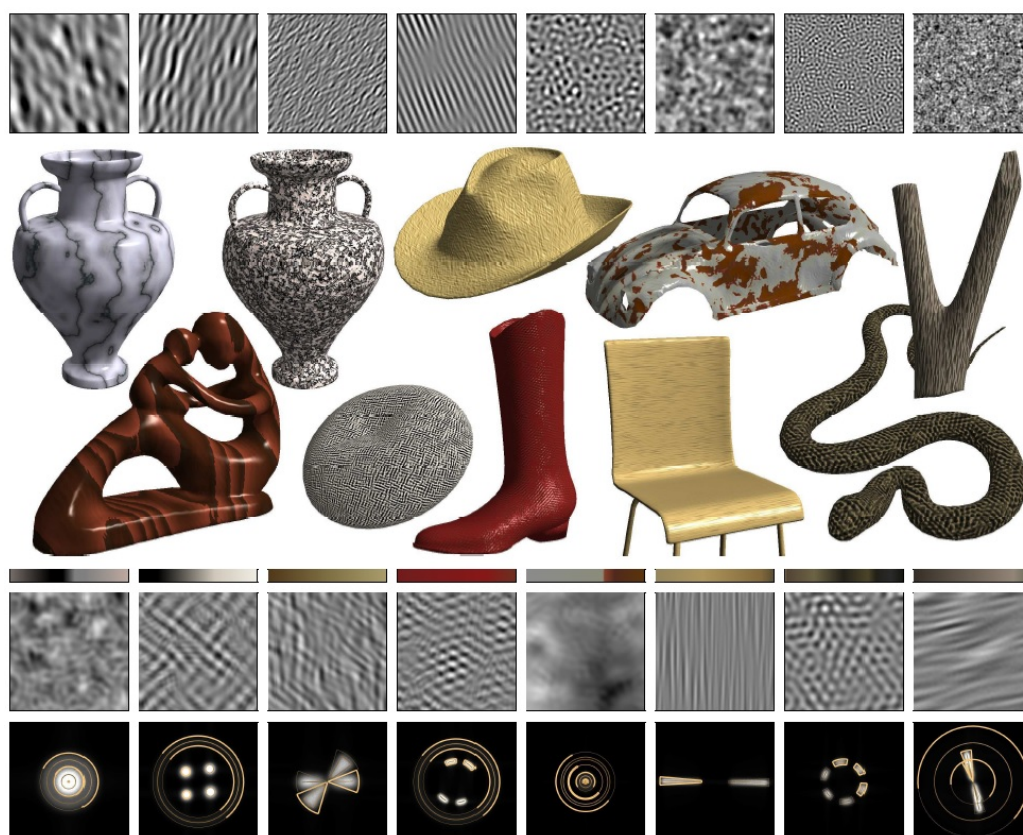
La synthèse non paramétrique consiste à générer une texture de taille arbitraire à partir d'un échantillon en entrée en exploitant des propriétés statistiques ou géométriques [WLKT09] (figure 1.10).



**FIGURE 1.10** – Génération d'une texture de taille arbitraire à partir d'un échantillon.

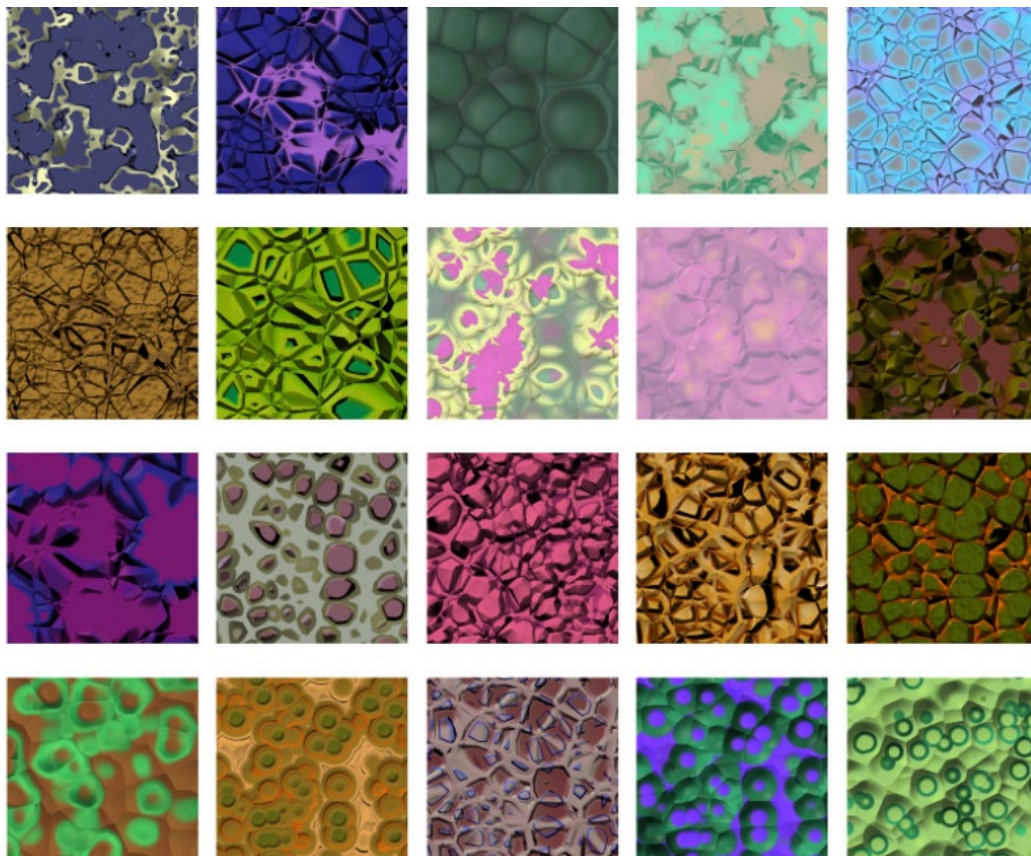


La synthèse procédurale [DLY+20], elle, utilise un algorithme ou une fonction mathématique pour automatiser la génération de contenus numériques, munis de paramètres permettant de contrôler l'apparence du résultat. La synthèse procédurale peut s'appliquer aussi bien à de la géométrie (par exemple, modèle 3D d'arbres, terrains) qu'à des textures ou matériaux. Dans le cadre de la synthèse de textures, différentes fonctions de *bruit* ont été proposées pour synthétiser des textures stochastiques (figure 1.11) (par exemple, un mur peint avec un aspect non uniforme) et des motifs structurés particuliers (par exemple, des cellules de peau d'animal) (figure 1.12).



**FIGURE 1.11** – Génération de textures de taille arbitraire à partir de fonctions de bruits, plaquées sur des objets 3D, et colorées via des tables de couleurs (voir [LLDD09]).

Certains bruits peuvent être utilisés comme fonctions de base de texture (*Texture Basis Functions*), c'est-à-dire un "support" ou un "guide" spatial pour effectuer de la synthèse de textures, en particulier dans le cas qui nous intéresse, c'est-à-dire celui des textures et matériaux de type *stochastique structuré* et potentiellement *non homogène*.



**FIGURE 1.12** – Génération de textures cellulaires par *texture basis function* de Worley [Wor96].

Les deux approches, non paramétrique et procédurale, permettent d'effectuer de la synthèse *par l'exemple* (*by example, data-driven*), c'est-à-dire synthétiser des textures à partir d'échantillons sous la forme d'images.

Actuellement, la synthèse procédurale domine dans l'industrie, car elle permet d'éditer et configurer les textures et matériaux à la demande (changement de couleurs, d'aspects, de formes, etc.) et choisir leur résolution de rendu. Les fonctions de bruit et des modèles plus complexes comme les fonctions de base de texture sont implémentées dans les outils comme Substance Designer vu plus haut. Comme on le verra en détail dans l'état de l'art, notre analyse soulève quatre limitations majeures des méthodes de synthèse existantes :

- la préservation et reproduction automatique des structures,
- dans le cas non homogène, la reproduction automatique de l'organisation spatiale des motifs en termes de relations d'adjacence et de transitions ;
- le passage à l'échelle ;
- l'édition interactive multi-échelle en temps réel.

Aucune des méthodes de synthèse existantes ne répond à toutes ces exigences.

**Modélisation procédurale inverse.** La *modélisation procédurale inverse* consiste à déterminer les valeurs des paramètres des modèles procéduraux de textures ou de matériaux à partir d'images. Cela nécessite d'avoir défini un modèle avec ses paramètres, afin de pouvoir ensuite analyser une image, estimer les valeurs des paramètres qui correspondent, et pouvoir reconstruire ou synthétiser une texture ou un matériau à partir du modèle.

La possibilité de pouvoir reconstruire automatiquement un graphe de l'outil Substance Designer à partir d'une image est actuellement hors d'atteinte. C'est un *problème mal posé*, car plusieurs graphes différents peuvent conduire à la même apparence. De plus, les graphes de matériaux sont souvent larges et complexes, voire composés eux-mêmes de nœuds programmables. C'est un domaine en forte demande dans l'industrie, à l'instar des travaux d'Adobe orientés vers leurs outils Substance Designer et Substance Alchemist ([SLH<sup>+</sup>20] *MATch : Differentiable Material Graphs for Procedural Material Capture*).



### 1.2.3 Spécificités des industries créatives

Le cahier des charges de l'industrie impose des contraintes de performances, de robustesse des outils logiciels, matériels ainsi que des algorithmes utilisés (notamment en termes d'hypothèses contraignantes). En outre, la dimension artistique et créative est essentielle : on ne communique pas toujours de la même manière avec un artiste, un scientifique et un développeur. Les sensibilités et les vocabulaires sont parfois différents : un artiste appréciera des paramètres et termes intuitifs (en lien avec ses actions concrètes), plutôt que des paramètres d'algorithmes ayant un sens mathématique. Parmi les spécificités et besoins des industries créatives, on peut citer :

- le besoin de contrôle artistique et le relâchement de contraintes algorithmiques (voir ci-dessous),
- le besoin de dépasser le clivage art et technologie et le développement de l'approche *Creative AI*, c'est-à-dire utiliser l'intelligence artificielle (comme le Machine Learning et le Deep Learning) pour augmenter la créativité et la productivité des artistes.

Concernant le besoin de contrôle artistique et le relâchement de contraintes algorithmiques, on peut citer l'exemple ci-après. La conférence annuelle *FMX*<sup>9</sup> est consacrée à l'animation, à l'informatique graphique et aux effets spéciaux. Elle y regroupe tous les acteurs majeurs internationaux des sociétés d'effets visuels du cinéma comme Weta Digital, ILM, MPC, Framestore, Mac Guff, Pixar, Disney et DreamWorks. Diverses sessions permettent de découvrir les dessous des films. D'autres montrent les dernières avancées de la recherche académique, notamment au travers de résumés et de sélections d'articles scientifiques de la conférence Siggraph. Ces industries suivent de très près les travaux du monde scientifique afin de s'en inspirer ou intégrer leurs avancées. En 2017, dans une session dédiée aux technologies appelée *Tools for tomorrow*, P. Selva de la société Weta Digital a présenté la session *Art Directable Nature - tools for large scale environments* (également à Eurographics 2017<sup>10</sup>). Il a évoqué la nécessité d'avoir à disposition des outils pour la création et la gestion d'environnements à grande échelle tout en préservant le contrôle artistique. Plusieurs exemples illustratifs de films, tels qu'*Avatar* et *Le Seigneur des Anneaux*, comprennent la gestion de mondes allant de l'échelle du centimètre, voire du millimètre, jusqu'à plusieurs kilomètres. Cette présentation illustre la spécificité des films en termes de narration

---

9. <https://fmx.de/en/home>

10. <https://projet.liris.cnrs.fr/eg2017/index.php/industrial-program/>

(*story telling*), à savoir passer avant l'absolue exactitude physique. À titre d'exemple, le directeur artistique peut vouloir déplacer un objet comme un rocher ou un arbre où il le désire, et ce quels que soient les modèles sous-jacents à base de physique et de modèles mathématiques. Dans les travaux scientifiques récents en Informatique Graphique, on peut citer WorldBrush ([EVC<sup>+</sup>15], *WorldBrush : Interactive Example-based Synthesis of Procedural Virtual Worlds*, par Busto *et al.*) comme outil répondant aux besoins de contrôler des distributions d'éléments naturels comme des rochers, des arbres et des plantes. Cependant, d'autres travaux, notamment en synthèse de textures par l'exemple, cherchant à automatiser des tâches habituellement réalisées par des artistes, manquent de paramètres intuitifs leur permettant de contrôler le résultat. À ce titre, on peut citer le keynote de la conférence *SCA 2018* donnés par J.P. Lewis (*Open Problems in Character Animation for Games and VFX*)<sup>11</sup>.

---

11. <https://sca2018.inria.fr/program/invited-speaker/>

# Chapitre 2

## Verrous Scientifiques et Techniques, Contributions

Dans ce chapitre, on présente le contexte, les problématiques et les verrous scientifiques et techniques abordés dans cette thèse, qui concernent la synthèse de textures et, plus généralement, celles des matériaux. Les industries et applications visées sont principalement celles du film et du jeu vidéo, dont les besoins ont été exposés précédemment. Nous présentons ensuite succinctement les contributions auxquelles ce travail de thèse a conduit.

### 2.1 Contexte et problématique

La modélisation et la gestion de l'apparence des scènes virtuelles 3D hautement détaillées que l'on trouve dans les jeux vidéo et l'industrie du film sont de plus en plus réalistes et complexes. Les demandes en termes d'automatisation des tâches de création de contenu associées par des infographistes sont très fortes, notamment pour les textures et matériaux.

Dans cette thèse, nous nous intéressons aux textures stochastiques structurées (et par extension aux matériaux) qui sont parmi les plus répandues dans les environnements virtuels. Ces textures, caractérisées par des propriétés qui varient spatialement (non stationnaires), sont particulièrement difficiles à reproduire par les méthodes de synthèse par l'exemple et ont fait l'objet de peu de travaux de recherche. Les méthodes existantes, pour la plupart guidées par les statistiques, s'appliquent à ce type de textures seulement si des propriétés de régularité simples peuvent en être extraites, ou

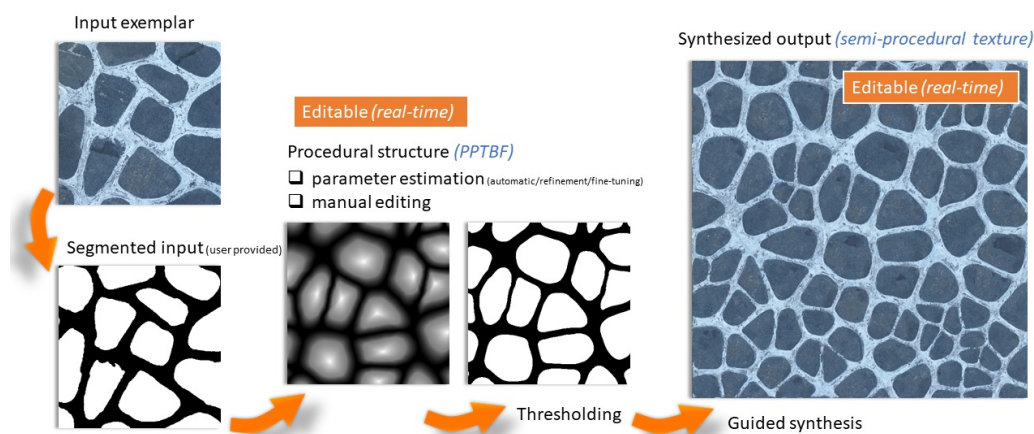
bien en ayant recours à des cartes de labels, en général créées manuellement, pour encoder les variations et/ou la répartition spatiale des motifs extraits des exemples. Dans ces conditions, des résultats très fidèles aux exemples peuvent être obtenus, mais la nécessité de créer des cartes de labels constitue un frein important. En outre, ces méthodes offrent des possibilités de contrôle et d'édition très restreintes. Par ailleurs, des méthodes de synthèse procédurales à base de graphes de fonctions ont été développées, avec pour objectif de produire des textures et des matériaux avec plusieurs avantages : une paramétrisation permettant de contrôler le résultat, un coût de stockage faible, et une synthèse possible à la volée sur GPU. Cette approche, devenue le standard industriel, requière une grande expertise et un processus fastidieux de création par essai et erreur pour chaque texture ou matériau à produire. La modélisation procédurale inverse est un champ de recherche dont l'objectif est de parvenir à déterminer automatiquement les graphes procéduraux et les valeurs de leurs paramètres à partir d'exemples de textures ou de matériaux. Les approches actuelles reposent sur des collections de textures et de matériaux et des algorithmes de classification, supervisée ou non, mais ne garantissent pas une ressemblance visuelle avec les exemples en entrée du fait de leur grande diversité possible.

## 2.2 Objectif de la thèse et contributions

L'objectif des travaux de cette thèse est de favoriser le passage à l'échelle de la synthèse de textures stochastiques structurées. Nous avons plus particulièrement étudié le cas de textures à deux motifs dominants (par exemple, de l'écorce d'arbre, de la tôle avec des tâches de rouille, une chaussée constituée de pavés, etc.). Nous proposons une nouvelle méthode de synthèse de textures et de matériaux par l'exemple qui s'appuie sur un modèle procédural original pour représenter la structure, que nous définissons dans ces travaux comme la répartition spatiale des motifs dans le plan de la texture. Les contributions de cette thèse comprennent :

- un modèle procédural de structure pour des textures à deux motifs dominants ;
  - une méthode de synthèse de textures et de matériaux semi-procédurale ;
  - une collection de 150 cartes de structure extraites de textures ;
- ainsi que le code source de la méthode développée mis à disposition de la communauté. Notre méthode prend en entrée un exemple de texture, et une

carte de labels binaire dont chaque pixel est associé à un motif de l'exemple d'entrée. La sortie est une texture de taille arbitraire pouvant être générée à la volée, dont la structure peut être éditée en temps réel. La structure procédurale générée sert de support à la synthèse de la couleur de la texture ou des propriétés des matériaux, selon une approche par l'exemple guidée par les statistiques locales (figure 2.1).

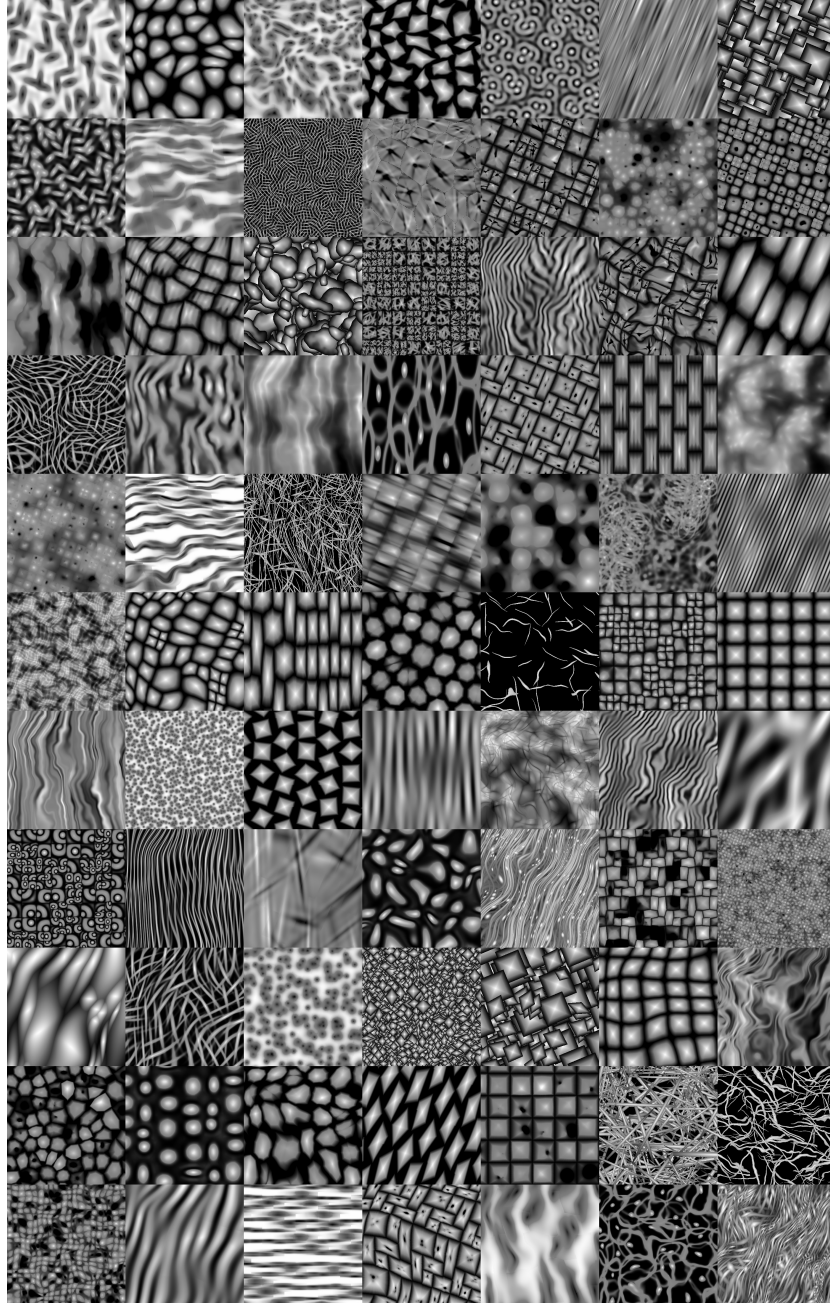


**FIGURE 2.1** – Modèle de textures semi-procédurales. Séparation de la synthèse de structure de type procédurale, de la synthèse de la couleur par pixel. À partir d'une texture d'exemple et sa carte de structure (segmentation manuelle), on détermine le modèle procédural le plus proche (estimation de ses paramètres) puis sa structure par seuillage. Ensuite vient la phase de colorisation (post-traitement).

La méthode proposée tire ainsi parti du meilleur du monde de la synthèse par l'exemple et de la synthèse procédurale : elle offre à la fois des résultats fidèles aux exemples, une représentation à faible coût de stockage, la possibilité de synthétiser à la volée, et un degré élevé de contrôle. Notre contribution peut ainsi être vue comme un modèle d'apparence permettant de décrire de façon semi-procédurale les détails d'une scène virtuelle 3D qui interagissent avec la lumière, ce que nous nommons modèle d'apparence semi-procédural.

## 2.3 Modèle procédural de structure

Notre modèle procédural de structure utilise une nouvelle fonction de bruit générique appelée PPTBF (*Point Process Texture Basis Function*), sans limite (infinie), sans répétition, cohérente dans tout le plan 2D, éditable et contrôlable. Notre modèle se fonde sur une observation simple : les textures contiennent des structures de base (par exemple : cellules, fissures, tâches) qui peuvent être caractérisées par des images binaires contenant des éléments eux-mêmes caractérisés par leurs distributions, interactions et formes visuelles. Tout ceci peut être reproduit procéduralement par notre modèle, qui englobe les modèles de bruits précédents et permet de simuler des phénomènes plus riches et complexes comme des distributions de points non-uniformes (par exemple : aléatoire complet, clusters, attraction/répulsion, quasi-régulier), des formes étendues (par exemple : anisotropie, épaisseur), des mixtures d'éléments (exemple : mélange, empilement), avec possibilité d'organisation spatiale de plus haut niveau (par corrélations des noyaux) (figure 2.2). Concernant l'évaluation de la capacité de notre modèle à reproduire des structures, nous montrons comment, à partir d'une image, estimer ses paramètres afin d'obtenir un modèle procédural perceptuellement similaire. Différentes approches sont proposées : automatique, semi-automatique (guidée par l'utilisateur), ou par édition manuelle d'outils interactifs, grâce à la mise en place d'une métrique de similarité perceptuelle entre textures et la création d'une banque de descripteurs d'images de structures binaires par échantillonnage de l'espace des paramètres de la PPTBF.

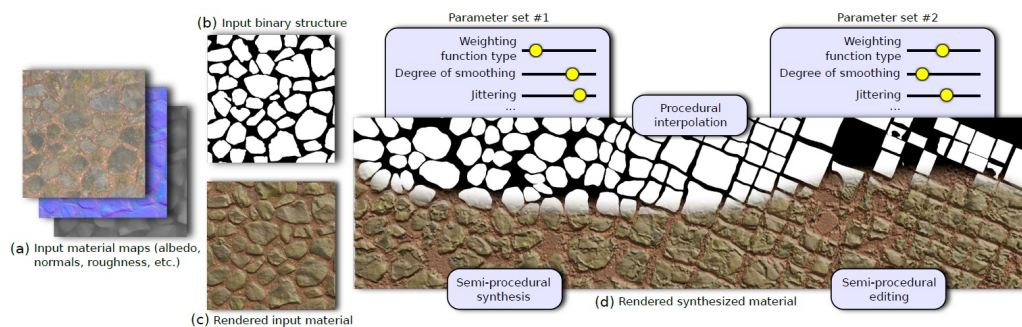


**FIGURE 2.2** – Quelques exemples de PPTBF. Un extrait de notre base de données de PPTBF.

## 2.4 Synthèse de la couleur ou des propriétés des matériaux

La couleur ou les propriétés des matériaux sont synthétisées en utilisant une approche d'optimisation parallèle automatique par l'exemple, engendrant une similitude visuelle avec l'exemple (amélioration de la cohérence de la structure, bonne correspondance visuelle même lorsque la structure n'est pas parfaite), préservant ses propriétés procédurales, et ce, de manière automatisée, rapide (interactive, voire temps réel) et contrôlable (figure 2.3).

Cette thèse s'intéresse également à la gestion et à la reproduction automatique de l'organisation spatiale de leurs motifs, notamment en termes de relation d'adjacence et de leurs transitions. L'utilisation de cartes de labels permet de segmenter les motifs des textures en entrée et de guider les méthodes de synthèse. Cependant, peu de méthodes se sont intéressées à la synthèse automatique de cartes de labels. Nous montrons comment notre modèle de textures semi-procédurales permet de synthétiser des cartes de labels à la volée et s'en servir comme guide lors de la phase de l'optimisation des détails d'apparence (couleur et autres propriétés des matériaux). Pour des arrangements de labels plus complexes, il est nécessaire d'aller plus loin et d'étudier des méthodes plus avancées, potentiellement plus gourmandes en temps de calcul et consommation mémoire. Nous présentons quelques résultats de travaux en cours.



**FIGURE 2.3** – Synthèse de matériaux procéduraux par l'exemple variant spatialement.



## 2.5 Publications, collaborations et valorisation

Ce travail de thèse a conduit aux communications suivantes.

### Article dans une revue internationale avec comité de lecture

[GAD<sup>+</sup>20] *Semi-Procedural Textures Using Point Process Texture Basis Functions*, Computer Graphics Forum, en collaboration avec B. Benes au HPCG (High-Performance Graphics Group, Purdue, USA) et E. Galin au LIRIS (CNRS, France).

### Communication dans une conférence internationale avec comité de lecture

[GAD<sup>+</sup>20] *Semi-Procedural Textures Using Point Process Texture Basis Functions*, EGSR 2020, 29 juin au 3 juillet 2020, en collaboration avec B. Benes au HPCG (High-Performance Graphics Group, Purdue, USA) et E. Galin au LIRIS (CNRS, France). **Prix : Mention Honorable décernée par le comité de sélection du meilleur article**

### Communications dans une conférence nationale sans comité de lecture, avec actes

- (i) *Semi-Procedural Textures Using Point Process Texture Basis Functions*, JFIG 2020 (Journées Françaises d’Informatique Graphique), 27 avril 2020, LORIA (Nancy, France).
- (ii) *Procedural Texture Extrapolation*, Journée thématique du GdR IG-RV, 4 avril 2019, LIRIS (CNRS, Lyon, France).

### Valorisation

Des efforts importants de développement ont permis de diffuser le code source lié à cette thèse, ainsi que différentes interfaces d’utilisation, sous licence open source. Un dépôt *GitHub* public dédié a été créé<sup>1</sup>.

Afin d’augmenter la visibilité du projet, nous avons obtenu le label de *reproductibilité*. Plus de détails sont disponibles sur le site du *Graphics Replicability Stamp Initiative*<sup>2</sup>.

---

1. <https://github.com/ASTex-ICube/semiproctex>

2. <http://www.replicabilitystamp.org/>

## Projet de Recherche ANR

Ce travail de thèse a permis de collaborer avec des acteurs du projet de recherche ANR *HDWorlds (Huge Digital Worlds)*<sup>3</sup>. Ce projet s'intéresse au fait de produire des modèles 3D massifs représentant des mondes virtuels à grande échelle avec un niveau de détails élevé, ce qui est un défi majeur en infographie. L'objectif du projet de recherche HDWorlds est de surmonter cette limitation afin de synthétiser de grandes scènes détaillées en utilisant une approche procédurale.

## Communications nationales sans comité de lecture et sans acte (présentations, vulgarisation scientifique)

- (i) *Semi-Procedural Textures Using Point Process Texture Basis Functions*, Les journées du Campus d'Illkirch, 19 avril 2021, en ligne.
- (ii) *Creative AI for Texture Synthesis*, Data Science and Artificial Intelligence Workshop, 7 novembre 2019, laboratoire ICube (Strasbourg, France).
- (iii) *Extrapolation de textures pour la création de mondes virtuels de film ou de jeu vidéo*, Séminaire des doctorantes et doctorants en informatique, SIF-doctorants, 2 juillet 2019, Paris.
- (iv) *Recherche en Informatique Graphique*, Présentation à des lycéens, 28 mars 2019, laboratoire ICube (Strasbourg, France).
- (v) *Mixing graphics and compute : texture synthesis and out-of-core on-demand production caching library*, development of scientific computing software at ICube research Lab [Strasbourg, 7th March 2019]
- (vi) *Management and control of Appearance for texture synthesis* - PhD Meeting, laboratoire ICube [Strasbourg, 22 janvier 2019]
- (vii) *Statistical and procedural representations for the control and scalability of texture/material synthesis*, Journée "Poster" de l'école doctorale MSII (Mathematics, Sciences of the Information and the Engineer), 2 octobre 2017, Strasbourg, France.
- (viii) *Démonstrations logicielles et mini-exposé aux étudiants de 1ère année de Licence de Maths-Informatique*, janvier 2018/2019/2020, laboratoire ICube (Strasbourg, France).

---

3. <http://hdworlds.unistra.fr/index.php/Accueil>

## 2.6 Plan de la thèse

Après cette **Introduction** aux problématiques, verrous scientifiques et techniques associés à la synthèse de textures et de matériaux, le plan du manuscrit se poursuit comme suit :

**Partie 2 : État de l’Art.** Dans cette partie, on présente un état de l’art des techniques de génération de contenu associée aux textures et matériaux. Plus précisément, le chapitre 3 traite des *Modèles de Textures et Différentes Approches de Synthèse*. Le chapitre 4 traite du *Contrôle de la Synthèse des Textures et Matériaux Structurés et Non Homogènes*. Le chapitre 5 traite de la *Modélisation Procédurale Inverse de Textures et de Matériaux*. Le chapitre 6, *État de l’art : Conclusion*, conclut cette partie.

**Partie 3 : Modèle d’Apparence Semi-Procédural.** Cette partie est consacrée aux travaux de recherche et contributions réalisés durant cette thèse.

(i) **Approche Semi-Procédurale.** Le chapitre 7, *Approche Semi-Procédurale*, présente un aperçu de notre modèle stochastique procédural d’apparence. Les chapitres qui suivent traitent de chacun de ses composants.

(ii) **Synthèse de Structures.** Le chapitre 8 présente un aperçu de notre *Modèle de Structures Procédurales PPTBF* (Point Process Texture Basis Functions), et le chapitre 9 une *Évaluation du Modèle Procédural de Structures PPTBF* illustrant la gamme de structures synthétisables. Le chapitre 10, *Estimation des Paramètres des Structures Procédurales PPTBF*, présente notre approche de modélisation semi-procédurale inverse.

(iii) **Synthèse de Textures et Matériaux.** Le chapitre 11 traite de la *Synthèse de Textures et Matériaux Semi-Procéduraux*, et le chapitre 12 présente une *Évaluation du Modèle Semi-Procédural* illustrant la gamme de textures et matériaux synthétisables.

**Partie 4 : Conclusion.** Le chapitre 13 dresse le bilan des travaux réalisés durant cette thèse, et le dernier chapitre 14 traite des perspectives possibles de poursuite de ces travaux.

Deuxième partie

État de l'Art

# Chapitre 3

## Modèles de Textures et Différentes Approches de Synthèse

Dans ce chapitre, nous présentons un état de l'art de la représentation et de la synthèse de textures, puis son extension aux matériaux. Nous portons une attention particulière à la synthèse par l'exemple, et à la représentation et à la prise en compte de la *structure*. Nous distinguons les méthodes de synthèse *non paramétriques* et les méthodes *procédurales*. Dans le chapitre suivant (chapitre 4), nous présentons les méthodes de l'état de l'art proposées pour prendre en compte les variations spatiales propres aux textures structurées et non homogènes. Nous traitons ensuite d'estimation des paramètres de modèles procéduraux à partir d'images (chapite 5).

### 3.1 Introduction

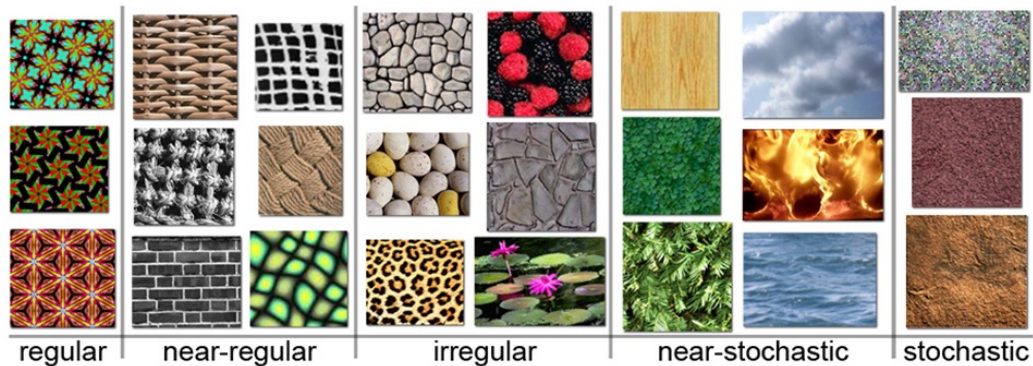
#### 3.1.1 Notion de texture et taxonomie

**Qu'est-ce qu'une texture ?** La notion de *texture* n'est pas générique et commune, elle n'admet pas de définition consensuelle. En effet, différentes communautés (vision par ordinateur, informatique graphique, artistes) proposent différents points de vue, vocabulaires, algorithmes et outils. Même si nous adoptons ici le point de vue de l'informatique graphique, nous aborderons quelques travaux issus d'autres communautés ayant des liens avec la

synthèse de textures.

Dans [WLKT09], *State of the art in example-based texture synthesis*, Wei *et al.* définissent une texture comme une image contenant des *motifs* élémentaires qui se répètent. Un motif peut aussi bien être défini par des caractéristiques géométriques que statistiques, en termes de distribution de couleurs par exemple, avec une certaine *homogénéité*. La notion d’homogénéité est dépendante de l’échelle d’observation. À différentes échelles d’observation, un même motif peut avoir une apparence plus ou moins homogène. On considérera ici que l’échelle d’observation est fixée de telle sorte qu’un motif est homogène. On parlera de *textures homogènes* pour qualifier des textures répétant un unique motif, avec la possibilité d’avoir des variations aléatoires ne changeant pas l’apparence globale de la texture. On parlera de textures *non homogènes* pour des textures comportant plusieurs motifs visuellement distinguables.

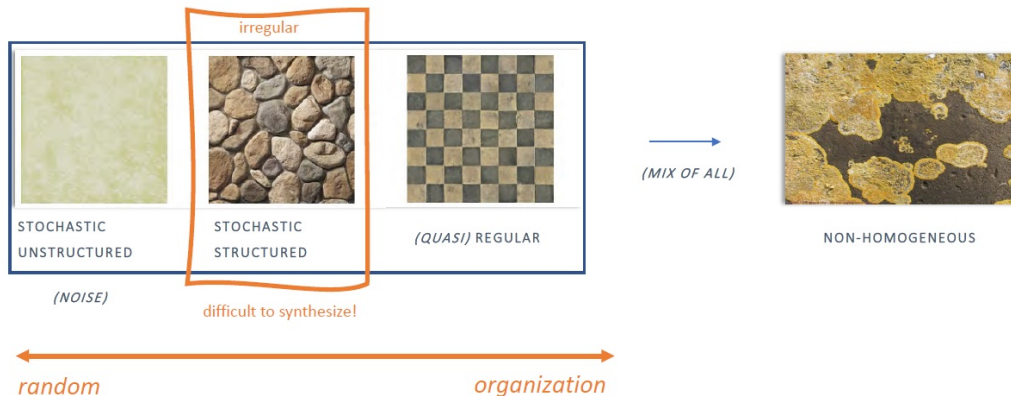
Il existe différents types de textures et différentes classifications possibles. Nous reprenons la classification proposée par Lin dans [Lin04] (figure 3.1) : des plus régulières et quasi-régulières, avec pour certaines des arrangements spatiaux particuliers, aux plus irrégulières, aux plus aléatoires ou *stochastiques*.



**FIGURE 3.1** – Différents types de textures. Quelques exemples de textures des plus structurées aux plus stochastiques (ou aléatoires), voir [Lin04].

L’HDR de Sylvain Lefebvre [Lef14] propose une classification plus nuancée autour de la notion de structure et d’organisation spatiale : régulier et quasi-régulier, structuré organisé, stochastique structuré, stochastique non structuré.

Nous nous intéressons plus spécifiquement aux textures stochastiques *structurées*, illustrées dans la figure 3.2. Celles-ci présentent un aspect global aléatoire, mais avec des structures à une échelle plus fine. Les textures non homogènes, plus complexes, sont caractérisées par différents motifs (figure 3.2 à droite, avec le lichen et la surface sur laquelle il se développe).



**FIGURE 3.2** – Différents types de textures. Quelques exemples de textures des plus structurées aux plus stochastiques (ou aléatoires).

**Qu'est-ce que la structure ?** Dans notre contexte, c'est une notion visuelle, associée à certaines variations spatiales dans les textures, correspondant à des transitions, séparations, ou frontières plus ou moins marquées entre des motifs. La structure peut résulter de motifs similaires possédant des bords francs, tout en étant dans une texture homogène à l'échelle d'observation, comme les galets de la figure 3.2. Dans une texture non homogène, comme le lichen dans la même figure, la structure est la conséquence du passage d'un motif à un autre (du lichen vers son support). La notion de structure peut aller plus loin et prendre en compte l'*organisation spatiale* des motifs, autrement dit leurs relations d'adjacences, à plus ou moins grande distance. Dans ce manuscrit, nous nous limiterons à une prise en compte de la structure comme une partition du plan en régions correspondant à des motifs similaires ou différents.

La principale caractéristique des textures stochastiques structurées est que la structure comporte des variations spatiales aléatoires, qui ne peuvent pas être décrites de façon simple, comme c'est le cas par exemple pour un



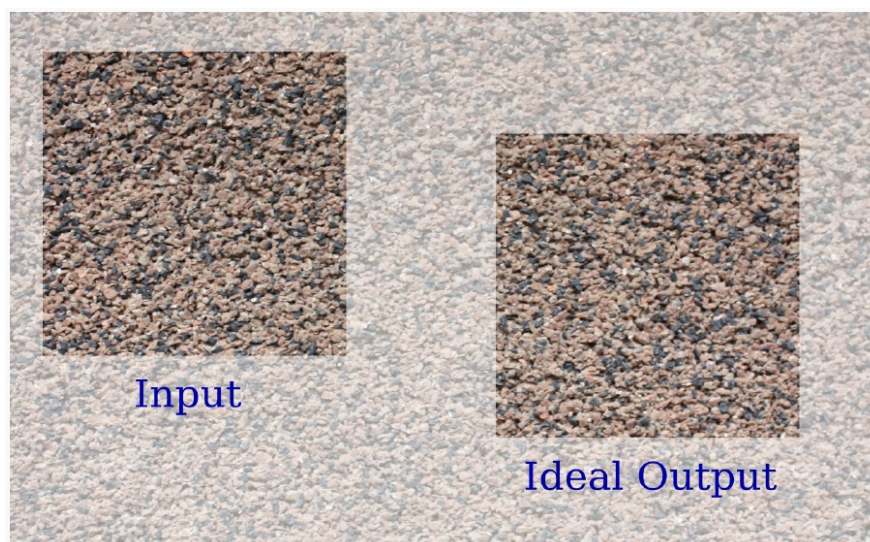
damier. Une solution proposée en synthèse de textures par l'exemple consiste à utiliser des cartes de textons ou de labels pour *guider* spatialement la synthèse. Un des problèmes consiste alors à produire ces guides spatiaux.

### 3.1.2 Modèles de texture

Différents modèles ont été proposés pour représenter les textures. Nous distinguerons les approches *statistiques* et les approches *structurelles*.

#### Approches statistiques

Les approches statistiques correspondent à une approche classique en vision par ordinateur, traitement du signal, statistiques et physique statistique. Une texture est représentée comme la réalisation d'un processus stochastique, dont les algorithmes de génération peuvent se caractériser ainsi : étant donné une image de texture d'entrée, le but est de produire une texture étant à la fois visuellement similaire et différente au niveau des pixels de la texture d'entrée, afin d'introduire de la variété. L'image de sortie est idéalement perçue comme une autre partie du même grand morceau de matériau homogène d'où provient la texture d'entrée (figure 3.3).



**FIGURE 3.3** – Synthèse de textures par l'exemple. Texture vue comme la réalisation d'un processus stochastique.



Afin de pouvoir modéliser et générer de nouvelles textures, l'idée est de faire des hypothèses de stationnarité et d'ergodicité. Dans ces modèles, on trouvera ceux issus des MRF (*Markov Random Fields*) et les modèles de *bruits procéduraux*, notamment les fonctions de bruits. Dans les approches par MRF, les algorithmes de génération suivent des hypothèses de stationnarité et de localité. La stationnarité indique une similarité si l'on considère une fenêtre d'observation glissante sur la texture. La localité indique que la valeur d'un pixel, comme sa couleur, ne dépend que de ses pixels proches et est indépendant du reste de la texture. On verra que les hypothèses liées au modèle MRF contraignent et limitent la modélisation de textures stochastiques structurées.

Dans les approches par bruits procéduraux, une hypothèse couramment formulée est la caractérisation par la densité spectrale de puissance (PSD, *Power Spectral Density*) [LLC<sup>+</sup>10], qui correspond aux textures *gaussiennes*. Cette hypothèse a permis de développer des algorithmes et des représentations spécifiques utilisant l'analyse de Fourier. Nous verrons que cette représentation ne permet de prendre en compte les structures que de façon limitée.

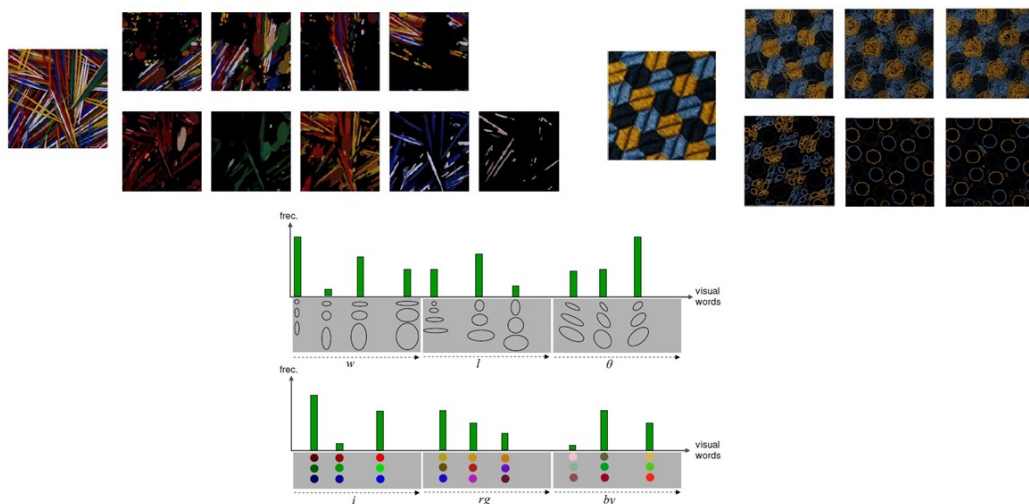
L'utilisation de statistiques d'ordre plus élevé a été proposée pour représenter des textures structurées, comme l'autocorrélation, les matrices de co-occurrence, ou les matrices d'aura [QY05]. Ces représentations restent peu adaptées à la représentation de structures stochastiques structurées.

Des modèles à base de réponses de filtres ont été proposés pour tenter de capturer et reproduire des statistiques d'ordre supérieur, par exemple des ondelettes à différents niveaux d'échelles. Avec le développement des réseaux de neurones convolutionnels dans les années 2010, de nouvelles représentations statistiques ont vu le jour, s'appuyant sur des descripteurs appris à plusieurs niveaux d'échelles sur de grandes collections d'images. Ces représentations ont fait leurs preuves en synthèse de textures, mais, comme nous le verrons, elles restent limitées par leurs performances, la résolution des images produites et les possibilités de contrôle du résultat.

## Approches structurelles

Une première approche structurelle consiste à représenter une texture comme une composition d'éléments de formes distinguables, distribués selon des règles de positionnement spatial. Dans [Jul81], *Textons, the elements of texture perception, and their interactions*, Julesz présente des expériences

psychovisuelles pour discriminer les textures. Il revient sur une de ses hypothèses précédentes, à savoir que des textures avec des statistiques d'ordre 2 identiques ne peuvent être discriminées. Pour cela, il montre que des textures peuvent être discriminées par le système visuel pré-attentif grâce à des unités élémentaires, appelées *textons*, caractérisées par des formes allongées de largeurs spécifiques, orientations et différents ratios de tailles, et leurs terminaisons, leur statistique du premier ordre ayant un rôle perceptuel important. Dans [AV12], *Texton theory revisited : A bag-of-words approach to combine textons*, Alvarez et Vanrell proposent de déterminer la composition d'une image/texture sous la forme d'un dictionnaire de textons, dont les variations d'attributs tels que leurs formes, tailles, longueurs, orientations et couleurs sont apprises par une quantification de l'espace des paramètres. C'est une représentation par sac de mots (*bag of words*), soit des histogrammes d'occurrences de mots ou caractéristiques locales visuelles, couramment utilisé en recherche d'images par le contenu et en classification d'images (figure 3.4).



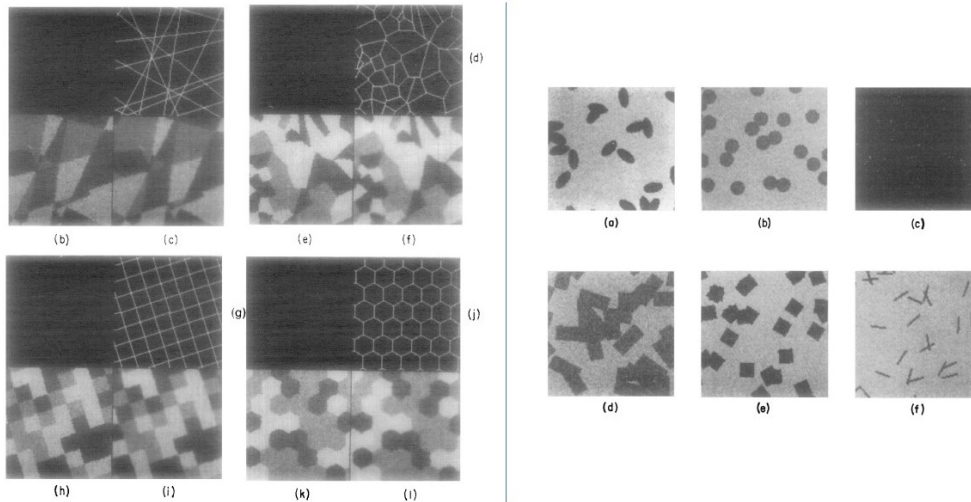
**FIGURE 3.4** – Descripteur de textons perceptuels. Caractéristiques perceptuelles des textures encodées par des combinaisons de textons (*p-blobs*), selon des paramètres de forme, taille et couleur [AV12].

Ils notent qu'une quantification uniforme de l'espace des paramètres ne prend pas en compte certaines corrélations perceptuelles entre eux, ce qui diminue le pouvoir discriminant du descripteur. Ce problème de schéma d'échantillonnage perceptuel se retrouve dans la plupart des travaux de ce type, à base

de descripteurs et de métrique de similarité entre eux, requérant de mettre en place des stratégies plus évoluées ou empiriques.

La notion de *texton*, comme motif élémentaire se répétant, a été reprise dans de nombreux travaux en informatique graphique qui se sont intéressés à des textures stochastiques structurées. En particulier, les idées de *masques de textons* [ZZV<sup>+</sup>03], et de *cartes de labels* [HJO<sup>+</sup>01] pour des textures non homogènes, ont été introduites pour guider la synthèse de textures en s'appuyant sur des algorithmes initialement conçus pour des textures homogènes et non structurées.

**Processus aléatoires** Dans [SA79], *Random Pattern Generation Processes*, Schachter et Ahuja s'intéressent à l'analyse des *motifs* présents dans les images, plus particulièrement leur modélisation par des processus géométriques aléatoires (figure 3.5). Les auteurs proposent également de caractériser les éléments géométriques par différents descripteurs statistiques, tels que l'autocorrélation. D'autres travaux de recherche traitent de sujets similaires, comme dans [AR81], *Mosaic Models for Textures*, de Ahuja et Rosenfield, où ils englobent plusieurs modèles dans la catégorie des mosaïques. Ce type de modèle est limité dans la variété des textures qu'il peut représenter, et aucune méthode de synthèse par l'exemple n'a été proposée.



**FIGURE 3.5** – Modèles de motifs aléatoires. Différentes modélisations de processus aléatoires pour générer des structures cellulaires et de bombing (par empilement) [SA79].

## 3.2 Génération non paramétrique

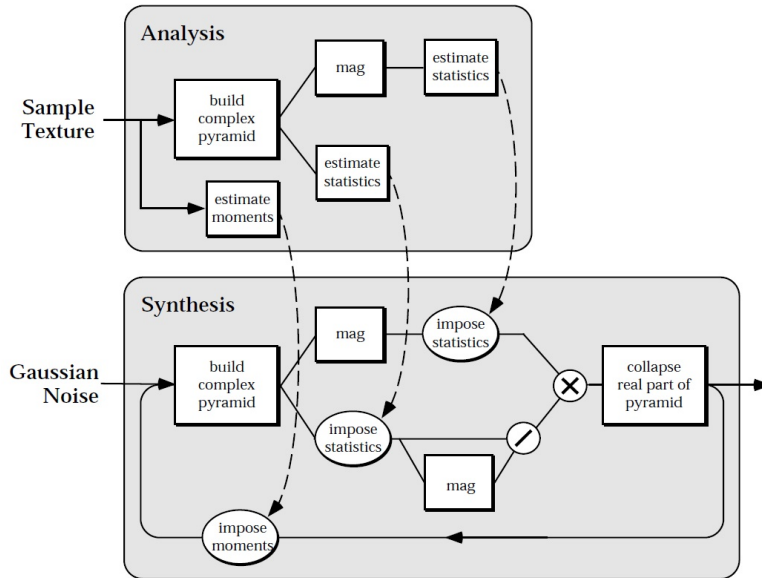
On entend par méthodes *non paramétriques* les méthodes cherchant à générer des textures ressemblant à un exemple donné en entrée, soit sur la base de propriétés statistiques, soit selon une approche géométrique, cherchant à éviter ou minimiser des artefacts visuels comme les répétitions, alignements et discontinuités. Dans la suite, on liste quelques travaux majeurs dans le domaine en rapport avec nos travaux.

### 3.2.1 Approches statistiques Multi-Échelle

Ce type de méthodes génère des textures à l'aide de contraintes statistiques. L'algorithme commence par extraire des statistiques significatives de l'image d'entrée (exemple : la distribution de couleurs, de coefficients de Fourier, de coefficients d'ondelettes, de réponses de filtres multi-échelle et multi-orientations, etc.). Ensuite, il calcule une image de sortie, initialisée par un bruit blanc, en imposant les statistiques de l'entrée itérativement (*algorithme multi-passes*).

Dans [HB95] *Pyramid-based Texture Analysis/Synthesis*, Heeger et Bergen choisissent de générer des textures encodées sous la forme d'histogrammes de réponses de filtres. Une des limitations est liée à une hypothèse forte de textures homogènes en entrée. Par la suite, [PS99] dans *Texture modeling and synthesis using joint statistics of complex wavelet coefficients*, Portilla et Simoncelli utilise une décomposition sur une base d'ondelettes multi-échelle avec de meilleures propriétés d'analyse et d'extraction du contenu spatial et fréquentiel des textures d'entrée, tout en imposant des contraintes croisées entre échelles et entre différentes caractéristiques statistiques des textures, telles que les quatre premiers moments : moyenne/espérance, variance, coefficient d'asymétrie et le kurtosis (très fréquent dans la nature) (voir le diagramme de l'algorithme 3.6). Ils utilisent des statistiques d'ordre 2 pour capturer les structures locales. Mais il reste de problèmes pour capturer la localisation des phases, donc des structures ([OL81] *The importance of phase in signals*). De plus, l'algorithme ne converge pas forcément.

Ces approches, notamment celle de [PS99], sont très proches des algorithmes par apprentissage profond (deep learning), à l'exclusion des filtres (ondelettes) qui ne sont plus donnés manuellement, mais générés automatiquement par optimisation.



**FIGURE 3.6** – Approche par statistiques multi-échelle et ondelettes. Diagramme de l’algorithme de synthèse par statistiques multi-échelle et ondelettes [PS99].

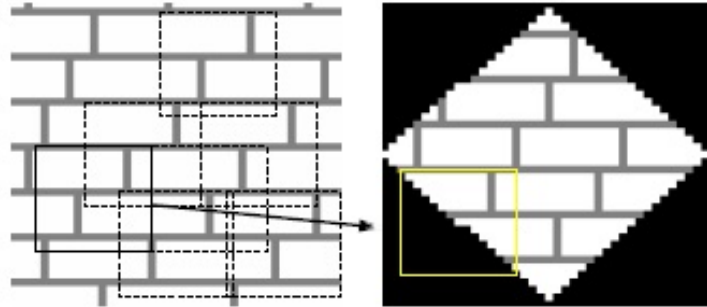
### 3.2.2 Approches par pixel et par patch

Les méthodes par pixel et par patch consistent, algorithmiquement, à générer une nouvelle image par des assemblages et réarrangements du contenu d’une image donnée en entrée. Les algorithmes cherchent à maximiser les ressemblances locales entre l’image générée et celle donnée en entrée, en suivant les hypothèses de stationnarité et de localité des modèles MRF.

#### Approches par pixel

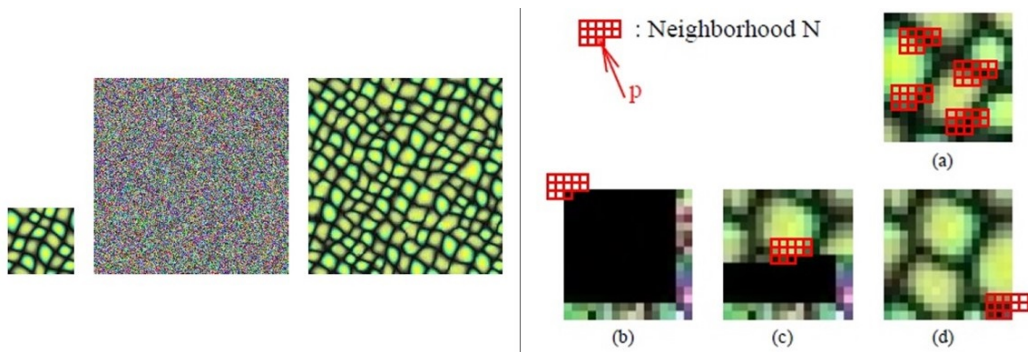
Les méthodes par pixel consistent à générer des textures, en suivant un ordre prédéfini, principalement séquentiellement, pixel par pixel.

Dans [EL99] *Texture synthesis by non-parametric sampling*, Efros et Leung génèrent des textures de manière progressive, pixel par pixel, en choisissant le pixel de l’entrée, dont le voisinage est le plus similaire à celui de la sortie en cours de synthèse (figure 3.7).



**FIGURE 3.7** – Génération par échantillonnage de pixels. Génération d’une texture (à droite) à partir d’un exemple (à gauche), en copiant pixel par pixel en sortie, le pixel en entrée dont le voisinage est le plus similaire dans la sortie déjà synthétisée [EL99].

Dans [WL00], *Fast Texture Synthesis Using Tree-structured Vector Quantization*, Wei et Levoy améliorent l’approche en accélérant la synthèse grâce à une structure accélératrice de recherche de plus proches voisins dans un espace multidimensionnel de voisinages. L’algorithme, initialisé par un bruit aléatoire, prend également mieux en compte les structures larges, en utilisant une approche multi-échelle allant du plus grossier au plus fin (figure 3.8).

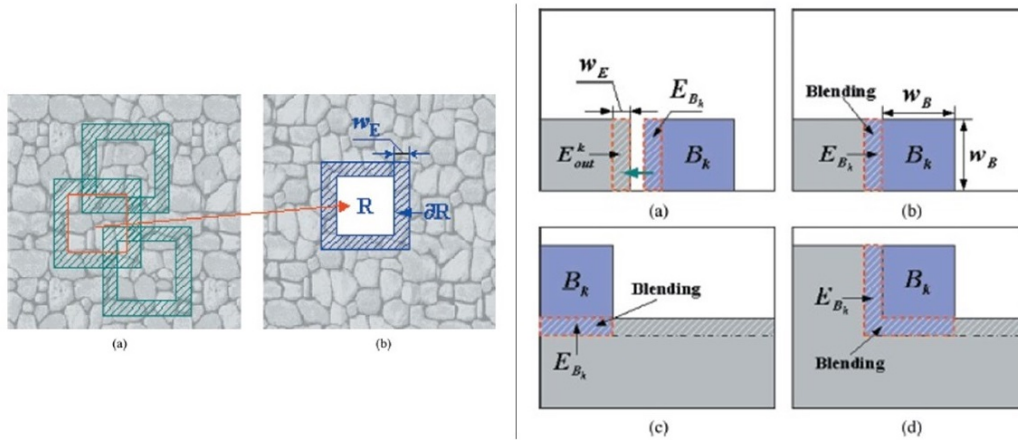


**FIGURE 3.8** – Génération par échantillonnage de pixels. À gauche : génération d’une texture (à droite) à partir d’un exemple (à gauche), en modifiant itérativement une texture de bruit (au milieu) afin qu’elle ressemble visuellement à l’entrée. À droite : synthèse par recherche de plus proches voisins (meilleurs candidats) [WL00].

## Approches par patch

Les méthodes par patch sont basées sur les mêmes principes que les méthodes par pixel, mais copient des régions de pixels, appelés patches, à la place. Elles permettent de mieux conserver la similarité locale avec l'image d'entrée. Les méthodes se distinguent par la taille des patches, leurs formes et leurs stratégies pour éviter les discontinuités visibles entre eux.

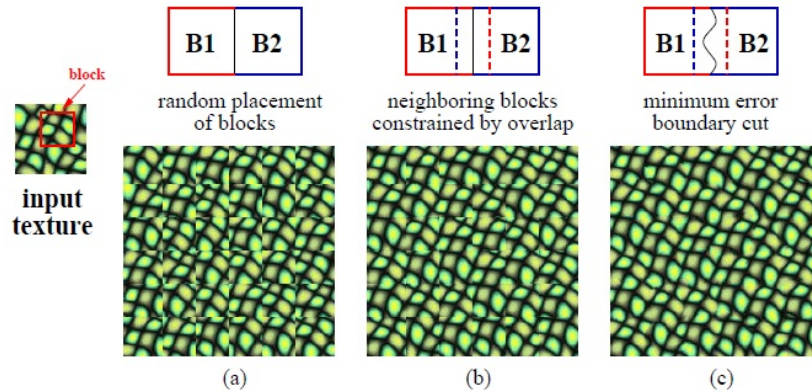
Dans [LLX<sup>+</sup>01] *Real-time texture synthesis by patch-based sampling*, Liang *et al.* proposent de générer une texture par patch afin de renforcer la similarité visuelle locale. Chaque patch possède une taille donnée et est augmenté de *bordures*. Chaque patch est sélectionné parmi une liste de meilleurs candidats selon une métrique pénalisant les dissimilarités dans les zones de recouvrement entre patches (figure 3.9). Une fonction de mélange (blending) est appliquée sur les zones de recouvrement afin de diminuer les discontinuités.



**FIGURE 3.9** – Patch-based sampling. À gauche : recherche de candidats satisfaisant les contraintes de recouvrement sur les bordures. À droite : mélange des tuiles sur les zones de recouvrement [LLX<sup>+</sup>01].

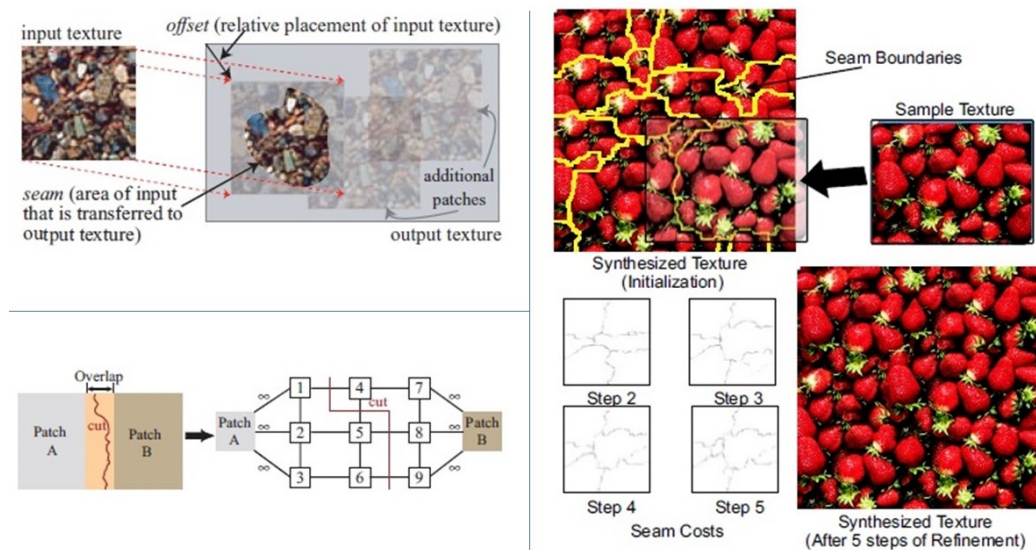
Dans [EF01], *Image Quilting for Texture Synthesis and Transfer*, Efros et Freeman améliore les résultats en effectuant des coutures optimales dans les zones de recouvrement, au lieu de masquer les artefacts par des mélanges (figure 3.10).





**FIGURE 3.10** – *Image Quilting*. Les tuiles qui se recouvrent sont cousues par une couture optimale dans les zones de recouvrement [EF01].

Dans [KSE+03], *Graphcut Textures : Image and Video Synthesis Using Graph Cuts*, Kwatra *et al.* généralisent l’approche par patch en autorisant des tailles et formes arbitraires de patch. Ceci permet de mieux s’adapter au contenu de l’image d’entrée et éviter la distinction des pavages en grille régulière sous-jacents des méthodes par patch précédentes (figure 3.11).



**FIGURE 3.11** – *Graphcut*. Formes et tailles de patch arbitraires. Coupe optimale sur un graphe où les pixels des zones de recouvrement sont les noeuds [KSE+03].



### Approche hybride : par pixel et par patch

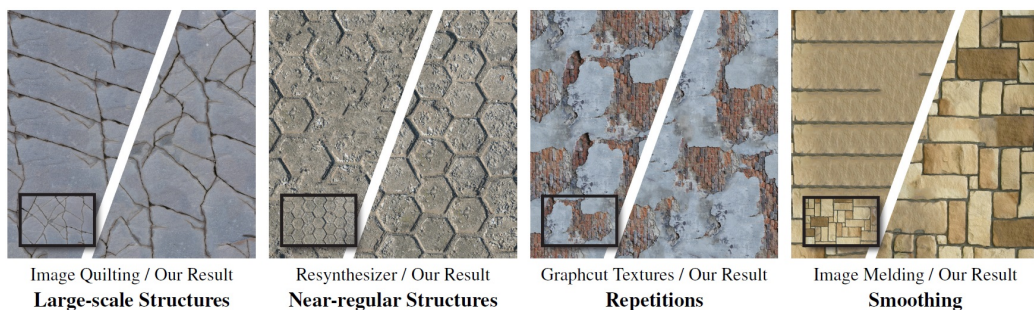
Dans [KEBK05], *Texture Optimization for Example-based Synthesis*, Kwatra *et al.* proposent une approche par pixel, qui va également, entre autre, favoriser, sous contraintes, la création de larges zones issues de l'entrée, sous forme de patches. Visuellement, les patches ressemblent aux coutures optimales génériques de la méthode *Graphcut* [KSE+03] (figure 3.12).



**FIGURE 3.12** – Texture Optimization : favoritisation de copie de larges régions de l'entrée. Exemple d'entrée avec son espace de coordonnées de textures, et sa synthèse favorisant la copie de larges régions [KEBK05].

Toute la texture est générée de manière globale, par un mécanisme d'optimisation basé sur l'algorithme *EM* (*Expectation-Minimization*) [DLR77]. L'algorithme en 2 passes permet de sélectionner des pixels selon une fonction de coût avec des considérations locales et globales afin de permettre de reproduire des variations à plus grande échelle dans l'image d'exemple.

Dans [KNL+15], *Self Tuning Texture Optimization*, Kaspar *et al.* améliorent l'approche [KEBK05] en sélectionnant automatiquement ses paramètres, réduisant les répétitions, ajoutant un mécanisme d'initialisation intelligente (*smart initialization*), cherchant à faire correspondre une grille sur certaines caractéristiques de la texture, utile pour traiter les textures quasi-régulières à régulières sans pénaliser les textures sans régularité particulière. Elle utilise également un histogramme global pour mieux respecter les fréquences des couleurs dans l'entrée. Cette approche est une des méthodes les plus robustes, versatiles et produisant des résultats de qualité. Elle dispose de mécanismes pour s'adapter à la fois aux textures stochastiques et celles plus structurées (quasi-régulières à régulières), ainsi que pour certaines structures étendues (figure 3.13).



**FIGURE 3.13** – Self-Tuning Texture Optimization : qualité, efficacité et robustesse. Problèmes classiques de synthèse et comparaison avec [KNL<sup>+</sup>15].

Cependant, lorsque l'entrée est trop stochastique, son mécanisme recherchant de la régularité à travers une grille peut échouer à reproduire les structures (figure 3.14). Si l'image est trop basse résolution et les structures trop denses, la méthode peut échouer à détecter les contours qui normalement l'aident durant la synthèse. Son temps de génération est trop long (plusieurs minutes pour des images de 1024x1024) pour des applications interactives, voir temps réel.



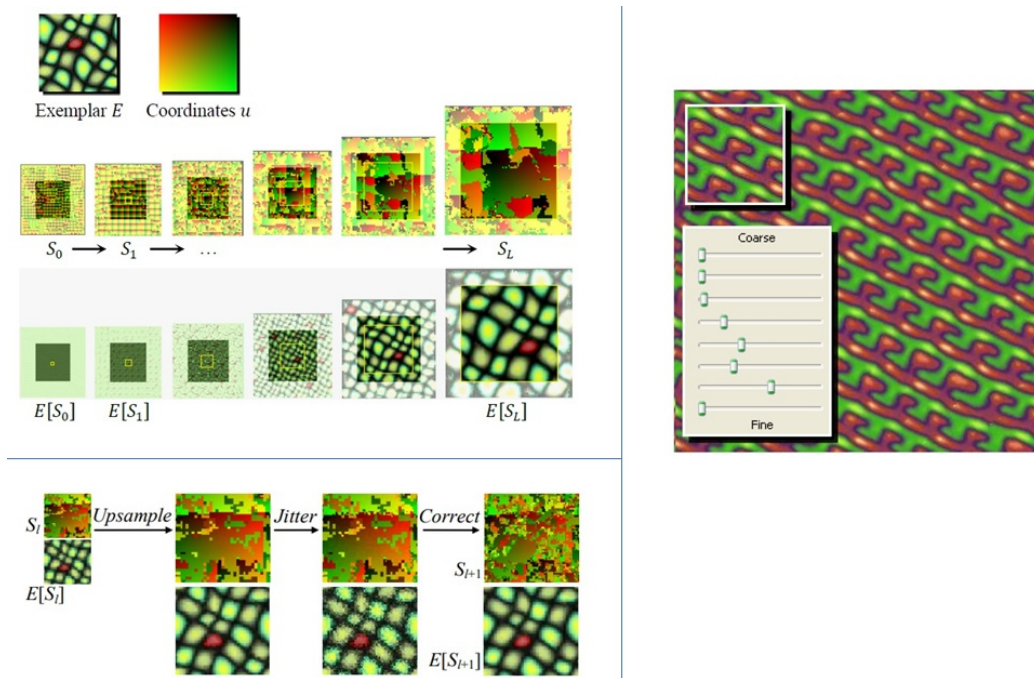
**FIGURE 3.14** – Échec de la reproduction de structures stochastiques [KNL<sup>+</sup>15].

## Améliorations et avancées majeures des méthodes par pixel

On présente ici quatre travaux qui ont eu un impact majeur sur les performances de la synthèse par pixel, notamment pour la paralléliser et l'accélérer sur GPU, que nous appellerons *méthodes par pixel temps réel*.

Dans [WL02], *Order-independent texture synthesis*, Li-Yi et Levoy proposent un algorithme permettant de s'affranchir de la contrainte de générer les textures séquentiellement, ouvrant les portes de la parallélisation de la synthèse.

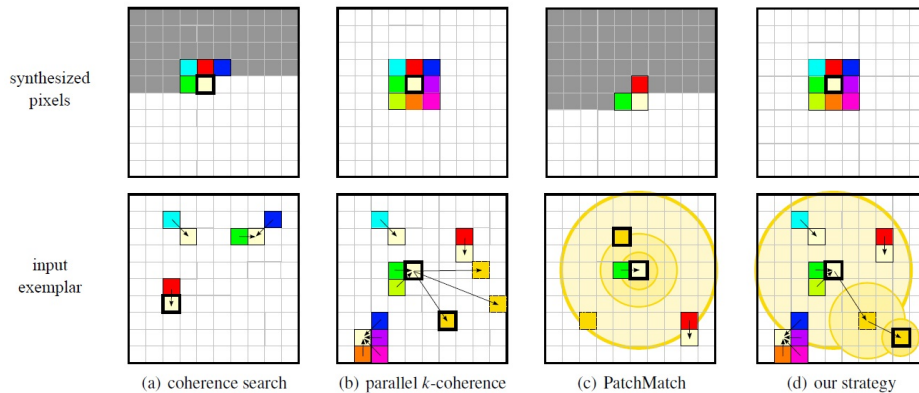
Dans [LH05], *Parallel Controllable Texture Synthesis, PCTS*, Lefebvre et Hoppe proposent un algorithme de synthèse par pixel multi-échelle temps réel accéléré sur GPU (figure 3.15). Une pré-étape d'initialisation permet de rechercher et sauvegarder des informations de plus proches voisins afin d'accélérer la recherche en temps réel.



**FIGURE 3.15** – Synthèse par pixel multi-échelle temps réel sur GPU. En haut à gauche : synthèse multi-échelle. En bas gauche : 3 étapes par niveau d'échelle (upsampling, jittering et correction). Droite : édition multi-échelle de l'aléatoire [LH05].

Dans [LH06], *Appearance-space Texture Synthesis, ASTS*, Lefebvre et Hoppe améliorent PCTS en augmentant les voisinages de couleurs par des informations comme la radiance, de façon à pouvoir synthétiser des matériaux. Les voisinages contiennent plus d’informations locales et à plus grande échelle, et améliorent la cohérence des pixels choisis en sortie. Ces voisinages sont réduits en taille par PCA pour être utilisés en temps réel sur GPU.

Dans [BELS10], *Instant Texture Synthesis by Numbers*, Busto *et al.* améliorent l’approche image analogies et accélèrent les recherches de plus proches voisins, en se passant de l’étape d’initialisation pour enregistrer des voisinages proches, en remplaçant la recherche de type *k-coherence* par une marche aléatoire (*coherent random walk*), consistant à chercher autour de la position en cours en effectuant des sauts de positions aléatoires, dans des cercles de tailles décroissantes (figure 3.16). C’est une extension de *Patch Match*, qui sont les premiers à avoir proposé une marche aléatoire ([BSFG09] *Patch-Match : A randomized correspondence algorithm for structural image editing*, de Barnes *et al.*).

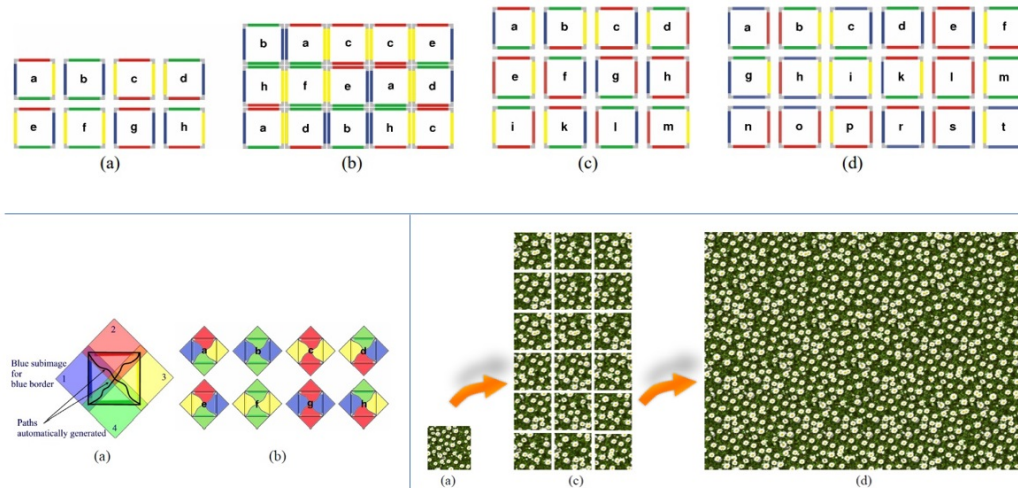


**FIGURE 3.16** – Recherche de pixels candidats accélérée par une marche aléatoire cohérente [BELS10].

Les trois dernières méthodes citées offrent d’excellentes performances pour de la synthèse en temps réel, non bornée spatialement. Pour reproduire des textures stochastiques structurées et des textures non homogènes, elles nécessitent d’être guidées spatialement. Dans nos travaux, nous proposons de générer des cartes de guidage définies de façon procédurale, et nous verrons que nous utilisons ces méthodes pour produire des textures stochastiques structurées en exploitant ces cartes de guidage.

### 3.2.3 Approche par pavage

Dans [CSHD03], *Wang tiles for image and texture generation*, Cohen *et al.* proposent une approche de synthèse par pavage non périodique de tuiles régulières (carrées). L'utilisateur peut ensuite les remplir de textures, motifs, distributions de points, d'objets 3D, etc. Chaque côté d'une tuile possède une couleur, et un pavage valide nécessite de juxtaposer les mêmes couleurs par coté partagé. Concernant la synthèse de textures, les tuiles sont remplies par 4 morceaux de textures tournées à 45 degrés. Les transitions entre tuiles sont générées par Quilting [EF01], avec des coutures optimales (figure 3.17).



**FIGURE 3.17** – Approche par pavage : Wang Tiles. Pavage par Wang Tiles appliqué à la synthèse de textures [CSHD03]. En haut : exemples de Wang Tiles créés à partir de différents nombres de couleurs horizontales et verticales. Bas gauche : confection d'une tuile à partir de 4 morceaux de textures, cousus par Quilting. Bas droite : exemple de synthèse à partir d'une image exemple et 18 Wang Tiles.

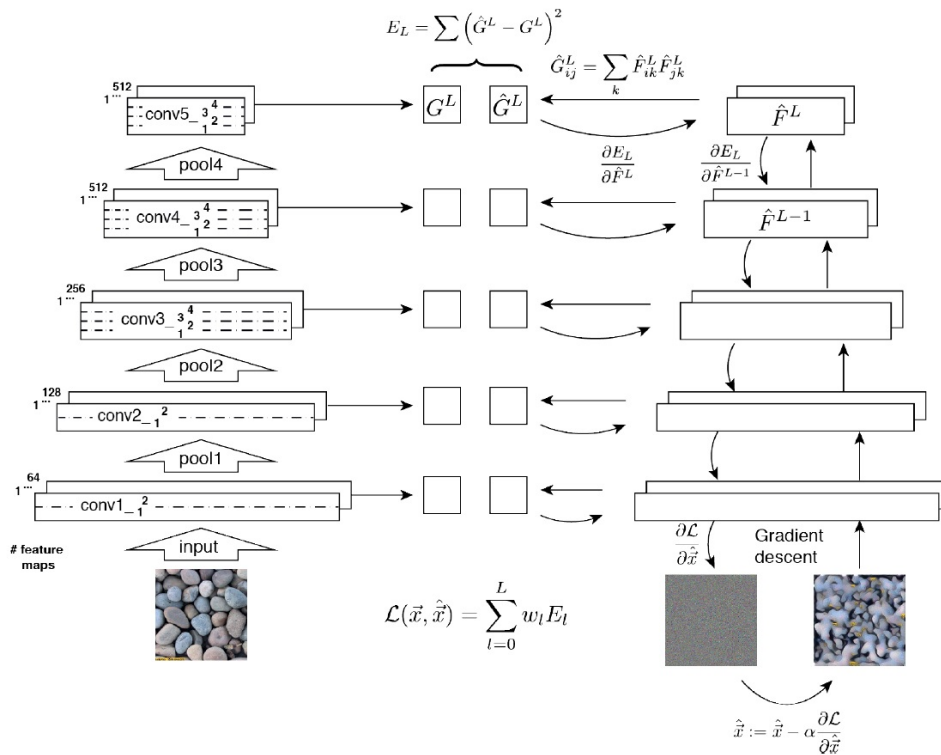
Contrairement aux approches précédentes par pavage périodique de tuiles régulières construit à la volée, ici le pavage est apériodique et pré-calculé. De nombreuses approches par pavage ont été proposées par la suite pour améliorer les Wang Tiles, mais ces travaux dépassent le cadre de cette thèse.



### 3.2.4 Approche par apprentissage profond

Les approches par apprentissage profond (*deep learning*) sont proches des algorithmes statistiques multi-échelle vus précédemment (par exemple [PS99]), générant des textures à l'aide de contraintes statistiques. Cependant, les filtres (comme les ondelettes) ne sont plus donnés manuellement, mais calculés automatiquement par l'entraînement de réseaux de neurones.

Dans [GEB15], *Texture Synthesis Using Convolutional Neural Networks*, Gatys *et al.* proposent une synthèse de texture par des réseaux de neurones convolutionnels (*CNN : Convolutional Neural Networks*), en utilisant les matrices de Gram des cartes de caractéristiques (feature maps) comme descripteurs statistiques de contraintes, pour chacune des couches (layers) du réseau. Elles encodent des relations entre les cartes de caractéristiques, à la manière de statistiques jointes (*corrélations*) (figure 3.18).



**FIGURE 3.18** – Génération de textures par CNN. Optimisation par descente de gradient sur la fonction de coût, liée aux matrices de Gram sur les réponses de caractéristiques (*feature maps*) par couche du CNN. Voir [GEB15].

Dans [SCO17], *Deep Correlations for Texture Synthesis*, Sendik et Cohen-Or étendent le travail précédent [GEB15] en ajoutant des contraintes supplémentaires dans la fonction de coût, par exemple une matrice de structure qui encode cette fois les relations intra-caractéristiques à l'intérieur d'une carte de caractéristiques.

Ces méthodes de synthèse de textures à l'aide de CNN s'appuient sur une description statistique apprise à partir d'ensembles de jeux de données d'entraînement, après un long et coûteux processus. Ces méthodes ne sont pas adaptées aux passages à l'échelle, car elles opèrent sur des domaines finis, et ont des limitations importantes de mémoires et de taille d'image de sortie (même taille que l'entrée). Ces approches semblent produire des images de bonne qualité. Mais la reproductibilité des résultats montre des problèmes de convergence, des problèmes aux bords, des distorsions, des mélanges de couleurs qui n'existent pas dans l'image d'entrée, etc. Lorsqu'on les teste sur des textures stochastiques structurées, elles échouent à reproduire les structures (voir 3.19).



**FIGURE 3.19** – Deep Correlations [SCO17] : échec de reproduction de textures stochastiques structurées par CNN.



## 3.3 Synthèse par méthodes procédurales

On entend par *procédurales* les méthodes qui impliquent des fonctions mathématiques ou des algorithmes munis de paramètres permettant de contrôler l'apparence du résultat de la synthèse.

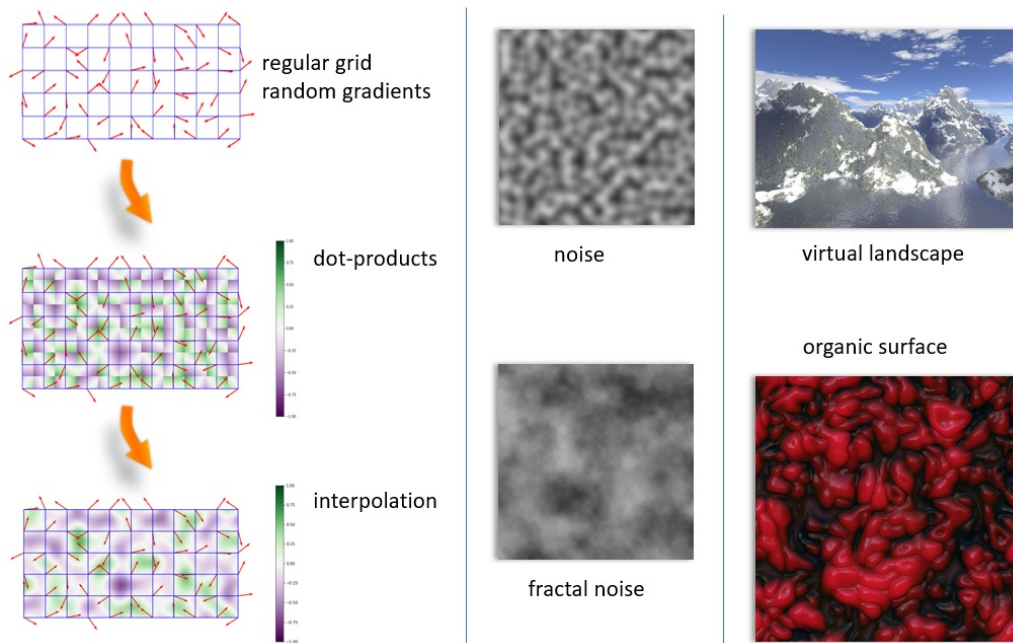
### 3.3.1 Bruits procéduraux

Dans [LLC<sup>+</sup>10], *State of the Art in Procedural Noise Functions*, Lagae *et al.* présentent un état de l'art des fonctions de bruits procédurales. Des évolutions ont été proposées depuis, notamment en ce qui concerne une extension de la gamme des textures synthétisables, ainsi que de leur contrôle et de leur performance. Dans la suite, on présente les modèles et approches les plus liés à nos travaux.

#### Bruit de Perlin

Dans [Per85], *An image synthesizer*, Perlin propose une des fonctions de bruit les plus utilisées encore aujourd'hui, notamment dans l'industrie, qui a été récompensée par un prix en 1997 (*Academy Award for Technical Achievement*). Cette fonction procédurale est construite grâce à une grille régulière, en 1, 2 ou 3 dimensions, sur laquelle est stockée une valeur aléatoire d'un vecteur gradient sur chacun de ces nœuds. En chaque nœud, le bruit vaut 0 et on a une valeur de gradient aléatoire. Et entre chaque nœud, la valeur du bruit provient de l'interpolation des gradients de manière lisse (figure 3.20). Le bruit de Perlin est à bande limitée, stationnaire (invariant par translation) et isotrope (invariant par rotation). Différents types de textures, par exemple avec une apparence fractale, peuvent être créées par combinaison linéaire de bruits mis à l'échelle, afin de créer des apparences plus complexes :

$$f(x) = \sum_i^n \frac{1}{2^i} \text{noise}(2^i x)$$



**FIGURE 3.20** – Bruit de Perlin (voir [Per85]). À gauche : algorithme en 3 étapes. Au milieu : exemple de bruit et de bruit fractal. À droite : exemples d'application. (illustrations issues de la page Wikipedia Perlin Noise<sup>1</sup>).

1. [https://en.wikipedia.org/wiki/Perlin\\_noise](https://en.wikipedia.org/wiki/Perlin_noise)

## Bruits à convolution parcimonieuse (*Sparse Convolution*)

**Sparse Convolution** Dans [Lew84], *Texture synthesis for digital painting*, Lewis introduit le concept de bruit à convolution parcimonieuse (*sparse convolution*) qui vise à générer des fonctions de bruit comme une somme pondérée de noyaux à diverses positions données par une variable aléatoire suivant une loi de probabilité donnée, ici Poisson (pour de l'aléatoire complet). Sa formulation concrète, explicitée dans [Lew89], *Algorithms for solid noise synthesis*, consiste en une convolution d'un noyau 3D  $h(\rho)$  avec un processus ponctuel de Poisson  $\gamma$  :

$$\eta(\rho) = \int_{R^3} \gamma(\sigma) h(\rho - \sigma) d\sigma$$

où le processus de Poisson consiste en des impulsions aléatoires en valeur et position (où  $\rho_k$  est la position de la  $k$ -ième impulsion) :

$$\gamma(\rho) = \sum a_k \delta(\rho - \rho_k)$$

Ce qui se réduit à la somme :

$$\eta(\rho) = \sum a_k h(\rho - \rho_k)$$

**Spot Noise** Dans [VW91], *Spot noise texture synthesis for data visualization*, Van Wijk propose une fonction de bruit appelée *Spot Noise*, permettant de générer des textures stationnaires et anisotropes, grâce à l'utilisation de noyaux génériques  $h$ , pouvant être des motifs visuels :

$$f(x) = \sum a_i h(x - x_i)$$

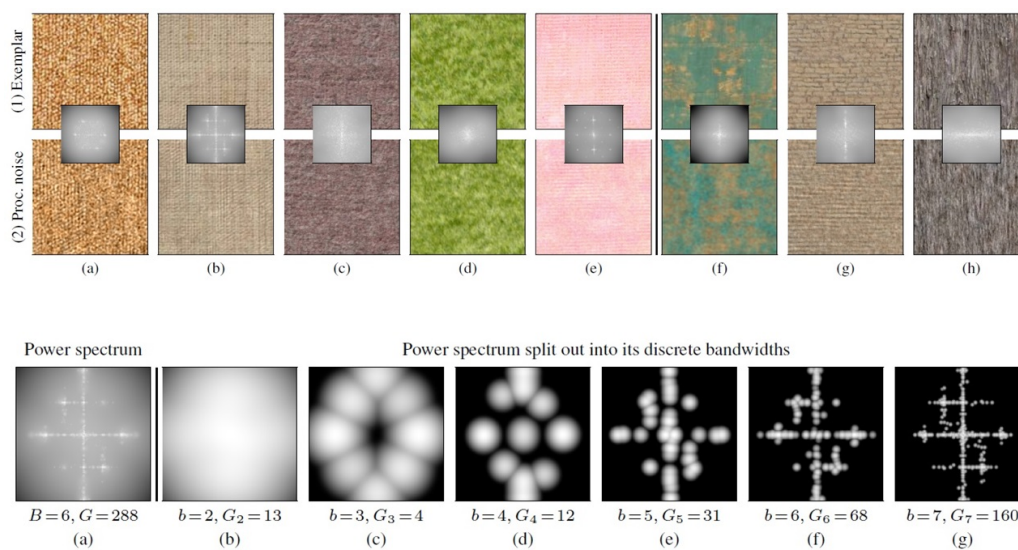
**Gabor Noise** Dans [LLDD09], *Procedural noise using sparse Gabor convolution*, Lagae *et al.* proposent une fonction de bruit à convolution parcimonieuse avec, pour noyau, des fonctions de Gabor. Le contrôle spectral est amélioré en sélectionnant, pour les noyaux, des valeurs aléatoires pour certains paramètres comme l'orientation. Le contrôle utilisateur se fait dans le domaine spectral, grâce à un éditeur de bande de fréquences, en dessinant des bandes (*min*, *max*) pour la fréquence et l'orientation, afin de contrôler la texture dans le domaine spatial (figure 3.21).



**FIGURE 3.21** – Synthèse de motifs stochastiques par fonctions de bruit procédural. Génération de textures de taille arbitraire par fonctions de bruits procéduraux, plaquées sur des objets 3D, et potentiellement colorés via des tables de couleurs (voir [LLDD09]).

## Bruit procédural par l'exemple (*Noise By-Example*)

Dans [GLLD12], *Gabor noise by example*, Galerne *et al.* présentent une méthode pour générer des textures de bruits de Gabor procéduraux à partir d'une image d'exemple en entrée. Le spectre de puissance de l'entrée est estimé et approximé comme une somme de gaussiennes à bandes limitées. Un algorithme d'optimisation cherche à déterminer les paramètres de ses noyaux pour se rapprocher le plus possible du spectre de l'entrée. La texture d'entrée doit être gaussienne, ce qui limite des applications pour générer des textures stochastiques structurées, voir figure 3.22. Le bruit par l'exemple, n'est pas une texture par l'exemple. Il n'y a pas de structures dans le bruit.



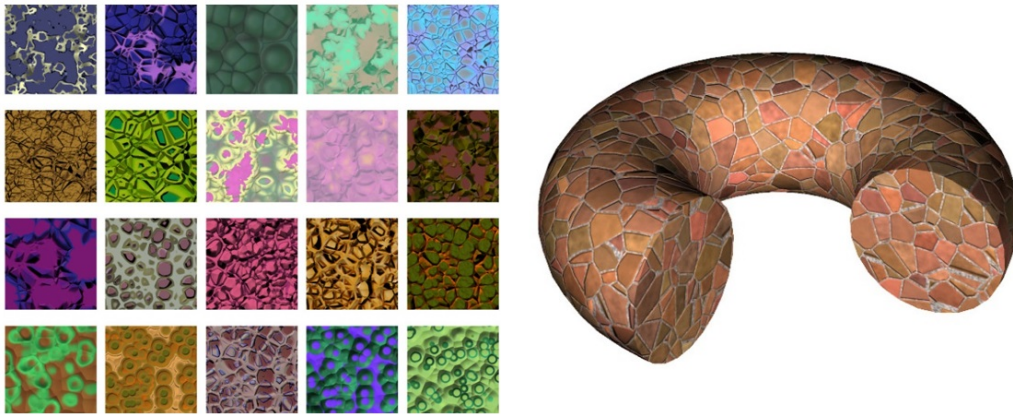
**FIGURE 3.22** – Gabor Noise par l'exemple. Résultats de synthèse de bruit par l'exemple (voir [GLLD12]). En haut : génération de textures gaussiennes de (a) à (e), et non gaussiennes de (f) à (h). En bas : illustration de l'estimation et la décomposition d'un spectre de puissance en somme de gaussiennes à bandes limitées.

### 3.3.2 Fonctions de base de texture (*Texture Basis Functions*)

Les fonctions de base de textures (*Texture Basis Functions*) sont classées dans la catégorie des bruits procéduraux. Ils permettent généralement de générer des textures structurées, difficiles, voire impossible, à générer avec les précédentes méthodes.

#### Bruits cellulaires

Dans [Wor96], *A cellular texture basis function*, Worley présente une fonction de bruit permettant de partitionner le plan en cellules aléatoires. Cette fonction permet de générer des surfaces texturées ressemblant à des zones pavées, de la peau (organique), et des éléments naturels comme des rochers, des montagnes, cratères, à partir d'ensembles de points. Très proche d'un partitionnement en cellules de Voronoï, l'algorithme utilise la distance aux  $n^{\text{th}}$  plus proches voisins ( $n^{\text{th}}$ -closest point basis function). De nouveaux motifs peuvent être générés par des combinaisons linéaires pondérées de ces fonctions de distance (figure 3.23).



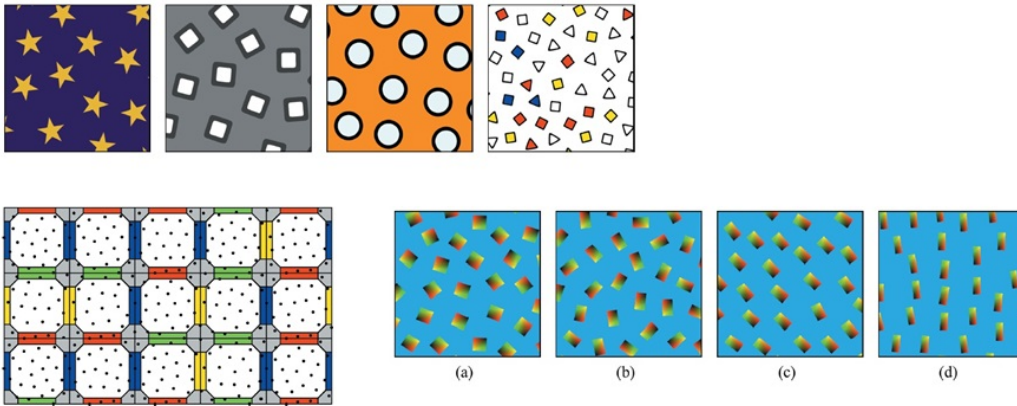
**FIGURE 3.23** – Synthèse de motifs cellulaires par fonctions de base de texture de Worley [Wor96].

Dans [QY01], *A generalized cellular texture basis function*, Qin et Yang proposent une extension des fonctions de base de Worley en ajoutant des combinaisons non linéaires de fonctions de distances, enrichissant la gamme des textures représentables.



### 3.3.3 Autres approches et motifs procéduraux

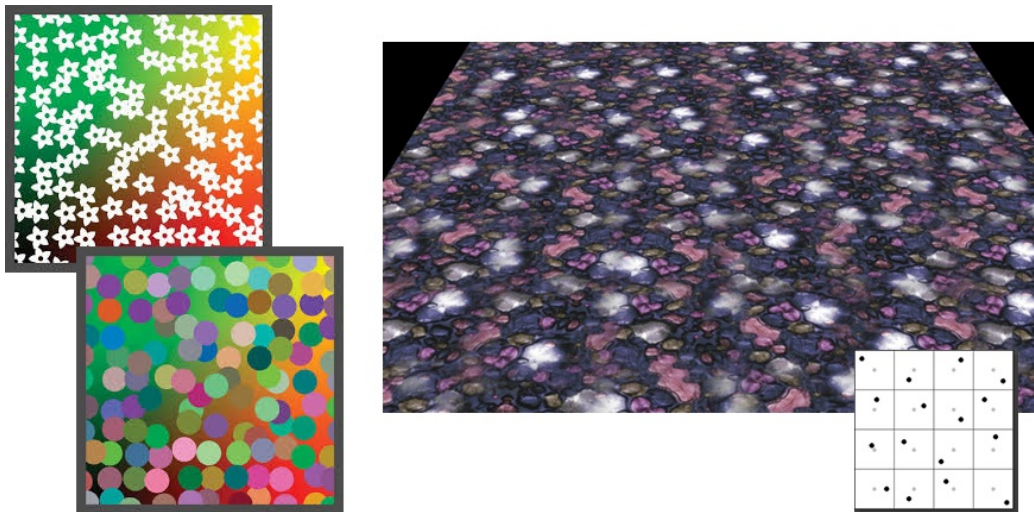
**Distribution d'éléments procéduraux.** Dans [LD05], *A procedural object distribution function*, Lagae et Dutré proposent une nouvelle *texture basis function* permettant de générer des éléments procéduraux, en contrôlant leurs distributions, leurs formes et certains attributs comme leurs tailles, orientations et couleurs. Les distributions d'éléments de type *Poisson disk* sont contrôlées par un pavage stochastique du plan en tuiles, généré à la volée, répartissant les éléments dans chacune d'elles. Des Wang Tiles sont également générés à la volée pour assurer la diversité grâce à leur pavage apériodique (figure 3.24).



**FIGURE 3.24** – Procedural Object Distribution Function. Exemples de génération de motifs procéduraux (haut), ne se recouvrant pas, distribués dans des Wang Tiles (bas gauche), et dont les paramètres d'apparence peuvent changer (taille, orientation, couleur) (bas droite) (voir [LD05]).

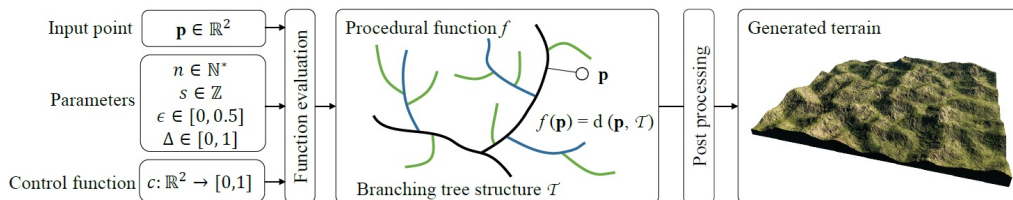
**Bombing** Dans [SA79], *Random pattern generation processes*, les modèles de bombing permettent de générer des processus où les éléments peuvent s'empiler comme des feuilles qui tombent avec un mécanisme de priorité (figure 3.25, provenant du chapitre *Texture Bombing* du livre *GPU Gems* [F+04]). Le principe est de déposer des motifs ou éléments sur des positions aléatoires, générées par un processus de Poisson. Il existe des implémentations GPU du bombing ([F+04] *GPU gems : programming techniques, tips, and tricks for real-time graphics*, de Fernando *et al.*).





**FIGURE 3.25** – Bombing. Modèle de bombing avec empilement d’éléments avec des priorités (probabilités) (voir [SA79]).

**Branchements.** Dans [GBG<sup>+</sup>19], *Dendry : A procedural model for dendritic patterns*, Gaillard *et al.* proposent une fonction procédurale calculable localement qui génère des motifs de branchement à différentes échelles, avec comme application la synthèse de terrains avec des réseaux de rivières (figure 3.26).



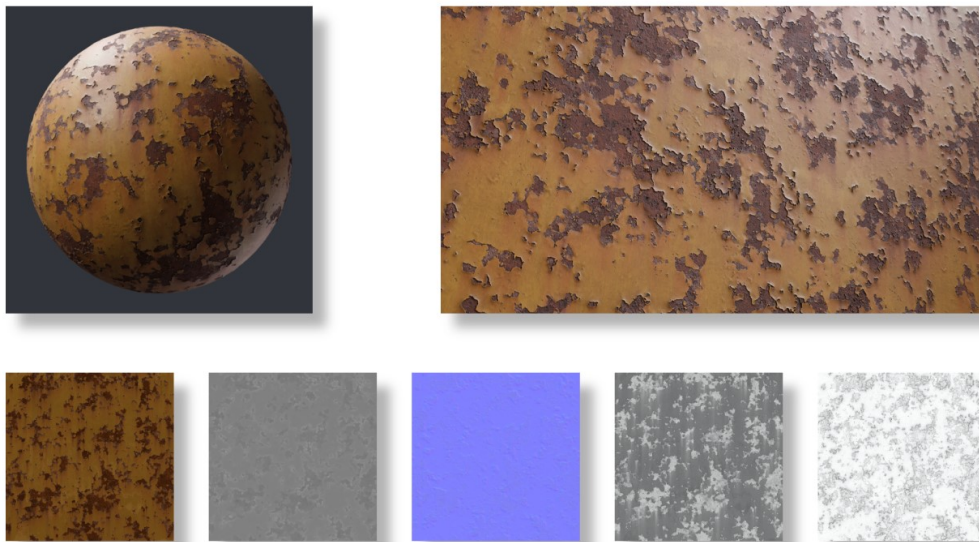
**FIGURE 3.26** – Dendry : génération procédurale de motifs de branchements. La fonction évalue la distance d’un point à une structure d’arbre généré procéduralement, à partir de paramètres et d’une fonction de contrôle (voir [GBG<sup>+</sup>19]).

Il existe peu de travaux de recherche permettant de générer ces différents types de textures par l’exemple (inverse procedural modeling).

## 3.4 Représentations et génération des matériaux

### 3.4.1 Représentations des matériaux

La littérature sur les matériaux est très vaste. On se limite ici aux matériaux représentables sous forme de couches de textures (figure 3.27).



**FIGURE 3.27** – Matériau PBR sous forme de couches de textures. Matériau encodant des propriétés classiques utilisées par les moteurs de rendu actuels. En haut : un matériau plaqué sur un objet 3D et un zoom. En bas : les différentes couches de textures composant le matériau (albedo, hauteur, normale, rugosité, occlusion ambiante). [référence : *rusted paint metal surface - PBR0494*]<sup>2</sup>

---

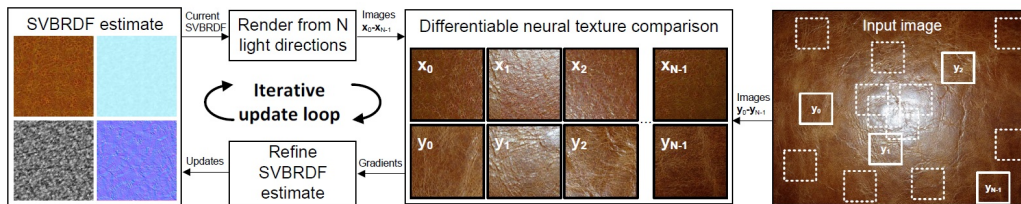
2. <https://www.textures.com/download/PBR0558/138718>

## 3.4.2 Acquisition et synthèse de matériaux

### Automatisation de l'acquisition de matériaux SV-BRDF

Depuis quelques années, il y a une forte demande de l'industrie pour l'automatisation et la simplification des processus d'acquisition des matériaux.

Dans [AAL16], *Reflectance modeling by neural texture synthesis*, Aitala et al. proposent une méthode pour déterminer/extraire les couches de matériaux SV-BRDF automatiquement à partir d'une simple photo prise d'un téléphone portable (figure 3.28). Ils se basent sur des descripteurs de réseaux de neurones de Gatys [GEB15], et une méthode d'optimisation pour s'approcher du matériau d'entrée par différents rendus du matériau procédural en cours en modifiant des paramètres d'éclairage. Cette méthode est proche des approches statistiques multi-échelle de [PS99] et des réseaux de neurones comme [GEB15], mais : le rendu du matériau procédural permet de recalculer ses descripteurs et vérifier les contraintes statistiques. Cette méthode se base sur une hypothèse forte de stationnarité du matériau d'entrée, ce qui limite le champ des matériaux représentables.



**FIGURE 3.28** – Reflectance modeling by neural texture synthesis. Acquisition de matériaux SV-BRDF avec un mobile [AAL16].

Dans [LSC18], *Materials for masses : SV-BRDF acquisition with a single mobile phone image*, Li et al. proposent une approche similaire avec une couche de rendu pour vérifier les contraintes statistiques, avec cette fois une approche par *encoder-decoder* : un encodeur et 3 décodeurs pour déterminer les 3 couches d'albedo (diffuse), normal et de rugosité, à partir d'un simple appareil photo (figure 3.29). Cette méthode permet de gérer une gamme de matériaux plus larges que les stationnaires, mais pour aider et guider ses algorithmes, elle utilise en interne un classifieur servant à déterminer le type du matériau, ne garantissant pas de toujours trouver un bon candidat. Ce type d'approche par classification, associée à une base de données de modèles

paramétriques, posent souvent des problèmes d'incohérences visuelles, due au manque de garantie de faire correspondre la sortie avec l'entrée.

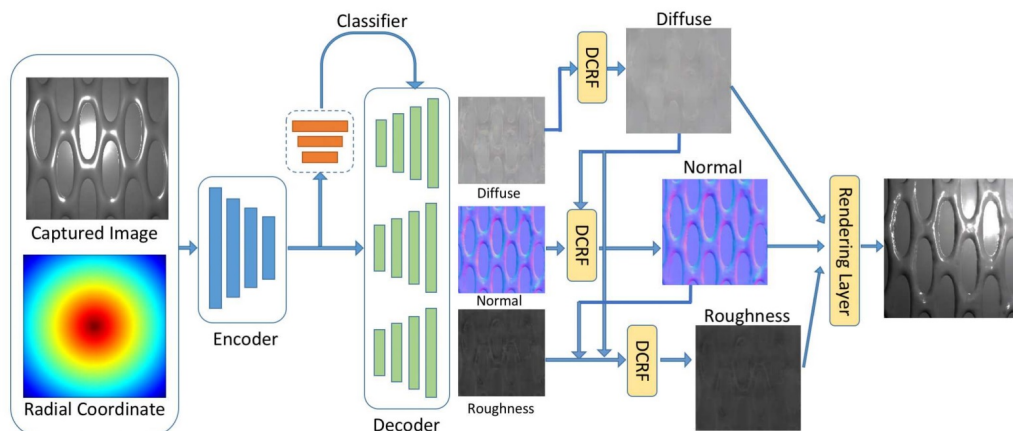


FIGURE 3.29 – Acquisition de matériaux SV-BRDF avec un mobile [LSC18].

Dans cette thèse, on utilise une base de données de matériaux en couches de textures, provenant notamment du site <https://www.textures.com/>. Il existe de nombreux autres travaux sur l'acquisition avec des matériels et dispositifs plus ou moins complexes, mais qui sortent du cadre de cette thèse.

# Chapitre 4

## Contrôle de la Synthèse des Textures et Matériaux Structurés et Non Homogènes

Nous avons vu que les méthodes de synthèse de textures et de matériaux par l'exemple présentées dans le chapitre précédent ont des capacités limitées à reproduire d'une part les structures stochastiques, et d'autre part les textures non homogènes. Dans ce chapitre, on s'intéresse aux méthodes qui permettent d'introduire du contrôle spatial de façon *explicite* dans la synthèse de textures, et quelles sont les possibilités d'édition de l'utilisateur.

### 4.1 Contrôle des structures

On s'intéresse ici aux travaux qui tentent d'améliorer la reproduction de structures dans un contexte de textures homogènes, voire de proposer d'éditer les structures.

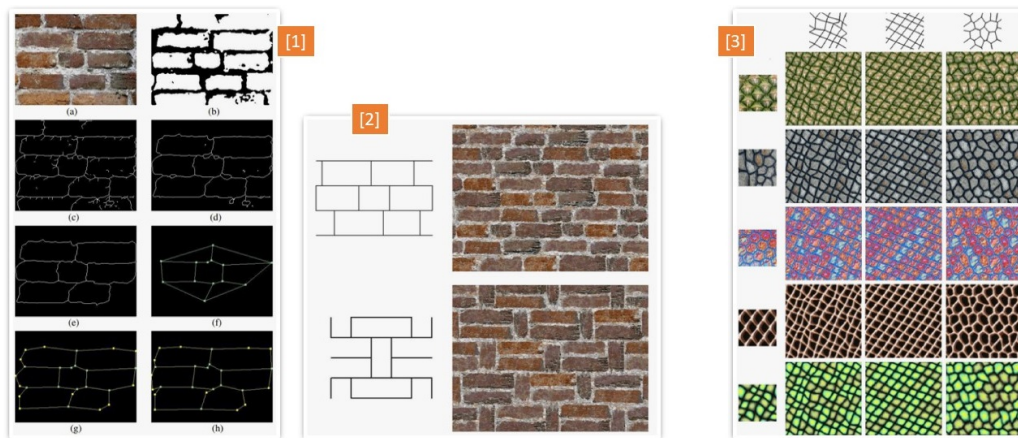
#### 4.1.1 Approches géométriques

Dans sa thèse [Fra05], *Decoupling structure and appearance for example-driven detail synthesis*, Frasson propose un modèle permettant de découpler la structure de l'apparence. À partir d'une carte à deux labels séparant les motifs extraits manuellement d'une texture, une première passe s'attache à synthétiser la structure de manière géométrique, sous forme d'un maillage.



La couleur est ensuite générée par la méthode par l'exemple [EF01], *Image Quilting*, guidée par la structure. Dans [DZ06], *Real-time structured texture synthesis and editing using image-mesh analogies*, Dischler et Zara reprennent ce principe et proposent d'éditer le maillage interactivement, qui sert ensuite à guider une synthèse de texture selon la méthode PCTS [LH05] (figure 1.2).

Dans [WQC<sup>+</sup>14], *Interactive texture design and synthesis from mesh sketches*, Wang et al. proposent une extension du travail précédent, [DZ06], permettant de fournir une esquisse ou un croquis à partir duquel le maillage sera calculé (figure 1.2).



**FIGURE 4.1** – Synthèse contrôlable de textures structurées. [1] Extraction de la structure d'une image d'exemple sous forme de maillage par optimisation et [2] synthèse avec édition de la structure [DZ06]. [3] Version proposant un croquis en entrée [GKHF14]. Dans les deux cas, la synthèse est contrainte par la structure selon l'approche *Image Analogies* ([HJO<sup>+</sup>01]).

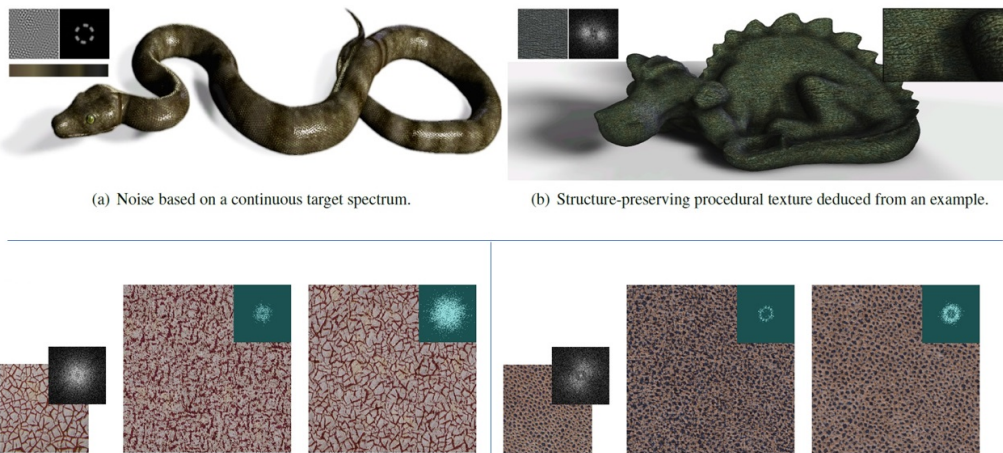
Cette approche ne peut s'appliquer qu'à des textures dont la structure a une topologie (ou connexité) de type grille et ne variant pas spatialement, ce qui limite la gamme de textures représentables.

## 4.1.2 Approches procédurales

**Fonctions de bruit procédurales** Si les modèles de textures et les méthodes de synthèse procédurale ont évolué vers plus de rapidité, qualité et diversification des types de structures générées, elles sont toujours limitées à

des types de motifs stochastiques ayant peu de structures. Quelques travaux ont contribué à étendre la gamme des textures de bruits procéduraux générés, par plus d’anisotropie, plus de contrôle local et une meilleure gestion de la reproduction des structures.

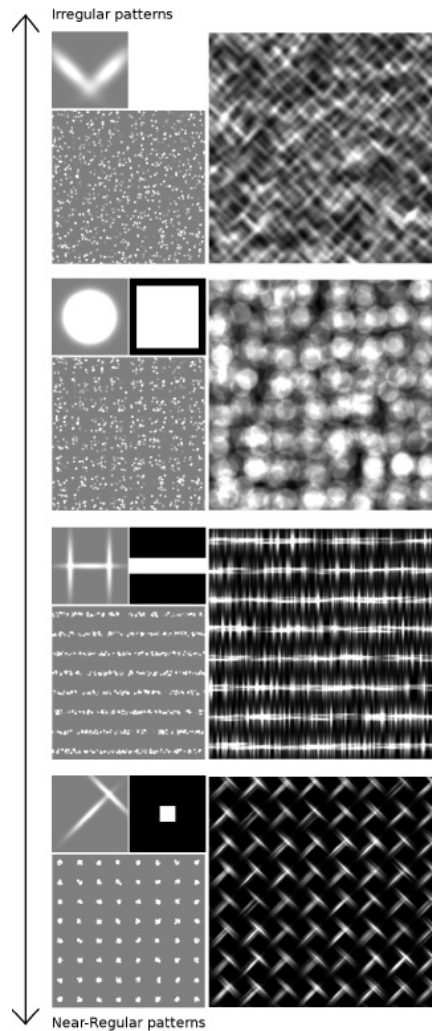
Dans [GSV<sup>+</sup>14], *Local Random-Phase Noise for procedural texturing*, LRPN, Gilet *et al.* proposent un modèle à base de séries de Fourier pour créer des motifs locaux. Leur approche consiste à fixer certaines phases, ce qui leur permet de générer de la structure. Cependant, certains motifs structurés possèdent des phases réparties dans tout le spectre, ce qui peut détériorer, mélanger, voire répéter les structures (figure 4.2).



**FIGURE 4.2** – LRPN : Local Random Phase Noise. Bruit par l’exemple pouvant conserver certaines structures, selon un paramètre utilisateur ( $\in [0; 1]$ ). Certaines structures ne peuvent pas être reproduites, [GSV<sup>+</sup>14].

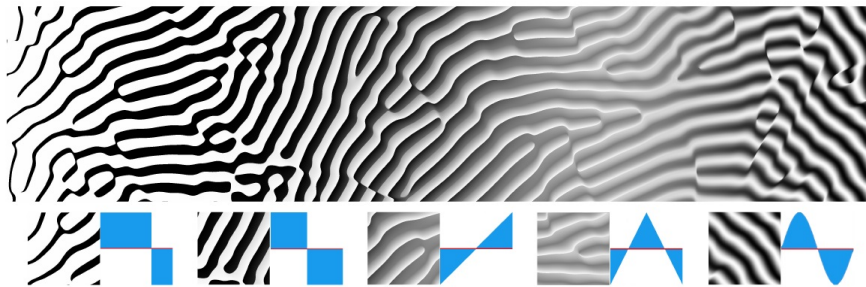
Dans [PGDG16], *Procedural texture synthesis by locally controlled spot noise*, Pavie *et al.* proposent une amélioration du bruit LRPN, [GSV<sup>+</sup>14], étendent la gamme de textures de bruit en utilisant des gaussiennes anisotropes (figure 4.3). Les motifs générés sont des structures répétitives avec de petites variations aléatoires. Le contrôle spatial est plus facile, avec plusieurs fonctionnalités pour l’utilisateur, mais le contrôle spectral est perdu. Dans [CGG19], *Local spot noise for procedural surface details synthesis*, Cavalier *et al.* proposent d’améliorer le *locally controlled spot noise*, [PGDG16], notamment pour le filtrer.





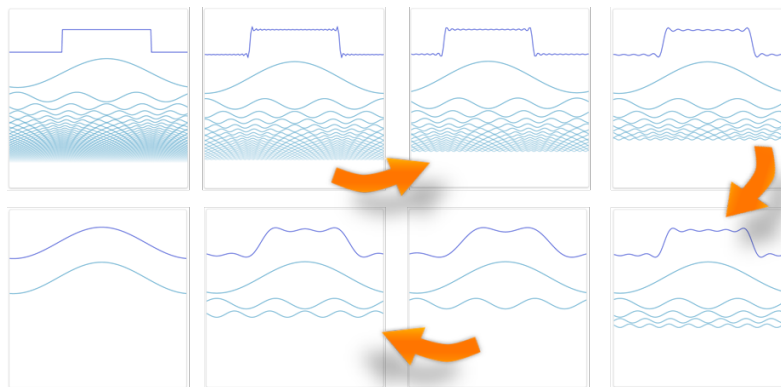
**FIGURE 4.3** – Locally Controlled Spot Noise. En haut à gauche : noyau et profil de densité périodique. En bas à gauche : distribution d’impulsions. Droite : bruit synthétisé de l’irrégulier vers le quasi-régulier, [PGDG16].

Dans [TEZ<sup>+</sup>19], *Procedural phasor noise*, Tricard *et al.* remplacent la génération d’un bruit d’intensité par un bruit de phases. Ils appliquent un front d’onde sinusoïdale ou périodique quelconque pour maintenir un contraste optimal le long de ce front d’ondes (figure 4.4).



**FIGURE 4.4** – Procedural Phasor Noise. En haut : interpolation de motifs procéduraux avec contrôle du contraste. En bas : les différents profils de fonctions périodiques utilisés pour le contrôle. [TEZ<sup>+</sup>19].

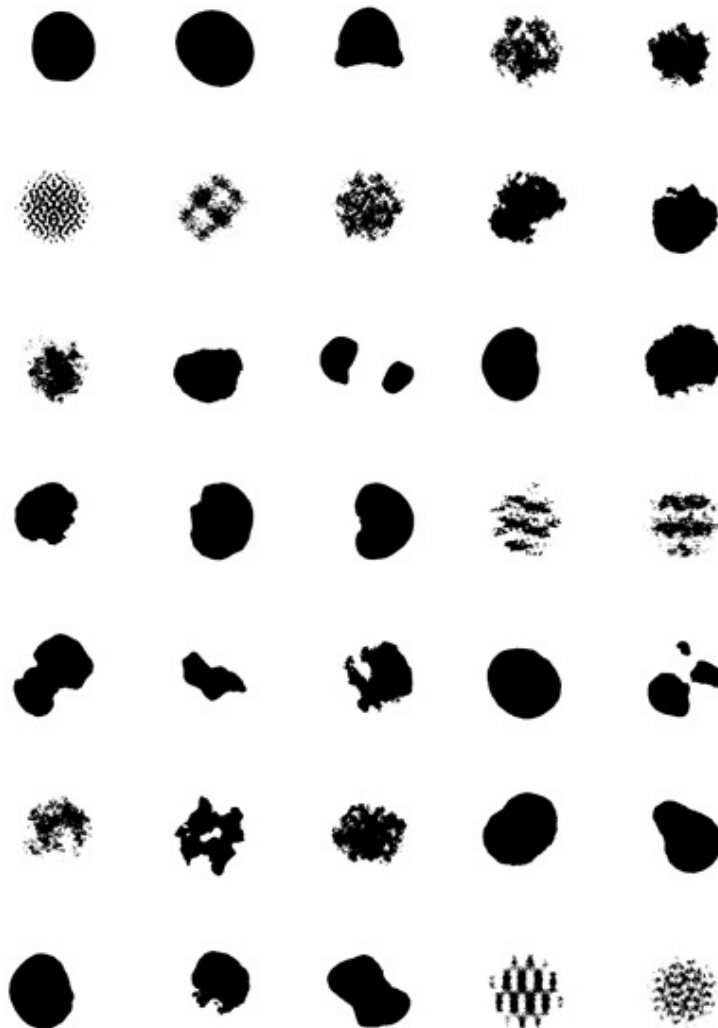
**Congruence de phases** Les phases sont la composante essentielle de la structure ([OL81] *The importance of phase in signals*, Oppenheim). Le point clé consiste à contrôler la position spatiale des phases pour créer de la structure. La figure 4.5 illustre la décomposition d’une forme structurée en une somme de sinusoides. Cette image ne montre pas les phases, mais pour créer des bords durs, il faut contrôler et synchroniser un nombre important de fonctions sinusoidales, dans une grande partie du spectre. On parle de congruence de phases. Cette approche a peu été développée et reste à l’état d’ouverture potentielle pour de futurs travaux en synthèse de textures.



**FIGURE 4.5** – Décomposition de signaux sur la base des cosinus et sinus sans information de phase<sup>1</sup>.

1. <http://www.jezzamon.com/fourier/index.html>

Sur la page web personnelle de J.P. Lewis<sup>2</sup>, on trouve l'évocation d'un travail traitant de synthèse par congruence de phases (figure 4.6), mais il n'y pas plus de détail à part une image de noyaux aux motifs variés, structurés et stochastiques structurés, qui auraient été générés par congruence de phases.



**FIGURE 4.6** – Synthèse de textures par cohérence de phases. Dans *Coherent Phase Texture Synthesis*, J.P. Lewis présente la synthèse de détails structurés en gérant la cohérence de phases des noyaux dans la sparse convolution.

---

2. <http://scribblethink.org/>

### 4.1.3 Gestion des variations spatiales

Dans [ZZV<sup>+</sup>03], *Synthesis of Progressively-variant Textures on Arbitrary Surfaces*, Zhang et al. proposent une méthode permettant de générer des textures homogènes variant spatialement, en se basant sur des textures sous forme de masques de textons, c'est-à-dire des images binaires encodant la distribution spatiale d'un motif élémentaire (figure 4.7). Un nouveau masque de textons peut être généré par édition du masque initial grâce à des opérations de traitement d'images comme des opérations morphologiques (érosion, dilatation) ou des déformations, servant à guider la synthèse d'une nouvelle texture. Les auteurs ont utilisé une méthode de synthèse par pixels séquentielle (antérieure aux méthodes temps réel). Nous verrons dans la section suivante que le principe de la carte de textons peut s'étendre au cas de textures non homogènes.

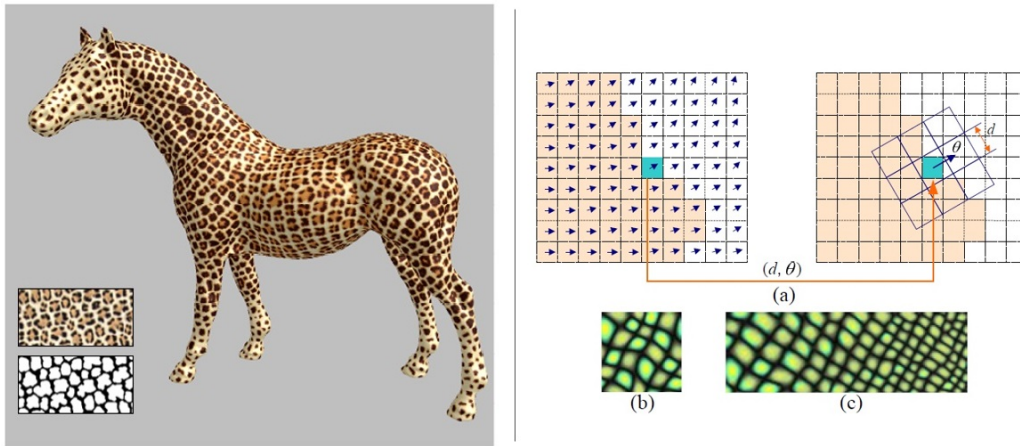


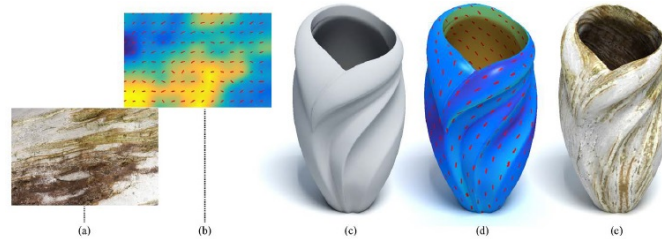
FIGURE 4.7 – Masques de textons et champs de distorsion ([ZZV<sup>+</sup>03]).

## 4.2 Reproduction, contrôle et édition des textures non homogènes

On s'intéresse ici au cas de textures non homogènes qui comportent, par définition, plusieurs motifs distinguables visuellement.

### 4.2.1 Guidage par champs de vecteurs 2D

Dans [ZSL<sup>+</sup>17], *Analysis and controlled synthesis of inhomogeneous textures*, Zhou *et al.* proposent une méthode de synthèse de textures non homogènes, par analyse de l'exemple, afin d'extraire des champs de vecteurs 2D capturant les variations basses fréquences de certaines caractéristiques de l'image d'entrée, puis synthèse à partir d'un champ de vecteurs 2D cible pouvant correspondre aux directions principales de la surface d'un objet 3D (figure 4.8). Dans cette méthode, les textures homogènes sont vues comme comportant des variations continues d'un motif à un autre, et donc sans structure du point de vue de notre définition de la structure.



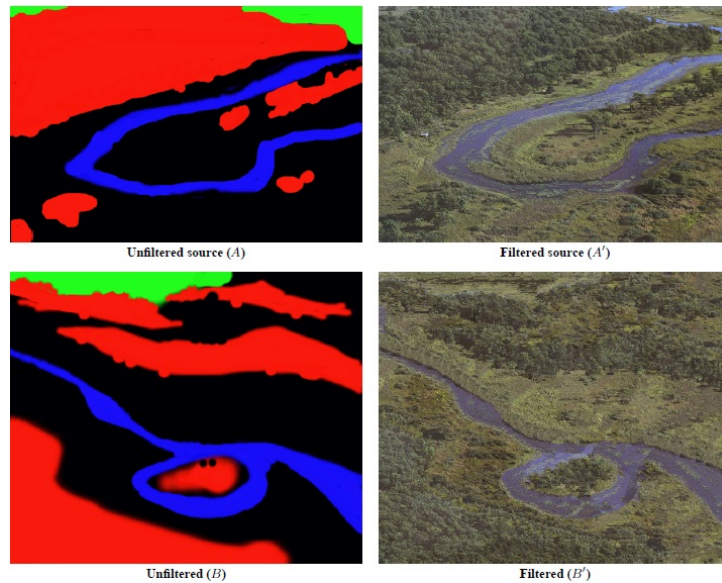
**FIGURE 4.8** – Extraction d'un champ de vecteurs 2D d'un exemple d'entrée pour contrôler la synthèse d'une texture sur un objet 3D ([ZSL<sup>+</sup>17])

### 4.2.2 Guidage par cartes de labels (*Texture-by-Numbers*)

Une approche courante pour guider spatialement la synthèse de textures non homogènes structurées est l'approche *Texture-by-Numbers*. Dans [HJO<sup>+</sup>01], *Image Analogies : Texture By Numbers*, une carte de labels dédiée est utilisée pour contrôler le placement des motifs dans l'image de sortie. Une carte de labels est d'abord extraite manuellement ou automatiquement de l'image d'exemple, comme dans un processus de segmentation. Un label représente alors une sous-texture ou un motif. Ensuite, une nouvelle carte de labels est dessinée manuellement en modifiant l'organisation spatiale des labels (figure 4.9).

Cette approche peut s'appliquer à quasiment toutes les méthodes par pixel et par patch, notamment les méthodes par pixel temps réel :

- [LH05], *Parallel Controllable Texture Synthesis, PCTS* ;
- [LH06], *Appearance-space Texture Synthesis, ASTS* ;
- [BELS10], *Instant Texture Synthesis by Numbers*.



**FIGURE 4.9** – Utilisation d’une carte de labels pour contrôler le placement de contenu à partir de l’exemple d’entrée. En haut : image d’entrée et sa carte de labels associée. En bas : nouvelle carte de labels guidant sa texture synthétisée associée ([HJO<sup>+</sup>01] *Image Analogies : Texture By Numbers*)

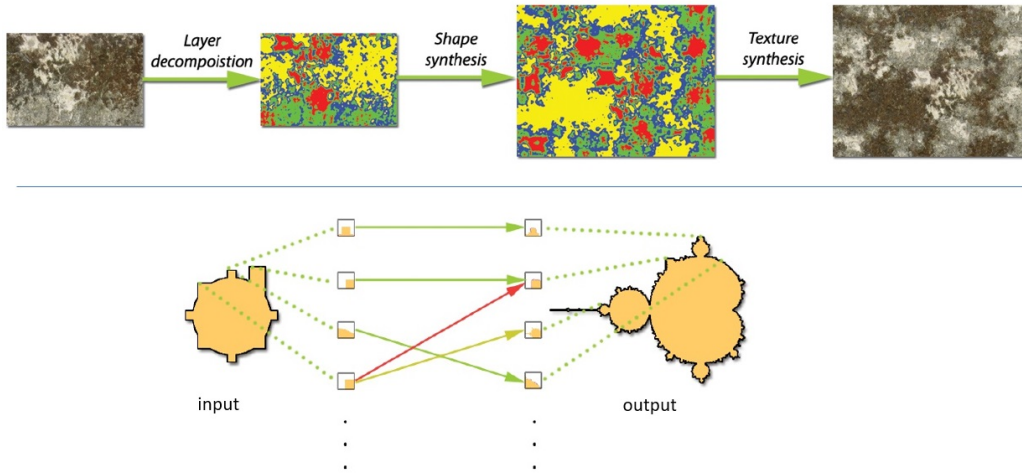
**Généralisation : Gestion de contraintes multiples** Dans [RB06], *Constrained texture synthesis via energy minimization*, Ramanarayanan et Bala présente une approche par minimisation de contraintes (*CMS, constrained minimization synthesis*), qui étend l’approche Image Analogies, en permettant d’ajouter un nombre arbitraire de contraintes basées sur des images fournies par l’utilisateur.

Ces méthodes considèrent que les cartes de labels sont produites de façon manuelle ou semi-automatique, que ce soit pour les textures d’entrée ou celles de sortie. Les travaux qui suivent se sont attaqués à ce problème.

**Approche multi-Couches géométrique** Dans [RCOL09], *Layered Shape Synthesis : Automatic Generation of Control Maps for Non-stationary Textures*, Rosenberger *et al.* proposent une méthode de génération automatique de cartes de labels basée sur un exemple. Ces cartes sont modélisées comme une superposition de plusieurs couches, où les parties visibles sont remplies par une texture plus homogène. Une fois l’image en entrée décomposée en un



petit nombre de couches, un algorithme de synthèse de formes par l'exemple génère automatiquement un nouvel ensemble de couches, préservant les caractéristiques locales et globales de la carte de l'entrée. Ces cartes servent ensuite à guider le processus de synthèse de textures (figure 4.10).



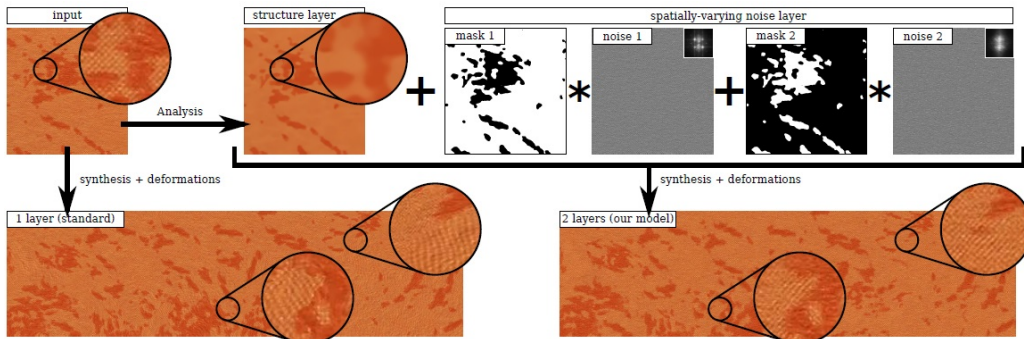
**FIGURE 4.10** – Layered Shape Synthesis. Génération automatique de cartes de contrôle pour des textures non homogènes sous forme de couches (*layers*), ([RCOL09]). Haut : décomposition de l'entrée en couches, puis synthèse de nouvelles couches de cartes. Bas : génération d'une couche (à droite) par l'exemple d'entrée (à gauche). Des morceaux de contours sont assemblés comme des patches.

**Extraction et synthèse de cartes de labels multi-échelle** Dans *Multi-Scale Label-Map Extraction for Texture Synthesis* [LSA<sup>+</sup>16], des cartes de labels sont extraites automatiquement à plusieurs échelles (figure 4.11). Suite à une extraction *hiérarchique*, les transitions de labels résultantes sont transmises d'échelles grossières à fines. Ainsi, une transition existant à une échelle existe à toutes les échelles plus fines. Une application d'édition d'images sous la forme d'un outil de peinture est proposée. Elle se base sur une synthèse de textures par patch et l'utilisation d'un descripteur statistique encodant chaque patch en un histogramme de labels (extrêmement compacte), ce qui permet d'accélérer l'édition de textures de haute résolution. La principale limitation est qu'il n'y a pas de gestion des transitions et des relations d'adjacence des motifs, provoquant des incohérences visuelles.



**FIGURE 4.11** – Multi-Scale Label-Map Extraction for Texture Synthesis. Extraction automatique de cartes de labels à différentes échelles [LSA<sup>+</sup>16].

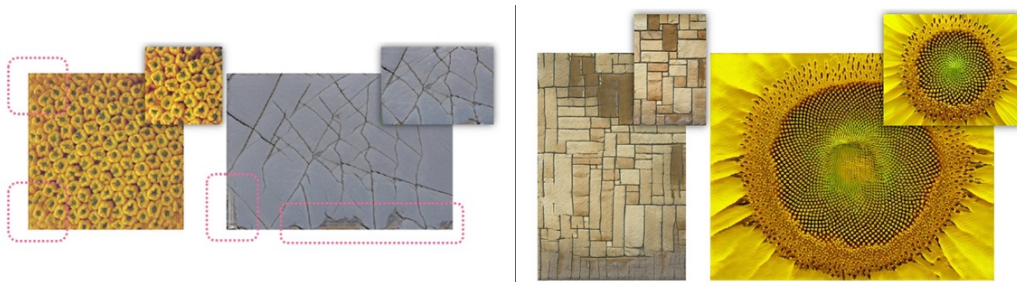
**Approche de textures à deux couches : structure et détails** Dans [GSDC17], *Bi-Layer textures : a Model for Synthesis and Deformation of Composite Textures*, Guingo *et al.* proposent un modèle à deux couches pour séparer la synthèse de la structure de ses détails. La structure est générée par des approches de synthèse de textures par l'exemple. Les détails sont synthétisés sous forme de bruits procéduraux, synchronisés par des couches ou masques de structure. Un inconvénient de cette méthode est la difficulté de contrôle de la zone de structure, comme les formes aux bords (par exemple des lichens), voir figure 4.12.



**FIGURE 4.12** – Bi-Layer Textures. Décomposition d'une texture en une couche de structure et une couche de bruit. La couche de bruit capture un bruit Gaussien variant spatialement comme un mélange de bruits stationnaires. Les textures sont synthétisées à la volée par synthèse synchronisée des couches. De la variété peut être ajoutée à l'étape de synthèse en déformant la couche de structure tout en préservant l'apparence aux échelles fines, encodée dans la couche de bruit [GSDC17].

### 4.2.3 Génération par apprentissage automatique

Plusieurs méthodes par apprentissage automatique, utilisant des réseaux de neurones convolutionnels, ont cherché à reproduire des textures non homogènes, apprenant la distribution spatiale des motifs dans les images. Dans [ZZB<sup>+</sup>18], *Non-stationary texture synthesis by adversarial expansion*, Zhou *et al.* proposent une approche basée sur des réseaux antagonistes (adverses) (GANs, *Generative Adversarial Network*). Le but est de pouvoir synthétiser des textures non stationnaires, contenant des structures à grande échelle, et des textures variant spatialement et non homogènes, qui sont des textures difficiles à synthétiser par les méthodes précédentes. Les limitations concernent les temps d'apprentissage, le coût en stockage, les problèmes de convergence des GANs, et le fait que cette approche est plutôt un moyen d'agrandir une image (2 fois) que de faire de la réelle synthèse (en fait, c'est une sous-partie du domaine). Par ailleurs, des problèmes aux bordures et coins des images synthétisées que l'auteur propose de couper manuellement (*crop*) peuvent apparaître. De plus, lorsque l'exemple d'entrée est trop irrégulier, comme des briques irrégulières de murs, la méthode échoue à préserver les structures (voir 4.13). Enfin, les GANs ont des difficultés à reproduire tous les modes constituant les exemples d'entrée, et donc ont tendance à ne modéliser qu'une sous-partie du modèle latent sous-jacent.

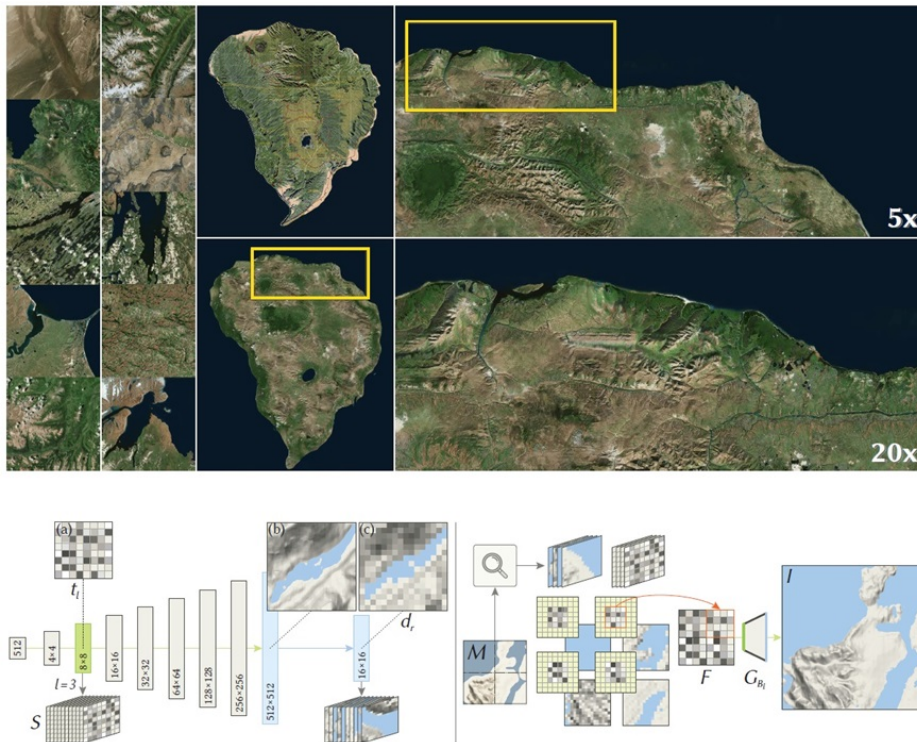


**FIGURE 4.13** – Deep Learning, CNNs et GANs : échec de la reproduction de structures stochastiques et problèmes aux bords [ZZB<sup>+</sup>18].

Cette approche par apprentissage automatique présente des résultats visuellement proches, parfois de bonne qualité, cependant il est difficile de reproduire à l'identique leurs résultats. En effet, l'algorithme semble instable et une image proche en entrée, mais légèrement plus stochastique, peut le faire échouer à reproduire les structures, même à l'échelle locale. De plus, il

est difficile de passer à l'échelle, par exemple pour texturer un grand domaine. Il n'y a pas de modèle interne de structure, ce qui empêche de contrôler des structures à larges échelles.

Dans [FAW19], *TileGAN : Synthesis of large-scale non-homogeneous textures*, Frühstück *et al.* proposent une méthode pour générer des textures extrêmement larges avec des GANs, avec des détails à plusieurs niveaux d'échelles, qu'ils illustrent par de la synthèse d'images aériennes. L'algorithme utilise des espaces latents pour générer et contrôler le contenu à diverses échelles (figure 4.14). La taille des images générées, bien que très grande, semble finie. Le matériel et ressources requis demandent une configuration matérielle très performante en termes de mémoire RAM, de GPU, et avec un processus d'entraînement très long.



**FIGURE 4.14** – TileGAN : synthèse non homogène multi-échelle. Synthèse multi-échelle d'images aériennes. En haut : (à gauche) des images d'apprentissage, et (à droite) des images générées à différents niveaux d'échelle avec leurs détails [FAW19]. En bas : utilisation d'espaces latents pour générer et contrôler le contenu à diverses échelles.

## 4.3 Gestion de l'organisation spatiale

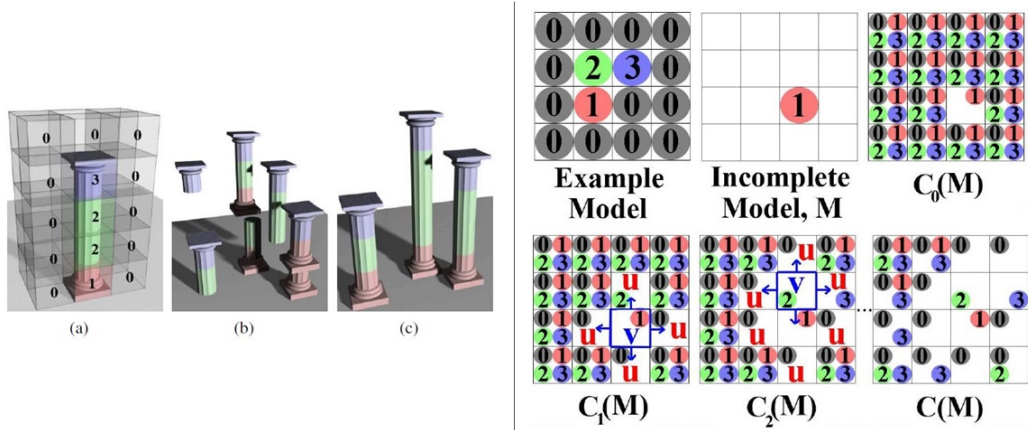
L'organisation spatiale des motifs dans le résultat de la synthèse, telle que gérée par les méthodes présentées plus haut, n'est pas contrôlée explicitement. Celle-ci résulte de l'algorithme de synthèse. Pour un contrôle explicite, des contraintes supplémentaires doivent être introduites.

### 4.3.1 Synthèse par contraintes sur les relations d'adjacence de blocs

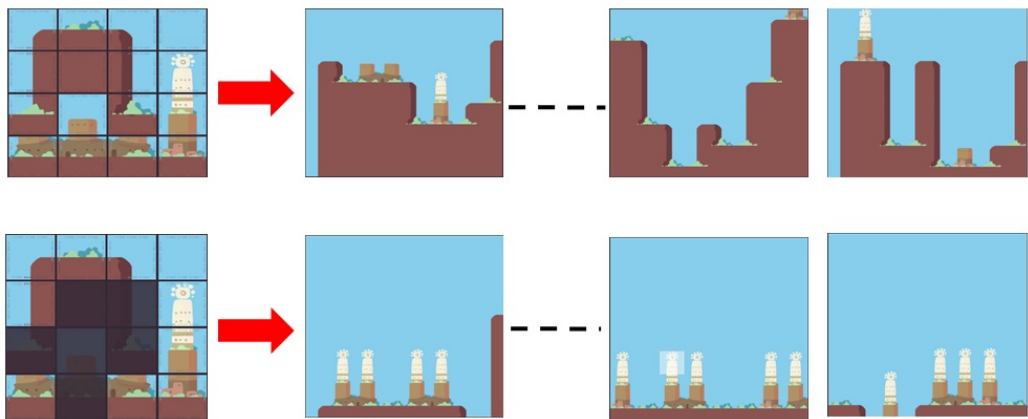
**Approche DMS : Discrete Model Synthesis.** Dans [Mer07], *Example-based model synthesis*, Merrell présente son approche de "synthèse de modèles discrets", où il prend en compte les relations d'adjacence entre les éléments de base, appelés *blocs*, constituant une image ou un modèle 3D en entrée (figure 4.15). Un graphe d'adjacence est pré-calculé et utilisé pour guider le choix du prochain bloc lors de la synthèse, à une position donnée (voir 4.16). Merrell a démontré que l'algorithme résout un problème de satisfaction de contraintes de type 3-SAT planaire, dont la complexité augmente rapidement avec le nombre de blocs. Dans le cas général, décider si un résultat de synthèse peut être obtenu à partir d'un graphe d'adjacence donné est un problème NP-difficile. Cette complexité limite l'application de l'algorithme à de faibles nombres de blocs. Dans [MM08], *Continuous Model Synthesis* et [MM11], Merrell améliore le travail précédent, notamment en permettant l'ajout de contraintes par l'utilisateur pour plus de contrôle.

**Approche WFC : Wave Function Collapse.** Dans [KS17], *WaveFunctionCollapse is constraint solving in the wild*, Karth et Smith proposent un modèle inspiré de la mécanique quantique pour définir une fonction d'onde caractérisant les différents états possibles en chacun des points, et une contraction des états en un seul. Comme expliqué par Merrell en 2021 dans un document, *Comparing Model Synthesis and Wave Function Collapse* [Mer21], ce modèle est très proche de DMS. WFC semble fait pour de petites images 2D. Il l'illustre sur de nombreuses images exemples 2D, voir figure 4.17.





**FIGURE 4.15** – DMS : Discrete Model Synthesis. À gauche : (a) modèle 3D découpé manuellement en morceaux avec un identifiant/label, (b) une version incohérente et (c) une valide. À droite, un exemple de l’algorithme en 2D : en partant du morceau d’ID 1, on propage dans chaque cellule la liste de tous les index de morceaux qui créeraient un objet valide en termes de relation d’adjacence de ces parties. Ensuite, pour chaque cas, on propage successivement les contraintes de la relation d’adjacence.



**FIGURE 4.16** – DMS : Discrete Model Synthesis. Génération automatique de textures ou images sous forme de grille à partir d’un ensemble de tuiles d’entrée. Les relations d’adjacence sont prises en compte pour sélectionner le contenu de chaque cellule.



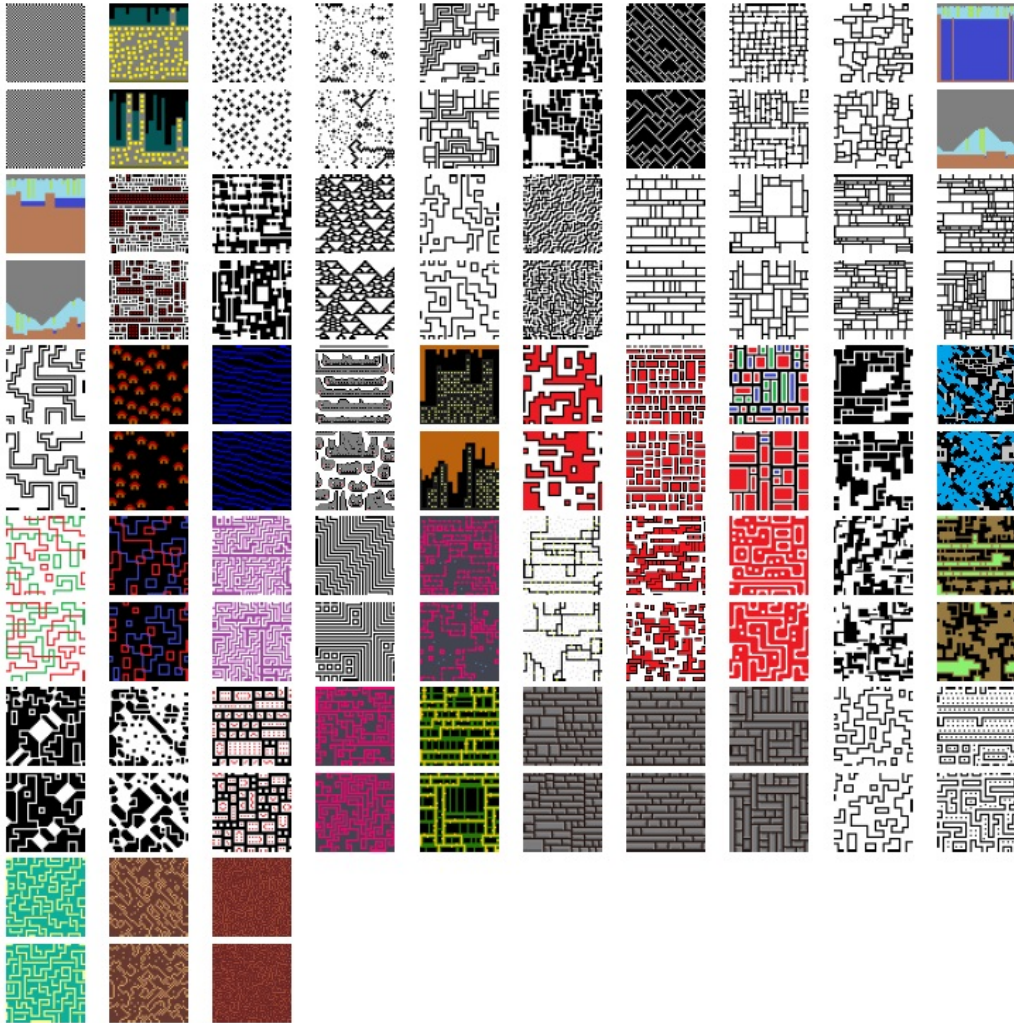


FIGURE 4.17 – Synthèse DMS vs. WFC

Comparaison de DMS (Discrete Model Synthesis) et WFC (Wave Function Collapse) par Paul Merrell, 2021 [Mer21]. Pour chaque couple d'image, à chaque fois DMS est en haut et WFC en bas.

Cette approche est bien fondée sur le plan théorique, mais souffre de complexités en temps et en mémoire difficilement compatibles avec des contraintes de synthèse de textures en temps réel, sauf pour de très petits nombres de blocs.

### 4.3.2 Arrangements et distributions d'éléments

On présente ici des travaux dans lesquels les textures sont représentées par des distributions spatiales d'éléments vectoriels, notamment sous forme de processus ponctuels.

Dans [HLT<sup>+</sup>09], *Appearance-guided synthesis of element arrangements by example*, Hurtut *et al.* s'intéressent aux distributions d'éléments vectoriels à base de processus ponctuels de Gibbs (figure 4.18). Leur approche permet d'extraire et déterminer des caractéristiques de la distribution à partir d'une image d'entrée (figure 4.19). Une limitation du modèle est la non prise en compte de la régularité des arrangements, ce qui demande des modèles plus évolués de processus ponctuels.

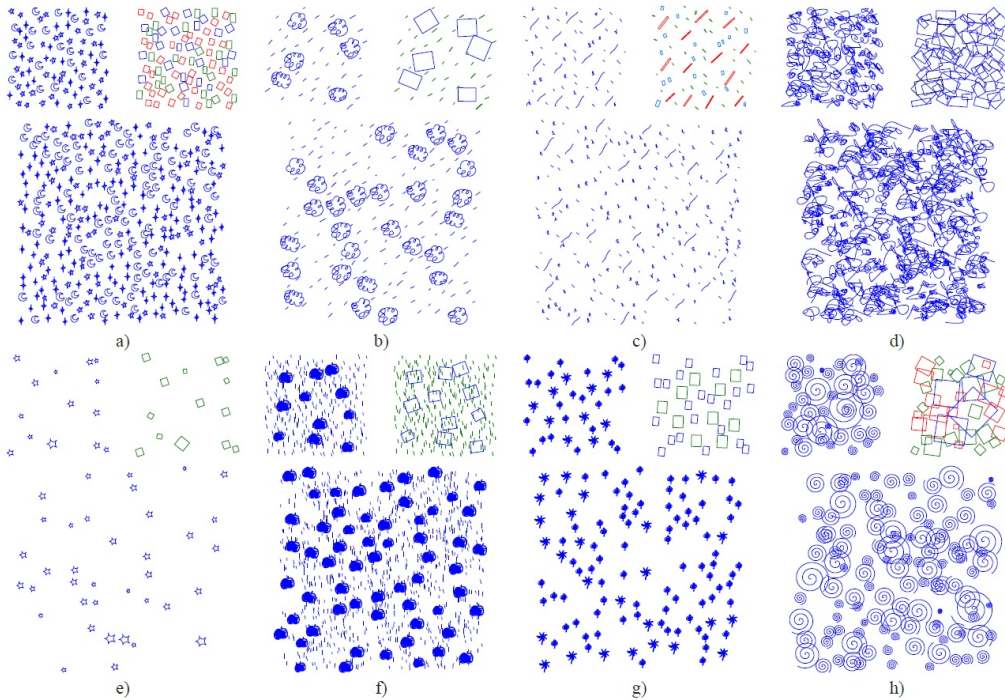


FIGURE 4.18 – Génération automatique d'arrangements d'éléments [HLT<sup>+</sup>09].

Dans [LHVT17], *Programmable 2d arrangements for element texture design*, Loi *et al.* s'intéressent à la mise place d'un framework de programmation pour créer des textures à base d'éléments en contrôlant leur arrangement spatial, formes et mélange (interaction).

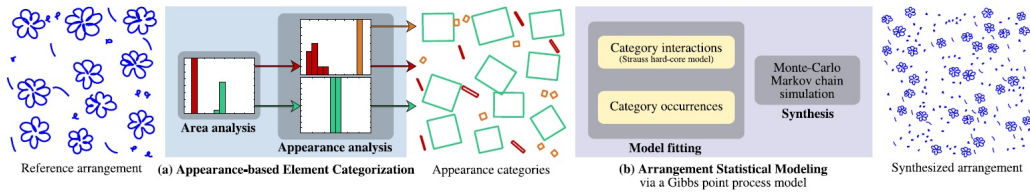


FIGURE 4.19 – Génération automatique d’arrangements d’éléments [HLT<sup>+</sup>09].

Il existe également d’autres travaux de recherche s’intéressant à la génération et au contrôle des distributions d’éléments comme [EVC<sup>+</sup>15] (*WorldBrush : Interactive Example-based Synthesis of Procedural*) et [GLCC17] (*EcoBrush : Interactive Control of Visually Consistent Large-Scale Ecosystems*), pour la modélisation de scènes virtuelles 3D en peignant des distributions statistiques avec un contrôle utilisateur avancé (métaphore de peinture, types de distributions, transitions entre distributions, etc.). Également les travaux de [ÖG12], *Analysis and synthesis of point distributions based on pair correlation*, qui se basent sur le descripteur des *pair correlation function*, *PCF*, permettant de caractériser les distributions des distances entre chacun des éléments deux à deux, illustrés par de la génération de forêts. Cette approche permet de gérer des processus ponctuels plus avancés et plusieurs classes d’éléments (figure 4.20).

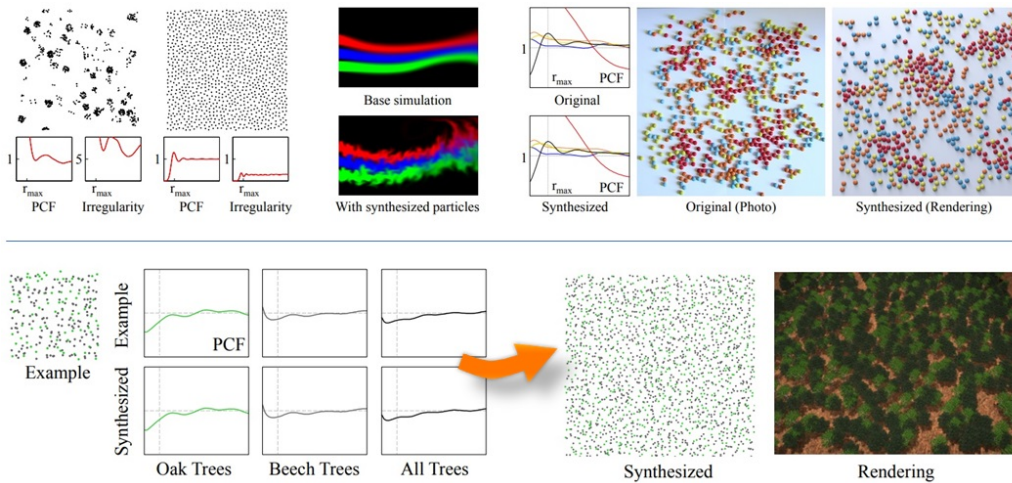


FIGURE 4.20 – Analyse et synthèse de distributions par *Pair Correlation Function* [ÖG12].

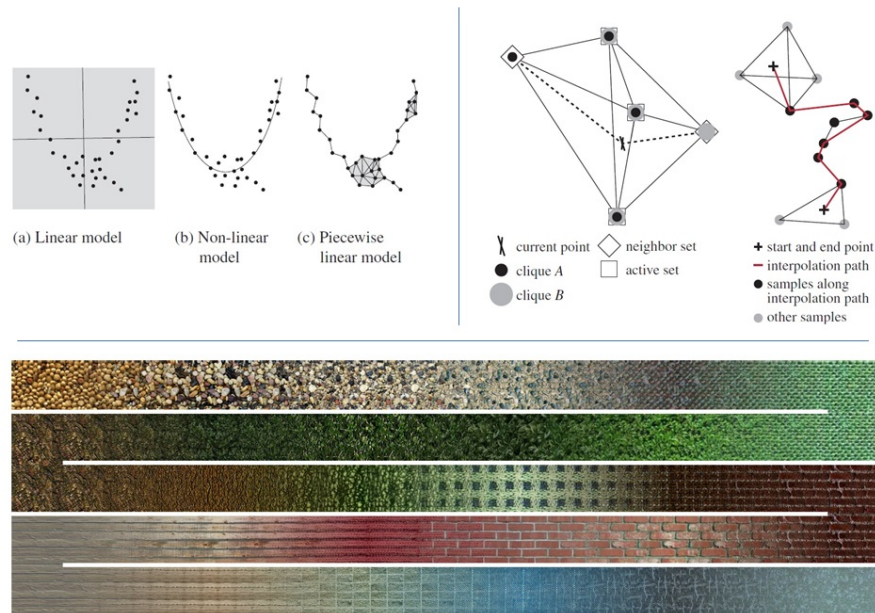


Ces travaux, en utilisant des processus ponctuels plus complexes que ceux de Poisson, permettent d’augmenter la gamme de phénomènes naturels représentables sous forme procédurale. Nous exploitons ces approches dans notre modèle procédural de structures que nous présentons dans le chapitre 8.

## 4.4 Interpolation de textures

Un outil supplémentaire courant en synthèse de textures est l’*interpolation*, qui permet d’étendre les textures produites. On présente ici une méthode en particulier, parmi plusieurs autres existantes.

Dans [MZD05], *Texture Design Using a Simplicial Complex of Morphable Textures*, Zhou *et al.* construisent un espace de textures sous la forme d’un complexe simplicial. Des cliques permettent de regrouper des apparences proches entre lesquelles interpoler, ou parcourir le plus court chemin. L’utilisateur peut naviguer dans cet espace pour ensuite interpoler les textures (figure 4.21). Le contrôle des structures par ces méthodes est difficile. Dans notre approche, nous proposons d’interpoler les structures afin de garder le contrôle sur celles-ci.



**FIGURE 4.21** – Interpolation de textures dans un espace de complexes simpliciaux [MZD05].

# Chapitre 5

## Modélisation Procédurale Inverse de Textures et de Matériaux

Dans ce chapitre, on présente un état de l’art de la synthèse de textures et de matériaux dans le cadre de la modélisation procédurale inverse. Comme précédemment, on s’attache à déterminer des points communs entre méthodes et approches, tout en soulevant leurs limitations.

### 5.1 Approches classiques

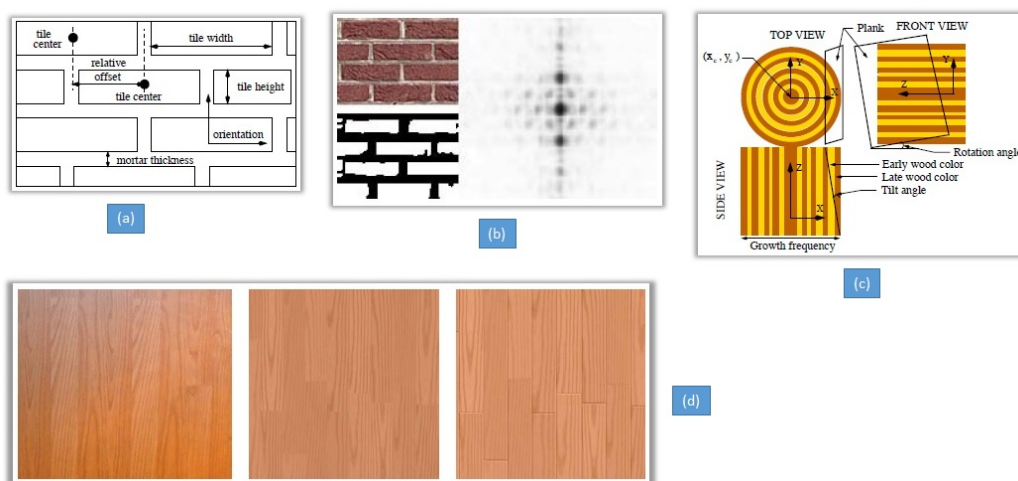
#### 5.1.1 Extraction et recherche de paramètres procéduraux

On s’intéresse ici à la *détermination des paramètres procéduraux d’un type de classe de textures donné*, afin de reproduire l’apparence visuelle d’une texture en entrée, et ensuite de pouvoir l’éditer.

#### Modèles de structures procédurales spécifiques

Dans [LP00], *Analysis and synthesis of structural textures*, Lefebvre et Poulin proposent une approche pour extraire les paramètres procéduraux d’une texture d’entrée de type structurée, en se restreignant à des motifs de pavages rectangulaires et de bois. Pour cela, deux modèles procéduraux simples sont proposés (figure 5.1), afin que l’étape d’estimation des paramètres permette de reproduire un masque binaire de structures similaires

aux motifs contenus dans l'entrée. Une fois la structure procédurale déterminée, la synthèse des détails de couleurs, basée sur la méthode d'Heeger and Bergen [HB95], est guidée par la structure. L'utilisateur peut ensuite créer des matériaux en éditant des paramètres tels que la rugosité, des coefficients de réflexions et déplacement map, etc. L'estimation des paramètres procéduraux (par exemple, taille, orientation, etc.) s'effectue à partir de la transformée de Fourier (FFT) du masque binaire de structure de l'entrée extraite automatiquement et/ou à l'aide de l'utilisateur. L'approche permet également de combiner les deux types de structures pour créer des pavages de planches en bois. L'utilisateur peut intervenir dans toutes les étapes (choix du modèle procédurale [pavage ou bois], modification et raffinement du masque de structures, édition des paramètres extraits) et peut ainsi guider l'algorithme.



**FIGURE 5.1** – Analyse et synthèse de textures structurées. (a) et (c) : modèles procéduraux de pavages rectangulaires et de bois. (b) : extraction du masque de structures et estimation des paramètres procéduraux à partir de sa FFT. (d) : exemple de synthèse : l'entrée à gauche, puis 2 exemples de synthèse avec possibilités d'imbrication des 2 types de structures (voir [LP00]).



## Modèle de texture procédurale : séparation de la structure et de la couleur

Dans [GKHF14], *Interactive Parameter Retrieval for Two-Tone Procedural Textures*, Gieseke *et al.* proposent un modèle de texture séparant la couleur et la structure. Dans ce modèle, une texture  $T$  est supposée contenir deux couleurs RGB principales  $\mathbf{c}_1$  et  $\mathbf{c}_2$ , ainsi qu'une carte de structure  $s_a(\mathbf{x})$  variant spatialement et indiquant comment mélanger (*blend map*) ces deux couleurs (figure 5.2). Cette carte de structure est une fonction scalaire continue,  $s_a(\mathbf{x}) : \mathbb{R}^2 \rightarrow [0, 1] \in \mathbb{R}$ , permet de mélanger les deux couleurs par une simple combinaison linéaire :

$$T_{\mathbf{a}, \mathbf{c}_1, \mathbf{c}_2}(\mathbf{x}) = (1 - s_a(\mathbf{x})) \mathbf{c}_1 + s_a(\mathbf{x}) \mathbf{c}_2$$

Cette structure  $s_a(\mathbf{x})$ , générée procéduralement, est paramétrée par un vecteur de paramètres  $\mathbf{a} \in \mathbb{R}^n$  en fonction de classes de textures prédéfinies, comme : des pavages, des cellules, du bois, du marbre, ainsi que des textures stochastiques basées sur des fonctions de bruits (Perlin, turbulence) comme du béton, de la rouille, des nuages et la surface de végétaux. À partir d'une image en entrée, après avoir extrait les deux couleurs par PCA, le modèle procédural de structure le plus proche est déterminé automatiquement par une recherche dans une banque de *shaders*, grâce à une représentation sous forme de *descripteurs de textures* caractérisées par des *caractéristiques globales* telles que la luminosité et le contraste, ainsi que des *caractéristiques locales* extraites grâce à des réponses de banques de filtres de type Gabor. La *métrique de similarité* utilisée permet de pondérer l'équilibre entre les caractéristiques globales et locales. Un point important consiste en l'échantillonnage des paramètres des *shaders*, permettant de précalculer une banque de descripteurs. Les auteurs indiquent l'importance d'un échantillonnage uniforme et aléatoire, car l'espace d'apparence n'est pas linéaire perceptuellement (tous les paramètres n'influent pas tous autant sur l'apparence et de manière non-linéaire). Le scénario principal est celui d'un artiste ayant pré-sélectionné une classe de textures ressemblant à une image d'entrée (par exemple des briques), et souhaitant déterminer les paramètres du modèle procédural automatiquement. La méthode consiste ici en une régression des paramètres du modèle procédural. Il n'y a pas d'étape de classification pour déterminer la classe de la texture d'entrée automatiquement, ce qui nécessiterait une métrique de similarités de types de textures et des techniques de *classification* ou *clustering*.

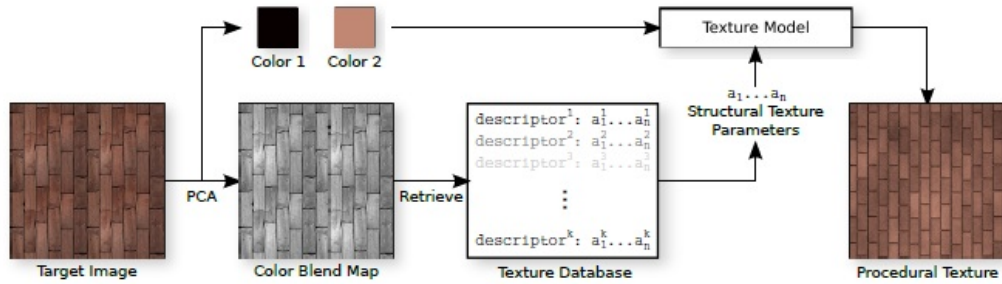
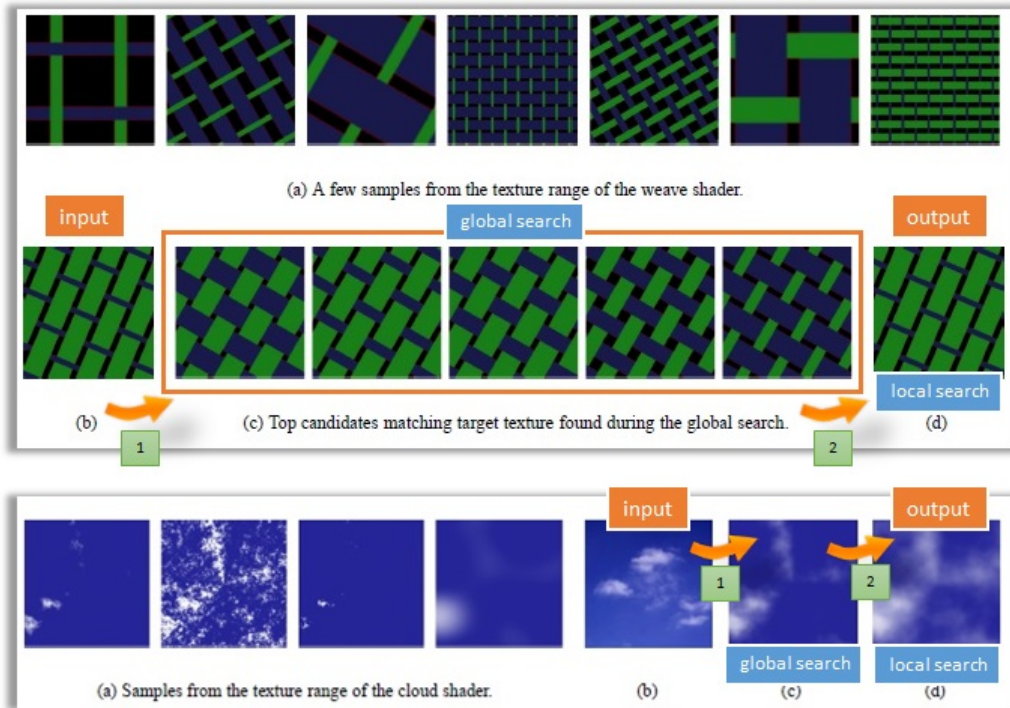


FIGURE 5.2 – Modèle de textures à deux couleurs et une couche de structure procédurale [GKHF14]

### 5.1.2 Extraction et recherche de modèles procéduraux et de leurs paramètres

L’approche précédente de [LP00] est difficilement généralisable et ne présente pas de tests sur des textures non structurées. Afin de pouvoir étendre leur méthode, il est nécessaire de créer une base de données de modèles procéduraux avec, pour chacun, une liste de paramètres spécifiques, et la sélection automatique du modèle nécessiterait des techniques de classification qui pourrait engendrer des problèmes de similarité si le modèle de structure n’est pas suffisamment proche visuellement. Dans [BD01], *Image-Driven Procedural Texture Specification*, Bourque et Dudek travaillent à l’automatisation de la sélection d’un *shader* le plus similaire parmi un catalogue de *shaders*, ainsi que de ces paramètres de textures procédurales à partir d’une image (figure 5.3). Leur approche est ensuite améliorée et étendue dans [BD04], *Procedural texture matching and transformation*, où ils proposent également un algorithme permettant de créer une séquence de textures entre deux ou plusieurs *shaders*, en essayant de rester proche perceptuellement (figure 5.4).

Plus précisément, dans [BD01], *Image-Driven Procedural Texture Specification*, Bourque et Dudek travaille à l’automatisation de la sélection des paramètres de textures procédurales à partir d’une image. Étant donné une texture à reproduire  $T$ , ainsi qu’un ensemble de textures procédurales  $(P_i)_{i \in \mathbb{N}}$  (par exemple, générateurs de briques, de nuages) avec, pour chacune, un jeu de paramètres et de valeurs valides  $\mathbf{x}_i$  (par exemple, largeur, longueur, espacement et orientation pour des briques; amplitude et lacunarité du bruit pour des nuages), on souhaite trouver une texture procédurale  $P_k$  avec une valeur possible de son jeu de paramètres  $\mathbf{x}_k^*$ , telle que  $P_k(\mathbf{x}_k^*)$  génère une



**FIGURE 5.3** – Spécification de textures procédurales basée image. Ligne du haut : plusieurs configurations d’un même modèle procédural de briques. Ligne du milieu : (b) entrée, (c) résultats d’une recherche globale de candidats similaires dans l’espace des paramètres, (d) résultat final d’un raffinement par recherche locale autour des candidats. En bas : même chose qu’au milieu (voir [BD01]).

texture similaire à  $T$  en entrée. Une recherche exhaustive étant trop contraignante, et souhaitant une interactivité, le processus est scindé en 2 étapes : une recherche globale approximative sur tous les types de textures en échantillonnant *uniformément* l’espace de leurs paramètres, puis un raffinement local des paramètres  $\mathbf{x}_i$  de la texture candidate précédente la plus proche de l’entrée  $T$ . Pour cela, ils se basent une notion de similarité perceptuelle, ou distance  $D$ , entre deux textures  $T_1$  et  $T_2$ , concrètement une MSE (*mean square difference*) sur leur contenu fréquentiel, à savoir les modules de leurs transformées de Fourier  $F_m$  :

$$D(T_1, T_2) = \sum_x \sum_y (F_m(T_1)[x, y] - F_m(T_2)[x, y])^2, \quad \text{sur tous les pixels } (x, y)$$

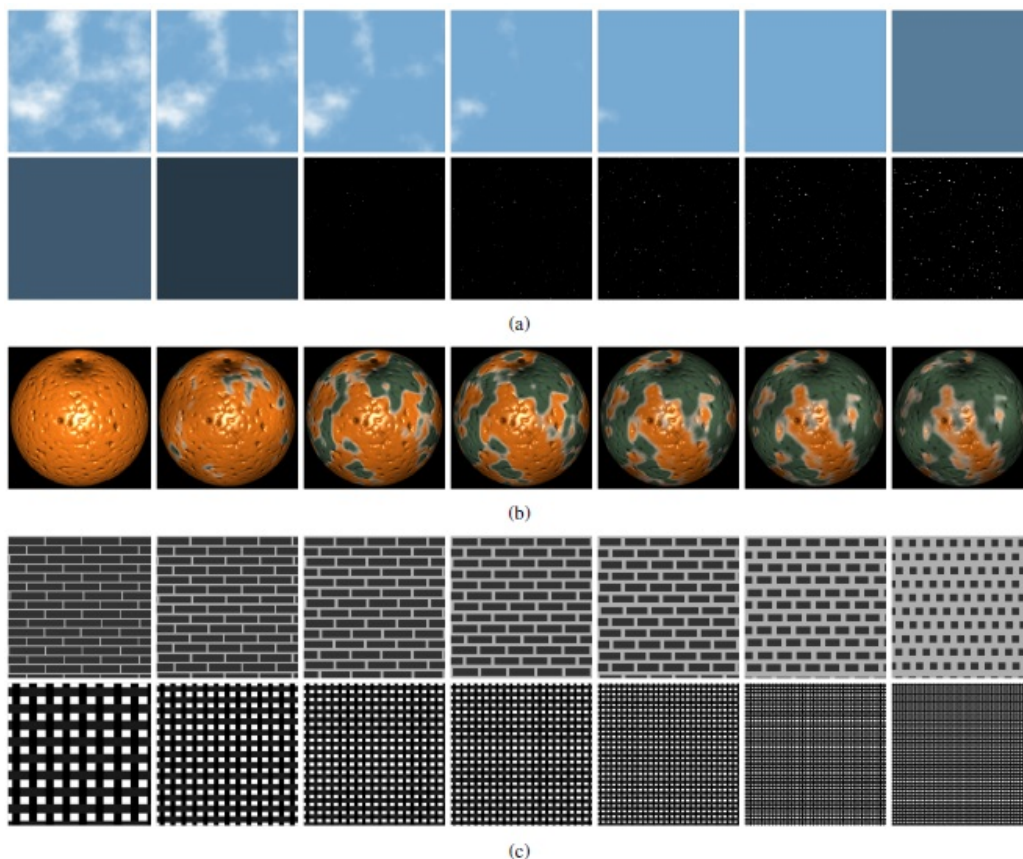


FIGURE 5.4 – Spécification de textures procédurales basée image [BD04]

La première étape peut être accélérée en précalculant toutes les transformations de Fourier. L'espace des paramètres et des solutions étant gigantesques, ils simplifient leur approche en fixant certains paramètres, en restreignant l'espace des valeurs possibles, testent leur méthode sur un très petit échantillon de textures (en réalité deux : brique et nuage). Il est difficile de valider et généraliser ce modèle, mais l'approche est intéressante. Se pose alors la question de qu'est-ce qu'une bonne métrique de similarité, ou plutôt qu'est-ce qu'une métrique adaptée? Existe-t-il une métrique générique pour tout type d'images, ou faut-il utiliser diverses métriques selon le type et contenu des textures comme évoqué dans leur article où, à chaque type de texture procédurale  $P_i$ , est associée une mesure de distance  $D_i$  adaptée à la discrimination de son contenu. Concernant leurs hypothèses, le modèle à reproduire est

supposé être contenu ou proche du modèle de texture procédurale. De plus, la recherche globale est censée trouver un candidat suffisamment proche de la solution pour que l'optimisation locale qui suit puisse converger sans rester coincé dans des minima locaux. Ceci est en fait difficilement vérifiable, à part expérimentalement sur un jeu de données particulier.

## 5.2 Décomposition sur une base de BRDFs

Dans [LBAD<sup>+</sup>06], *Inverse Shade Trees for Non-Parametric Material Representation and Editing*, Lawrence *et al.* s'intéressent à la possibilité d'extraire le graphe de nœuds d'un matériau à partir d'une image, afin de pouvoir l'éditer. Cette décomposition se base sur des fonctions 1D ou 2D qui capturent le comportement des BRDF et de leur variation spatiale, afin de modéliser des SV-BRDF. Ici, le but est de pouvoir éditer et rendre le matériau en temps réel. Seul le matériau est modifié, la structure de l'image reste inchangée.

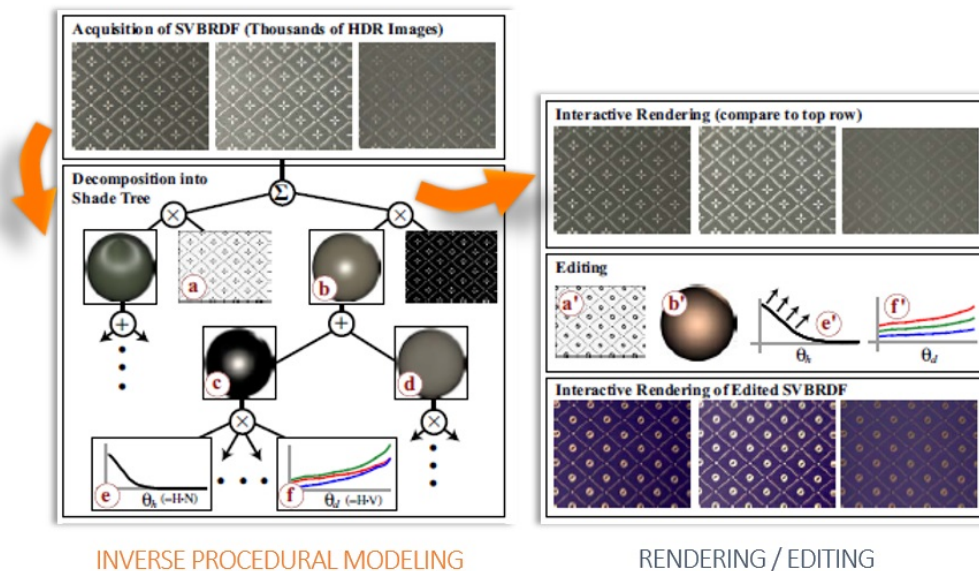


FIGURE 5.5 – *Inverse Shade Trees* [LBAD<sup>+</sup>06]. L'extraction du graphe de nœuds paramétriques à partir d'images permet une édition et un rendu interactif.



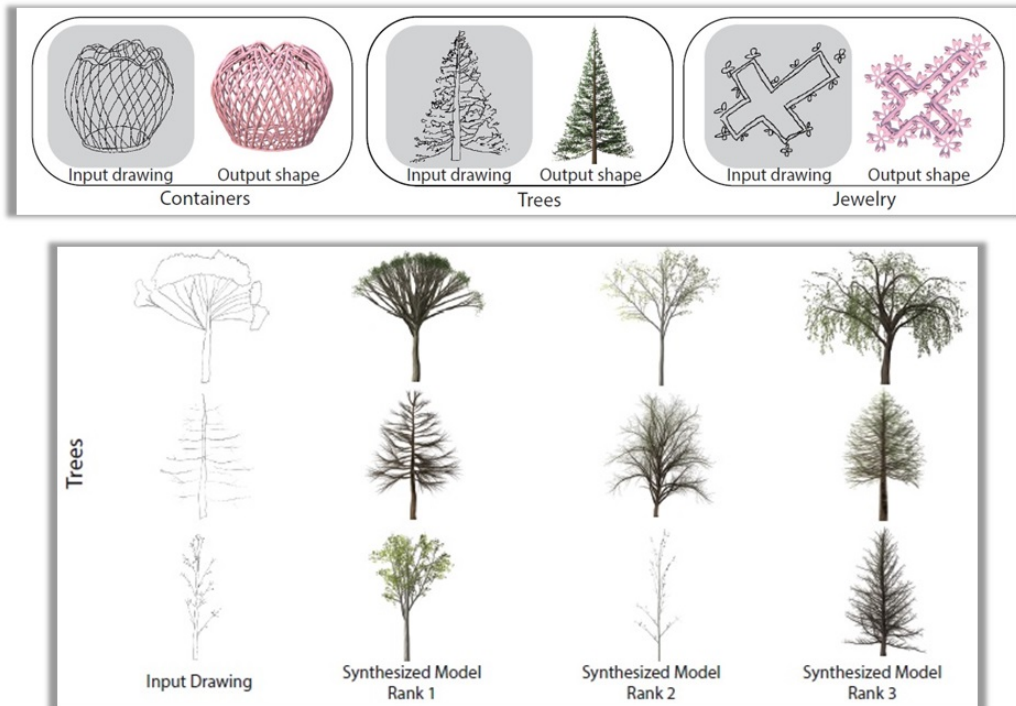
## 5.3 Approche par apprentissage automatique

On présente quelques méthodes et approches permettant de déterminer et générer des modèles procéduraux à partir d'exemples de textures, de matériaux, ou autres, selon diverses techniques propres au *machine learning* et au *Deep Learning*.

### 5.3.1 De croquis vers des modèles procéduraux

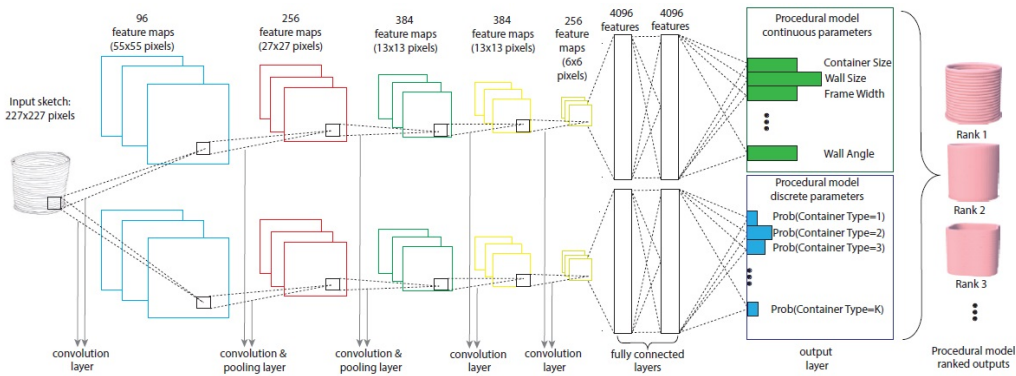
Dans [HKYM16], *Shape synthesis from sketches via procedural models and convolutional networks*, Huang *et al.* s'intéressent à la génération de modèles 3D procéduraux (par exemple : arbres, bijoux, vases) à partir de croquis 2D dessinés par un utilisateur (figure 5.6). Pour chaque catégorie d'objets, ils disposent d'un ensemble de règles génératives (un modèle procédural générique) permettant de générer des sous-catégories grâce à une combinaison de paramètres discrets (par exemple, des famille d'arbres : chêne, pin, érable, palmier, etc.) et d'autres continus (par exemple, largeur du tronc, hauteur, taille des feuilles, etc.). Étant donné un croquis 2D fourni par un utilisateur ainsi que sa catégorie (par exemple, "arbre"), leur approche consiste à déterminer les paramètres procéduraux (discrets et continus) afin d'être le plus ressemblant visuellement. De manière interactive, le système fournit en sortie un ensemble de paramètres procéduraux permettant de générer plusieurs candidats 3D selon un classement (ex : les 3 premiers), puis l'utilisateur est libre d'en choisir un et de l'éditer. En pratique, leur approche consiste en l'utilisation successive de deux réseaux de neurones convolutionnels (CNN), un premier pour déterminer les paramètres discrets de la sous-catégorie (par *classification*) et un second pour déterminer ses paramètres continus (par *régression*). Les réseaux utilisés sont des réseaux pré-entraînés existants (Alex-Net), construits pour des tâches de classification d'images naturelles, puis spécialisés par *transfer learning* en remplaçant la couche de neurones finale avec une nouvelle couche contenant soit le nombre de classes discrètes souhaité, soit le nombre de paramètres continus (figure 5.7). Une fois le premier réseau spécialisé pour déterminer les paramètres discrets par classification, ses poids sont ensuite utilisés pour initialiser et spécialiser le deuxième réseau utilisé pour déterminer les paramètres continus par régression. Ensuite, après l'étape d'apprentissage, les réseaux sont utilisables à la demande, de manière interactive, pour estimer les paramètres de modèles procéduraux à partir de croquis. Le jeu de données des croquis en entrée (image en niveau





**FIGURE 5.6** – Génération de modèles procéduraux à partir de croquis ([HKYM16] *Shape synthesis from sketches via procedural models and convolutional networks*).

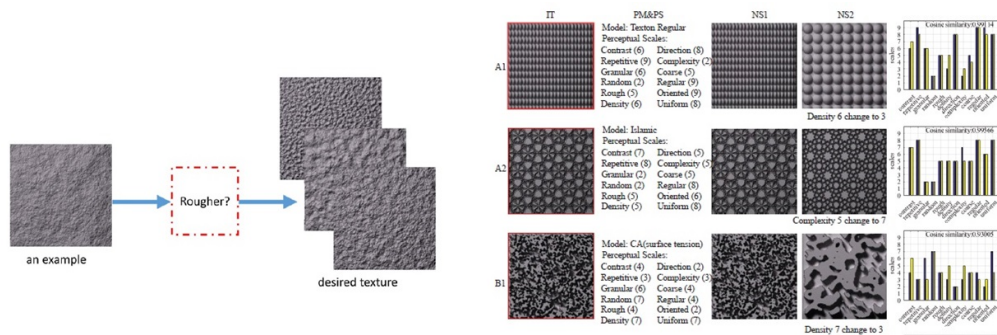
de gris) est généré automatiquement par échantillonnage des paramètres procéduraux continus (par *Poisson disk sampling*) pour toutes les combinaisons de paramètres discrets. Ces modèles 3D échantillonnés sont *simplifiés* automatiquement pour ressembler à de véritables croquis 2D utilisateurs (par exemple, morphologie mathématique). Enfin, le jeu de données est réduit manuellement (si des images sont trop similaires), et en utilisant un seuillage automatique sur les *feature vectors* (descripteurs) des croquis pour diminuer le nombre de données et réduire le temps d'apprentissage. On retrouve, une fois de plus, la difficulté du choix de l'échantillonnage de l'espace des paramètres procéduraux et des stratégies pour y pallier (par exemple, intervention manuelle). Le problème vient des descripteurs ne capturant pas assez d'informations perceptuelles discriminantes, voire également de la métrique de similarité.



**FIGURE 5.7** – Génération de modèles procéduraux à partir de croquis ([HKYM16] *Shape synthesis from sketches via procedural models and convolutional networks*).

### 5.3.2 De caractéristiques perceptuelles vers des textures procédurales

Dans [LGD<sup>+</sup>18], *Perception-driven procedural texture generation from examples*, Lui *et al.* s'intéressent à la génération de textures procédurales grâce à leurs caractéristiques perceptuelles, comme le contraste, les répétitions, la granularité, l'aléatoire, la rugosité, la densité, la directionnalité, la complexité structurelle, la rugosité (*coarseness*), la régularité, l'orientation locale et l'uniformité. Ces caractéristiques, qui représentent une valeur moyenne perceptuelle associée à une texture avec des valeurs d'intensité, peuvent être modifiées par l'utilisateur (figure 5.8).



**FIGURE 5.8** – Modélisation procédurale inverse basée sur des caractéristiques perceptuelles. Génération de textures visuellement similaires par modification de caractéristiques perceptuelles (contraste, densité, rugosité, etc.) [LGD<sup>+</sup>18].

**Apprentissage (figure 5.9)** À partir d'un jeu de textures procédurales donné (ex : cellulaire, fractal, Perlin...), un ensemble de textures est généré par échantillonnage de leurs paramètres (*uniformément* par soucis d'efficacité), puis un réseau de neurone dédié est chargé d'extraire automatiquement des caractéristiques (*features*) sous la forme d'un descripteur (un regroupement d'histogrammes). L'essentiel de leur approche se base sur le catalogue de descripteurs ainsi généré. Des données supplémentaires sont ajoutées pour toutes les textures générées, comme une valeur de similarité entre tous les couples de textures, ainsi qu'une valeur pour chacune des caractéristiques perceptuelles. Cette base de données de descripteurs est ensuite utilisée, dans le cadre d'un apprentissage supervisé, pour entraîner des modèles (*SVM : Support Vector Machine*) à déterminer le type de modèle procédural parmi le jeu de textures procédurales d'entrée (classification) et à déterminer chacune des caractéristiques perceptuelles avec leur intensité. En outre, un espace perceptuel de textures (*Perceptual Texture Space*), généré à partir des données d'entrée, permet de travailler dans un espace 3D représentant la similarité perceptuelle des textures plus proche de la perception humaine (et donc plus orienté utilisateur), grâce à une technique de réduction non-linéaire de la dimension.

**Génération (figure 5.9)** À la génération, à partir d'une texture d'entrée, le descripteur est extrait par réseau de neurone, puis la partie prédiction permet de déterminer le modèle procédural le plus proche, et ses caractéristiques perceptuelles, ensuite une recherche de plus proches candidats dans l'espace perceptuel des textures permet de raffiner le résultat, puis éditer ses caractéristiques perceptuelles.

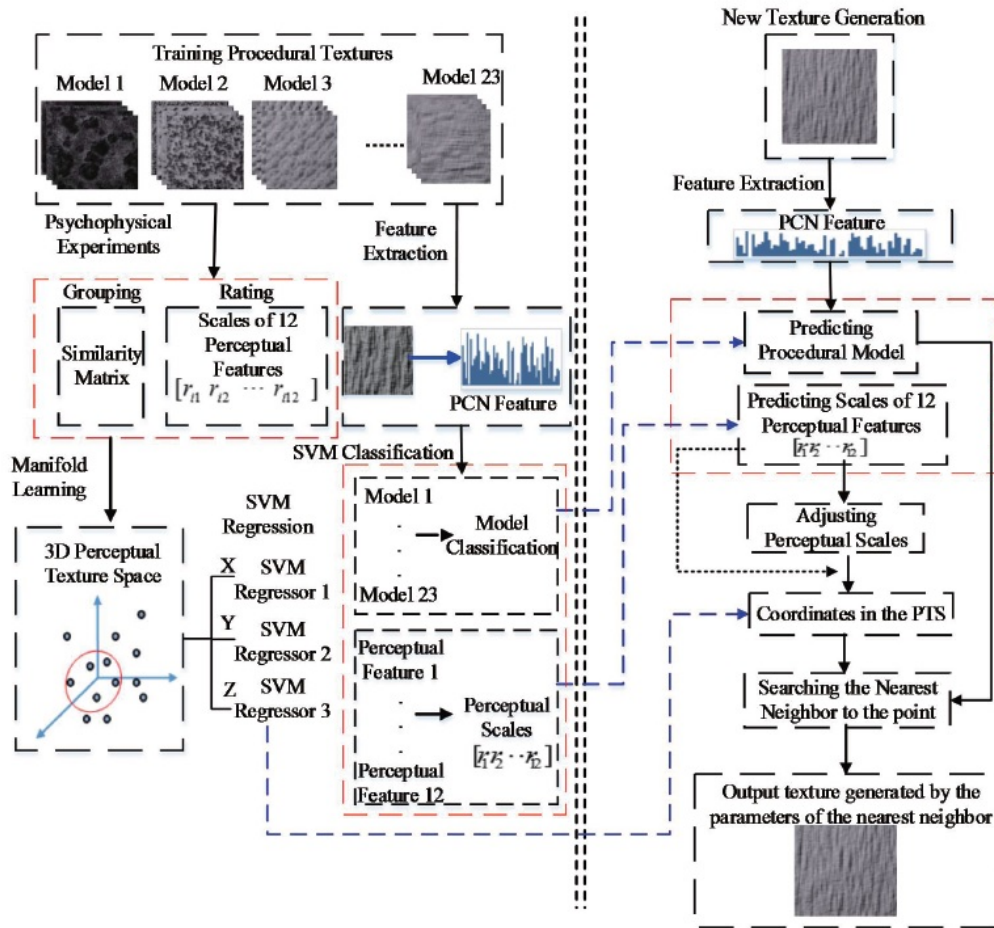


FIGURE 5.9 – Modélisation procédurale inverse basée sur des caractéristiques perceptuelles. Schéma général. À gauche : apprentissage pour l’analyse. À droite : synthèse par caractéristiques perceptuelles [LGD<sup>+</sup>18].

### 5.3.3 Apprentissage de préférences utilisateur et espaces latents

Dans [ZFWW18], *Gaussian material synthesis*, Zsolnai-Fehér *et al.* s'intéressent à l'automatisation de la génération de contenu dans les scènes 3D virtuelles, notamment l'aide à la sélection de matériaux. Les préférences de l'utilisateur, encodées sous forme de scores (*scalaire*) associés à ces matériaux préférés, sont apprises et stockées au fur et à mesure de l'utilisation d'un logiciel d'édition et de sélection de matériaux. La technique de Machine Learning est basée les processus gaussiens (*Gaussian Process*), selon deux modalités. La première utilise les GPR (*Gaussian Process Regression*) pour apprendre les paramètres des modèles procéduraux selon des scores de préférences, produisant une liste de plus proches candidats. La seconde utilise des GPLVM (*Gaussian Process Latent Variable Model*) pour naviguer dans un espace à 2D par réduction non-linéaire de la dimension, en projetant les candidats précédents dans le plan, et en associant les paramètres procéduraux du modèle à un score de similarité. Cet espace 2D, visualisé sous la forme d'une carte de scores colorée, permet de générer de nouveaux paramètres de matériaux, entre les candidats, grâce aux capacités prédictives du modèle GPLVM. L'utilisateur est alors libre de naviguer dans le plan en s'approchant ou s'éloignant des propositions (colorées par scores de préférences). Chaque matériau est visualisé grâce à un système de réseaux de neurones pré-entraîné pour transformer un jeu de paramètres d'un matériau procédural en image de rendu d'un même et unique modèle 3D statique (voir figure 5.10). L'approche a été testée sur une base de matériaux homogènes assez simples, sous forme de shaders programmables ([BS12] Physically-based shading at Disney), et non pas avec des matériaux en couches variant spatialement utilisés par les artistes dans les outils de texturing qui nous intéressent.

### 5.3.4 De textures vers des graphes de textures et matériaux procéduraux

Dans [HDR19], *A novel framework for inverse procedural texture modeling*, Hu *et al.* proposent une approche par modélisation procédurale inverse afin de générer des matériaux procéduraux en couches à partir d'une texture homogène en entrée. Leur méthode, en 2 étapes, se base sur des réseaux de neurones pour déterminer le type de modèle procédural, puis les paramètres procéduraux permettant d'obtenir le modèle le plus similaire (figure 5.11).

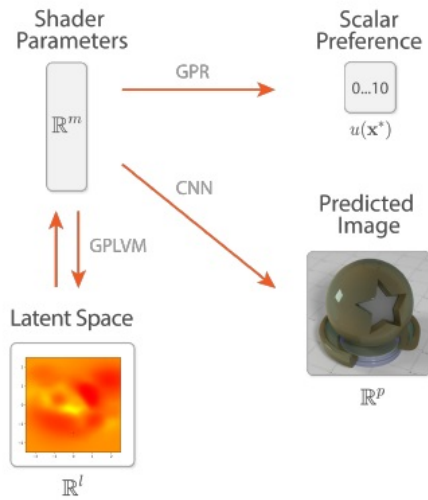


FIGURE 5.10 – Synthèse de matériaux par processus gaussiens (GPR, GPLVM). Apprentissage de scores de choix de matériaux préférés [ZFWW18].

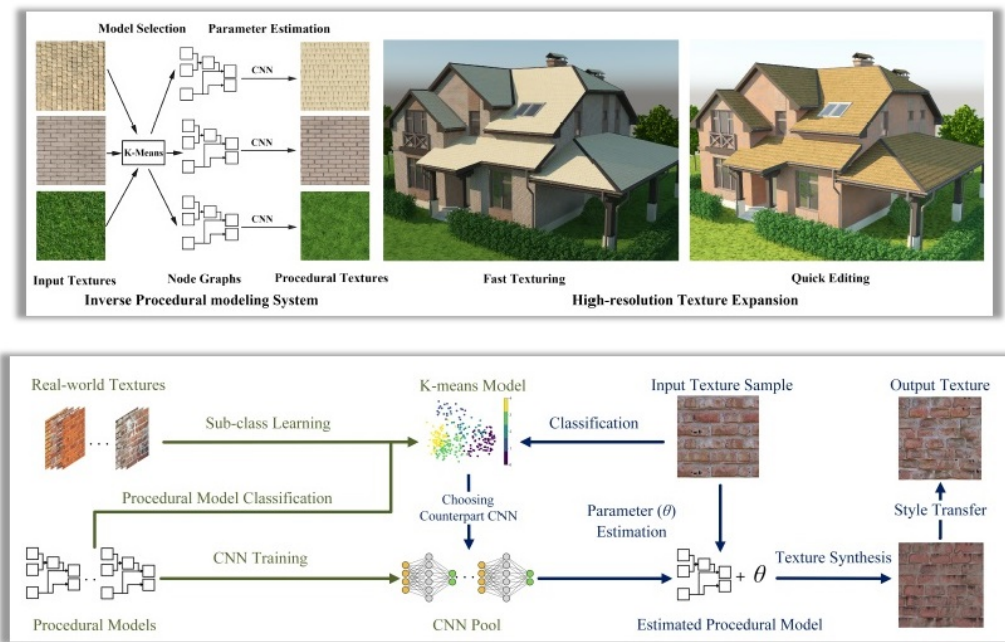


FIGURE 5.11 – Modélisation procédurale inverse de textures [HDR19].



Les résultats de la génération procédurale de [HDR19] ne permettent pas de reproduire l'apparence complexe de certaines textures ou matériaux, c'est pourquoi ils emploient une étape de transfert de style pour ajouter des détails d'apparence. Cependant, lors de cette étape, le modèle procédural n'existe plus au profit d'une représentation classique de texture par pixel, ce qui rompt l'interaction avec l'utilisateur.

# Chapitre 6

## État de l’art : Conclusion

Nous avons présenté un état de l’art des méthodes de synthèse de textures avec une attention particulière portée à la prise en compte de la structure et au contrôle de la synthèse dans le cas non homogène.

Au regard de nos objectifs, les méthodes les plus pertinentes sont celles permettant d’effectuer de la synthèse en temps réel, et de façon non bornée. En cela, les méthodes de synthèse non paramétriques *par pixel* (comme [LH05] et [LH06]), ainsi que les méthodes procédurales, sont les plus compétitives.

Les méthodes de synthèse par pixel, conçues au départ pour des textures vérifiant des hypothèses de stationnarité et de localité, nécessitent d’être guidées spatialement pour reproduire des structures et des textures non homogènes, ce qui peut être fait à l’aide de carte de labels [BELS10]. La limite de l’existant concerne ici les cartes de labels. Dans le cadre des textures stochastiques structurées, aucune méthode n’a été proposée pour générer ces cartes de guidage en temps réel et de façon contrôlée.

Du côté des méthodes procédurales, la reproduction de textures stochastiques structurées et de textures non homogènes demeure peu explorée. Parmi les fonctions de bruit [LLC<sup>+</sup>10], seul le bruit cellulaire de Worley [Wor96] apparaît comme capable de reproduire ce type de textures, mais avec une variété qui reste limitée.

Soulignons également que des progrès ont été faits récemment dans le domaine de l’estimation automatique des paramètres de modèles procéduraux à partir d’images, grâce notamment à l’apprentissage automatique et à l’apprentissage profond. Ceci s’applique notamment aux graphes de matériaux procéduraux.

L'approche que nous proposons dans cette thèse est hybride et cherche à tirer parti à la fois des avantages de la synthèse non paramétrique et de ceux de la synthèse procédurale. Nous choisissons de représenter de façon procédurale les cartes de guidage, à l'aide d'une fonction de texture de base suffisamment générale pour couvrir une grande variété de structures stochastiques. Ces cartes de guidages peuvent être générées *par l'exemple* par une méthode d'estimation de paramètres. Nous utilisons ensuite ces cartes de guidage dans le cadre d'une méthode de synthèse par pixel, elle-même *par l'exemple*, pour générer des textures et des matériaux. Nous qualifions notre approche de *semi-procédurale* en raison du recours à des méthodes non paramétriques et procédurales. Dans la partie suivante, nous présentons notre modèle.

Troisième partie

Modèle d'Apparence  
Semi-Procédural

## (i) INTRODUCTION

# Chapitre 7

## Approche *Semi-Procédurale*

Dans ce chapitre, on donne une description générale de notre approche de synthèse de textures et de matériaux, dite *semi-procédurale*, depuis sa conception. Nous présentons d’abord nos observations, à la fois dans le cadre de la création de matériaux par des artistes et sur des images de textures, qui ont guidé la construction de notre modèle d’un point de vue conceptuel. Nous présentons ensuite la formulation mathématique générale de notre modèle. Les détails du modèle (une *instanciation* particulière) seront exposés dans les chapitres suivants.

### 7.1 Observations sur les textures structurées

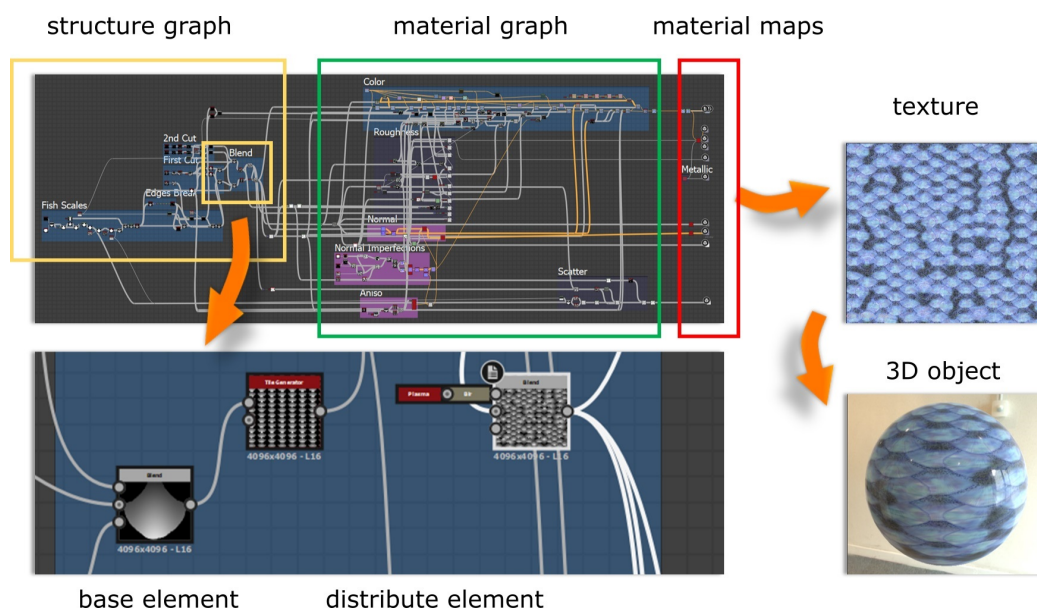
#### 7.1.1 Observations générales

Nous présentons ici deux méthodes artistiques de construction de textures qui nous ont orienté vers la définition d’un modèle de structures pertinent. Premièrement, nous décrivons la manière dont les artistes, infographistes, arrivent à reproduire des textures dans les films et jeux vidéos, grâce aux outils de graphes procéduraux comme Substance Designer d’Adobe Allegorithmic. Deuxièmement, nous décrivons comment procède les développeurs de type *technical director*, ou encore ceux de la *demoscene*.

**Structure dans un graphe de textures et de matériaux procéduraux.** Après analyse de plusieurs graphes de matériaux procéduraux de Substance Designer, on peut remarquer certains points communs. Pour gé-



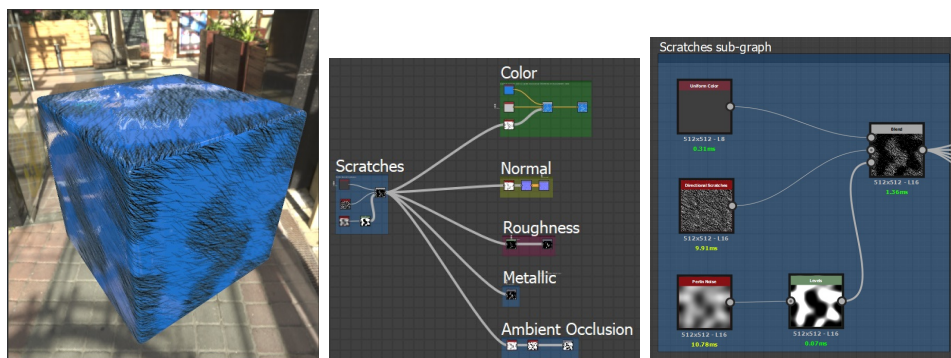
néer des couches de matériaux, on retrouve souvent des masques, binaires ou en niveaux de gris, sur lesquels viennent s'ajouter des détails de couleurs et de matériaux. On peut qualifier ces masques comme de la *structure*. À titre d'exemple, le graphe du matériau d'écailles de poisson (figure 7.1), illustre comment les différentes couches de matériaux sont modélisées à partir d'un graphe de structure définissant les écailles, sur lequel vient se greffer un autre graphe d'apparence modélisant les matériaux. Ici, la partie structure sert à modéliser la géométrie d'une écaille de poisson, sa primitive de base, puis à les distribuer dans l'espace en contrôlant leur répartition (par exemple, régulière ou aléatoire) ainsi que leur densité (nombre par unité de surface).



**FIGURE 7.1** – Création de couches de matériaux par des graphes de structure (primitives de bases et distributions spatiales) et de détails d'apparence (couleur et autres propriétés des matériaux).

On observe par ailleurs que les structures peuvent aussi être modélisées sous forme de bruits procéduraux, comme le bruit de Perlin. Le bruit sert à définir des régions plus ou moins étendues, avec des répartitions non uniformes et des motifs aléatoires. La figure 7.2 illustre un exemple de rayures.

Un autre point commun est que les parties du graphe définissant la struc-



**FIGURE 7.2** – Création de couches de matériaux par des graphes de structure (bruits et zones non homogènes) et de détails de d'apparence (couleur et autres propriétés des matériaux).

ture sont généralement plus simples et petits que ceux modélisant l'apparence (les détails de couleurs et de matériaux), qui à l'inverse sont souvent assez complexes.

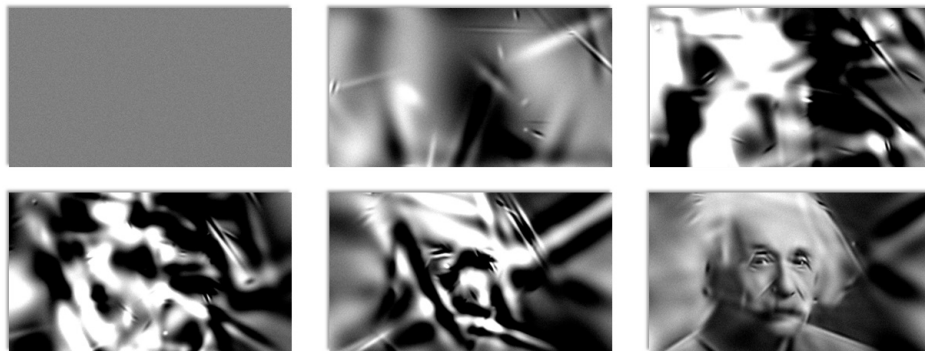
**Structure vue par des développeurs/Technical Directors.** Prenons l'exemple du développeur Inigo Quilez, qui illustre sur son site Shadertoy<sup>1</sup> des exemples de création de scènes virtuelles 3D entièrement procédurales. Celles-ci sont générées à partir de fonctions de distance signées (*Signed Distance Fields*, SDF) pour la modélisation des objets 3D, *ce qui peut être vu comme une partie structure*, puis une étape de texturing et shading indépendante pour ajouter des détails d'apparence (couleur et autres propriétés des matériaux). On y trouve également un exemple d'une approche de reconstruction d'images à partir de sommes de noyaux de Gabor (figure 7.3). La démo<sup>2</sup> est illustrée par une animation artistique où les noyaux apparaissent un à un. L'application, à l'origine de cette démo artistique, utilise des réseaux de neurones pour déterminer les paramètres des noyaux à partir d'une image en niveaux de gris fournie en entrée<sup>3</sup>. Cet exemple montre qu'il est possible de créer de la structure (ici essentiellement des contours) en contrôlant les positions, les orientations, les fréquences et les phases d'un ensemble de noyaux de Gabor. Cette approche représente en fait un cas particulier de dé-

1. <https://www.shadertoy.com/>

2. Démo : <https://www.shadertoy.com/view/4ljSRR>

3. Info : <https://mzucker.github.io/2018/04/27/image-fitting-tensorflow-rewrite.html>

composition d'une image sur une base d'ondelettes, le noyau de Gabor étant en effet souvent appelé ondelette de Gabor. Il s'agit d'une approche inspirée de l'analyse de Fourier, où la structure est représentée par une somme pondérée de fonctions de base. Dans le cas d'une décomposition en ondelettes, contrairement à l'analyse de Fourier, les fonctions sont à support spatial fini. Cette approche s'apparente à la définition d'un bruit par convolution éparse de noyaux de Gabor avec la différence que, pour le bruit, les positions ainsi que d'autres paramètres sont tirés complètement au hasard.



**FIGURE 7.3** – Exemple de modélisation par fonctions de Gabor. Une photo d'Einstein est modélisée à partir de noyaux de Gabor. Animation artistique, pas à pas, pour mieux visualiser le placement des noyaux (de haut en bas, et de gauche à droite).

### Conclusion et remarques

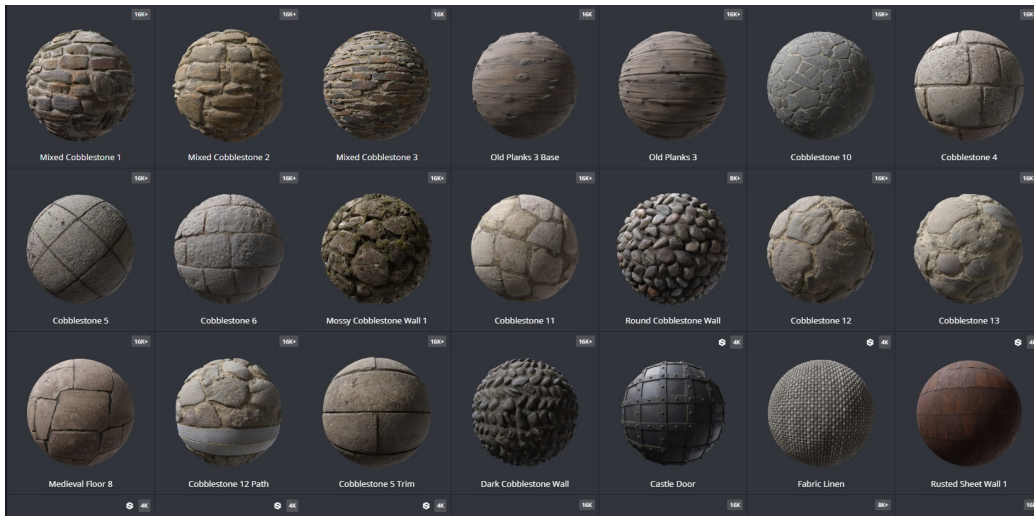
Pour conclure, nous constatons que la structure est souvent séparée des détails de couleur lors de la construction d'un graphe de texture, et que cette structure peut être définie à l'aide d'un bruit ou d'une somme de noyaux qui, lorsque les positions sont tirées au hasard, s'apparente à la définition d'un bruit de convolution éparse.

Ces observations ont été réalisées sur des textures stochastiques structurées, mais semblent également se vérifier sur d'autres catégories de structures. Cependant, pour le confirmer, il faudrait réaliser une étude plus approfondie. Nous nous inspirons de ces observations pour notre modèle semi-procédural de textures, en séparant la structure et les couleurs, comme nous le présentons dans la section suivante.

## 7.1.2 Les textures stochastiques structurées courantes

### Données en entrée

Nous nous sommes plus particulièrement intéressés à une façon de reproduire par notre modèle des textures stochastiques structurées courantes ou observables dans la nature. Nous sommes partis de l'observation d'images de textures naturelles, notamment à partir de l'album de Brodatz [Bro66], *Textures : A photographic album for artists and designers*, et du site internet <https://www.textures.com/> contenant un très grand nombre de textures naturelles et procédurales ainsi que des matériaux scannés et procéduraux (figure 7.4).

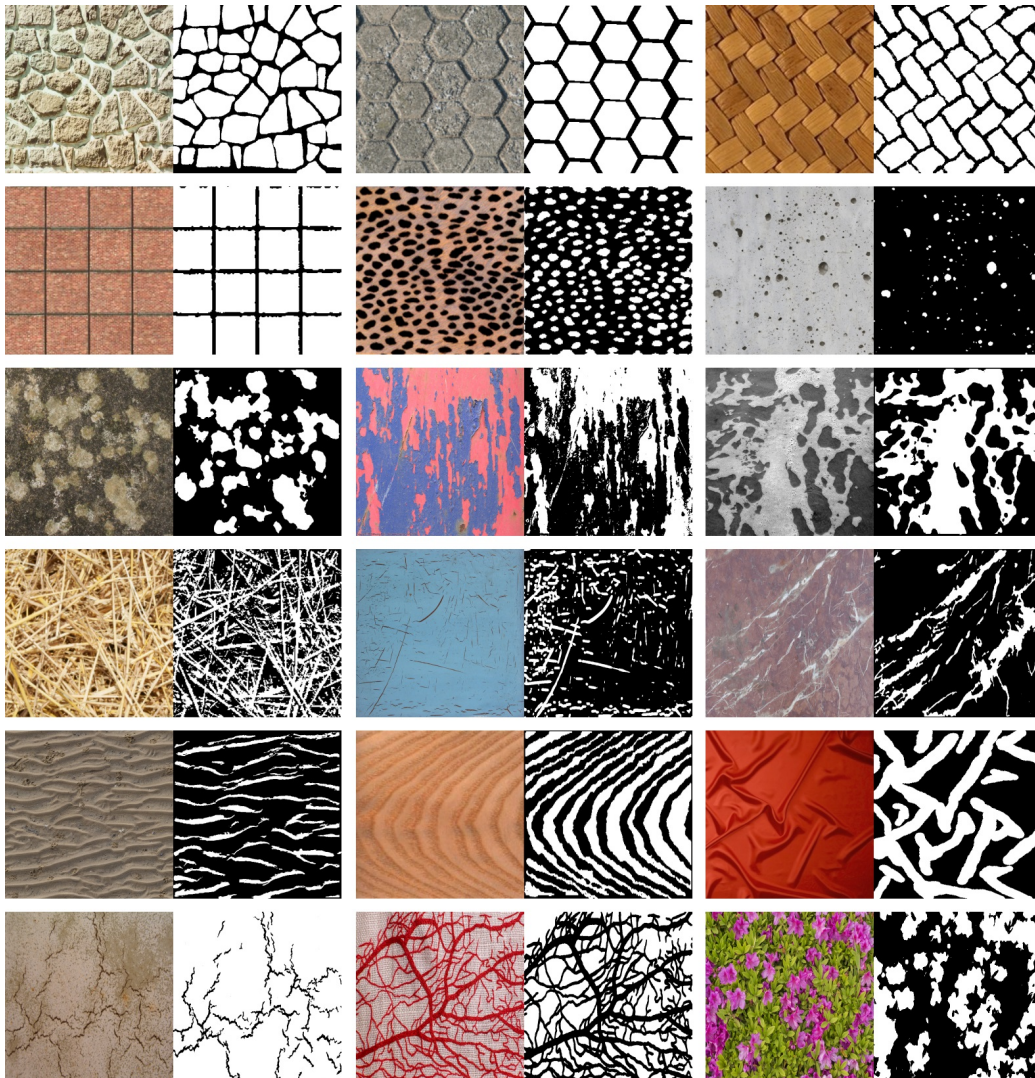


**FIGURE 7.4** – Accès à des bases de données de textures et matériaux, procéduraux ou non, scannés, etc. (voir <https://www.textures.com/>).

### Observations

De nombreuses textures contiennent divers types de structures stochastiques spatiales comme des cellules, fissures, grains, rayures, points, tâches, carreaux, vaguelettes et flocons : il s'agit d'un ensemble d'éléments visuellement et géométriquement bien identifiables. La figure 7.5 a été créée par segmentation manuelle de textures. Elle illustre une représentation dans laquelle la structure 12 est identifiée par une image binaire.

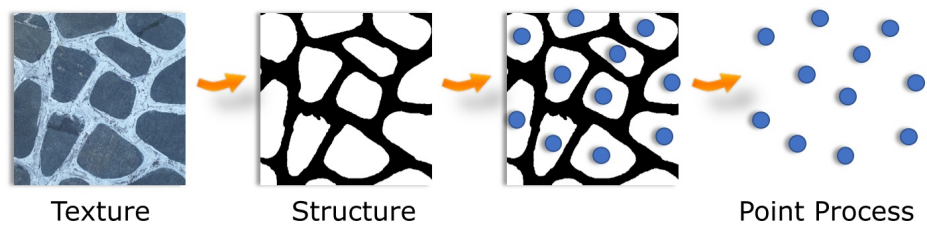




**FIGURE 7.5** – Caractérisation des textures naturelles par leurs *structures*.

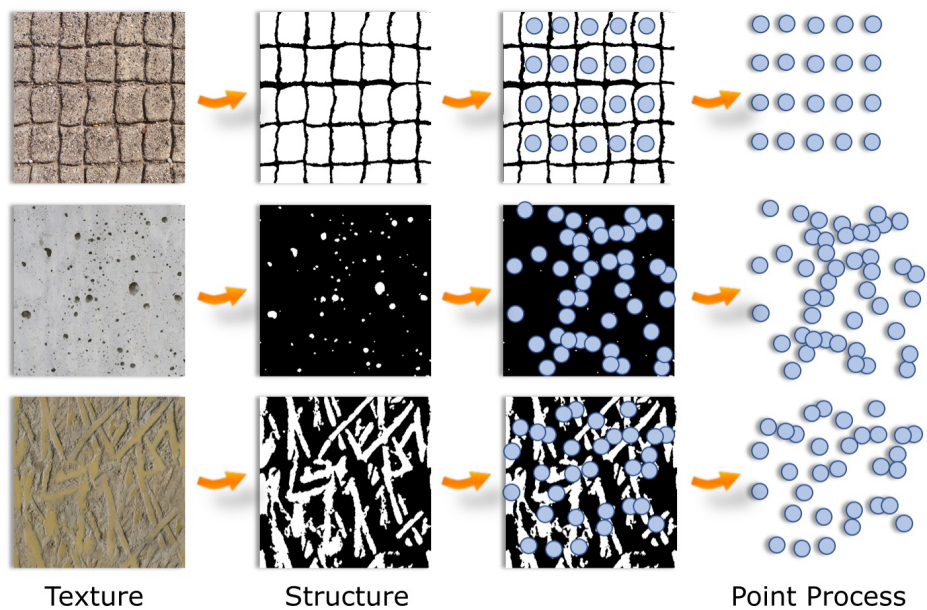
En observant ces images, on peut faire apparaître trois caractéristiques intéressantes :

**Distribution spatiale des éléments.** Premièrement, les structures peuvent être caractérisées par la **distribution spatiale de leurs éléments** localisables individuellement comme les pierres d'un carrelage (figure 7.6).



**FIGURE 7.6** – Texture et distribution spatiale de ses éléments.

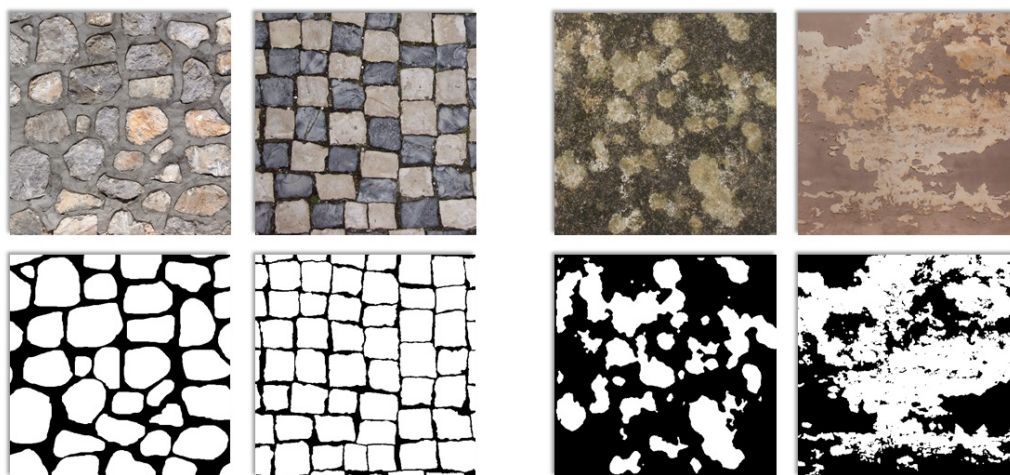
Les distributions peuvent être très variées selon les textures, allant de régulières à non-uniformes, voire complètement aléatoires, comme illustré sur la figure 7.7.



**FIGURE 7.7** – Exemples de distributions spatiales régulières, non-uniformes et aléatoires d'éléments.

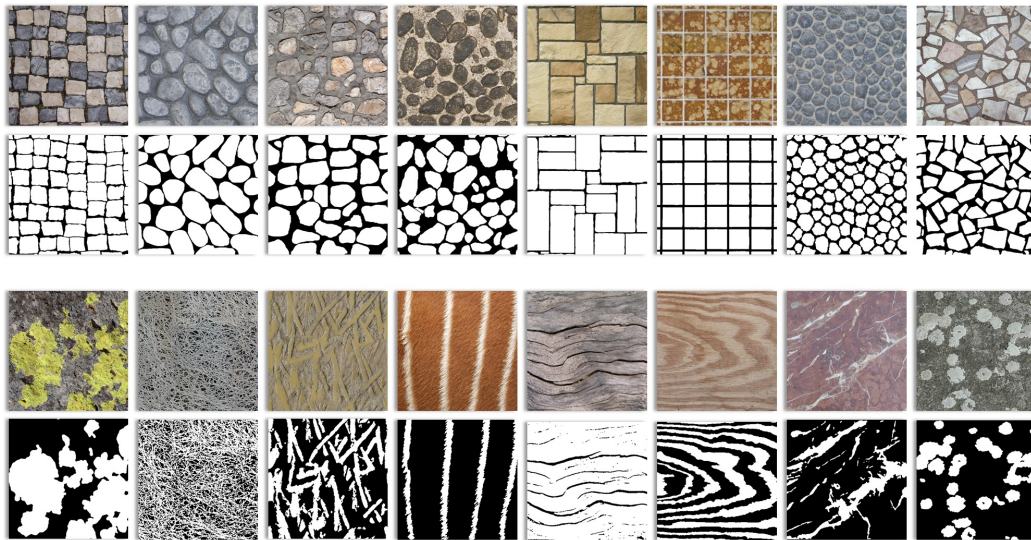


**Interactions mutuelles entre les éléments.** Deuxièmement, certaines structures révèlent des **interactions mutuelles entre leurs éléments**, par exemple en fusionnant des éléments comme avec des lichens (figure 7.8). Mais les éléments peuvent au contraire n'avoir aucun chevauchement, et se retrouver isolés spatialement, comme les pierres d'un mur ou les carreaux d'une mosaïque (figure 7.8).



**FIGURE 7.8** – Interactions mutuelles entre éléments. Différents types de fenêtres dont les éléments se recouvrent (à droite) ou non (à gauche). Fenêtres avec et sans recouvrement.

**Formes visuelles locales des éléments.** Enfin, les structures sont caractérisées par la **forme géométrique locale de leurs éléments**. On constate que les éléments stochastiques que l'on observe dans les textures naturelles ont souvent des formes visuelles générales assez simples : cellules, fissures, grains, rayures, tâches, carreaux, vaguelettes, écailles et copeaux (figure 7.9), même si leur contour peut dans certains cas avoir une forme géométrique complexe en s'apparentant parfois à une marche aléatoire.



**FIGURE 7.9** – Formes visuelles locales des éléments. Textures contenant divers types de structures stochastiques spatiales (cellules, fissures, grains, rayures, points, tâches, carreaux, vaguelettes et flocons).

## Conclusion

On vient de voir qu’il est possible de caractériser les structures stochastiques spatiales des textures par des images binaires. Dans une première approximation, les différents types de structures relevées semblent composés de distributions d’éléments assez simples. On a vu dans l’état de l’art que ces différents éléments visuels étaient modélisables par des fonctions de bruits, notamment par des techniques de convolution parcimonieuse (sparse convolution) et des *texture basis functions*.

Dans la suite, on montre comment, à partir de ces observations, il est possible de dériver un modèle procédural de structures. Pour cela, on introduit le concept de PPTBF (*Point Process Texture Basis Function*). On montre que cette représentation est en mesure de couvrir une gamme variée de structures stochastiques, incluant par exemple les cellules, les craquelures, les grains, les rayures, les tâches et les vaguelettes. On montre également que ce modèle peut s’étendre naturellement à la synthèse de structures non homogènes.

## 7.2 Vers un modèle générique procédural de structures

On décrit dans ce qui suit les trois composants qu'on propose pour caractériser la structure dans une texture.

### 7.2.1 Distribution spatiale des éléments

La figure 7.6 illustre le fait que l'on peut caractériser chaque élément visuel  $E_i$  par la position d'un point  $(x_i, y_i)$  du plan  $\mathbb{R}^2$  pouvant s'apparenter à un barycentre, que l'on appellera *feature point*. On modélise la distribution de l'ensemble de ces points par une somme d'impulsions de Dirac positionnées en chacun des *features points*  $(x_i, y_i)$ . Ainsi, étant donné un ensemble de *feature points*  $(x_i, y_i) \in \mathbb{R}^2$ , on note  $P$  la distribution telle que :

$$\forall (x, y) \in \mathbb{R}^2, \quad P(x, y) = \sum_i \delta(x - x_i, y - y_i)$$

Ces points sont à la base du contrôle de la spatialisation des éléments visuels.  $P$ , ainsi défini, correspond à un processus ponctuel. En pratique, les points seront construits par un ensemble de cellules  $R_i$  générées par pavage, dans lesquelles on range et organise les éléments. On fournit les détails de ce modèle au chapitre suivant.

### 7.2.2 Formes visuelles locales des éléments

On introduit une fonction  $F$ , que l'on nomme *feature function* pour représenter la forme d'un élément structurel d'une texture. Grâce aux propriétés de la distribution de Dirac, distribuer ces éléments dans l'espace revient à calculer une convolution entre  $F$  et le processus ponctuel  $P$  précédent, à savoir :

$$\forall (x, y) \in \mathbb{R}^2 \quad PPTBF(x, y) = P(x, y) * F(x, y)$$

Soit :

$$PPTBF(x, y) = \sum_i F(x - x_i, y - y_i)$$

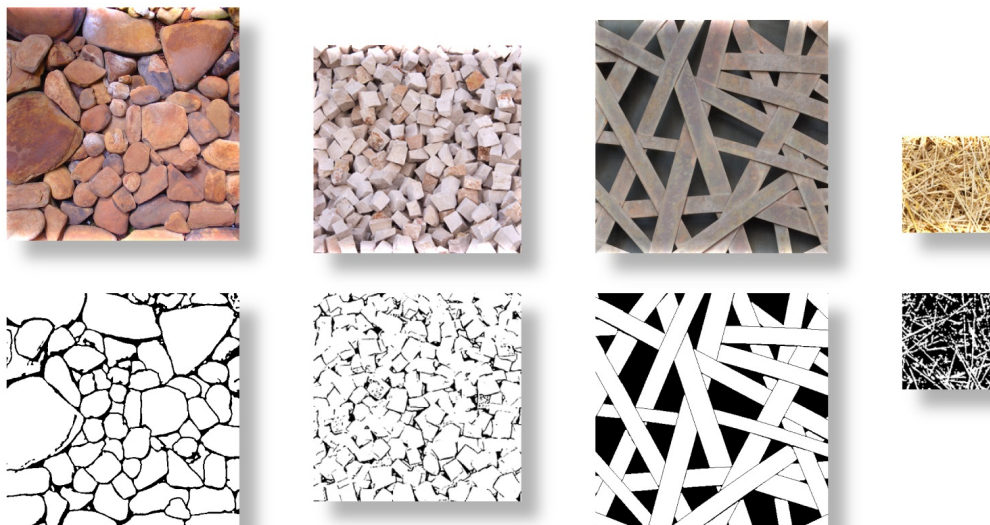
En pratique, on définit  $F$  comme un bruit. Dans l'état de l'art, on a vu que certains modèles de bruits procéduraux utilisent des sommes de noyaux

variant localement (convolution éparse). Pour définir  $F$  comme un bruit, on construit donc un deuxième processus ponctuel aléatoire local  $P_F$ , borné à l'intérieur des cellules  $R_i$  et permettant de sommer des noyaux,  $f_k$  soit :

$$\forall (x, y) \in \mathbb{R}^2 \quad F(x, y) = P_F(x, y) * f_k(x, y) = \sum_k f_k(x - x_k, y - y_k)$$

Par ailleurs, dans un souci de généralité, on propose de remplacer la somme précédente de la convolution éparse par un opérateur plus général de *mixture* de noyaux,  $\mathcal{M}$ . Il permet de simuler différentes manières de combiner les  $f_k$  (c.-à-d. autrement que de les additionner), par exemple de les empiler (voir figure 7.10), l'empilement étant utilisé notamment par le bombing :

$$F(x, y) = \mathcal{M}_k(f_k(x - x_k, y - y_k))$$



**FIGURE 7.10** – Certains types de textures contiennent des éléments qui s'empilent de manière aléatoire.

En pratique, on choisit de représenter  $\mathcal{M}$  par une somme, comme pour la convolution, mais pondérée par une fonction  $\mu_k(\mathbf{x})$  indiquant comment

contribue chaque noyau  $f_k$  :

$$F(\mathbf{x}) = \sum_k \mu_k(\mathbf{x}) f_k(\mathbf{x} - \mathbf{x}_k)$$

$\mu_k(\mathbf{x})$  dépend de l'opérateur de *mixture* souhaité.

On ne détaille pas plus cette partie. On verra au prochain chapitre quels types de fonctions  $f_k$  et quels  $\mu_k(\mathbf{x})$  nous avons choisi d'implémenter.

### 7.2.3 Interactions mutuelles entre éléments

Afin de contraindre spatialement les formes visuelles, on introduit une fonction de fenêtrage  $W$  pour *window function*, définissant l'extension spatiale des motifs visuels locaux et comment ils interagissent avec leurs voisins. En pratique, la fonction feature  $F$  précédente est modulée par  $W$  par multiplication, soit :

$$\forall (x, y) \in \mathbb{R}^2 \quad PPTBF(x, y) = P(x, y) * (W(x, y) F(x, y))$$

On distingue au moins deux familles de fenêtres : celles *sans recouvrement* (par exemple : cellules, pavages, briques, etc.), et celles *avec recouvrement* (par exemple : tâches éparées telles la rouille, le lichen, etc.). Nous unissons ces deux caractéristiques par le biais d'une unique fonction de fenêtrage générique définie sous la forme d'une combinaison linéaire d'une base de fonctions de fenêtrage  $w_j$  :

$$\forall (x, y) \in \mathbb{R}^2 \quad W(x, y) = \sum_j \lambda_j w_j(x, y), \quad \text{avec} \quad \sum_j \lambda_j = 1$$

Les  $\lambda_j$  sont des scalaires indiquant les poids dans la base.

On ne détaille pas plus cette partie. On verra au prochain chapitre quelle base de fenêtres  $w_j$  nous avons choisi d'implémenter.

### 7.2.4 Formulation discrète de la PPTBF

Pour résumer, le concept de PPTBF est représenté par trois composants :

Caractéristique	Modélisation	Fonction
distribution spatiale des éléments	processus ponctuel	$P$
interactions mutuelles entre éléments	fonction Window	$W$
formes visuelles locales des éléments	fonction Feature	$F$

Ceci se traduit, en formulation discrète, sous la forme du calcul de sommes suivant :

$$PPTBF(\mathbf{x}) = \sum_i \left( \left( \sum_j \lambda_j w_j (\mathbf{x} - \mathbf{x}_i) \right) \left( \sum_k \mu_k(\mathbf{x}) f_k (\mathbf{x} - (\mathbf{x}_i + \mathbf{x}_k)) \right) \right)$$

Note : En pratique, on verra qu'en tout point  $\mathbf{x}$ , les sommes des contributions de chaque *feature points*  $\mathbf{x}_i$  s'effectueront dans un voisinage réduit  $\mathcal{N}_i(\mathbf{x})$  autour du point  $\mathbf{x}$ .

Note : Les fonctions  $F$  et  $W$  sont définies en tout point du plan de la même façon. Leurs paramètres, eux, peuvent varier spatialement, et modifier l'apparence des structures générées selon la position dans le plan.

## 7.2.5 Transformations spatiales

On a vu qu'alors même que les éléments composant une structure sont assez simples, leur contour peut avoir une géométrie complexe, s'apparentant souvent à une marche aléatoire ou mouvement Brownien (figure 7.11). Ces formes aléatoires peuvent être modélisées à l'aide d'une déformation stochastique spatiale. Tout type de champ aléatoire de déformation est envisageable, notamment le bruit de Perlin, ou une de ses variantes fractales. Cette déformation peut être appliquée au modèle de PPTBF que nous venons de définir, sous la forme d'un opérateur de déformation  $\mathcal{D}$  agissant sur les coordonnées des positions  $\mathbf{x}$  auxquelles la PPTBF est évaluée :

$$PPTBF_{deformed}(\mathbf{x}) = PPTBF(\mathcal{D}(\mathbf{x}))$$

où le  $\mathcal{D}$  est par exemple un bruit de type *fractal Brownian motion* (fBm).

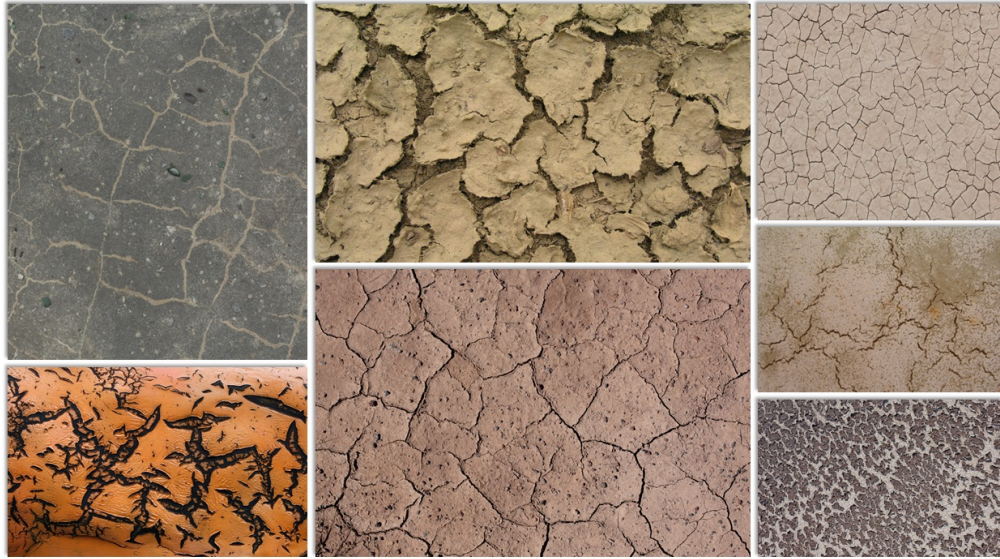
En plus de ces déformations aléatoires, le modèle PPTBF inclut également trois transformations spatiales (figure 7.12) : homothétie, translation et rotation.

$$PPTBF_{transformed}(\mathbf{x}) = PPTBF(\mathcal{T}(\mathbf{x}))$$

où le  $\mathcal{T}$  est la matrice permettant d'appliquer ces trois transformations.

Note : On préfère utiliser le formalisme  $\mathcal{T}$  pour les transformations spatiales, même s'il est courant d'indiquer les transformations inverses  $\mathcal{T}^{-1}$  (c.-à-d. de sens inversés) à la place.





**FIGURE 7.11** – Structures ayant des contours qui ressemblent à une marche aléatoire ou mouvement Brownien.



**FIGURE 7.12** – Ajout de transformations spatiales. L'ajout de transformations spatiales permet de modifier la résolution de la structure et adapter son apparence.

## 7.2.6 Génération des structures visuelles

La fonction PPTF définit un champ scalaire qu'on utilise pour générer des structures binaires par seuillage (voir figure 7.13). Une même famille de PPTBF avec des poids de fonction fenêtre différents génère différentes topologies après seuillage. Il est important de remarquer que la structure n'a pas besoin d'être très précise, car elle ne correspond qu'à la structure globale sous-jacente d'une texture et non pas à la texture finale.

Un paramètre  $\tau$  contrôle la valeur de seuillage :

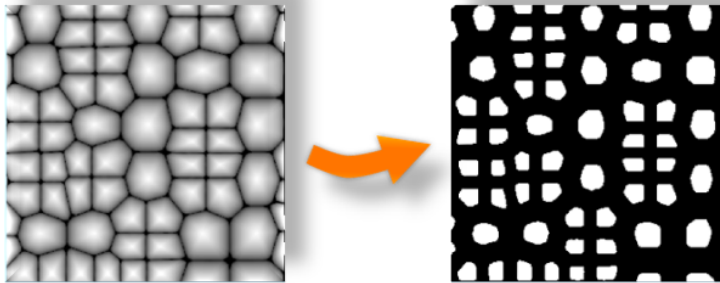
$$Structure(x, y; \tau) = U_{\tau}( PPTBF(x, y) )$$

où  $U_{\tau}$  est définie comme suit :

$$U_{\tau}(x) = U(x) * \delta(x - \tau) = U(x - \tau)$$

avec  $U$  la fonction de Heaviside (ou échelon unité) :

$$U(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$$

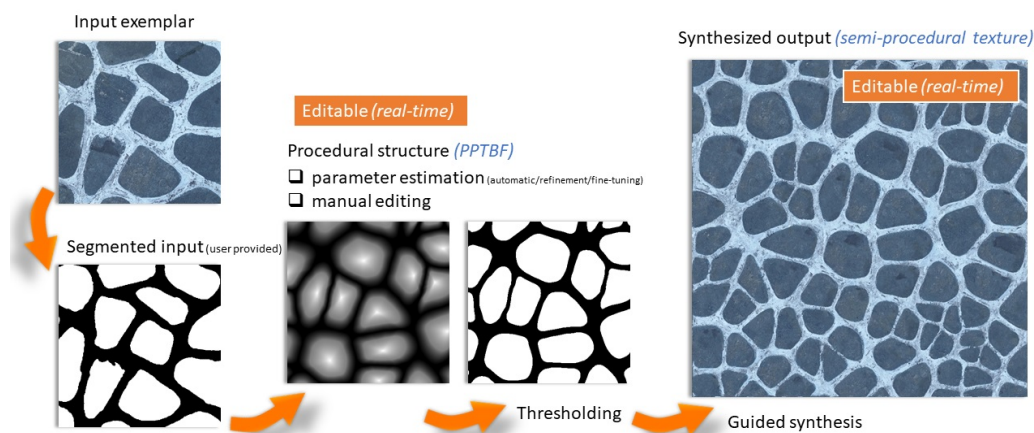


**FIGURE 7.13** – Génération de structures visuelles par seuillage. Le seuillage définit les structures visuelles.

### 7.3 L'Approche *semi-procédurale*

Dans l'état de l'art, on a présenté deux grandes familles d'algorithmes de synthèse de textures : la synthèse de textures par l'exemple et la modélisation procédurale. Les méthodes de synthèse par l'exemple requièrent de la stationnarité (homogénéité) et échouent dans de nombreux cas à préserver les structures (notamment globales ou étendues). Elles sont difficiles à contrôler et éditer à cause du manque de paramètres. La modélisation procédurale requiert, elle, une expertise et une création par essai et erreur. La modélisation procédurale inverse tente d'apporter une solution à ce problème, mais elle a besoin d'algorithmes de classification (supervisée ou non), qui ne peuvent pas garantir une ressemblance visuelle avec la texture exemple.

**Séparation structure procédurale et détails de couleurs.** Pour palier ces limitations, nous proposons une synthèse de textures capable de reproduire de façon automatique ou semi-automatique les structures observées dans des textures homogènes, à partir de petits échantillons de texture et d'une modélisation procédurale de la structure sous la forme d'une fonction générique. Pour cela, nous avons développé une méthode de synthèse de textures dites *semi-procédurales*. Elle consiste à séparer la synthèse de la structure, de la synthèse des détails de couleur. La structure est procédurale et utilise la PPTBF que nous avons introduite à la section précédente. Elle n'est pas bornée (infinie), sans répétition, cohérente dans tout le plan 2D et éditable à l'aide d'un ensemble de paramètres. Les détails de couleur sont ajoutés à partir d'exemples segmentés en utilisant une approche d'optimisation parallèle automatique engendrant une similitude visuelle. Cette méthode cherche ainsi à unifier la synthèse procédurale et la synthèse par l'exemple (figure 7.14).



**FIGURE 7.14** – Modèle de textures semi-procédurales. Séparation de la synthèse de structure de type procédurale, de la synthèse de la couleur par PCTS. À partir d’une texture d’exemple et sa carte de structure (segmentation manuelle), on détermine le modèle procédural le plus proche (estimation de ses paramètres) puis sa structure par seuillage. Ensuite vient la phase de colorisation (post-traitement).

Concrètement, à partir d’un modèle de structures procédurales PPTBF, une valeur de seuillage génère une carte de structure binaire qui nous permet d’utiliser une approche temps réel de génération de textures par l’exemple comme [LH05], *Parallel Controllable Texture Synthesis* et [BELS10], *Instant Texture Synthesis by Numbers*. Cependant, des adaptations sont nécessaires et seront expliqués au chapitre concernant la synthèse de textures et de matériaux 11.

Il est important de noter que nous ne générons pas directement des cartes de labels comme requises par l’approche *Texture-by-Numbers* [BELS10], mais que nous générons des champs scalaires qui, une fois seuillés, sont utilisables comme des cartes de labels. Cette indirection est essentielle et permet de conserver la structure sous la forme d’un modèle procédural unique. En effet, on ne souhaite pas avoir recours à toute une collection de modèles procéduraux spécialisés pour la génération de cartes binaires de structures.

Si l’on considère les outils de graphes de textures et de matériaux procéduraux, notre approche permet de remplacer les sous-graphes de structures par



## (ii) SYNTHÈSE DE STRUCTURES



# Chapitre 8

## Modèle de Structures Procédurales *PPTBF*

Dans cette partie, sur la base du concept de PPTBF présenté dans le chapitre précédent, nous développons une instance particulière pour laquelle nous précisons des choix de processus ponctuels (ou *point process*), fonction de fenêtrage (ou *window function*) et fonction de forme (ou *feature function*).

### 8.1 Instanciation du concept de PPTBF

On s'appuie sur la PPTBF vue au chapitre précédent, à savoir :

$$\forall \mathbf{x} \in \mathbb{R}^n, i \in \mathbb{N}^* \quad \left\{ \begin{array}{l} P(\mathbf{x}) = \sum_i \delta(\mathbf{x} - \mathbf{x}_i) \\ PPTBF(\mathbf{x}) = P(\mathbf{x}) * (W(\mathbf{x}) F(\mathbf{x})) \\ PPTBF(\mathbf{x}) = \sum_i W(\mathbf{x} - \mathbf{x}_i) F(\mathbf{x} - \mathbf{x}_i) \end{array} \right.$$

où  $P$  est une distribution de points,  $W$  une fonction de fenêtrage (*window function*) et  $F$  la fonction de forme (*feature function*). On montre une implémentation incrémentale du modèle, chaque composant pouvant être remplacé ou étendu pour augmenter le type de structures pouvant être couvert par le modèle. Les choix présentés dans ce chapitre ont, entre autres, été motivés par les bases de textures observées et par une implémentation temps réel, ou au minimum interactive, accélérée par programmation parallèle sur plusieurs

processeurs et/ou sur carte graphique. Dans la suite, on montre comment plusieurs variétés de *structures* sont encodées dans les 3 composantes de la PPTBF,  $P$ ,  $F$  et  $W$ .

## 8.2 Distributions procédurales de points

On s'intéresse ici à la modélisation du composant  $P$  (*processus ponctuel*) de la PPTBF, caractérisé par un ensemble de points  $\{\mathbf{x}_i \mid \mathbf{x}_i \in \mathbb{R}^n\}_{i \in \mathbb{N}^*}$ , nommés *feature points* :

$$\forall \mathbf{x} \in \mathbb{R}^n, i \in \mathbb{N}^* \quad \left\{ \begin{array}{l} P(\mathbf{x}) = \sum_i \delta(\mathbf{x} - \mathbf{x}_i) \\ PPTBF(\mathbf{x}) = P(\mathbf{x}) * (W(\mathbf{x}) F(\mathbf{x})) \\ PPTBF(\mathbf{x}) = \sum_i W(\mathbf{x} - \mathbf{x}_i) F(\mathbf{x} - \mathbf{x}_i) \end{array} \right.$$

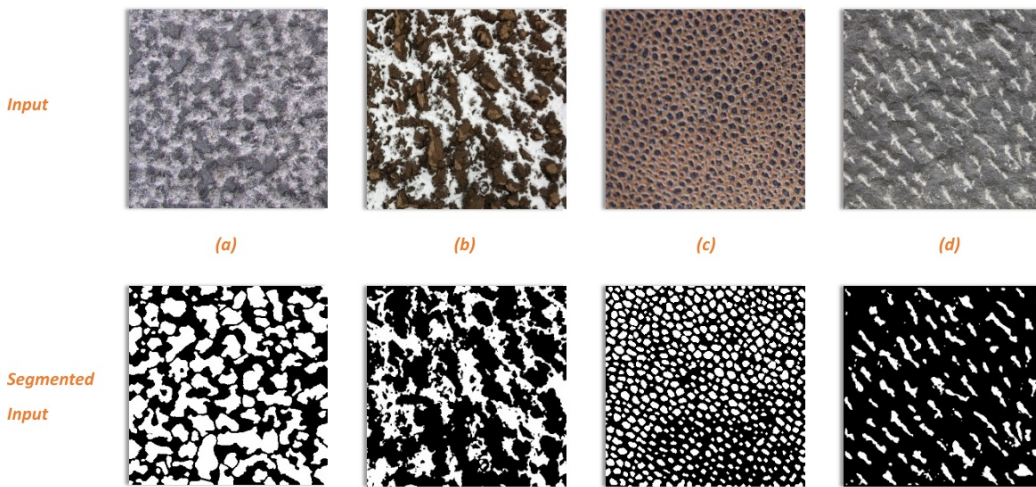
### 8.2.1 Objectifs

On souhaite pouvoir *synthétiser* des processus ponctuels spatiaux (*spatial point process*). Il en existe plusieurs types, le plus simple et le plus communément utilisé étant le processus ponctuel de Poisson. Il correspond à de l'aléatoire complet : les points sont indépendants les uns des autres et leur densité moyenne par unité de surface est constante sur  $\mathbb{R}^n$ . Il est utilisé comme base pour la génération de bruit par convolution éparse. Notre objectif est de proposer d'autres modèles, afin notamment de prendre en compte des phénomènes de regroupements ou d'interactions entre les points.

Dans un premier temps, on dresse une liste des phénomènes et types de processus ponctuels que l'on souhaite reproduire. Ensuite, on montre comment mettre en place un *framework générique commun* pour les simuler. Pour cela, on se base sur des méthodes procédurales, de manière à être efficace, avoir un jeu de paramètres contrôlables réduit et intuitif. Un des objectifs est de permettre une estimation des paramètres lors de phases d'analyse.

## 8.2.2 Différents phénomènes et types de processus ponctuels

À partir de l'observation de notre base de données de textures et de matériaux, nous avons constaté que quatre types principaux de processus ponctuels, comme illustré sur la figure 8.1, et présentés ci-dessous, était fréquemment, voire majoritairement, observables. Bien que des processus plus complexes existent dans la nature, comme par exemple la distribution d'espèces de végétaux, qui sont en compétition pour les ressources, nous ne les avons pas inclus dans notre approche. En fait, ce type de distribution sort du cadre de la génération des textures. Par ailleurs, ils nécessitent des simulations complexes, qui ne permettent pas une génération procédurale à la volée et sur un domaine non borné (infini).



**FIGURE 8.1** – Différents types de processus ponctuels. Distributions uniformes complètement aléatoires (Poisson) (a), distributions groupées avec agrégats (Cox) (b), approximations des distributions ponctuelles de Poisson-disque (Gibbs) (c), ainsi que les distributions quasi-régulières à régulières (d).

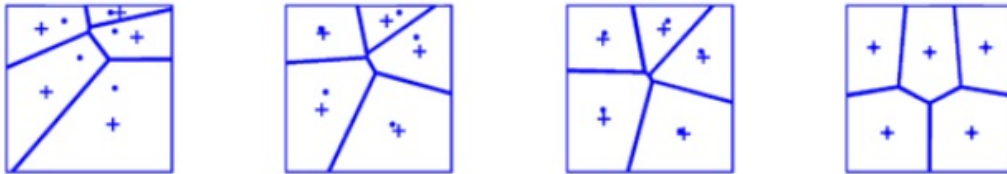
**Les Processus Ponctuels de Poisson** (figure 8.1 a) définissent l'aléatoire spatial complet, c'est-à-dire que l'intensité, qui est une mesure moyenne de densité de points, est constante sur  $\mathbb{R}^2$ .

**Les Processus Ponctuels de Cox** (figure 8.1 b), également appelés processus de Poisson doublement stochastiques, considèrent que l'intensité varie spatialement et qu'elle est elle-même un bruit (ou champ aléatoire).

Dans ce cas, les points ne sont pas distribués uniformément et forment des agrégats. Diverses techniques ont été proposées pour générer différents types de processus Cox. Le processus de Neyman-Scott est une approche directe qui utilise le processus ponctuel de Poisson pour définir les centres d'agrégats. Autour de ces centres, les points sont répartis selon une certaine fonction de densité de probabilité (PDF, pour Probability Density Function). Un cas particulier est le processus de Matern, qui consiste à tirer uniformément les points à l'intérieur de disques définis autour des centres d'agrégats.

**Les Processus Ponctuels de Gibbs** (figure 8.1 c) prennent en compte des interactions entre les points. Un exemple est le processus de Lennard-Jones qui produit une attraction à longues échelles et répulsion à petites échelles. Les distributions de type disque de Poisson (aussi appelé bruit bleu), distribue les points de manière à ce qu'ils ne sont jamais trop proches ni trop éloignés les uns des autres.

L'algorithme de Lloyd (voir [Llo82], *Least squares quantization in PCM*) peut être utilisé pour augmenter l'espacement minimum entre paires de points à partir d'un ensemble de points quelconque. L'algorithme construit le diagramme de Voronoï de l'ensemble des points et déplace ceux-ci dans la direction du centre de gravité de leur cellule de Voronoï. Ce processus est ensuite réitéré.



**FIGURE 8.2** – Algorithme de Lloyd. Processus de relaxation (voir <https://de.wikipedia.org/wiki/Hard-core-Prozess>).

### Processus ponctuels quasi-réguliers et réguliers

Ce type de processus est généralement basé sur des règles de placement, par exemple en utilisant les centres de gravité des cellules d'un pavage périodique du plan. La technique du *jittering* peut ensuite être utilisée pour perturber aléatoirement la position des centres, et ainsi définir des distribu-

tions quasi-régulières, notamment lorsque l’amplitude de la perturbation est faible (figure 8.1 d).

### 8.2.3 Génération procédurale

On souhaite générer les quatre types de processus ponctuels précédents procéduralement. En effet, rappelons qu’une génération procédurale possède plusieurs avantages : elle est parfaitement parallélisable, et du coup permet une génération temps réel sur GPU, elle est définie sur un domaine spatial non borné, ce qui permet d’éviter toute limitation de la taille du domaine, elle est extrêmement compacte en évitant toute forme de stockage dans des tableaux et elle est paramétrable, ce qui permet de générer et contrôler facilement de très nombreuses variantes.

#### Tessellations de l’espace et processus ponctuels

Les distributions de points (uniformes et avec agrégations) sont générées par dualité avec des tessellations du plan  $\mathbb{R}^2$ . À chaque cellule de la tessellation est associé un point. Généralement, les cellules d’une tessellation, qui peuvent avoir une forme géométrique quelconque, se nomment *prototiles* (en anglais *prototiles*). Nos prototiles se limitent à des cellules rectangulaires  $R_i$ . Ce choix est motivé par la simplicité et l’efficacité en matière de calculs : il est facile de tirer une position aléatoire dans un rectangle, alors que le problème peut devenir difficile si la forme géométrique de la prototile est plus complexe. Par ailleurs, il est plus commode de déterminer l’appartenance d’un point à un rectangle que l’appartenance d’un point à une forme géométrique quelconque. Plutôt que de traiter des tessellations quelconques, on se limite donc à des pavages rectangulaires du plan, majoritairement non-uniformes et irréguliers.

Ci-dessous, un exemple de pavage parfaitement régulier (figure 8.3).

À partir du pavage, on cherche à générer une distribution  $P$  d’éléments, exprimée par une somme d’impulsions de Dirac à des positions aléatoires  $\mathbf{x}_i$ , que nous appelons *feature points* :

$$P(x, y) = \sum_i \delta(x - x_i, y - y_i)$$

$$P(\mathbf{x}) = \sum_i \delta(\mathbf{x} - \mathbf{x}_i)$$



**FIGURE 8.3** – Pavage de l’Espace et Prototiles. La tessellation est formée de l’addition d’une prototile à chaque noeud d’un réseau spatial.

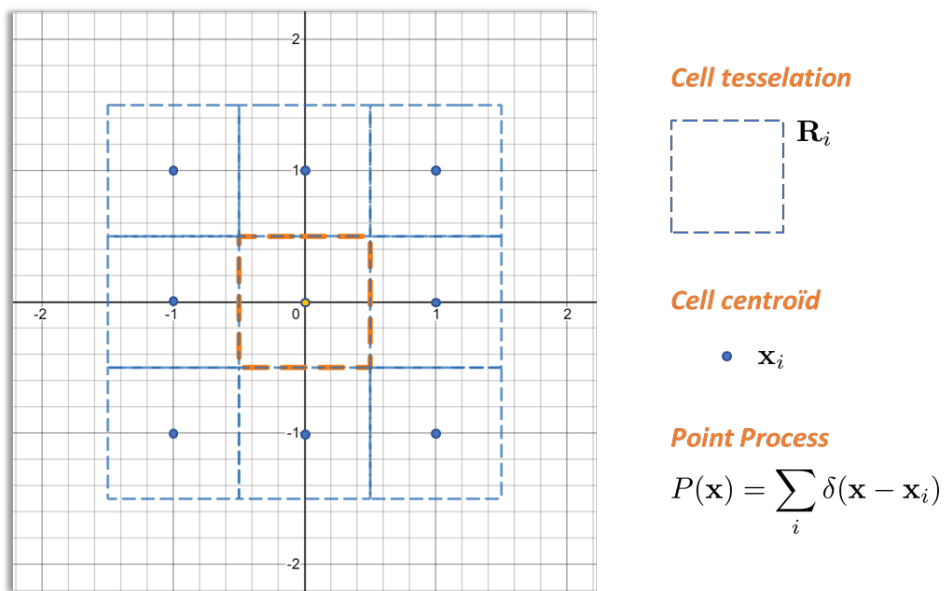
Les points  $x_i$  sont choisis à l’intérieur des cellules  $R_i$  en utilisant une variable aléatoire uniforme et en appliquant le principe de *jittering* à la position centrale de  $R_i$ . Un seul et unique point est tiré par  $R_i$ , comme illustré dans les figures 8.4 et 8.5. Un facteur d’amplitude qu’on appelle paramètre de *jittering* permet de gérer l’amplitude de l’aléa : une valeur nulle indique que c’est le point central qui est toujours donné. Dans ce cas, l’aléa est nul. Une valeur de 1 permet de tirer un point quelconque à l’intérieur de  $R_i$ . C’est l’aléa maximum.

Un pavage régulier basé sur des cellules  $R_i$  carrées peut facilement être utilisé comme approximation du processus ponctuel de Poisson. Il s’agit d’une approximation, car la distribution n’est pas complètement aléatoire étant donné que l’on a toujours nécessairement un point par  $R_i$ . Une approximation de meilleure qualité consisterait à sélectionner un nombre variable de points au hasard pour chaque  $R_i$ , selon une loi de Poisson. La densité moyenne est alors constante et correspond à l’espérance de la loi. Notre choix de n’avoir qu’un point par  $R_i$  permet cependant de garantir une meilleure performance de calcul, sans pour autant avoir d’impact négatif sur la qualité et variété des textures générées. Le pavage régulier en carré est un cas très particulier de pavage. Il est possible d’imaginer un très grand nombre de pavages réguliers ou irréguliers en rectangles. On décrit dans ce qui suit les pavages que nous avons réalisés. Dans tous les cas, notre principal critère est l’efficacité et la simplicité des calculs.

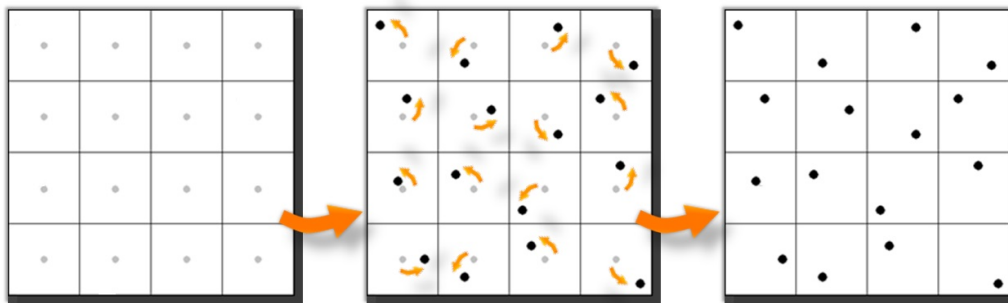
#### Décalage de Cellules

Un paramètre *decalx* permet de décaler les cellules horizontalement comme illustré sur la figure 8.6.



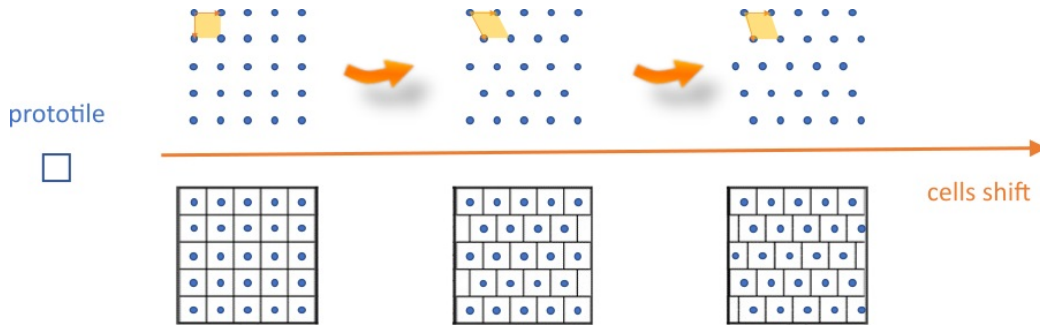


**FIGURE 8.4** – Modèle de processus ponctuel. Un pavage de l'espace en régions rectangulaires  $R_i$  et un tirage aléatoire d'un point  $x_i$  dans chacune d'elle définit un processus ponctuel  $P$ . L'exemple montre le cas particulier de régions carrées de taille constante et un paramètre de *jittering* nul.



**FIGURE 8.5** – Processus ponctuels et paramètre d'aléas. On tire un point  $\mathbf{x}_i$  aléatoirement dans chaque cellule  $R_i$  du pavage, ce qui génère un processus ponctuel  $P$ . Un paramètre de *jittering*  $j$  permet de passer d'une distribution stochastique à une distribution régulière et périodique.

Par exemple, si l'on souhaite décaler régulièrement des cellules sur l'axe



**FIGURE 8.6** – Décalage de cellules. Un paramètre  $decalx$  permet de décaler les cellules horizontalement.

horizontal de manière périodique, avec une périodicité de 3 :

$$\forall (i, j) \in \mathbb{Z}^2 \quad T(x, y) = \sum_i \sum_j \delta(x - ia - (|j|\%3)T_x, y - jb) * rect(x, y)$$

$$\forall (i, j) \in \mathbb{Z}^2 \quad T(x, y) = \sum_i \sum_j rect(x - ia - (|j|\%3)T_x, y - jb)$$

À titre d'exemple, comme illustré dans la figure 8.6, on peut souhaiter créer un agencement spatial périodique se répétant toutes les 2 ou 3 lignes horizontales, ce qui peut s'exprimer sous la forme d'un entier  $p_y$  indiquant la période :

$$\forall p_y \in \mathbb{N}_* \quad \forall j \in \mathbb{Z} \quad x \mapsto x - \frac{|j|\%p_y}{p_y}$$

$$\forall p_y \in \mathbb{N}_* \quad \forall j \in \mathbb{Z} \quad \begin{cases} x \mapsto x - \frac{|j|\%p_y}{p_y} \\ y \mapsto y \end{cases}$$

#### Pavages plus complexes

Les différents pavages suivent la formule :

$$\forall (i, j) \in \mathbb{Z}^2 \quad T(x, y) = \sum_i \sum_j \delta(x - ia, y - jb) * prototile(x, y)$$

Par exemple, le pavage du bas de la figure 8.7, s'exprime par deux fonctions rectangulaires translatées et mise à l'échelle, et distribuées par un réseau de vecteur  $(a, b)$  :

$$\forall (i, j) \in \mathbb{Z}^2 \quad T(x, y) = \sum_i \sum_j \delta(x - ia, y - jb) * \left( \text{rect}(x, y) + \text{rect}\left(\frac{x-1}{2}, \frac{y}{2}\right) \right)$$

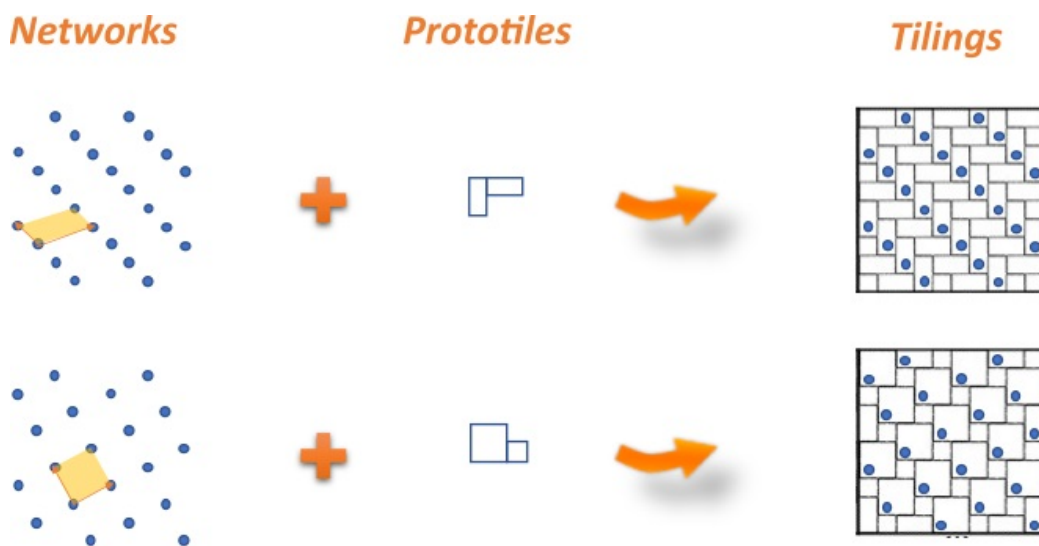


FIGURE 8.7 – Pavage de l'espace et prototiles.

## Regular Tilings

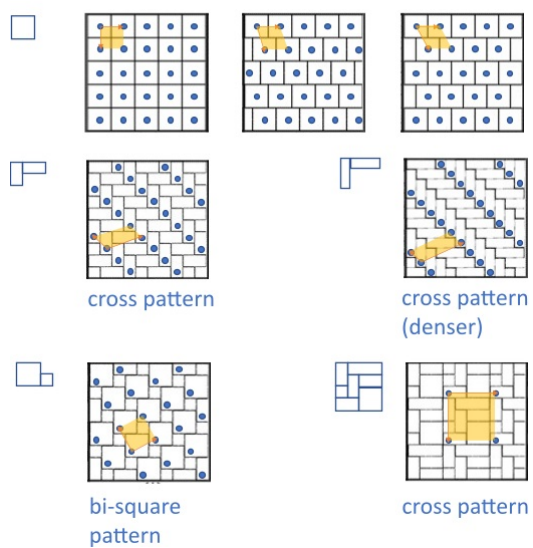
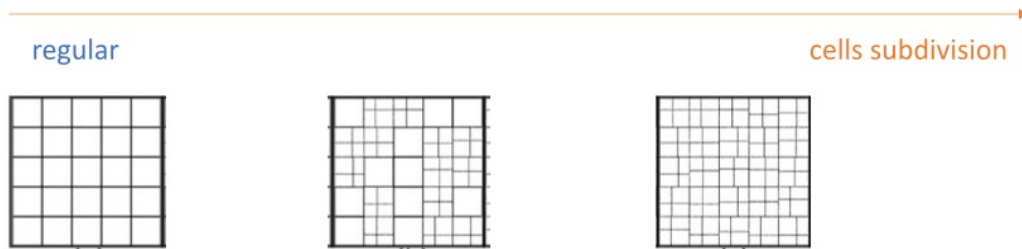


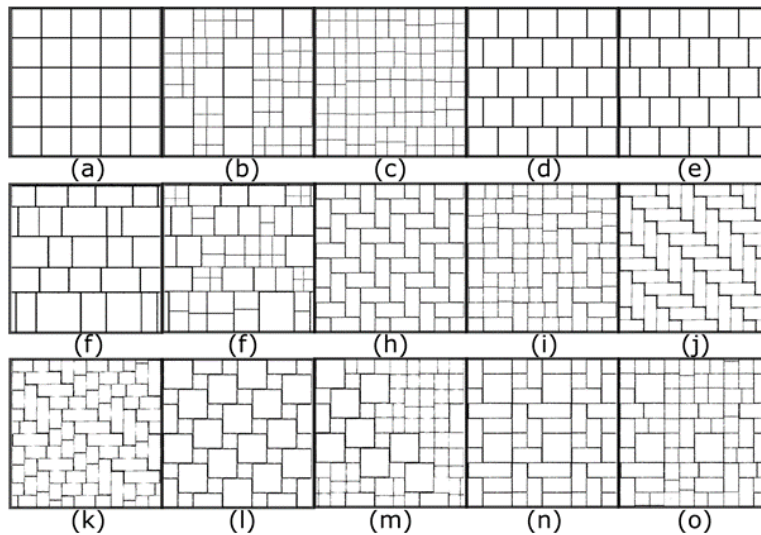
FIGURE 8.8 – Tessellations régulières et réseaux.

### Subdivision de Cellules

Un groupe de paramètres permet de subdiviser les cellules horizontalement et/ou verticalement comme illustré sur les figures 8.9 et 8.10.

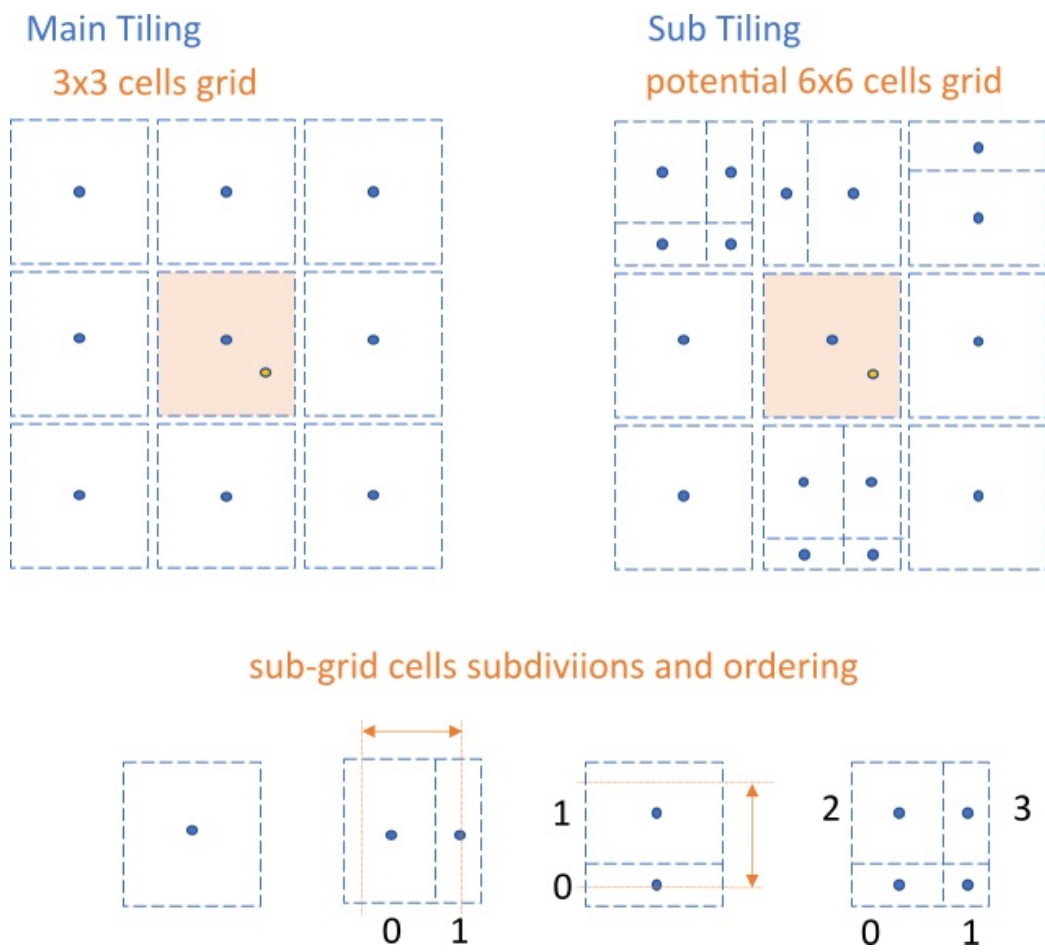


**FIGURE 8.9** – Subdivision de cellules. Un groupe de paramètres permet de subdiviser les cellules horizontalement et ou verticalement.



**FIGURE 8.10** – Différents types de prototiles.

Pour cela, deux paramètres permettent de contrôler la subdivision des cellules horizontalement et/ou verticalement selon des probabilités  $p_{subx}$  et  $p_{suby}$ . De plus, deux paramètres permettent de contrôler l'amplitude de l'éloignement par rapport aux centres des cellules,  $j_{itx}$  et  $j_{ity}$ . L'action de ces paramètres est illustré sur la figure 8.11.



**FIGURE 8.11** – Probabilités de subdivision de cellules. Des paramètres permettent de subdiviser les cellules horizontalement et ou verticalement avec un contrôle sur le positionnement des coupures (les indices correspondent à l'indexation des sous-tuiles).



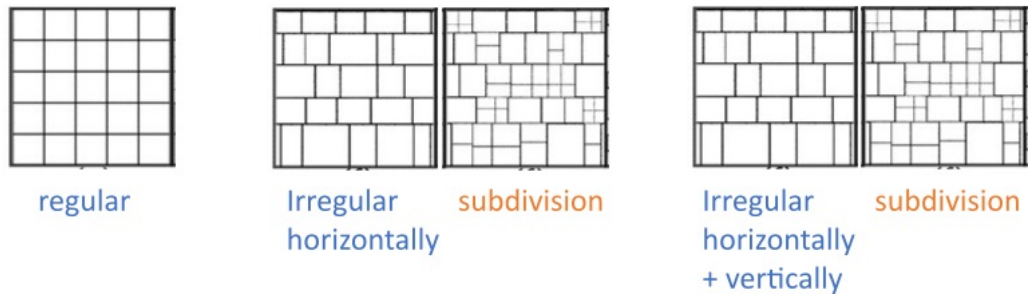


FIGURE 8.12 – Tessellations irrégulières

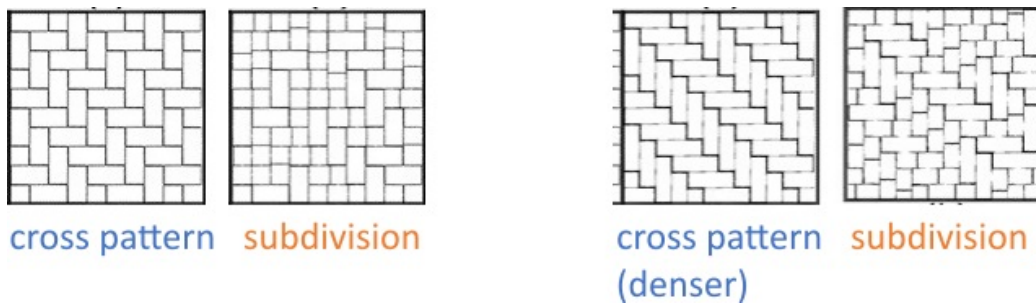


FIGURE 8.13 – Tessellations irrégulières

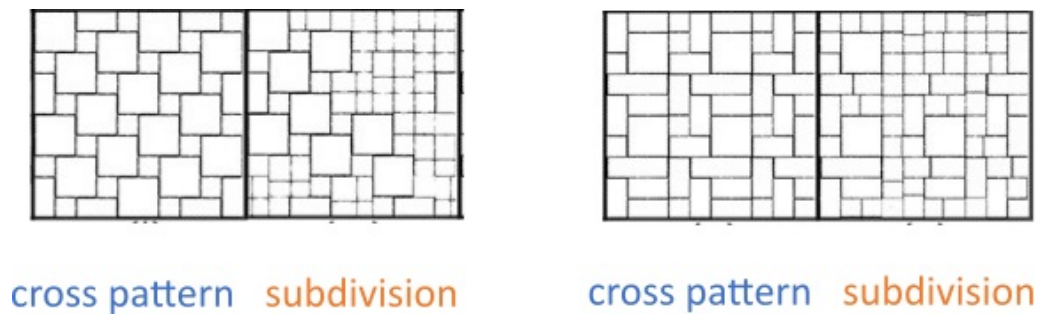


FIGURE 8.14 – Tessellations irrégulières

## 8.2.4 Résumé des paramètres des *processus ponctuels* $P$

Le tableau ci-dessous résume les paramètres liés à notre génération procédurale des processus ponctuels.

Symbole	Description	Valeur
TESSELLATION		
$\nu_T$	type de pavage (e.g. régulier, irrégulier...)	$\mathbb{N}_*$
$p_{subx}$	probabilité de subdivision d'une cellule $R_i$	[ 0, ..., 1 ]
$p_{suby}$	probabilité de subdivision d'une cellule $R_i$	[ 0, ..., 1 ]
$jit_x$	aléas sur la position de la coupure verticale $R_i$	[ 0, ..., 1 ]
$jit_y$	aléas sur la position de la coupure horizontale $R_i$	[ 0, ..., 1 ]
$unit$	unité de mesure ou taille unitaire	$\mathbb{R}$
PROCESSUS PONCTUEL		
$jitter$	jittering (paramètre global de l'aléatoire)	[ 0, ..., 1 ]
$n_{iter}$	nb d'itérations de relaxation (répulsion)	$\mathbb{N}$

**TABLE 8.1** – Liste des paramètres des processus ponctuels.

Mais en pratique, afin de simplifier l'étape d'estimation des paramètres du modèle procédural, nous avons fixé les paramètres  $(p_{subx}, p_{suby})$ ,  $(jit_x, jit_y)$  et  $n_{iter}$  pour créer des collections pré-définies de pavages, un pavage étant alors identifié par un numéro particulier  $\nu_T$ . Ce nombre peut être étendu aisément, sachant qu'une multitude d'autres pavages en cellules rectangulaires sont imaginables, en plus de ceux qu'on a développés ici. Toutefois, nos expérimentations nous ont permis d'obtenir un modèle suffisamment expressif et contrôlable pour reproduire la majorité des structures observées avec pour deux seuls paramètres  $\nu_T$  et  $jitter$ .

## 8.2.5 Déterminisme et générateur de nombres pseudo-aléatoires *PRNG*

Chaque cellule  $R_i$  du plan doit être caractérisée par un identifiant unique. Pour cela, on utilise le centroïde  $x_i$  auquel est associé un nombre pseudo-aléatoire qui va gouverner l'ensemble des propriétés aléatoires de ses paramètres, comme une *graine* (seed). Ceci est particulièrement important pour obtenir un comportement déterministe, à savoir permettre une reproductibilité des résultats. En pratique, on utilise une fonction de hashage sur les coordonnées spatiales du centroïde  $x_i$  des régions  $R_i$ .

Pour le générateur de nombres pseudo-aléatoires, plusieurs possibilités existent. On peut par exemple utiliser le générateur congruentiel linéaire utilisé par Lagae *et al.* dans [LLDD09], *Procedural noise using sparse Gabor convolution* :

$$y_{n+1} = (a y_n + c) \bmod m$$

où  $a$  est le multiplicateur,  $c$  l'incrément et  $m$  le module.

$$x_n = \frac{y_n}{m} \in [0; 1]$$

$$y_{n+1} = (3039177861 y_n) \bmod m$$

Avec la seed (graine) qui va varier pour chaque cellule  $R_i$  :

$$x_0 = (y \% N) * N + (x \% N)$$

## 8.3 Contraintes spatiales et interactions mutuelles entre éléments

On s'intéresse ici à la modélisation du composant *fonction de fenêtrage*  $W$  (*Window Function*) de la *PPTBF*, caractérisé par sa collection de points  $\{\mathbf{x}_i \mid \mathbf{x}_i \in \mathbb{R}^n\}_{i \in \mathbb{N}^*}$ , nommés *features points* :

$$\forall \mathbf{x} \in \mathbb{R}^n, i \in \mathbb{N}^* \quad \left\{ \begin{array}{l} P(\mathbf{x}) = \sum_i \delta(\mathbf{x} - \mathbf{x}_i) \\ PPTBF(\mathbf{x}) = P(\mathbf{x}) * (W(\mathbf{x}) F(\mathbf{x})) \\ PPTBF(\mathbf{x}) = \sum_i W(\mathbf{x} - \mathbf{x}_i) F(\mathbf{x} - \mathbf{x}_i) \end{array} \right.$$

### 8.3.1 Fenêtre générique

La fonction  $w$  définit l'extension spatiale des motifs visuels locaux (features) et comment ils interagissent avec leurs voisins. Cette fenêtre générique est une combinaison linéaire de fenêtres de base, se recouvrant ou non. Centrées en chaque feature point  $\mathbf{x}_i$  de chaque cellule  $R_i$ , tel que :

$$w(\mathbf{x} - \mathbf{x}_i) = \sum_{j=1}^{n_w} \frac{\omega_j}{N_j} w_j(\mathbf{x} - \mathbf{x}_i)$$

où le paramètre  $n_w$  est le nombre de fonctions de base  $w_j(\mathbf{x})$ ,  $\omega_j$  sont les poids des fonctions et  $N_j$  est un coefficient de normalisation.

Initialement, les fonctions avaient un profil gaussien, mais nous avons finalement testé plusieurs profils et choisi de contrôler la vitesse de décroissance ou croissance par une exponentielle, de paramètre  $\sigma_j$  :

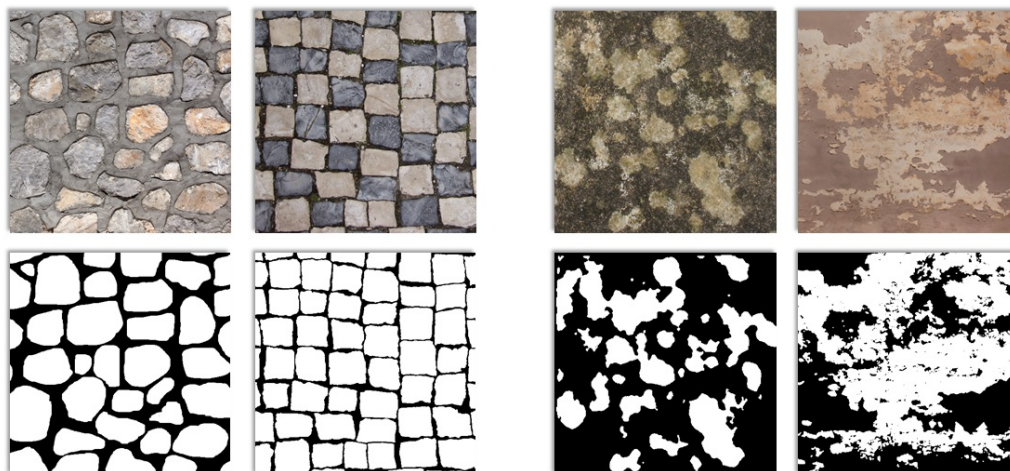
$$w_j(\mathbf{x} - \mathbf{x}_i) = \lfloor e^{-\sigma_j D_j(\mathbf{x} - \mathbf{x}_i)} \rfloor_{d_j}$$

ou, de manière équivalente, avec la fonction rectangulaire (porte,  $\Pi$ ) :

$$w_j(\mathbf{x} - \mathbf{x}_i) = (e^{-\sigma_j D_j(\mathbf{x} - \mathbf{x}_i)}) \Pi \left( \frac{\|\mathbf{x} - \mathbf{x}_i\|_2}{d_j} \right)$$

avec  $D_j$  la distance aux bords de la fenêtre. Ces fonctions distances sont tronquées par la variable  $d_j$ , qui définit leur extension spatiale.

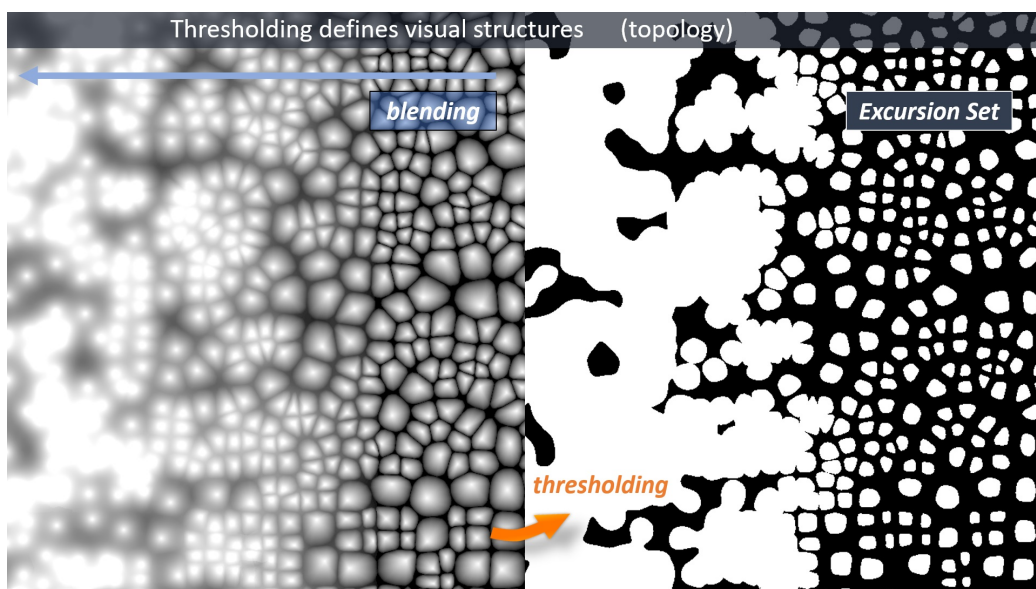
Pour la combinaison linéaire, nous nous sommes limités à deux fonctions : une sans recouvrement (dite cellulaire) et une avec recouvrement (figure 8.15). Ceci pour limiter les calculs et également pour simplifier l'étape de modélisation procédurale inverse au chapitre 10.



**FIGURE 8.15** – Fenêtres avec et sans recouvrement. Différents types de fenêtres dont les éléments se recouvrent (à droite) ou non (à gauche).

Ainsi, les poids  $\omega_j$  des fonctions de base étant normalisés tels que :  $\sum \omega_j = 1$ , on ne conserve qu'un seul paramètre  $\omega$  qui définit le taux de mélange des deux fonctions (figure 8.16).

La fenêtre définit l'extension spatiale des structures générées, et dans ce contexte, le profil de la fenêtre n'a pas une grande influence sur les caractéristiques structurelles. À l'inverse, son empreinte spatiale (forme 2D) s'avèrera être d'une importance primordiale.



**FIGURE 8.16** – Modèle générique de fenêtre. Mélange de fenêtres avec et sans recouvrement. Variation spatiale du paramètre de recouvrement  $\omega \in [0; 1]$  des fenêtres.



### 8.3.2 Fenêtre sans recouvrement

Deux mécanismes sont mis en place pour générer la fonction de fenêtrage de base sans recouvrement : l'un utilise le pavage procédural, l'autre utilise des cellules de Voronoï. En effet, les deux définissent un partitionnement de  $\mathbb{R}^2$  (c'est-à-dire sans chevauchement). Ils représentent des structures cellulaires visuellement attrayantes qui apparaissent par exemple dans certains modèles de craquelures et fissures. La mesure de distance utilisée pour définir les cellules de Voronoï détermine leurs formes géométriques. Pour plus de contrôle, leurs formes sont modifiables et les fenêtrages interpolables (figure 8.17).

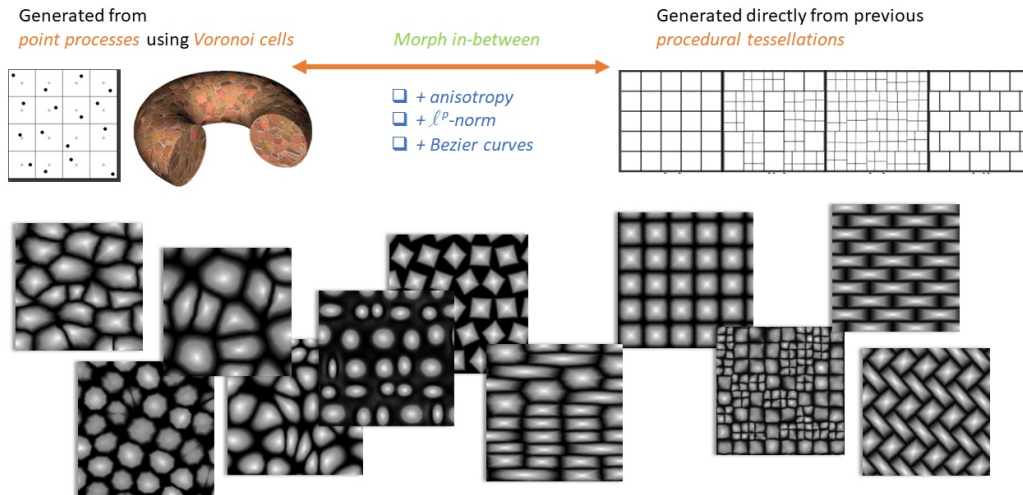


FIGURE 8.17 – Gestion des fenêtres sans recouvrement.

### Fenêtrage à partir du pavage procédural

Cette fenêtre s'exprime par une fonction distance  $D_t$  ( $t$  pour tessellation), pour tout point  $x$  sur une norme infinie :

$$D_t(\mathbf{x} - \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|_\infty$$

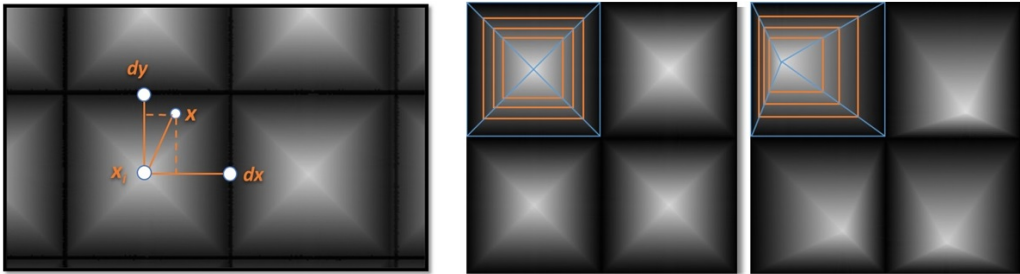
On la normalise par les dimensions  $\Delta\mathbf{R}_i$  de la cellule  $R_i$  :

$$D_t(\mathbf{x} - \mathbf{x}_i) = \frac{\|\mathbf{x} - \mathbf{x}_i\|_\infty}{\|\Delta\mathbf{R}_i\|_\infty}$$

$$\text{avec } \Delta\mathbf{R}_i = (|dx|, |dy|)$$

Pour toutes les cellules  $R_i$ , on souhaite également que la fonction prenne une valeur nulle sur les bords et maximale sur le *feature point*  $\mathbf{x}_i$  :

$$D_t(\mathbf{x} - \mathbf{x}_i) = 1 - \frac{\|\mathbf{x} - \mathbf{x}_i\|_\infty}{\|\Delta\mathbf{R}_i\|_\infty}$$



**FIGURE 8.18** – Fenêtres issues des pavages procéduraux. Fonction distance aux bords des fenêtres  $(dx, dy)$ , d'intensité maximale sur le centroïde  $x_i$  et nulle sur les bords. Visualisation des isolignes liées à la norme infinie.

## Fenêtrage à partir de cellules de Voronoï

La cellule de Voronoï d'un point  $\mathbf{x}$  d'un processus ponctuel  $P$  consiste en l'ensemble des points dont la distance à  $\mathbf{x}$  n'est pas plus grande que leurs distances aux autres points de  $P$ , soit :

$$V(\mathbf{x}) = \{ \mathbf{y} \in \mathbb{R}^2 \mid \forall \mathbf{z} \in P \setminus \{ \mathbf{x} \}, \|\mathbf{y} - \mathbf{x}\| \leq \|\mathbf{y} - \mathbf{z}\| \}$$

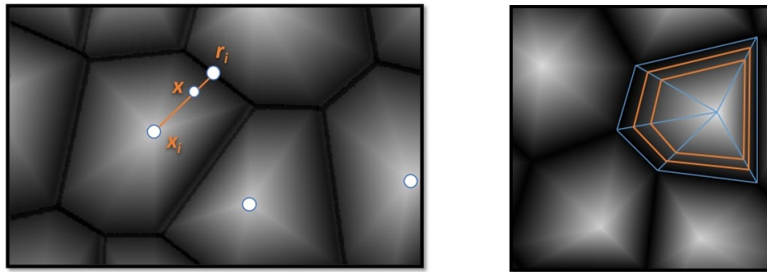
À partir de là, la tessellation de Voronoï, ou diagramme de Voronoï, est la partition ou décomposition de l'espace en cellules de Voronoï du processus ponctuel  $P$ . La distance  $D_v$  ( $v$  pour Voronoï) s'exprime par la norme  $\ell^p$  :

$$D_v(\mathbf{x} - \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|_p$$

On la normalise par la distance au point  $\mathbf{r}_i$  situé sur le bord de la cellule de Voronoï :

$$D_v(\mathbf{x} - \mathbf{x}_i) = \frac{\|\mathbf{x} - \mathbf{x}_i\|_p}{\|\mathbf{x} - \mathbf{r}_i\|_p}$$

Il n'y a pas de formule analytique pour déterminer le point  $\mathbf{r}_i$ , sur le bord de la cellule de Voronoï, avec la norme  $\ell^p$ . Il s'obtient par une recherche itérative le long du vecteur  $\overrightarrow{\|\mathbf{x}_i \mathbf{x}\|}$ .



**FIGURE 8.19** – Fenêtres issues des cellules de Voronoï. Fonction distance aux bords des fenêtres  $\mathbf{r}_i$ , d'intensité maximale sur le centroïde  $\mathbf{x}_i$  et nulle sur les bords. Visualisation des isolignes liées à la norme  $\ell^p$ .

En inversant la formule pour avoir des valeurs nulles aux bords, on a :

$$D_v(\mathbf{x} - \mathbf{x}_i) = 1 - \frac{\|\mathbf{x} - \mathbf{x}_i\|_p}{\|\mathbf{x} - \mathbf{r}_i\|_p}$$

## Interpolation des deux fenêtres

L'interpolation linéaire des deux fonctions distance issues des pavages procéduraux et des cellules de Voronoï permet d'étendre la gamme de formes des fenêtres, avec un paramètre  $\lambda$ . La distance résultante  $D_1$  est :

$$D_1(\mathbf{x} - \mathbf{x}_i) = (1 - \lambda) D_v(\mathbf{x} - \mathbf{x}_i) + \lambda D_t(\mathbf{x} - \mathbf{x}_i)$$

Ainsi :

$$D_1(\mathbf{x} - \mathbf{x}_i) = 1 - \left( (1 - \lambda) \frac{\|\mathbf{x} - \mathbf{x}_i\|_p}{\|\mathbf{x} - \mathbf{r}_i\|_p} + \lambda \frac{\|\mathbf{x} - \mathbf{x}_i\|_\infty}{\|\Delta \mathbf{R}_i\|_\infty} \right)$$

## Décroissance du profil des fenêtres

L'utilisation d'une fonction puissance (de paramètre  $d_1$ ) permet de contrôler la vitesse de décroissance (ou croissance) de la fonction window  $W_c$ , ce qui permet de rétrécir (ou d'élargir) le support spatial de la fenêtre :

$$w_1(\mathbf{x} - \mathbf{x}_i) = (D_1(\mathbf{x} - \mathbf{x}_i))^{d_1}$$

Nous préférons utiliser une exponentielle afin de garder un cadre commun avec l'ensemble de nos fenêtres, notamment celles avec recouvrement (voir suite). Une fois normalisée, on obtient :

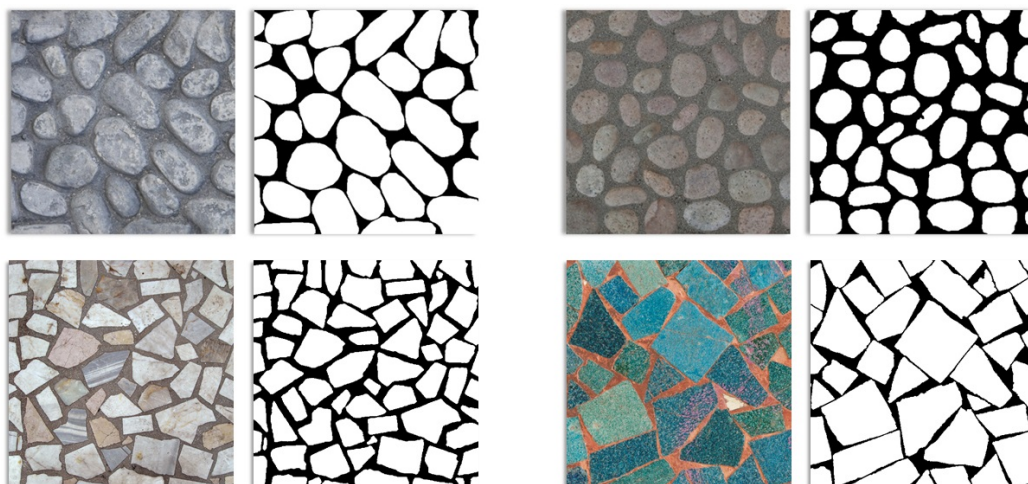
$$w_1(\mathbf{x} - \mathbf{x}_i) = \frac{e^{-\sigma_1(1-D_1(\mathbf{x}-\mathbf{x}_i))} - e^{-\sigma_1 d_1}}{1 - e^{-\sigma_1 d_1}}$$

où le paramètre  $\sigma_1$  sert à accélérer la décroissance du profil, et le  $d_1$  à élargir le support spatial du profil.

## Gestion de l'apparence et de la densité surfacique des fenêtres

La fenêtre sans recouvrement définie précédemment ne permet pas de couvrir correctement la gamme de structures que nous avons observée. La raison est qu'elle produit des structures essentiellement polygonales, que l'on ne retrouve quasiment que dans le cas des mosaïques, alors que la majorité des structures observées a plutôt des formes arrondies.

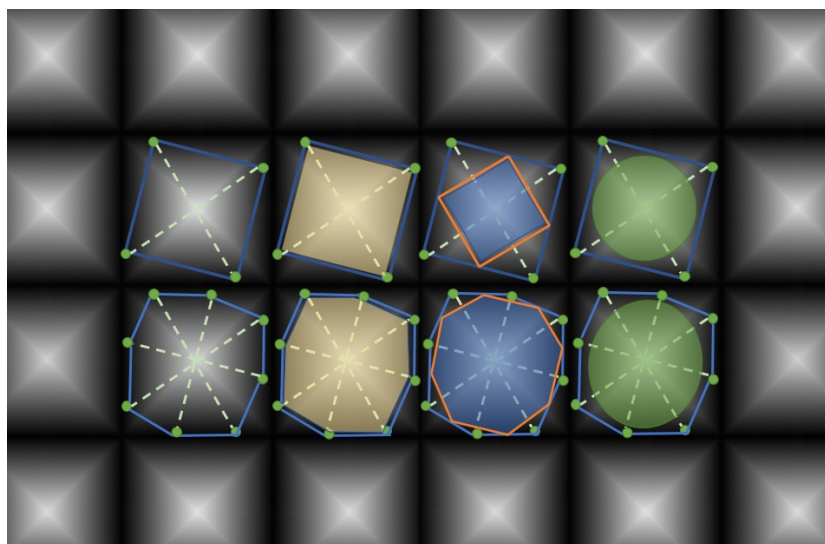
Afin d'étendre la gamme des formes engendrées par la fenêtre, on ajoute la possibilité de contrôler leur apparence qui peut varier de plus ou moins arrondi, à polygonale, donc taillée comme des mosaïques, du carrelage cassé, et créer des espacements entre les éléments. La figure 8.20 est un extrait de notre base de structures observées et présente un échantillon du type de structures cellulaires que l'on souhaite pouvoir modéliser à l'aide de la fonction de fenêtrage.



**FIGURE 8.20** – Fenêtres cellulaires arrondies ou à bords taillés. Différents types de fenêtres de formes et réduction de la densité.

Pour cela, on se donne la possibilité d'échantillonner le contour des cellules du pavage procédural, et déterminer des points de contrôle servant à générer de nouvelles formes à l'intérieur de l'enveloppe convexe associée : soit la forme géométrique tirée de sa courbe de Bézier, soit la région contenue dans l'enveloppe convexe (figure 8.21). Pour chaque cellule, à partir d'un nombre de points de contrôle  $n_v$  donné, on la découpe en secteurs angulaires uni-

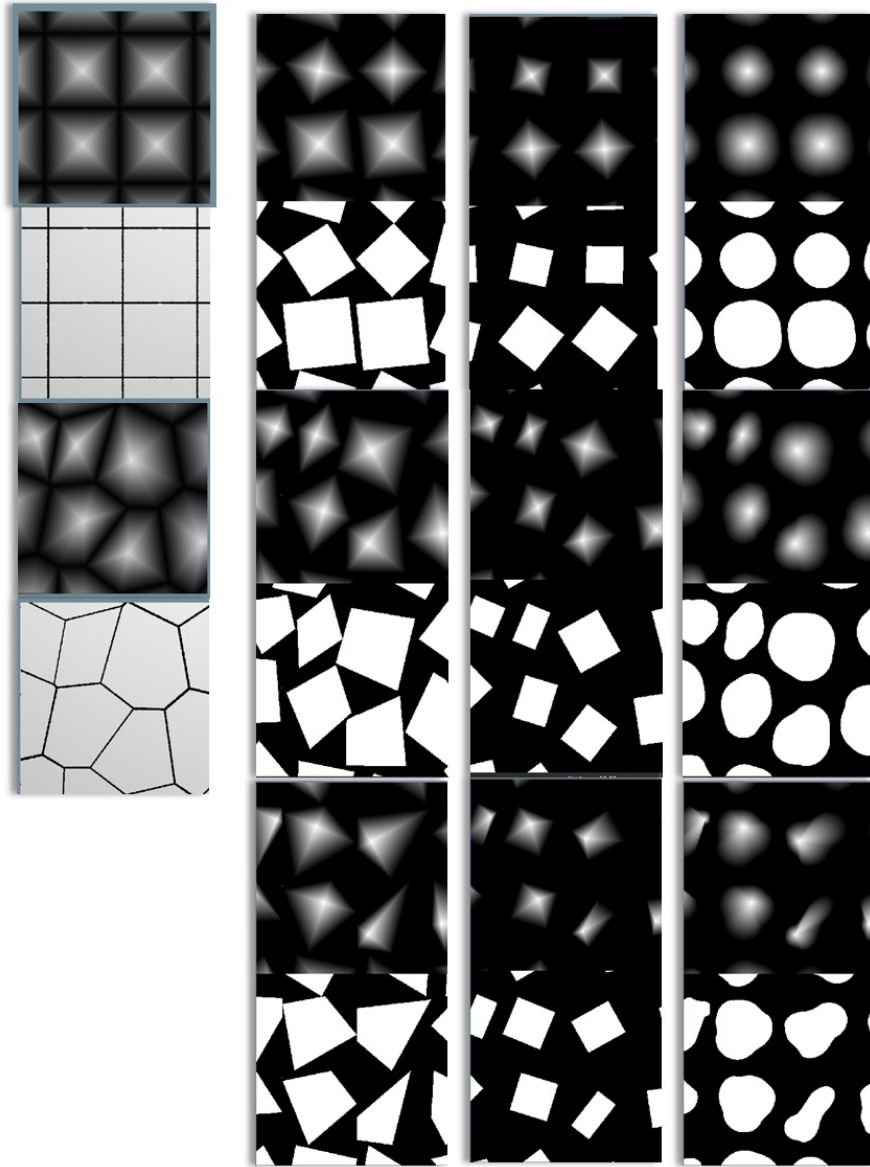
formes d'angle  $\alpha = 2\pi/2^{n_v}$ , en leur appliquant une rotation initiale aléatoire de déphasage tiré dans  $[0, \alpha]$ . Chacun des secteurs angulaires va générer un point de contrôle sur les bords de la cellule originale, déterminant l'enveloppe convexe d'une première région, ainsi que la courbe de Bézier associée (contrainte à l'intérieur). Enfin, un deuxième région est créée à partir des milieux de chaque segment de la première enveloppe convexe. La surface de cette région est plus petite et permet d'augmenter la densité des espacements (c.-à-d. créer du vide). Un paramètre scalaire continue permet de passer par chacune de ces 3 formes, interpoler linéairement entre chacun d'elles, afin de choisir un taux de bords adoucis ou durs (un peu comme les changements d'apparence des boules ou cercles unités en normes 1, 2 et infinie). Selon la valeur de la rotation aléatoire initiale pour chaque cellule, des motifs variés sont créés.



**FIGURE 8.21** – Formes des fenêtres sans recouvrement. En haut, de gauche à droite : (1) 4 points de contrôle, découpage en secteurs angulaires uniforme, et rotation initiale aléatoire ; (2) nouvelle forme en jaune dans l'enveloppe convexe ; (3) comme le (2) mais on prend le milieu de chaque côté de l'enveloppe convexe ; (4) courbe de Bézier créer par l'enveloppe convexe. En bas : même chose, avec 8 points de contrôle. Selon la valeur de la rotation aléatoire initiale pour chaque tuile, des motifs variés vont être créés.

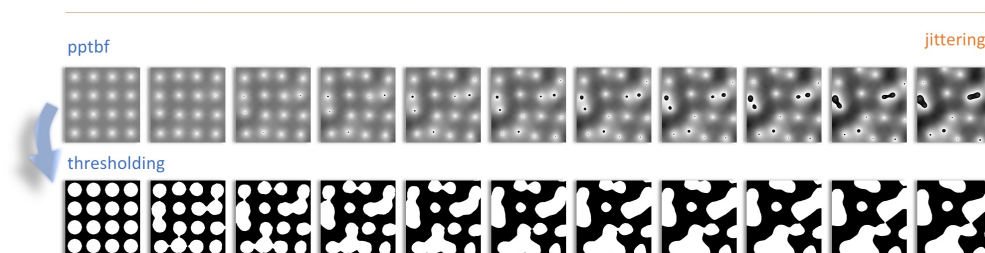
La figure 8.22 illustre la gestion de l'apparence et de la densité surfacique des fenêtres.





**FIGURE 8.22** – Gestion de l'apparence des fenêtres. De gauche à droite : (1) PPTBF standard, (2) régions dans l'enveloppe convexe avec 4 points de contrôle, (3) région dans l'enveloppe convexe issue des milieux des segments de la première enveloppe, (4) région dans la courbe de Bézier associée à la première enveloppe. En haut : fenêtres issues de pavages procéduraux. Au milieu : fenêtres issues de cellules de Voronoï sans jittering. Au milieu : fenêtres issues de cellules de Voronoï avec jittering (pour plus d'anisotropie).

Ci-dessous, un exemple illustrant l'effet du *jittering* sur la topologie des fenêtres : l'anisotropie créée des agrégats (clusters), avec des régions étendues non uniformes (figure 8.23).



**FIGURE 8.23** – Effet du *jittering* sur les fenêtres sans recouvrement. Création de clusters avec des régions étendues non uniformes. NOTE : l'image du haut contient un bug graphique qui se visualise sous forme de trous dans le champ scalaire de la PPTBF (dû à la division par un NAN dans le shader utilisé ici).

### 8.3.3 Fenêtres avec recouvrement

Le support spatial des fenêtres avec recouvrement est très simple. Il se base sur la norme  $\ell^p$  et permet de bénéficier de tout l'éventail des formes associées (figure 8.24) :

$$D_2(\mathbf{x}) = \|\mathbf{x}\|_p$$

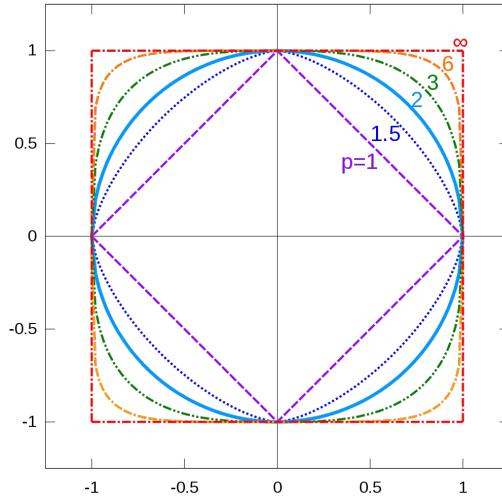
Centrée sur un feature point  $\mathbf{x}_i$  d'une cellule, on a (figure 8.25) :

$$D_2(\mathbf{x} - \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|_p$$

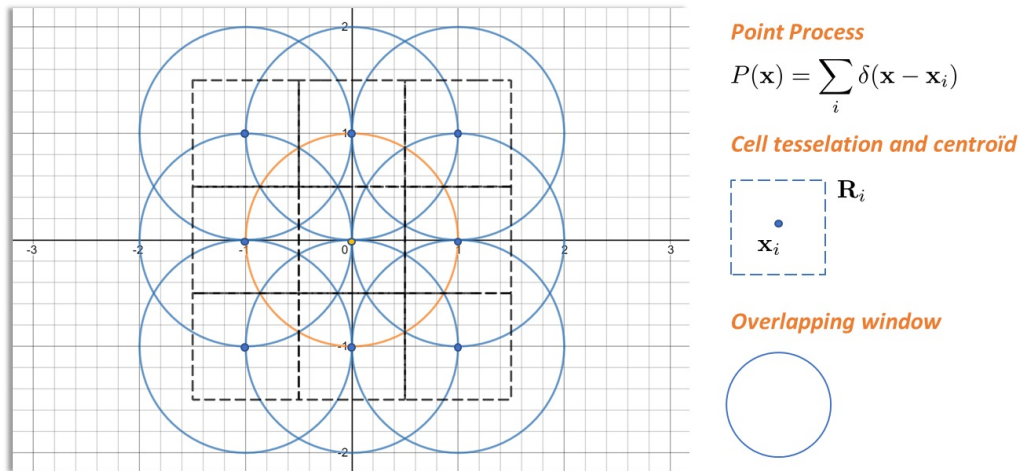
Nous avons testé plusieurs profils classiques de fenêtre issus du domaine du traitement du signal, comme : triangulaire, rectangulaire, gaussienne, tapered cosine, Tukey, Kaiser, etc. Comme pour les fenêtres sans recouvrement, on utilise deux paramètres d'apparence. Le paramètre  $\sigma_2$  sert à accélérer la décroissance du profil, et  $d_2$  sert à élargir le support spatial du profil.

En version normalisée et centrée sur les features points  $\mathbf{x}_i$  des cellules  $R_i$  on a pour la fonction fenêtre exponentielle (décroissante) :

$$w_2(\mathbf{x}) = \frac{e^{-\sigma_2 D_2(\mathbf{x})} - e^{-\sigma_2 d_2}}{1 - e^{-\sigma_2 d_2}}$$



**FIGURE 8.24** – Formes de base des fenêtres sans recouvrement. La norme  $\ell^p$  permet de représenter un large éventail de formes pour leur support spatial (voir [https://fr.wikipedia.org/wiki/Norme\\_\(math%C3%A9matiques\)](https://fr.wikipedia.org/wiki/Norme_(math%C3%A9matiques))).



**FIGURE 8.25** – Modèle de fenêtre avec recouvrement. Fenêtres à support circulaire (en fait, norme  $\ell^p$ ) placées sur chacun des centroides  $x_i$  du processus ponctuel sous-jacent (visualisé avec sa tessellation). Recouvrement et interaction des fenêtres associées au pavage des cellules les plus proches.

$$w_2(\mathbf{x}) * \delta(\mathbf{x} - \mathbf{x}_i) = w_2(\mathbf{x} - \mathbf{x}_i) = \frac{e^{-\sigma_2 D_2(\mathbf{x} - \mathbf{x}_i)} - e^{-\sigma_2 d_2}}{1 - e^{-\sigma_2 d_2}}$$

pour la fonction triangulaire :

$$w_2(\mathbf{x}) = 1 - \frac{D_2(\mathbf{x})}{d_2}$$

$$w_2(\mathbf{x}) * \delta(\mathbf{x} - \mathbf{x}_i) = w_2(\mathbf{x} - \mathbf{x}_i) = 1 - \frac{D_2(\mathbf{x} - \mathbf{x}_i)}{d_2}$$

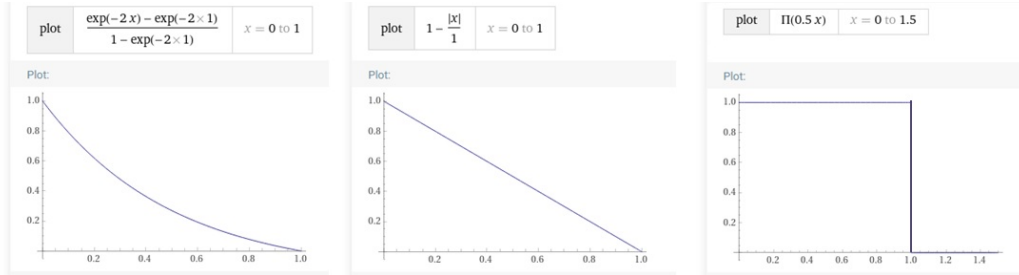
et pour la fonction rectangulaire :

$$w_2(\mathbf{x}) = \Pi\left(\frac{D_2(\mathbf{x})}{2d_2}\right)$$

$$w_2(\mathbf{x}) * \delta(\mathbf{x} - \mathbf{x}_i) = w_2(\mathbf{x} - \mathbf{x}_i) = \Pi\left(\frac{D_2(\mathbf{x} - \mathbf{x}_i)}{2d_2}\right)$$

Les autres type de profils n'ayant pas apporté plus de variété nous les avons abandonnés.

En pratique, afin de simplifier l'étape d'estimation des paramètres du modèle procédural dans le chapitre 10, nous aussi avons fixé les deux paramètres  $\sigma_2 = 2$  et  $d_2 = 1$ .



**FIGURE 8.26** – Profils des fenêtres avec recouvrement. Fenêtres : exponentielle (décroissante), triangulaire et rectangulaire. Courbes générées via le site WolframAlpha (<https://www.wolframalpha.com/>).

### 8.3.4 Synthèse des paramètres de la fonction *window* $W$

Le tableau 8.2 récapitule les paramètres de la fonction *window*  $W$ .

Symbole	Description	Valeur
FONCTION WINDOW		
$\omega$	combinaison linéaire des deux fenêtres de base	$[0, \dots, 1]$
$\ \cdot\ _c$	norme des fenêtres de base cellulaire	$[1, 2, \dots, \infty]$
$\lambda$	anisotropie des fenêtres de base cellulaire	$[0, 1]$
$n_v$	nombre de points de contrôle pour le lissage	$[3, \dots, 64]$
$l_c$	degré de lissage	$[0, \dots, 2]$
$\sigma_j$	sigma (largeur) de la fenêtre	float

**TABLE 8.2** – Liste des paramètres des fonctions *window*  $W$ .

## 8.4 Formes visuelles locales des éléments

On s'intéresse ici à la modélisation du composant *Feature Function*  $F$  de la *PPTBF* :

$$\forall \mathbf{x} \in \mathbb{R}^n, i \in \mathbb{N}^* \quad \left\{ \begin{array}{l} P(\mathbf{x}) = \sum_i \delta(\mathbf{x} - \mathbf{x}_i) \\ PPTBF(\mathbf{x}) = P(\mathbf{x}) * (W(\mathbf{x}) F(\mathbf{x})) \\ PPTBF(\mathbf{x}) = \sum_i W(\mathbf{x} - \mathbf{x}_i) F(\mathbf{x} - \mathbf{x}_i) \end{array} \right.$$

### 8.4.1 Objectifs

La fonction  $F$  modélise les caractéristiques visuelles des éléments (*feature*), c'est-à-dire les motifs locaux, comme des rayures, grains, points, tâches et tâches étendues. Pour la synthèse de textures, ce terme est généralement de première importance, car il influence fortement les caractéristiques visuelles structurelles des textures. Ici, l'objectif est de produire la structure globale de la texture sans aller jusqu'à des détails de formes géométriques précises. Une modélisation précise de composants hautes fréquences ou avec des formes spécifiques, telles que des fleurs de type pâquerettes, feuilles d'érables, etc., n'est pas requise, car elle peut être traitée ultérieurement lors de la phase de synthèse de détails colorés.

La fonction  $F$  a été construite de manière incrémentale, en incorporant les fonctionnalités une à une. Dans la suite, on présente sa construction pas à pas.

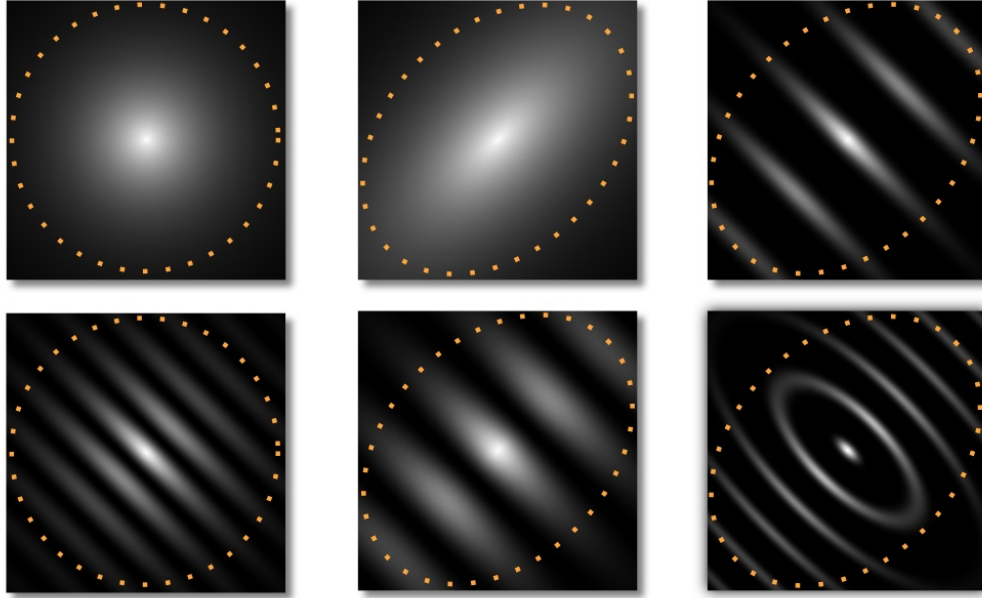
### 8.4.2 Objets de types filiformes ou constants

La fonction  $F$  est définie par une *mixture*<sup>1</sup> de noyaux de Gabor anisotropes, potentiellement filiformes, placés à des positions aléatoires à l'intérieur des cellules du pavage procédural (figure 8.27). On utilise des noyaux de Gabor parce qu'ils unifient les propriétés spatiales et fréquentielles.

---

1. Nous utilisons volontairement le terme anglais ici, qui peut se traduire en français par *mélange*, pour éviter toute confusion avec les autres types de mélanges effectués dans notre modèle.



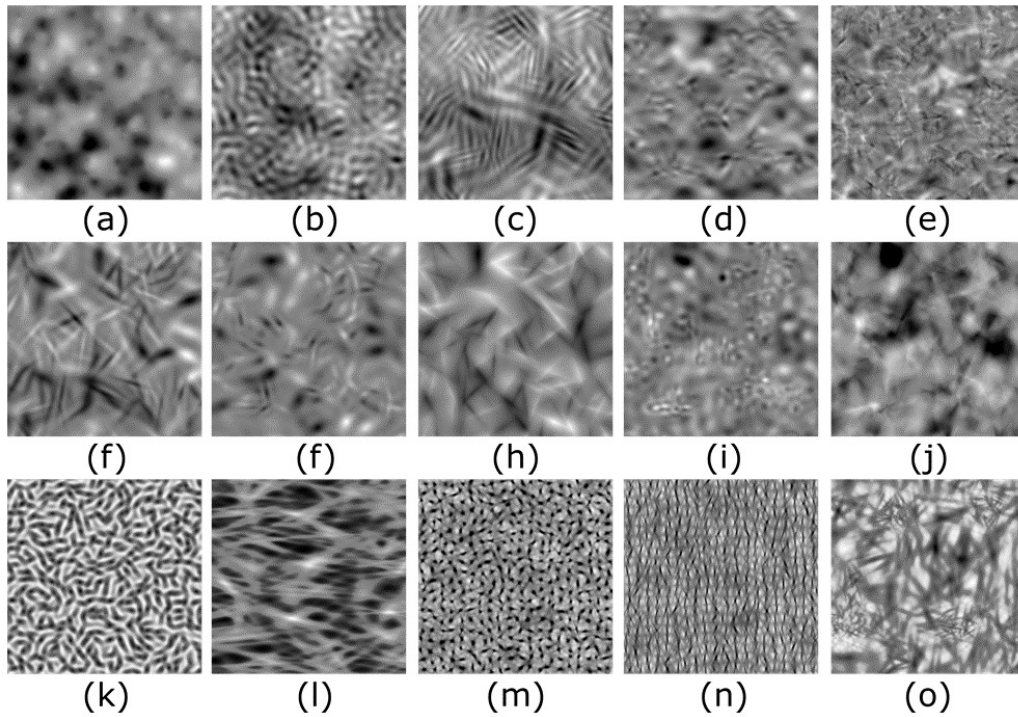


**FIGURE 8.27** – Noyaux de Gabor modifiés. À gauche : Fonction de Gabor isotrope (en haut est la fréquence 0). Au milieu : Fonction de Gabor anisotrope. À droite : augmentation de l'épaisseur  $\tau$  et courbure  $\kappa$ .

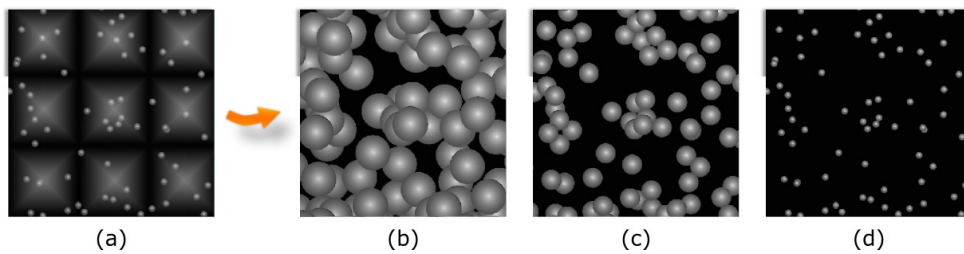
L'ajout de la gestion d'une forme d'anisotropie, d'épaisseur et de courbures des bandes, permet une modélisation plus efficace de caractéristiques anisotropes comme des fibres, des rayures et des plis. De même, notre gestion de mixture de noyaux et d'une meilleure prise en compte des corrélations entre les noyaux, étend la gamme des structures représentables par des fonctions de type bruits par convolution éparse avec des noyaux de Gabor (figure 8.28). Notre fonction de Gabor modifiée  $f(\mathbf{x})$  est composée, comme pour une fonction de Gabor classique, d'une onde  $\tilde{f}(\mathbf{x})$  modulée par une gaussienne  $\tilde{g}(\mathbf{x})$ . NOTE : nous avons choisi d'utiliser un paramètre  $\sigma$  qui évolue en réalité à l'inverse de la variance des gaussiennes traditionnelles. Pour la gaussienne, le paramètre  $\sigma$  est divisé par l'empreinte (footprint) de la cellule (figure 8.30) :

$$\tilde{g}_k(\mathbf{x}) = e^{-\frac{\sigma_k}{\text{footprint}_{R_i}} \|\mathbf{x}\|}$$

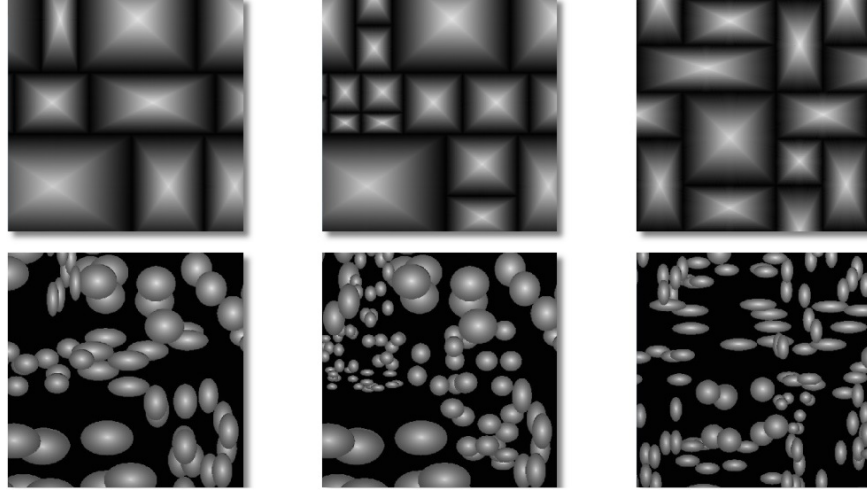
Pour ajouter de la variété et de l'aléatoire, chaque noyau de Gabor  $f_k$  dispose de son propre paramètre  $\sigma_k$  (lié à l'inverse de la variance) :



**FIGURE 8.28** – Différents types de features. L’anisotropie, l’épaisseur et de courbures des bandes permet une modélisation plus efficace de caractéristiques anisotropes comme des grains, des rayures et des plis. Notre modèle étend les fonctions de bruits.



**FIGURE 8.29** – Décroissance uniforme des fenêtres. En (a), visualisation des cellules  $R_i$  d’un pavage et leurs noyaux  $f_k$ . De (b) à (d), Décroissance uniforme du support spatial des noyaux  $f_k$ .



**FIGURE 8.30** – Supports spatiaux proportionnels aux cellules  $R_i$ . En haut, visualisation des cellules  $R_i$  de différentes tessellations. En bas, les supports spatiaux des noyaux  $f_k$  sont proportionnels aux cellules  $R_i$ .

$$\tilde{g}_k(\mathbf{x}) = e^{-\sigma_k \|\mathbf{x}\|}$$

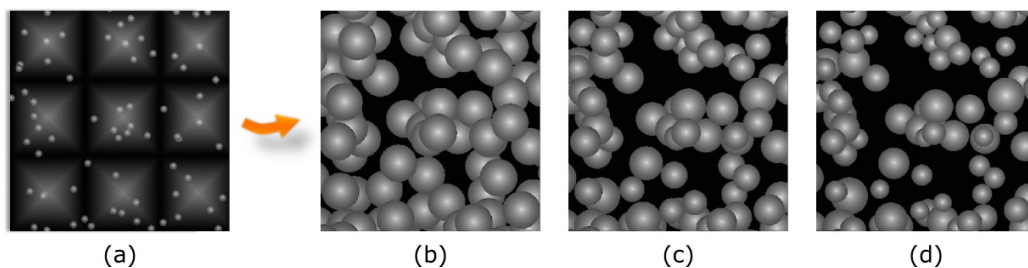
où les  $\sigma_{k|i}$  sont tirés au hasard proportionnellement à la taille de la cellule correspondante  $R_i$ , telle que :

$$\sigma_k = \sigma (1 + V_{\sigma_k} \Delta_\sigma) \quad \text{où} \quad V_{\sigma_k} \in [0; 1] \text{ suit une loi de probabilité } PDF_{\sigma}.$$

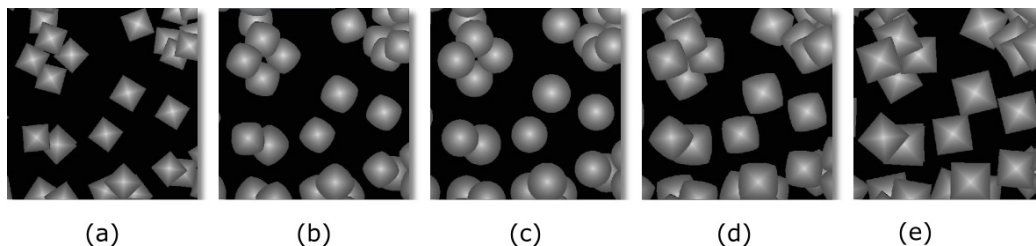
La valeur de  $\Delta_\sigma$  permet d'introduire de l'aléatoire dans l'extension spatiale des noyaux  $f_k$  (figure 8.31).

On souhaite pouvoir contrôler la forme du support spatial des noyaux de Gabor (figure 8.32). Pour cela, on utilise un paramètre  $p_F$  afin de modifier la norme  $p$  ( $l^p$ -norm) utilisée par l'exponentielle de la fonction de Gabor :  $\tilde{g}_k(\mathbf{x}) = e^{-\sigma_k \|\mathbf{x}\|_{p_F}}$

En ce qui concerne la composante fréquentielle du noyau de Gabor, le nombre des oscillations est contrôlé par un paramètre  $\phi$ . Il représente la



**FIGURE 8.31** – Décroissances non-uniformes aléatoires des fenêtres. En (a), visualisation des cellules  $R_i$  d'un pavage et leurs noyaux  $f_k$ . De (b) à (d), Décroissances non-uniformes aléatoires des supports spatiaux des noyaux  $f_k$ .



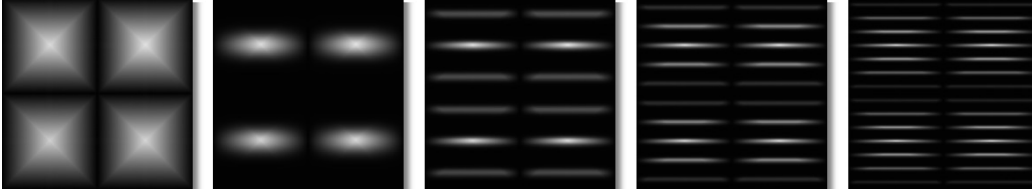
**FIGURE 8.32** – Forme du support spatial des noyaux. Variations de la forme des supports spatiaux des noyaux  $f_k$  en fonction de la norme : de  $\ell^1$  (a), en passant par  $\ell^2$  (c) jusque  $\ell^\infty$  (e).

fréquence unique du cosinus et correspond à une onde monochromatique. Lorsque  $\phi$  augmente, l'épaisseur des bandes diminue (figure 8.33).

Les ondes de nos fonctions de Gabor, normalement dans  $[-1, 1]$ , sont décalées pour obtenir uniquement des valeurs positives dans  $[0, 1]$ . Ceci permet de contrôler plus finement la combinaison par mixture des noyaux et la génération de structures plus complexes. De même, pour contrôler les phases, nous avons choisi de les rendre toutes nulles et leur contrôle est déporté dans la gestion de la position des noyaux sous forme de processus ponctuels secondaires et de combinaison de noyaux plus complexes. L'onde ne comporte qu'une seule fréquence  $\phi$ . On a alors :

$$\forall k \quad \phi_k = \phi \quad \text{avec} \quad \phi \in \mathbb{R}^+$$

$$\tilde{f}_k(\mathbf{x}) = 0.5 + 0.5 \cos(2\pi \phi \|\mathbf{x}\|)$$



**FIGURE 8.33** – Fréquence des bandes des noyaux  $f_k$ . Variation de la fréquence  $\phi$  des noyaux  $f_k$  pour une tessellation donnée. De gauche à droite,  $\phi = 0, 1, 3, 5, 7$ .

On introduit un paramètre d'anisotropie  $\eta$  permettant d'étirer spatialement les noyaux le long de l'axe  $Oy$  par rapport au système de coordonnées local centré sur le noyau d'indice  $k$ , tel que :

$$\forall x \in \mathbb{R} \quad f_\eta : x \mapsto \frac{x}{2^\eta} \quad \text{avec} \quad \eta \in \mathbb{R}^+$$

Pour cela, on utilise une matrice de mise à l'échelle non uniforme  $\mathcal{S}(\eta)$  :

$$\mathcal{S}(\eta) = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2^\eta} \end{pmatrix}$$

On a alors :

$$\tilde{g}_k(\mathbf{x}) = e^{-\sigma_k \|\mathcal{S}(\eta) \mathbf{x}\|_p}$$

$$\tilde{f}_k(\mathbf{x}) = 0.5 + 0.5 \cos(2\pi \phi \|\mathcal{S}(\eta) \mathbf{x}\|)$$

Afin d'orienter les bandes selon un angle  $\theta$ , on utilise une matrice de rotation  $\mathcal{R}(\theta)$ . Et pour augmenter le caractère stochastique, la diversité et complexifier l'apparence, on introduit des variations aléatoires pour chaque noyau (figure 8.34) par l'intermédiaire d'une variable aléatoire v.a.  $\theta_k$  telle que :

$$\theta_k = \theta V_{\theta_k} \quad \text{où} \quad V_{\theta_k} \in [-1; 1] \text{ suit une loi de probabilité } PDF_\theta.$$

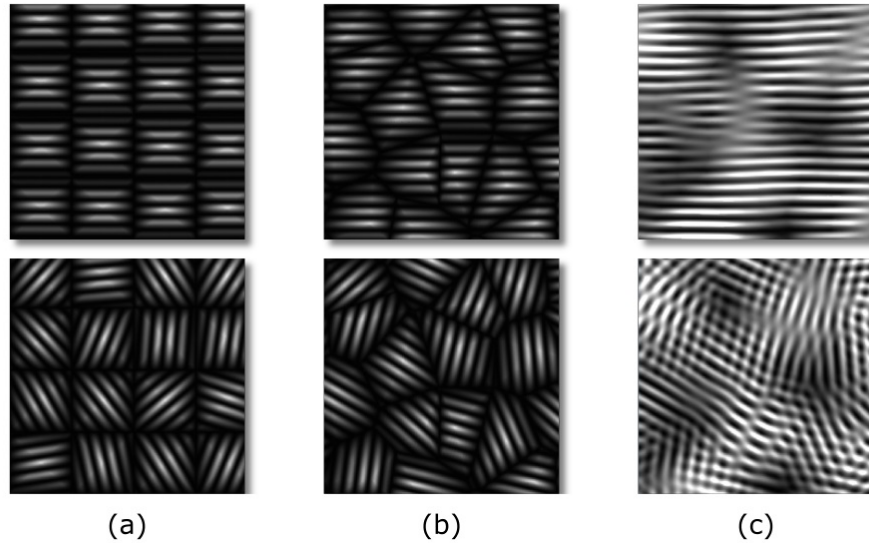
La matrice de rotation associée  $\mathcal{R}(\theta_k)$  transforme les positions spatiales :

$$T_{\mathcal{R}_{\theta_k}}(\mathbf{x}) = \mathcal{R}(\theta_k) \mathbf{x}$$

$$\mathcal{R}(\theta_k) = \begin{pmatrix} \cos(\theta_k) & -\sin(\theta_k) \\ \sin(\theta_k) & \cos(\theta_k) \end{pmatrix} \quad \text{avec } \theta_k \in [-\theta, \theta]$$

On a alors :

$$\begin{aligned} \tilde{g}_k(\mathbf{x}) &= e^{-\sigma_k \|\mathcal{R}(\theta_k) \mathcal{S}(\eta) \mathbf{x}\|_p} \\ \tilde{f}_k(\mathbf{x}) &= 0.5 + 0.5 \cos(2\pi \phi \|\mathcal{R}(\theta_k) \mathcal{S}(\eta) \mathbf{x}\|) \end{aligned}$$



**FIGURE 8.34** – Orientations aléatoires des noyaux  $f_k$ . En haut, noyaux  $f_k$  avec une orientation nulle pour différentes PPTBF (tessellation régulière, cellulaire [voronoï], fenêtres avec recouvrement), avec 1 seul noyau de fréquence  $\phi = 5$  par cellule  $R_i$ . En bas, mêmes PPTBF avec cette fois des orientations aléatoires telles que  $\theta = \frac{\pi}{2}$ .

Afin de pouvoir contrôler l'épaisseur des bandes, on ajoute un paramètre  $\tau$  sous la forme d'une fonction puissance de telle manière qu'une diminution

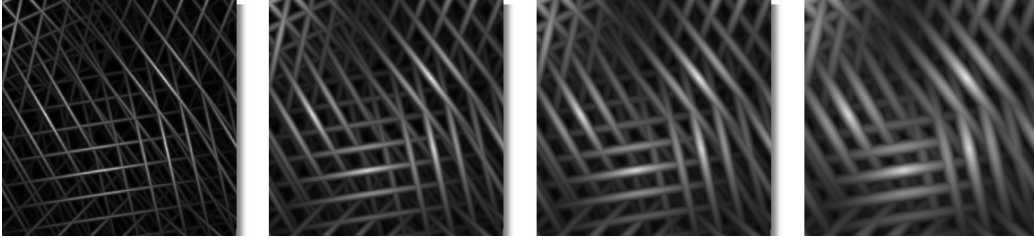


de  $\tau$  amincisse les bandes et une augmentation les élargisse (figure 8.35). Soit :

$$f_\tau : x \mapsto x^{1/\tau} \quad \text{avec} \quad \tau \in \mathbb{R}_+^*$$

On a alors :

$$\tilde{f}_k(\mathbf{x}) = \left( 0.5 + 0.5 \cos(2\pi\phi \|\mathcal{R}(\theta_k)\mathcal{S}(\eta)\mathbf{x}\|) \right)^{1/\tau}$$



**FIGURE 8.35** – Épaisseur des bandes des noyaux  $f_k$ . De gauche à droite : variation de l'épaisseur  $\tau = \{0.05; 0.25; 0.5; 1\}$  des bandes des noyaux  $f_k$  (avec une tessellation régulière, un seul noyau  $f_k$  par cellule  $R_i$  et une fréquence de  $\phi = 5$ ).

On introduit un paramètre  $\kappa$  qui contrôle la courbure des bandes : lorsque  $\kappa = 0$  les bandes sont rectilinéaires, quand  $\kappa = 1$  les bandes forment des cercles concentriques autour des barycentres des noyaux  $f_{k|i}$  (figure 8.36).

Pour cela, on utilise une matrice de mise à l'échelle non uniforme  $\mathcal{S}(\kappa)$  :

$$\mathcal{S}(\kappa) = \begin{pmatrix} \kappa & 0 \\ 0 & 1 \end{pmatrix}$$

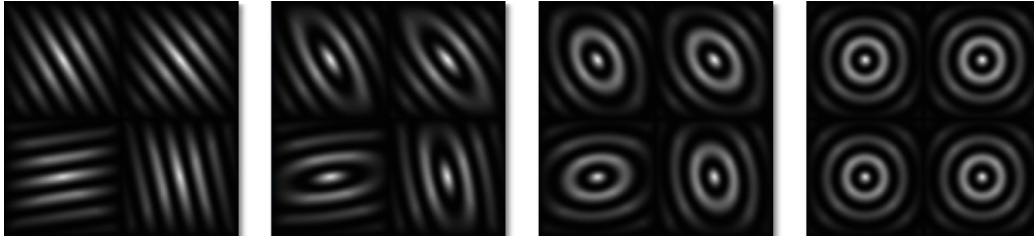
La distance  $\|\mathcal{S}(\kappa)\mathbf{x}\|_2 = \sqrt{(\kappa x)^2 + y^2}$  avec  $\mathbf{x} = (x, y)$ .

On a alors :

$$\forall k \quad \kappa_k = \kappa \quad \text{avec} \quad \kappa \in [0; 1]$$

$$\tilde{f}_k(\mathbf{x}) = \left( 0.5 + 0.5 \cos(2\pi\phi \|\mathcal{S}(\kappa)\mathcal{R}(\theta_k)\mathcal{S}(\eta)\mathbf{x}\|) \right)^{1/\tau}$$

Au final, le noyau peut-être vu comme un noyau de Gabor anisotrope et filiforme (*stringed Gabor function*), tel que :



**FIGURE 8.36** – Courbure des bandes des noyaux  $f_k$ . De gauche à droite : variation de la courbure  $\kappa = \{0; 0.33; 0.66; 1\}$  des bandes des noyaux  $f_k$  (avec une tessellation régulière, un seul noyau  $f_k$  par cellule  $R_i$  et une fréquence de  $\phi = 6$ ).

$$\mathbb{R}^2 \quad \longrightarrow \quad [0; 1]$$

$$f_k : \mathbf{x} \longmapsto e^{-\sigma_k \|\mathcal{R}(\theta_k) \mathcal{S}(\eta) \mathbf{x}\|_{p_F}} \left( 0.5 + 0.5 \cos(2 \pi \phi \|\mathcal{S}(\kappa) \mathcal{R}(\theta_k) \mathcal{S}(\eta) \mathbf{x}\|) \right)^{1/\tau}$$

avec les variables aléatoires v.a. orientation  $\theta_k$  et  $\sigma_k$  (lié à l'inverse de la variance), qui suivent des lois de probabilités  $\mathbb{P}_{\theta_k}$  et  $\mathbb{P}_{\sigma_k}$  telles que :

$$\theta_k = \theta V_{\theta_k} \quad \text{où } V_{\theta_k} \in [-1; 1] \quad \text{et } \theta_k \in [-\theta, \theta]$$

$$\sigma_k = \frac{\sigma (1 + V_{\sigma_k} \Delta_\sigma)}{\Delta R_i} \quad \text{où } V_{\sigma_k} \in [0; 1] \quad \text{et } \sigma_k \in \left[ \frac{\sigma}{\Delta R_i}, \frac{\sigma (1 + \Delta_\sigma)}{\Delta R_i} \right]$$

L'anisotropie (*anisotropy*), épaisseur (*thickness*) et courbure (*curliness*) permettent une modélisation plus efficace de motifs visuels anisotropes comme des rayures, fibres ou plis.

Le tableau 8.3 résume les paramètres des noyaux de Gabor anisotropes filiformes  $f_k$  de la *Feature Function F*.

### 8.4.3 Modèle de *mixture de noyaux*

Les formes visuelles sont modélisées par une *mixture* de noyaux de Gabor anisotropes filiformes, où les positions  $x_k$  des noyaux sont des positions aléatoires sélectionnées à l'intérieur des cellules  $R_i$  pour obtenir des décalages de phases aléatoires, et correspond à un processus ponctuel secondaire.

NOYAU  $f_k$  DE LA FEATURE FUNCTION  $F$

Symbole	Description	Domaine
OSCILLATIONS		
$\phi$	fréquence	$\mathbb{R}^+$
$\tau$	épaisseur	$\mathbb{R}_*^+$
$\kappa$	courbure	$[0; 1]$
SUPPORT SPATIAL		
$\sigma_f$	sigma des noyaux de Gabor (largeur)	$\mathbb{R}^+$
$\Delta_{\sigma_f}$	variation autour de $\sigma_f$	$[0; 1]$
$p_f$	norme $\ell^p$ (forme)	$[1; +\infty[$
TRANSFORMATIONS SPATIALES		
$\theta$	orientation du noyau (rotation)	$[0; \pi/2]$
$\eta$	anisotropie du noyau (étirement)	$\mathbb{R}^+$

TABLE 8.3 – Vue d'ensemble des paramètres de contrôle.

**Nombre aléatoire de noyaux de Gabor**

Le nombre de noyaux  $n_k$  est borné par  $[n_{min}, n_{max}]$ , ces deux paramètres contrôlant la quantité de fonctions de Gabor dans la mixture. Le nombre de noyaux  $n_i$  d'une cellule  $R_i$  est une variable aléatoire donnée par :

$$n_i = n_{min} + (n_{max} - n_{min})V_k \quad \text{où} \quad V_k \in [0; 1]$$

La variable aléatoire permet de faire varier le nombre de noyaux par cellule  $R_i$  comme illustré sur la figure 8.37.

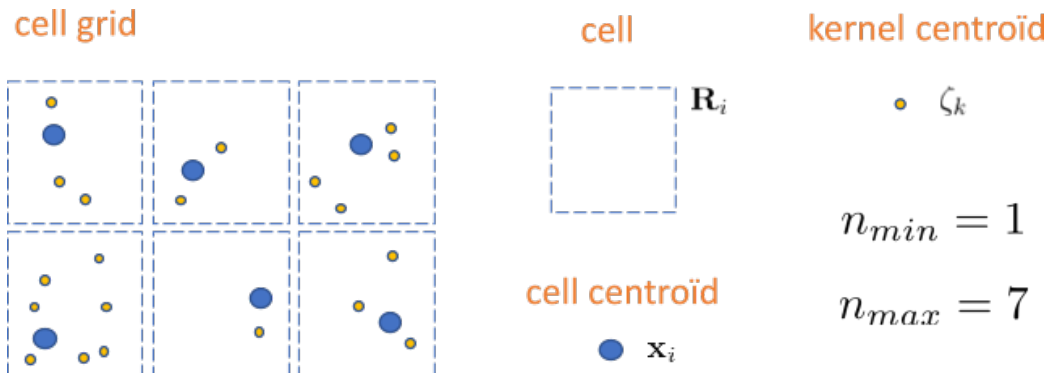
**Mixture des noyaux**

On modélise la mixture d'éléments  $f_k$  par des poids, et une somme pondérée.

$$F(\mathbf{x}) = \sum_k \mu_k(\mathbf{x}) a_k \delta(\mathbf{x} - \mathbf{x}_k) * f(\mathbf{x})$$

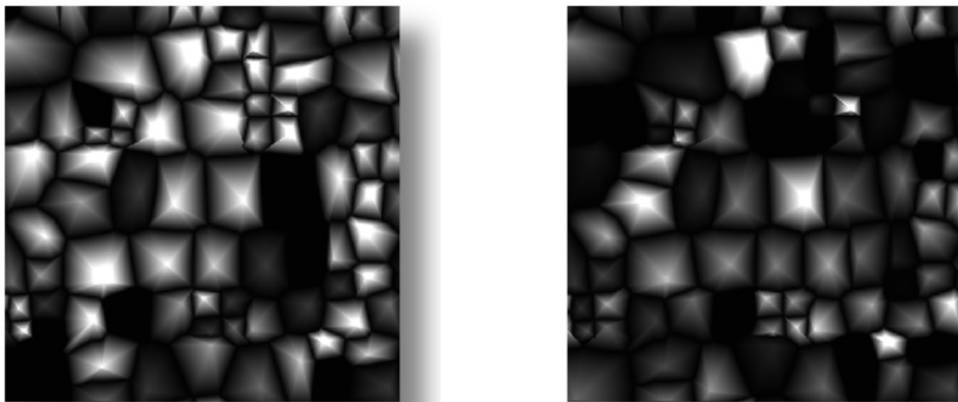
$$F(\mathbf{x}) = \sum_k \mu_k(\mathbf{x}) a_k f(\mathbf{x} - \mathbf{x}_k)$$

La fonction  $\mu_k(\mathbf{x})$  est lié à l'opérateur de mixture souhaité.  $a_k$  permet d'affecter des poids aux noyaux.

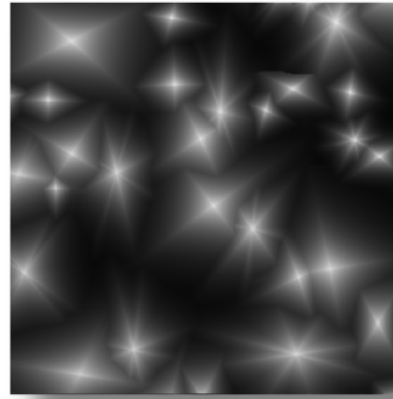
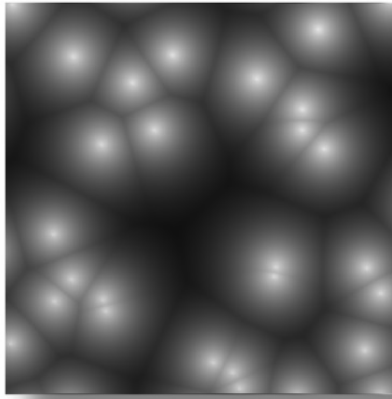


**FIGURE 8.37** – Nombre aléatoire de noyaux. Le nombre aléatoire de noyaux varie par cellule  $R_i$ .

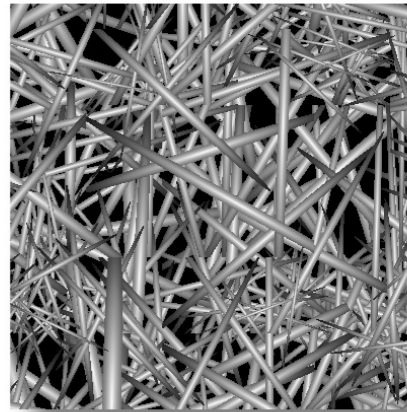
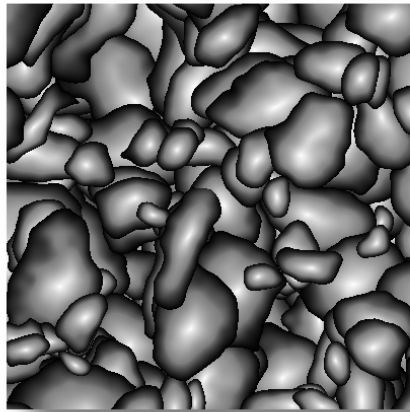
Les différents opérateurs de mixture que l'on a implémentés sont présentés sous une forme visuelle par les figures 8.38 à 8.40. Ils sont tous basés sur l'équation précédente en utilisant des choix particuliers pour  $\mu_k(\mathbf{x})$  et  $a_k$  que l'on ne détaille pas ici, car assez simples à mettre en œuvre.



**FIGURE 8.38** – Somme de noyaux. Modèle de mixture avec opérateur *addition* sur les intensités des noyaux. À gauche, somme avec poids uniforme. À droite, somme avec poids aléatoires générant plus de variations aléatoires comme des pics et des creux.



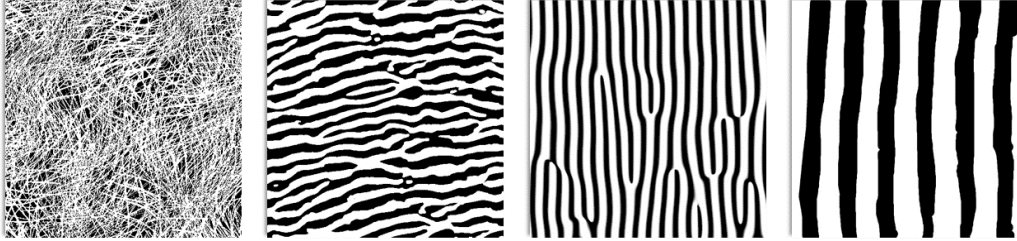
**FIGURE 8.39** – Union de noyaux. Modèle de mixture avec opérateur *max* sur les intensités des noyaux (avec différents nombres de noyaux, types de pavages et normes  $\ell^p$ ).



**FIGURE 8.40** – Empilements de noyaux. Modèle de mixture avec opérateur *max* sur des priorités aléatoires sur les noyaux.

#### 8.4.4 Distributions des noyaux de Gabor

Les positions des noyaux de Gabor peuvent être forcées à se rapprocher du point caractéristique  $x_i$  de la fonction de fenêtrage. Cela introduit une corrélation avec la cellule  $R_i$ .



**FIGURE 8.41** – Corrélations des noyaux. Le seuillage définit les structures visuelles. Différentes topologies se révèlent en fonction du recouvrement des fenêtres sous-jacentes.

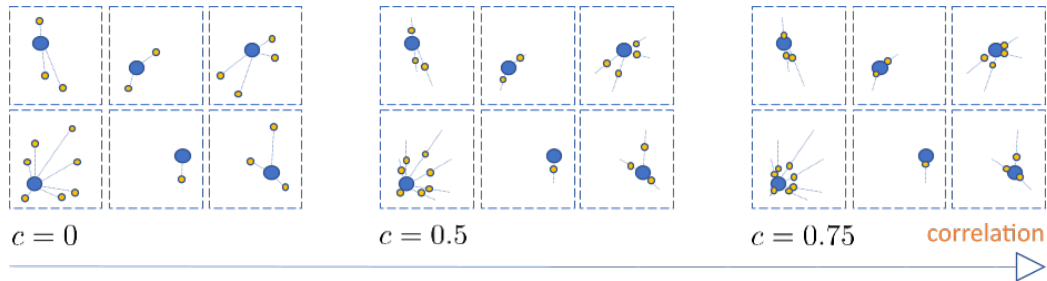
Pour les noyaux  $\mathbf{f}_{\mathbf{k}|i}$ , on tire uniformément des points  $\zeta_{k|i}$  dans le voisinage des centroïdes  $\mathbf{x}_i$  basé sur un coefficient de corrélation  $c \in [0, 1]$ , qui est un paramètre similaire au rayon des disques d'un processus ponctuel de *Matern* [IPSS08] (voir figure 8.42) :

$$\xi_{k|i} = c \mathbf{x}_i + (1 - c) (\mathbf{x}_i + \zeta_{k|i})$$

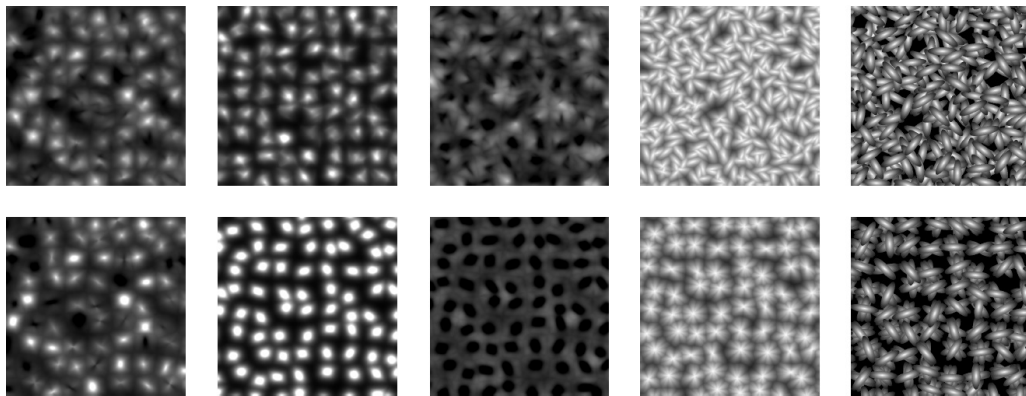
$$\zeta_{k|i} = \mathbf{x}_i + (1 - c) \zeta_{k|i}$$

$\zeta_{\mathbf{k}|i}$  étant une position aléatoire à l'intérieur de la cellule  $R_i$  issue du pavage.





**FIGURE 8.42** – Corrélation des noyaux. Les centroïdes  $\zeta_k$  des noyaux (en orange) sont tirés aléatoirement mais sous contrainte de rester proche des centroïdes  $x_i$  (en bleu) de leur cellule  $R_i$  sous-jacente.



**FIGURE 8.43** – Corrélation des noyaux. Exemples de PPTBF. Chaque colonne montre l'un des cinq modèles de mélange que nous avons mis en œuvre. La première ligne ne montre aucune corrélation avec le centroïde de la fenêtre, le second, une forte corrélation. Étant capable de gérer facilement les corrélations est l'une des propriétés clés de notre modèle. Comme on peut le constater, il a une forte influence sur le résultat visuel.

## 8.5 Variations spatiales et interpolations de structures

On souhaite pouvoir modéliser des variations spatiales de structures. De même que pour les BRDF, on introduit le terme de *SV-PPTBF* pour *Spatially-Varying PPTBF*.

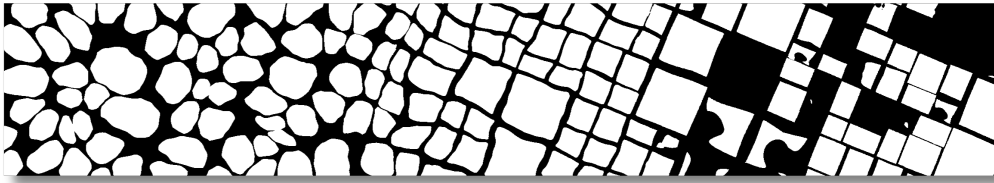


FIGURE 8.44 – Interpolations de PPTBF et structures associées

### 8.5.1 Métamorphose de PPTBFs

L'interpolation de PPTBF et de structures entre 2 points  $A$  et  $B \in \mathbb{R}^2$  :

$$\forall \mathbf{x} \in \mathbb{R}^2 \quad \text{et} \quad \mathbf{x} = \mathbf{x}_A + t(\mathbf{x}_B - \mathbf{x}_A) \quad \text{avec} \quad t \in [0; 1]$$

on interpole linéairement chacun des paramètres de la PPTBF.

Soit  $N^{PPTBF}$  le nombre de paramètres de la PPTBF, et  $\pi_k^{PPTBF}$  le  $k$ -ième de ses paramètres :

$$\forall k \in [1, N^{PPTBF}] \quad \pi_k^{PPTBF}(\mathbf{x}) = (1-m(\mathbf{x}))\pi_k^{PPTBF_A}(\mathbf{x}) + m(\mathbf{x})\pi_k^{PPTBF_B}(\mathbf{x})$$

avec des fonctions d'interpolation linéaires  $m(\mathbf{x})$ .

#### Gestion des transitions douces

Un des avantages majeurs de l'utilisation d'un unique modèle générique de PPTBF est que le morphing de structure des textures est direct. En revanche, lors de l'utilisation des modèles différents, de nouveaux nœuds de « transition » spécifiques doivent être créés manuellement, ce qui est une tâche très laborieuse, car le mélange trivial ne fonctionne généralement pas.

Tous les paramètres scalaires réels sont interpolés linéairement. Il y a cependant trois paramètres entiers,  $\nu_T$ ,  $f_t$  et  $n_i$  qui nécessitent un traitement spécial.

$\nu_T$  correspond au pavage sous-jacent : il sert à définir les features points et leur voisinage  $x_i \in \mathcal{N}_i(x)$ . Ceux-ci sont utilisés pour calculer les distances  $Dj$ . Leur nombre est fixé par la constante  $k$ . Les points étant ordonnés en fonction de la distance euclidienne croissante, nous proposons d'interpoler les valeurs de  $Dj$  des premières distances les plus proches, secondes les plus proches, ..., k-ième plus proches  $Dj$ . En pratique, cela fonctionne très bien, car les champs de distance représentent une solution courante pour le morphing de forme.

$f_t$  est le modèle de mixture. Il détermine comment les amplitudes  $a_k$  et la fonction de mixture  $\mu_k(x)$  sont calculées pour  $f_k(x)$ . Mais ce sont toutes des valeurs scalaires réels qui peuvent être interpolées.

Enfin,  $n_i$  est le nombre de noyaux de Gabor anisotropes  $f_k(x)$  par cellule  $R_i$ . Pour interpoler entre  $n_i^A$  et  $n_i^B$ , on prend le maximum  $n_i^{max} = \max(n_i^A; n_i^B)$ , ce qui fait disparaître linéairement les amplitudes  $a_k$ , où  $k > n_i^{max}$ , des noyaux qui sont en trop. Cela fait simplement disparaître le surplus de noyaux pendant l'interpolation.

Les figures 8.45, 8.46 et 8.47, illustrent des variations de PPTBF et de structures associées sur des bandes horizontales entre deux PPTBF et interpolation sur une bande intermédiaire.

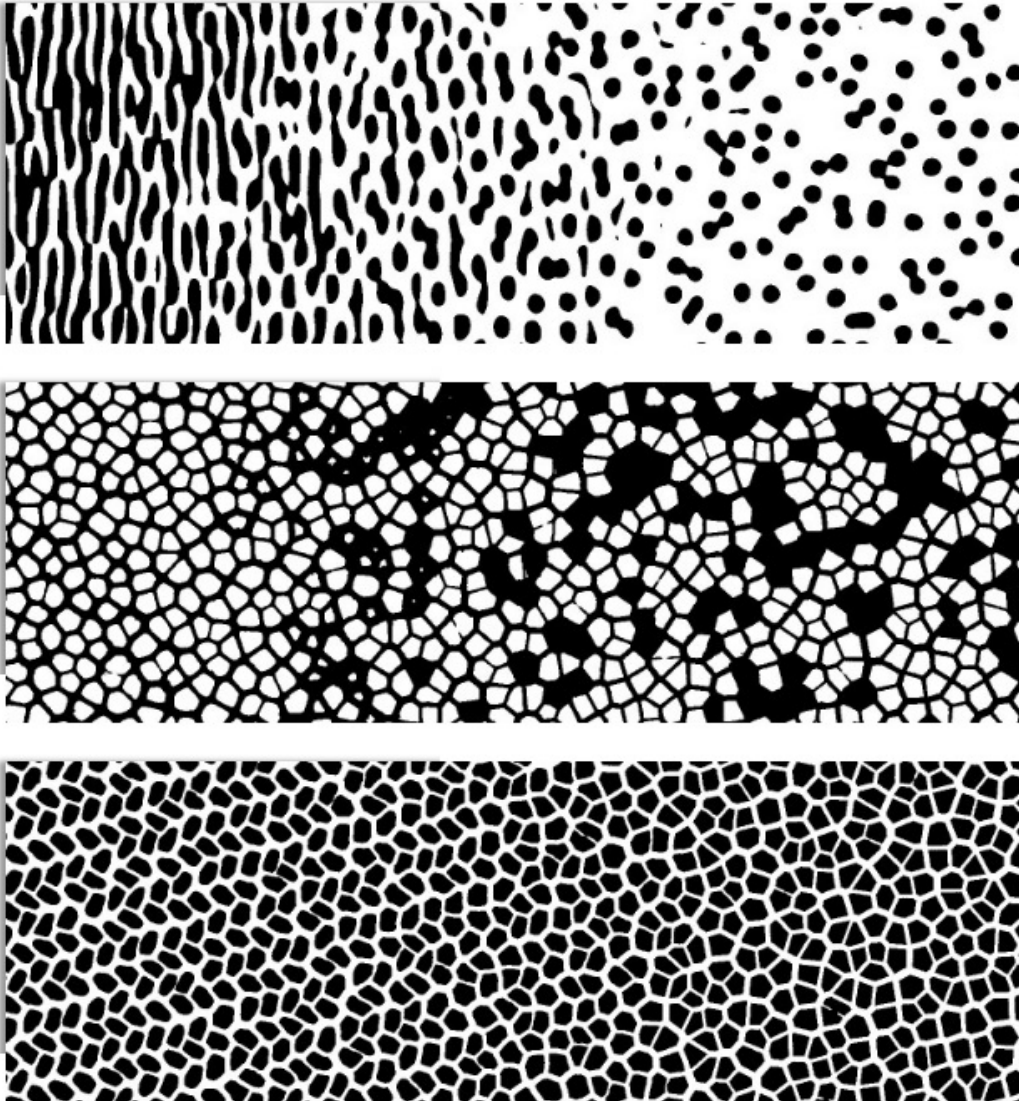


FIGURE 8.45 – Interpolations de structures

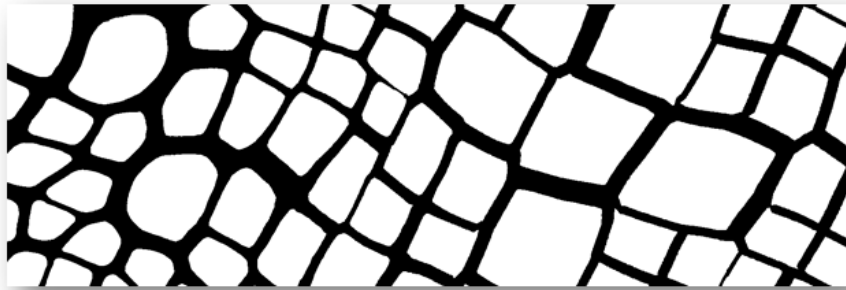


FIGURE 8.46 – Interpolations de structures.

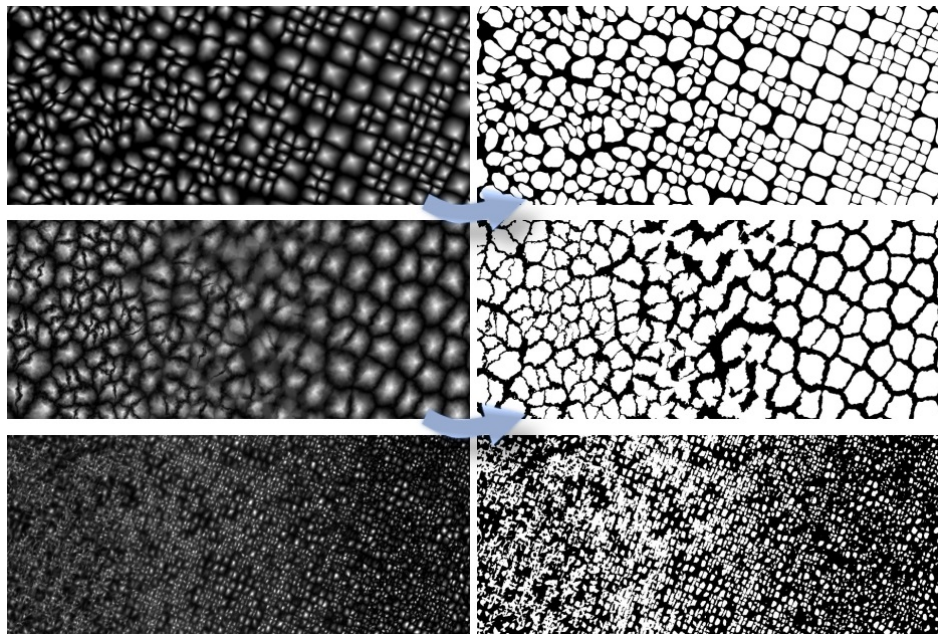


FIGURE 8.47 – Interpolations de structures

# Chapitre 9

## Évaluation du Modèle Procédural de Structures *PPTBF*

Dans cette partie, on présente une évaluation des résultats concernant la capacité de reproduction de structures visuelles à partir du modèle procédural PPTBF décrit au chapitre précédent. Par la suite, on compare nos résultats aux méthodes de l'état de l'art générant du bruit procédural par l'exemple. Enfin, on présente une étude des performances.

### 9.1 Approche expérimentale

Nous avons constitué une base de données d'environ 150 images de structures, obtenues en segmentant manuellement des textures provenant de différentes bases de données, afin d'évaluer la capacité de notre modèle procédural de structures PPTBF à reproduire ces structures. Les détails techniques permettant de déterminer les paramètres du modèle à partir d'une image d'entrée relèvent de la *modélisation procédurale inverse* et sont explicités dans le chapitre 10.

**Recherche reproductible.** On ne présente ici qu'une petite partie des résultats de notre évaluation.

L'ensemble complet des résultats est accessible sur un dépôt *GitHub* public<sup>1</sup>, dans les suppléments de la publication, sous forme de pages web


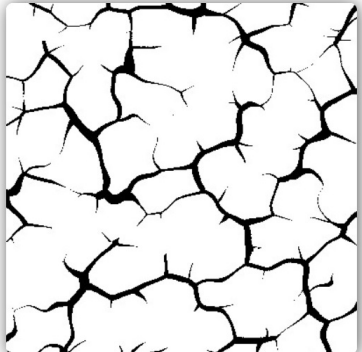
---

1. <https://github.com/ASTex-ICube/semiproctex>



(HTML), notamment le "*Supplemental Material 2*", et sa page "*1. Parameters estimation*". Ces données sont également directement accessibles en ligne<sup>2</sup>.

Pour chacune des images de structure en entrée, nous avons cherché à déterminer un modèle procédural visuellement semblable, et les valeurs des paramètres de celui-ci ont été sauvegardés dans un fichier texte (figure 9.1). Ce fichier stocke les paramètres de notre modèle procédural PPTBF. Son format est détaillé dans l'annexe B. La figure 9.1 est une illustration du site qui présente l'ensemble des résultats : à gauche la texture segmentée, au milieu le résultat d'une structure PPTBF d'aspect similaire et à droite un lien d'accès direct aux paramètres de PPTBF ayant permis d'obtenir le résultat du milieu.

Input binary structure	Procedural structure	PPTBF Parameters
		<p style="text-align: center;"><a href="#">Parameters</a></p>

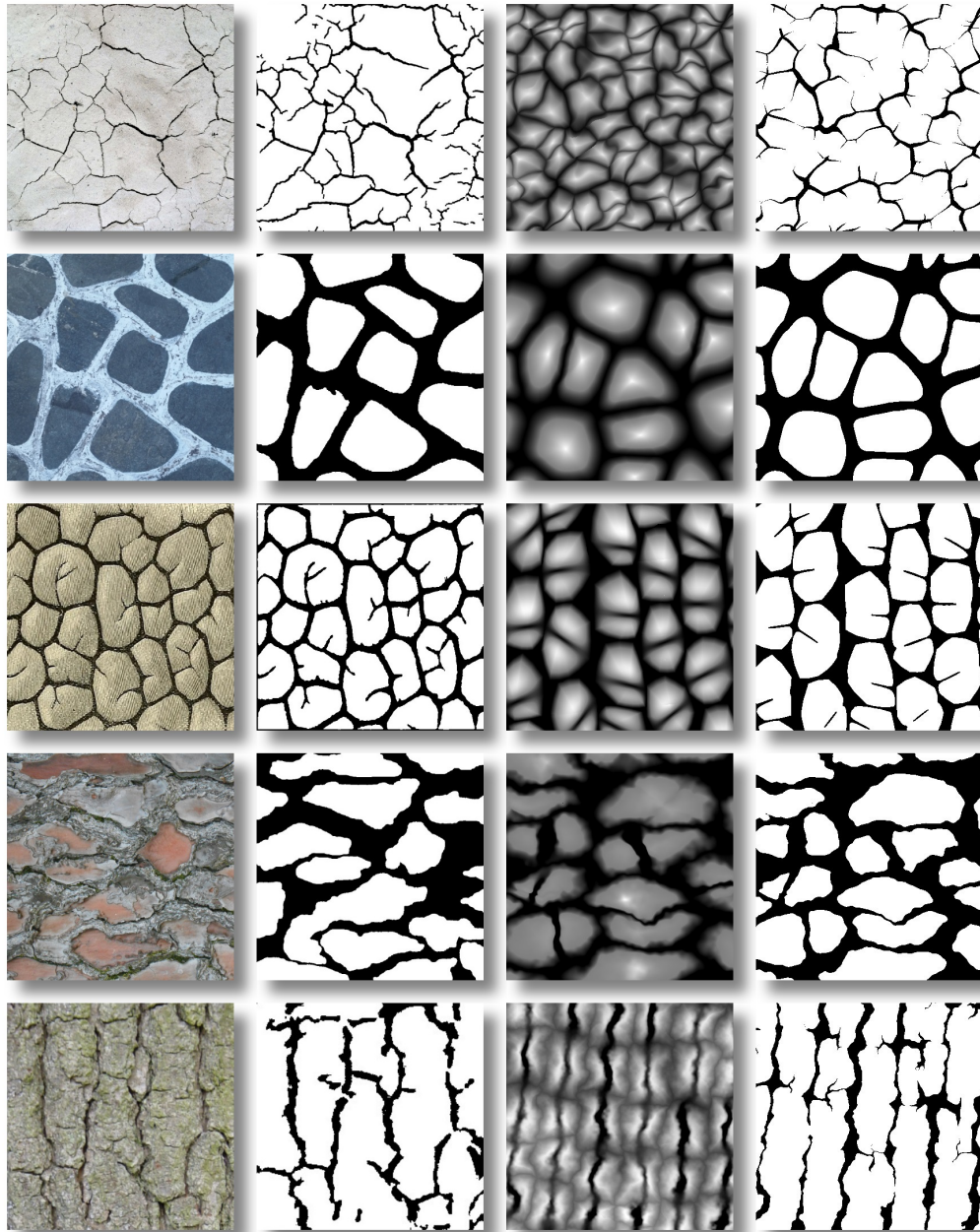
**FIGURE 9.1** – Exemple de reproduction de structures. *De gauche à droite : l'image de structure en entrée, un modèle procédural PPTBF visuellement semblable, un lien vers le fichier des paramètres du modèle PPTBF associé.*

Notons que le but n'est pas ici d'évaluer qualitativement et quantitativement le degré de la ressemblance, mais plutôt de montrer que la PPTBF est capable de couvrir une large gamme de structures que l'on observe fréquemment dans les bases de données de textures naturelles.

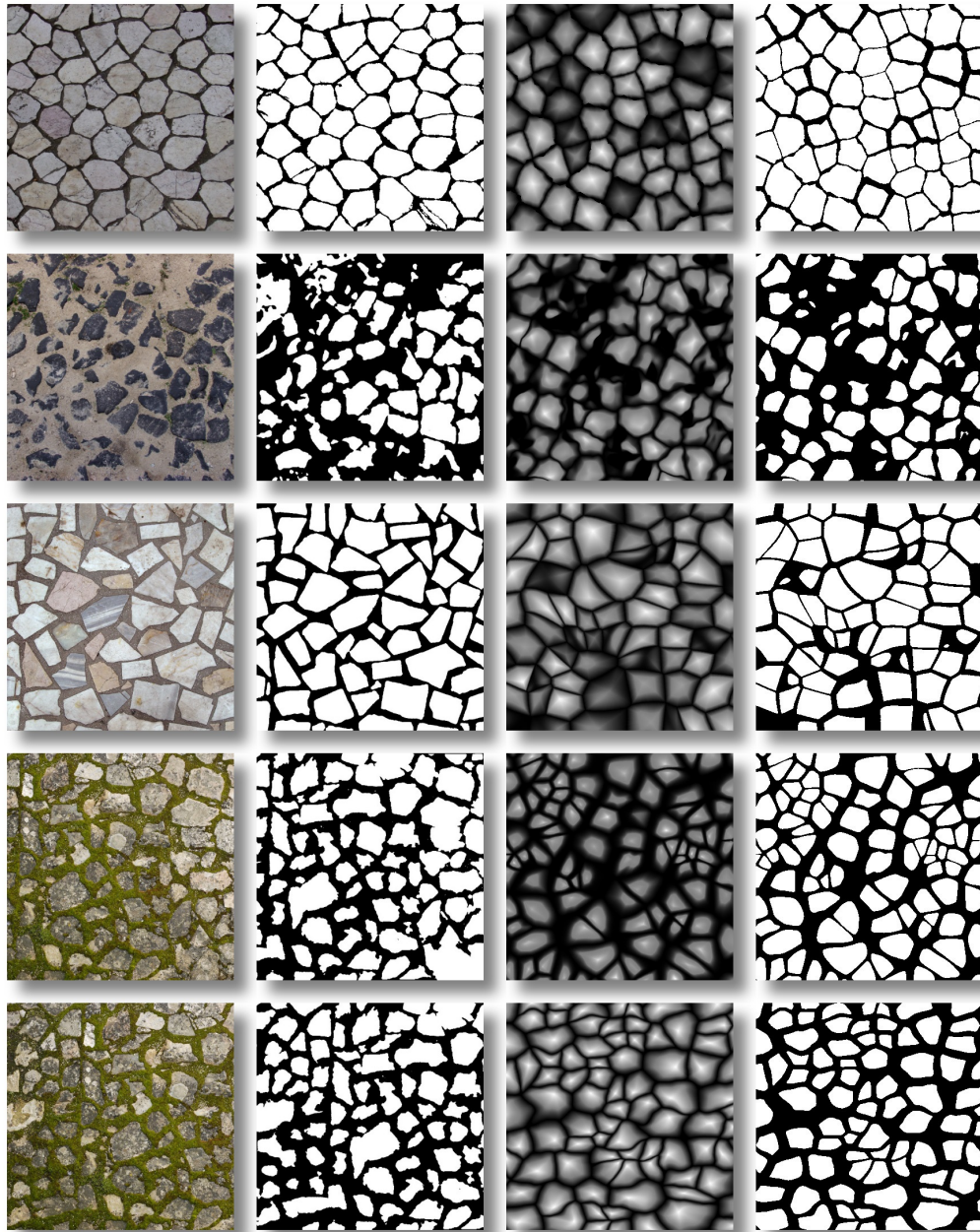
Dans la suite, on présente des exemples de résultats de synthèse de structures regroupés par catégories : cellules, tâches éparées, lignes, tâches éten-

2. [http://igg.unistra.fr/People/semiproctex/data/Supplemental2\\_v100.zip](http://igg.unistra.fr/People/semiproctex/data/Supplemental2_v100.zip)

dues, empilements, vaguelettes et pavages. C'est une proposition de classification dans le seul but de regrouper les résultats ayant des caractéristiques visuelles proches. On ne montre qu'un nombre restreint d'exemples extraits de la base complète dans chacun des cas. Les transformations (rotation, mise à l'échelle) et déformations (distorsion de l'espace par mouvement Brownien fractal) spatiales appliquées à la PPTBF permettent de s'adapter aux dimensions et formes des motifs d'entrée et ainsi d'augmenter "perceptuellement" le degré de similarité. Pour l'ensemble des résultats de cette section les figures montrent de gauche à droite : la texture, sa structure segmentée, le champ scalaire généré par la PPTBF, et une image de PPTBF seuillée.

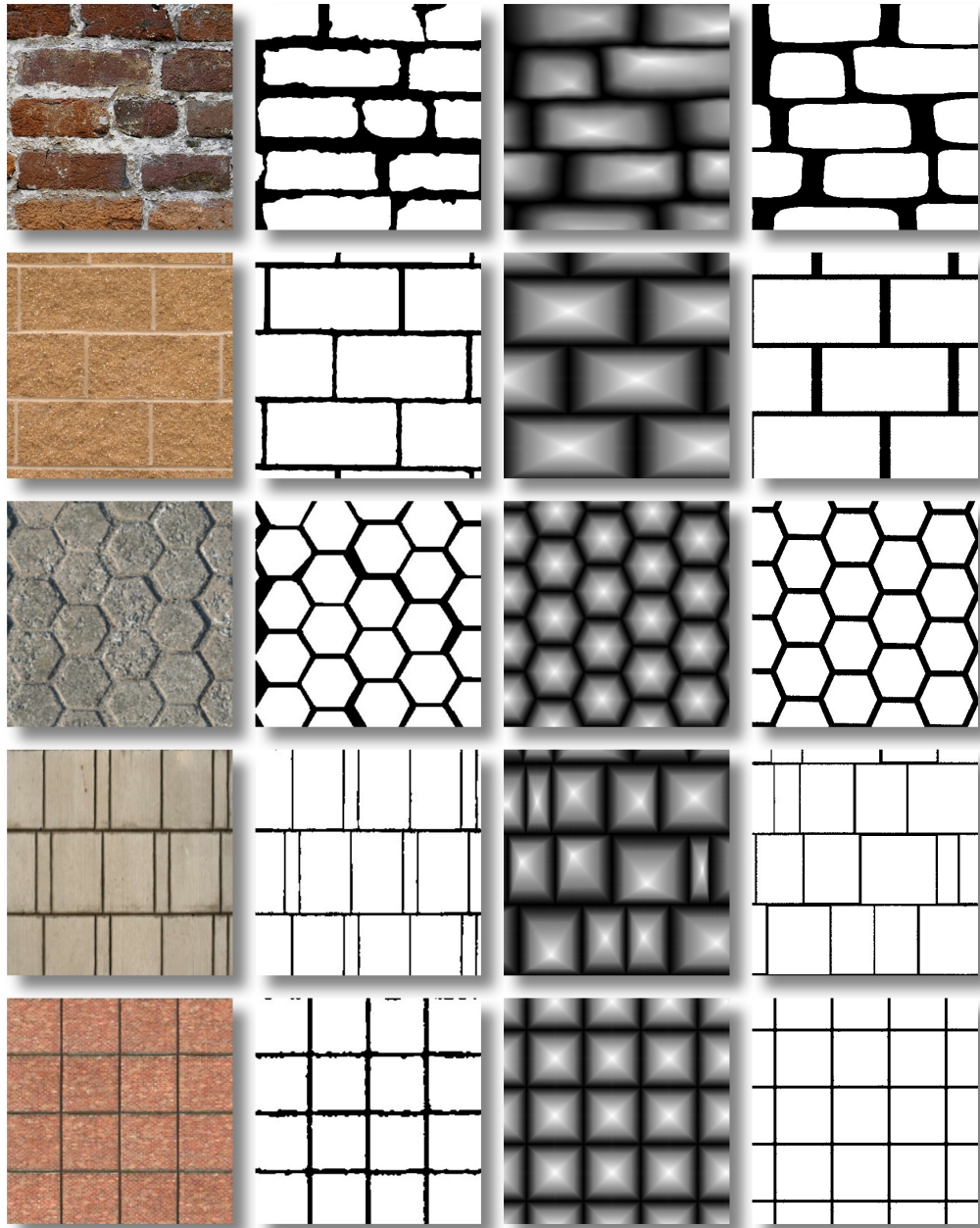


**FIGURE 9.2** – Résultats d'estimations de paramètres de cellules. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

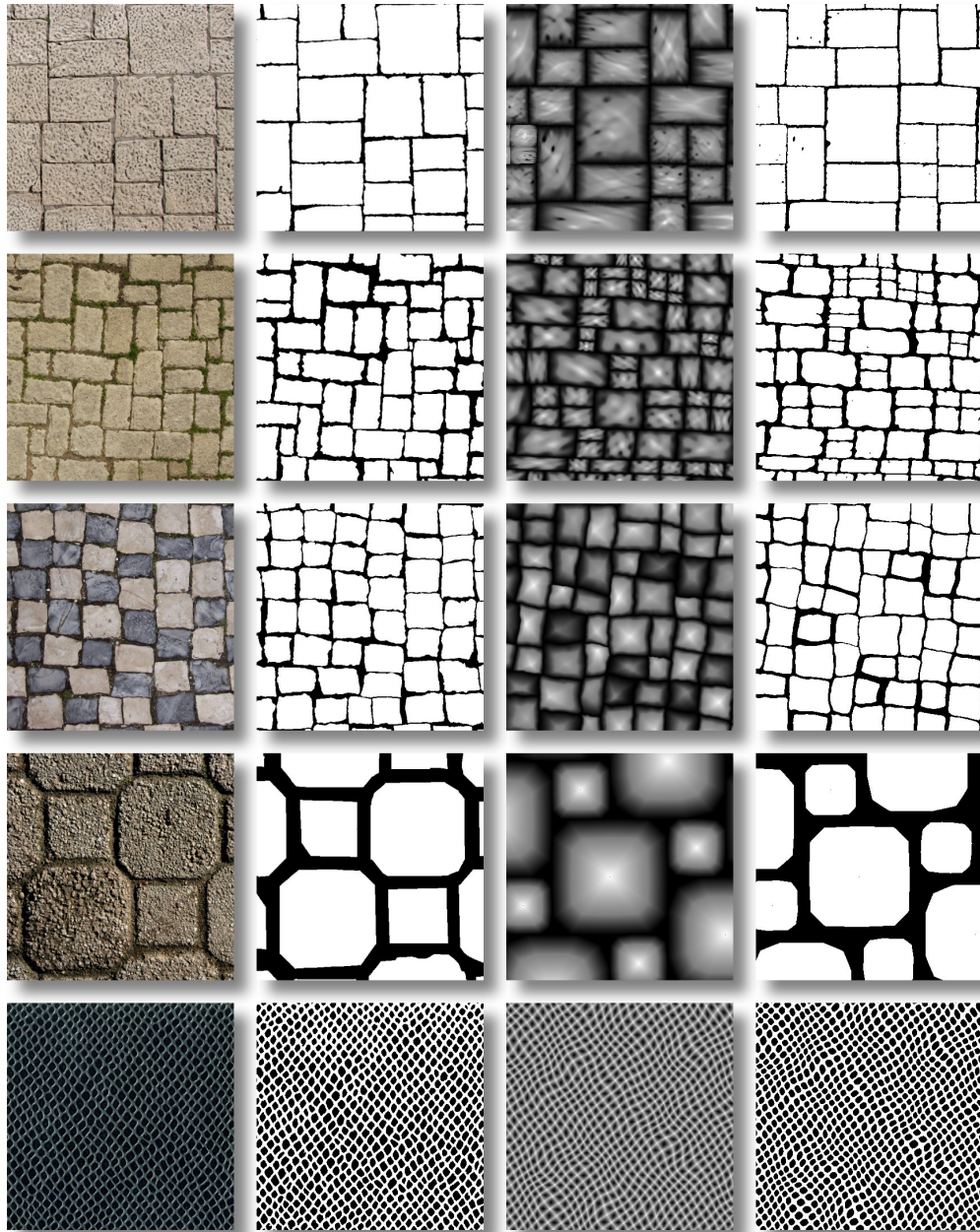


**FIGURE 9.3** – Résultats d'estimations de paramètres de cellules. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

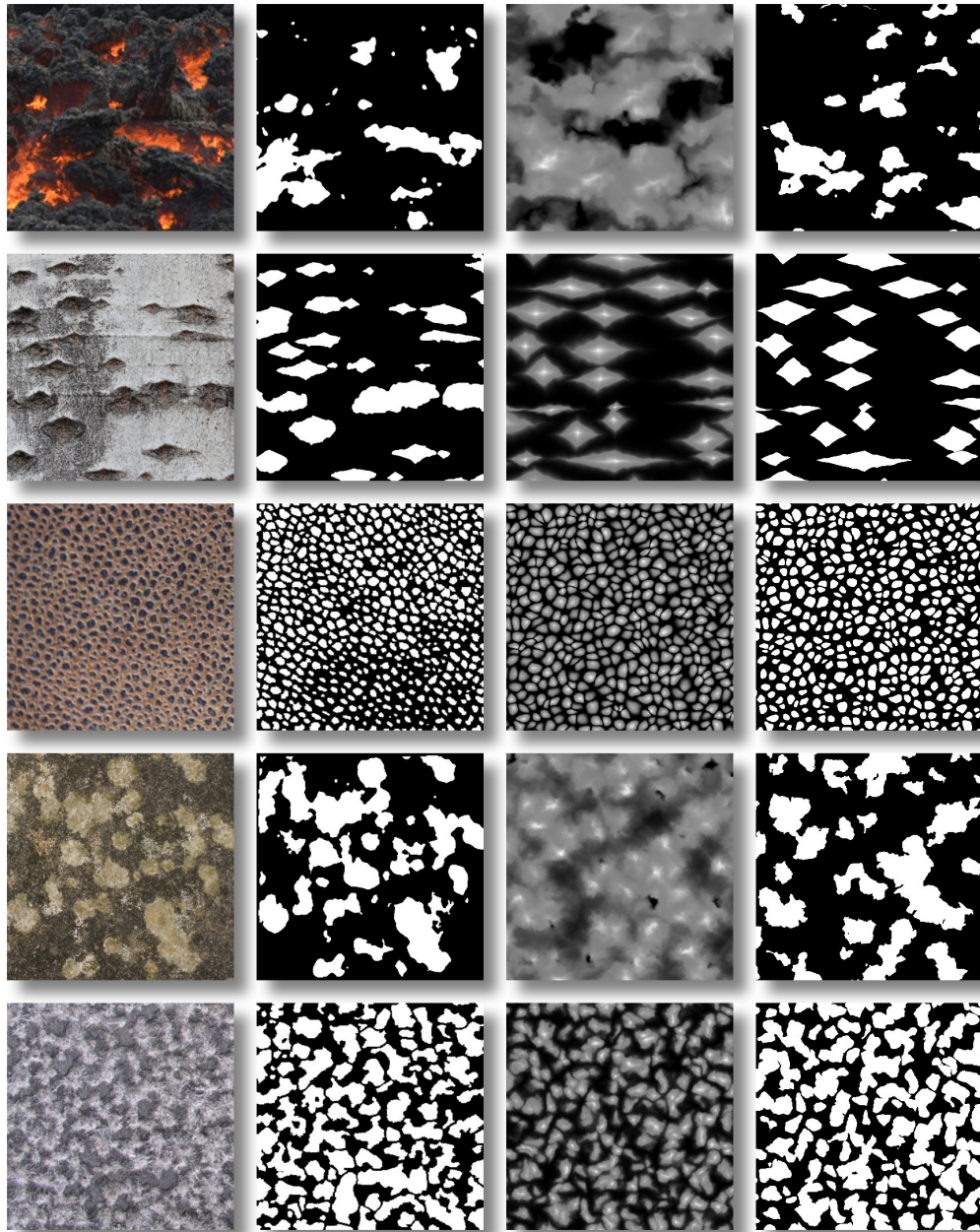




**FIGURE 9.4** – Résultats d'estimations de paramètres de pavages. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

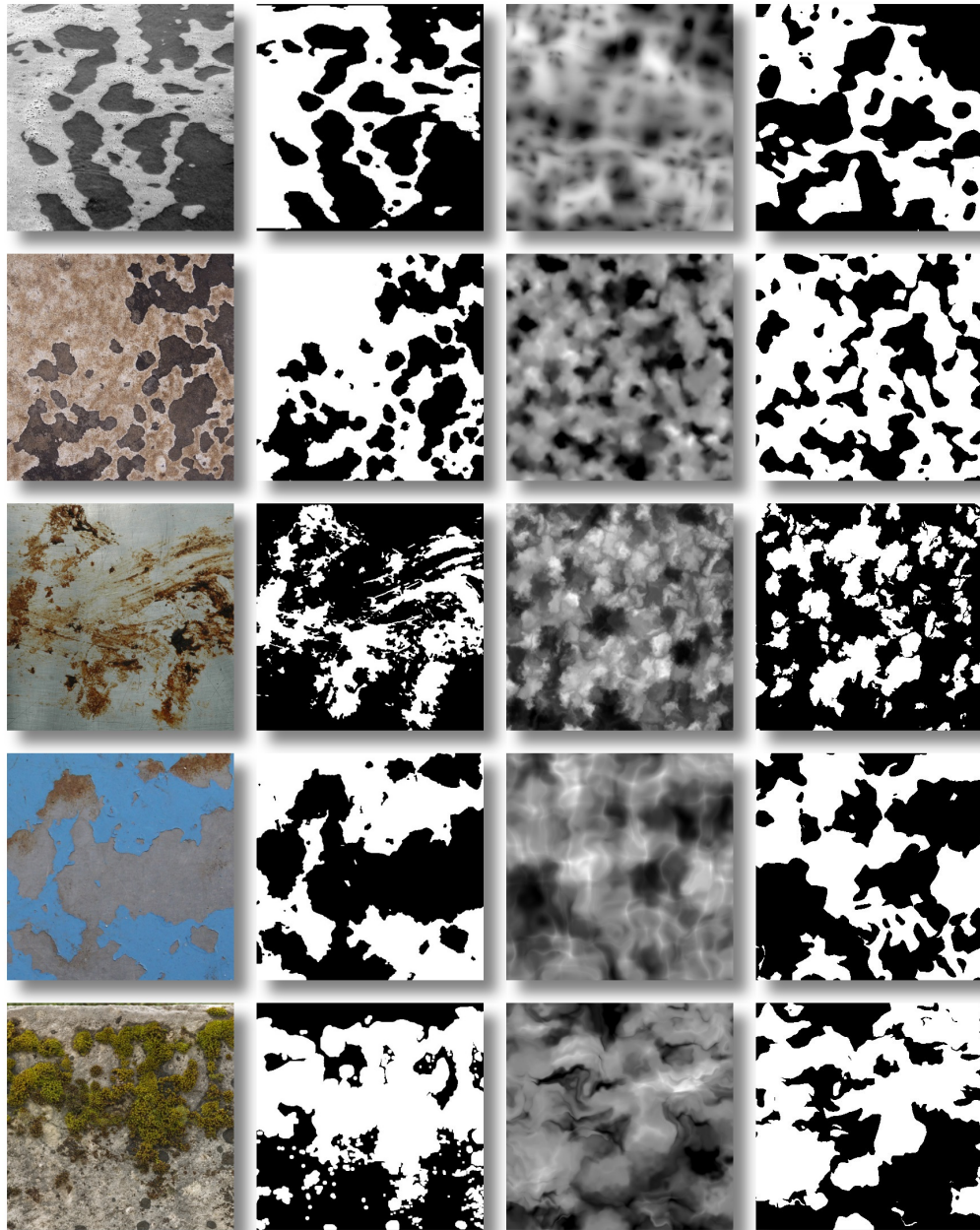


**FIGURE 9.5** – Résultats d'estimations de paramètres de pavages. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

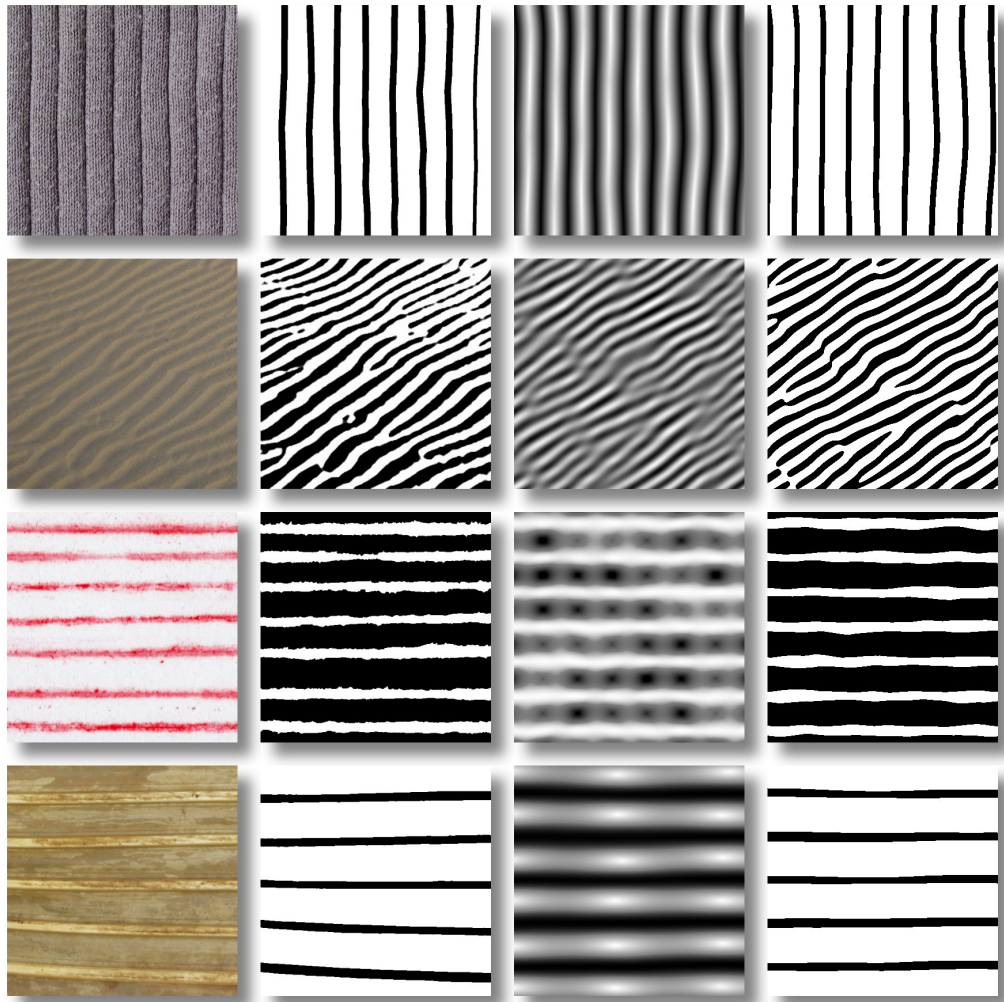


**FIGURE 9.6** – Résultats d'estimations de paramètres de tâches éparses. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

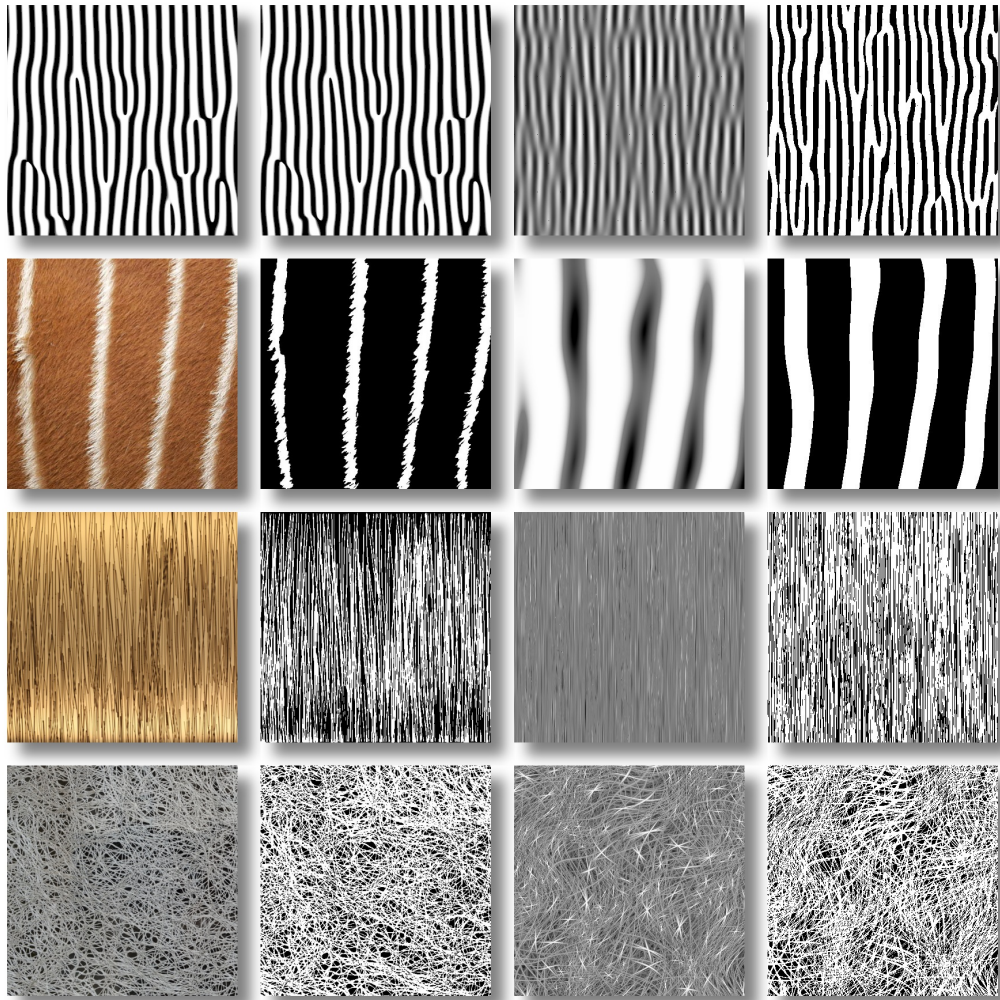




**FIGURE 9.7** – Résultats d'estimations de paramètres de tâches étendues. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

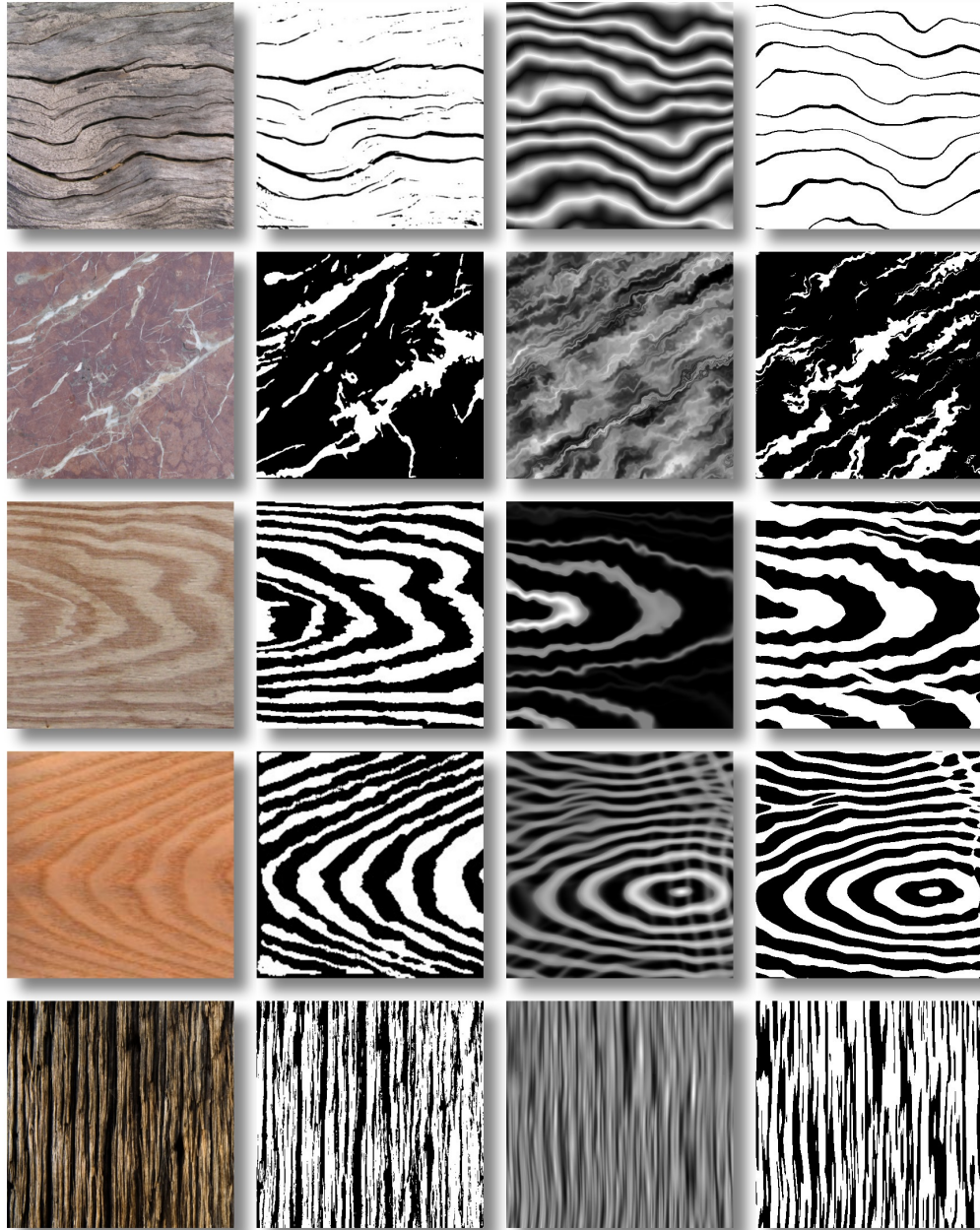


**FIGURE 9.8** – Résultats d'estimations de paramètres d'ondes. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seillée.*

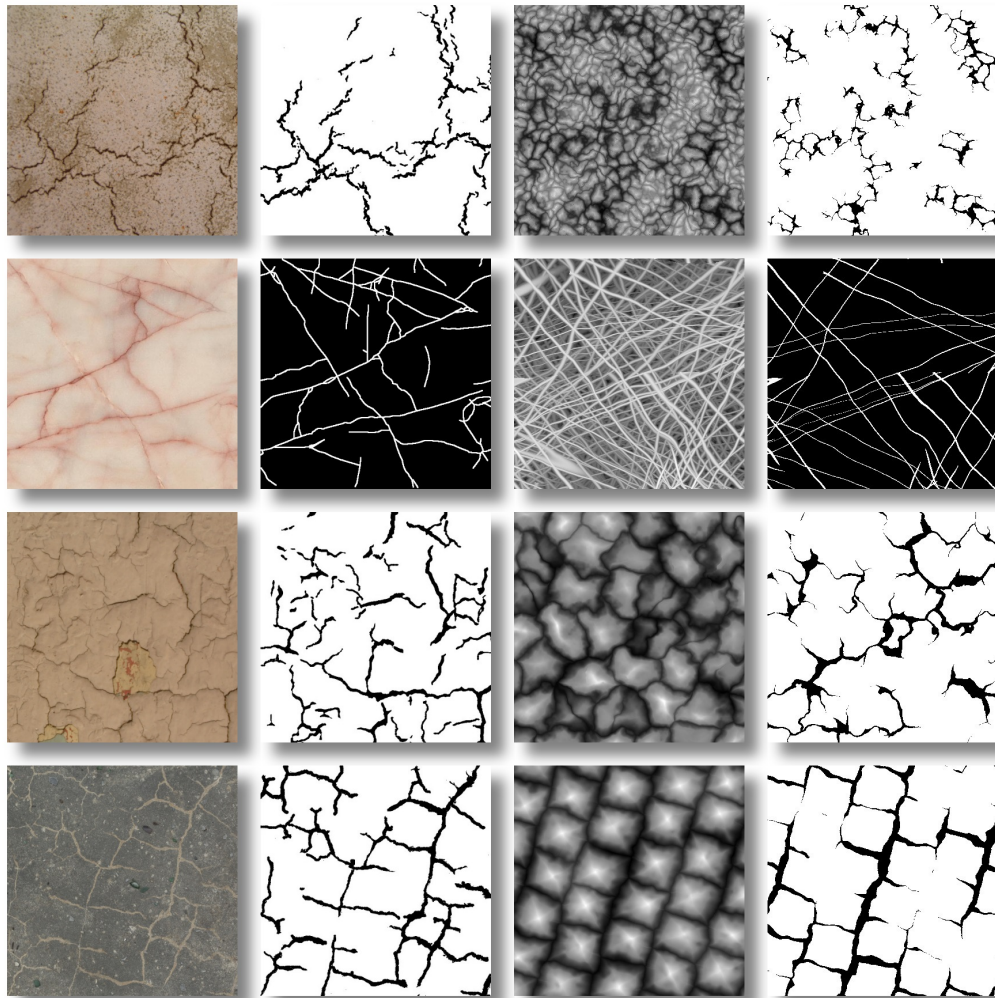


**FIGURE 9.9** – Résultats d'estimations de paramètres de lignes. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*

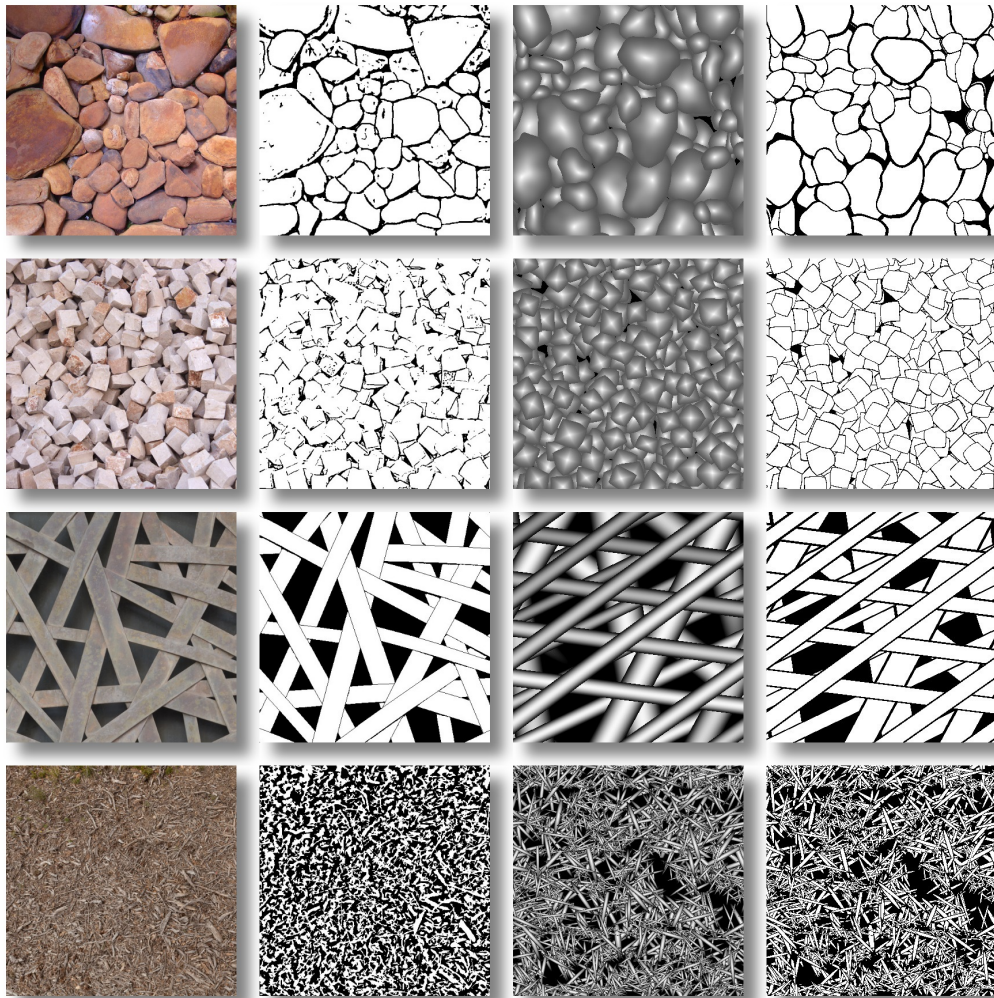




**FIGURE 9.10** – Résultats d'estimations de paramètres de grains. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*



**FIGURE 9.11** – Résultats d'estimations de paramètres de réseaux. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*



**FIGURE 9.12** – Résultats d'estimations de paramètres de piles. *De gauche à droite : texture, texture segmentée, PPTBF, PPTBF seuillée.*



## 9.2 Comparaisons aux méthodes du type *bruit par l'exemple*

Nous avons comparé nos résultats de synthèses de structures avec des approches de type bruit par l'exemple, *Noise by Example*, de l'état de l'art. Nous avons choisi les deux méthodes :

- *Gabor Noise by Example* [GLLD12];
- *Local Random-phase Noise for Procedural Texturing* [GSV+14].

Concernant [GSV+14], la valeur du paramètre  $r$ , indiquant le taux d'aléatoire dans les phases, est fixée à 0.2 (soit 20%). Nos comparaisons sont disponibles en ligne<sup>3</sup>.

Pour pouvoir comparer nos résultats à ces méthodes, qui, à la base, ne sont pas adaptées à des images binaires, des post-traitements ont été appliqués sur les résultats obtenus avec ces deux méthodes. En effet, les images binaires contiennent des hautes fréquences liées aux contours francs, ce qui induit un biais dans le domaine fréquentiel (apparition de hautes fréquences qui ne sont pas dans les structures). Pour chaque résultat de synthèse, la suite de post-traitements suivants a été appliqué :

1. Un filtre Gaussien a été appliqué avec, pour largeur de bande, soit  $\sigma = 1$  soit  $\sigma = 2$ .
2. Ensuite, l'image a été binarisée en utilisant l'algorithme d'Otsu [Ots79], *A threshold selection method from gray-level histograms*, pour déterminer le seuil approprié.
3. Puis l'image binaire a été filtrée en appliquant une fermeture morphologique avec un élément structurant en forme de disque de rayon 2, et les petites composantes connexes de surface inférieure à 128 pixels ont été supprimées.

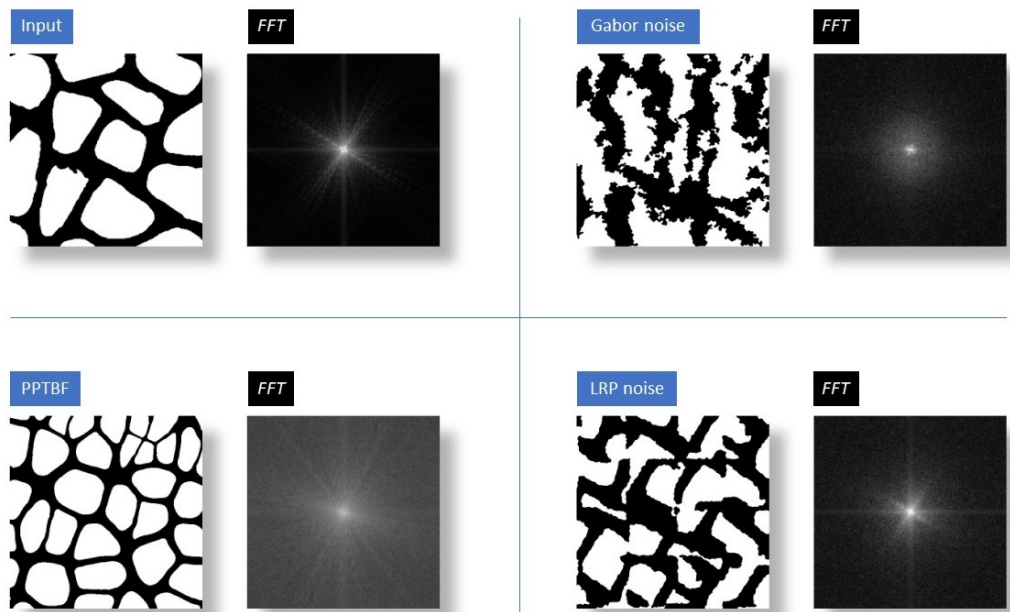
Ces post-traitements nous ont permis d'améliorer grandement la qualité visuelle des résultats des méthodes de bruit par l'exemple.

**Modèles à phases aléatoires** Comme on l'a souligné dans l'état de l'art, les approches basées sur la synthèse de micro-textures ou textures gaussiennes comme *Gabor Noise by Example* [GLLD12] ne permettent pas de générer des structures avec des bords francs. Il n'y a pas de contrôle précis sur les phases

---

3. [http://jgg.unistra.fr/People/semiproctex/data/Supplemental2\\_v100.zip](http://jgg.unistra.fr/People/semiproctex/data/Supplemental2_v100.zip)

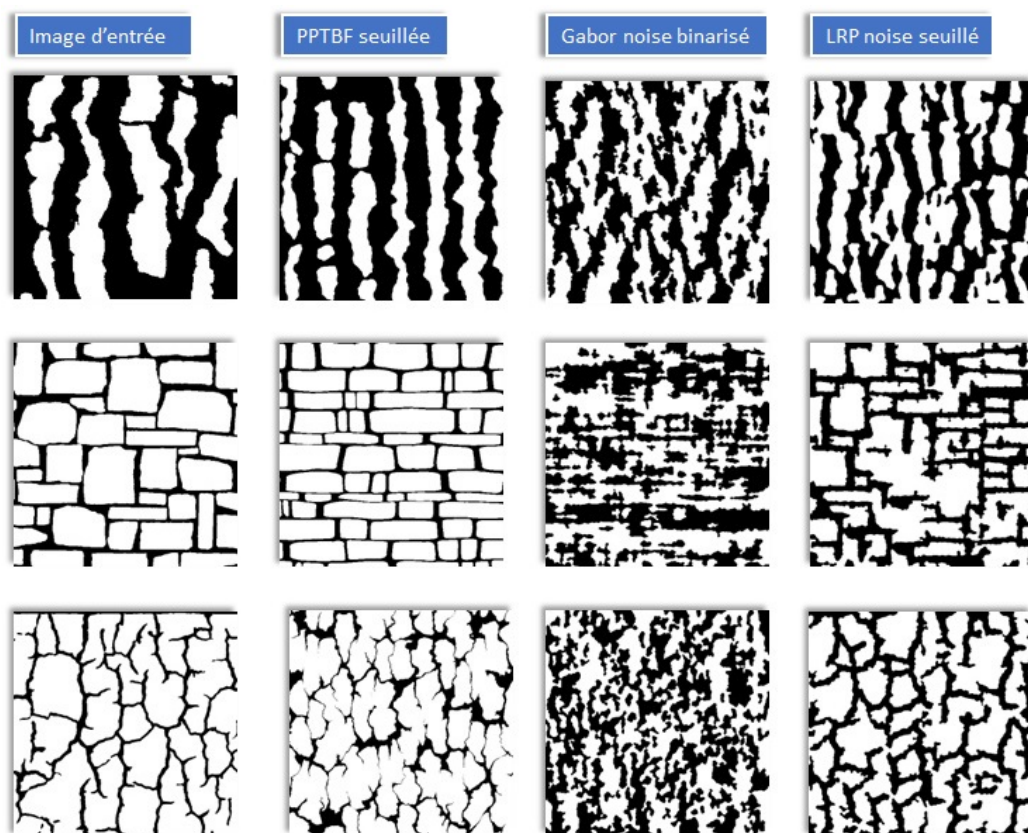
qui permettraient de synchroniser les fronts d'ondes de la base des exponentielles complexes de Fourier, ce sont des modèles de bruits à phase aléatoire (Random Phase Noise). Comme on peut le voir sur les figures 9.13, 9.14 et



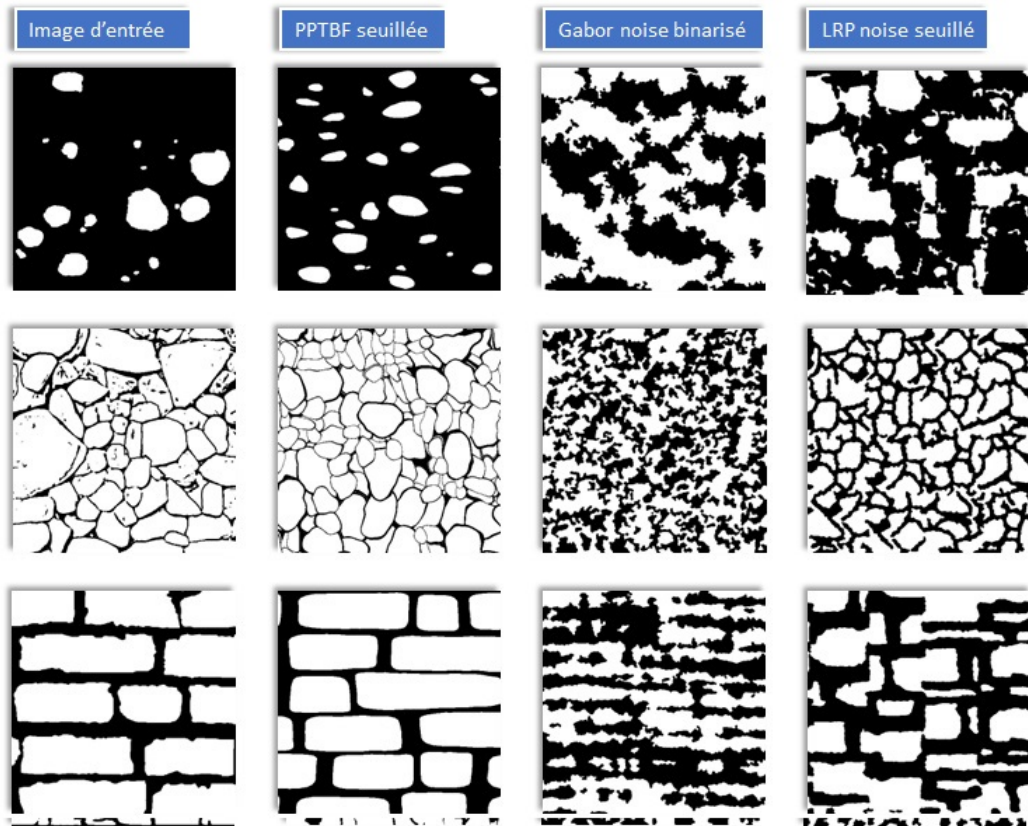
**FIGURE 9.13** – Comparaisons des méthodes de synthèses de structures (après avoir été post-traitées comme indiqué plus haut). Le modèle PPTBF est un modèle de structure explicite et cohérent sur le plan 2D, alors que les approches à base de phases aléatoires comme Gabor [GLLD12] et le LRP noise [GSV<sup>+</sup>14] ne permettent pas d’obtenir de modèles contrôlables et similaires à l’entrée. LRP noise reproduit localement de la structure mais n’est pas cohérent à des échelles plus grandes. Gabor noise prend en compte des cohérences à plus grandes échelles.

9.15, de manière générale, l’approche [GLLD12] n’arrive pas reproduire des bords francs. Les structures sont plus des tâches étendues et tâches éparées, ainsi que des lignes et ondes. Les informations de directions (orientation des motifs et formes) et les échelles semblent parfois être capturées (taille), mais pas les corrélations longues distances comme des lignes (ou formes étendues) traversant toute l’image.

**Modèles avec contrôle de phases aléatoires** Contrairement aux modèles à phases aléatoires et textures gaussiennes, l'approche de *Local Random-phase Noise for Procedural Texturing* [GSV<sup>+</sup>14] permet d'imposer un contrôle sur une sélection de phases (celles ayant le plus d'énergie). Il est alors possible de synthétiser des bords plus structurés. Comme on peut le voir sur les figures 9.13, 9.14 et 9.15, le modèle [GSV<sup>+</sup>14] n'arrive pas à capturer les corrélations longues distances des formes étendues, traversant toute l'image. Les structures sont mieux représentées, mais localement, on voit l'apparition de blocs, et la cohérence globale n'est pas toujours respectée.



**FIGURE 9.14** – Comparaisons : PPTBF et bruits procéduraux. De gauche à droite : image d'entrée, PPTBF seuillée, Gabor noise binarisé, LRP noise seuillé.

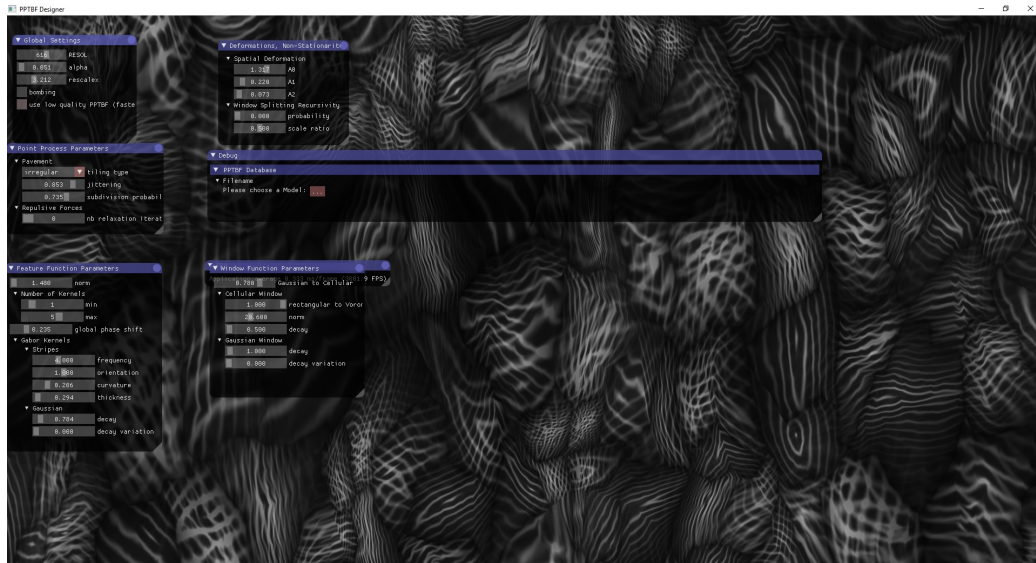


**FIGURE 9.15** – Comparaisons : PPTBF et bruits procéduraux. *De gauche à droite : image d'entrée, PPTBF seuillée, Gabor noise binarisé, LRP noise seuillé.*

## 9.3 Performances

Le modèle de structures procédurales PPTBF peut être généré et édité de manière interactive, voire temps-réel, selon la taille de l'image demandée. Nous avons développé plusieurs outils accélérés sur GPU, basés sur les langages C++ et la librairie graphique OpenGL (voir figure 9.16). En particulier, nous avons mesuré 9 ms pour une image  $256^2$ , 37 ms pour  $512^2$ , 153 ms pour  $1024^2$  et 606 ms pour  $2048^2$  sur une GeForce GTX 1060 avec 6 Go de RAM.

D'après nos expérimentations, les performances sont stables pour la plupart des combinaisons de paramètres de PPTBF utilisés, produisant différents types de structures. Cependant, l'utilisation de la relaxation de Lloyd sur les processus ponctuels, activée et hard-codée pour certains types de tiling, nécessite un algorithme itératif plus gourmand sur GPU.



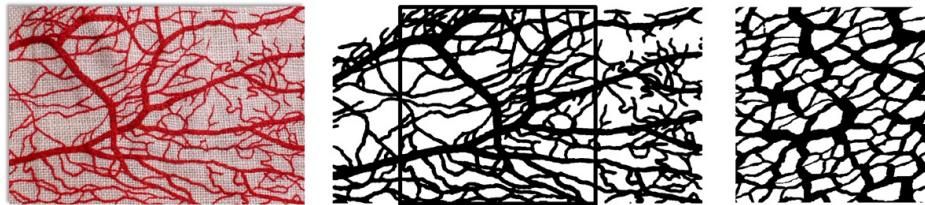
**FIGURE 9.16** – Performances : outil de génération et édition temps réel de notre modèle de structures procédurales PPTBF.



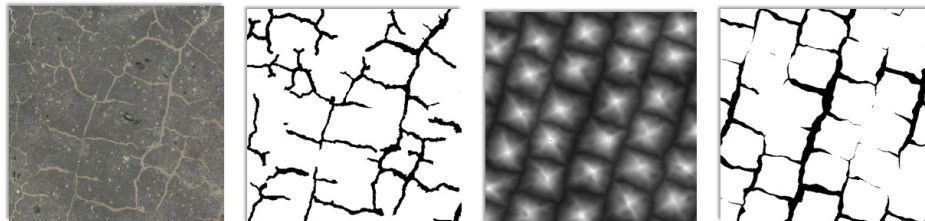
## 9.4 Limitations

**Gamme des structures représentables** Le but initial de notre modèle procédural de structures était d'élargir la gamme des textures représentables par un modèle de bruit de convolution éparse. Ce modèle permet également de traiter certaines structures quasi-régulières à régulières, mais en partie seulement.

Par contre, il ne permet pas de représenter des structures sémantiques complexes, par exemple des branchements (voir figures 9.17 et 9.18) ou, comme on l'avait déjà évoqué, des motifs précis comme par exemple des amas de feuilles d'érable.



**FIGURE 9.17** – La PPTBF ne gère pas les branchements. De gauche à droite : image d'entrée et sa structure binaire, PPTBF estimée.



**FIGURE 9.18** – La PPTBF ne gère pas les branchements. De gauche à droite : image d'entrée et sa structure binaire, PPTBF estimée.



# Chapitre 10

## Estimation des Paramètres des Structures Procédurales *PPTBF*

### 10.1 Introduction

Dans ce chapitre, nous décrivons une approche mise en oeuvre permettant de faciliter l'estimation des paramètres de la PPTBF en utilisant un exemple en entrée. Il ne s'agit pas d'une contribution, dans le sens où la méthode n'est pas automatique. Il s'agit essentiellement de présenter un travail préliminaire visant à apporter un premier outil à un utilisateur. En effet, cette thématique est très vaste et complexe, et nécessitera un travail plus poussé qui sort du cadre de cette thèse.

#### 10.1.1 Contexte

On souhaite pouvoir répondre au scénario suivant : *étant donné une image de structure (binaire) fournie en entrée, comment trouver la structure visuelle procédurale PPTBF la plus proche (c.-à-d. ses paramètres procéduraux), dont la version seuillée ressemble à la structure d'entrée ?* La figure 10.1 illustre notre approche.

Dans la suite, on explique chacune des étapes, nécessitant partiellement des manipulations par l'utilisateur, et mises en place pour estimer les paramètres procéduraux de PPTBF à partir d'images de structures binaires en entrée.

## By-example PPTBF parameters estimation

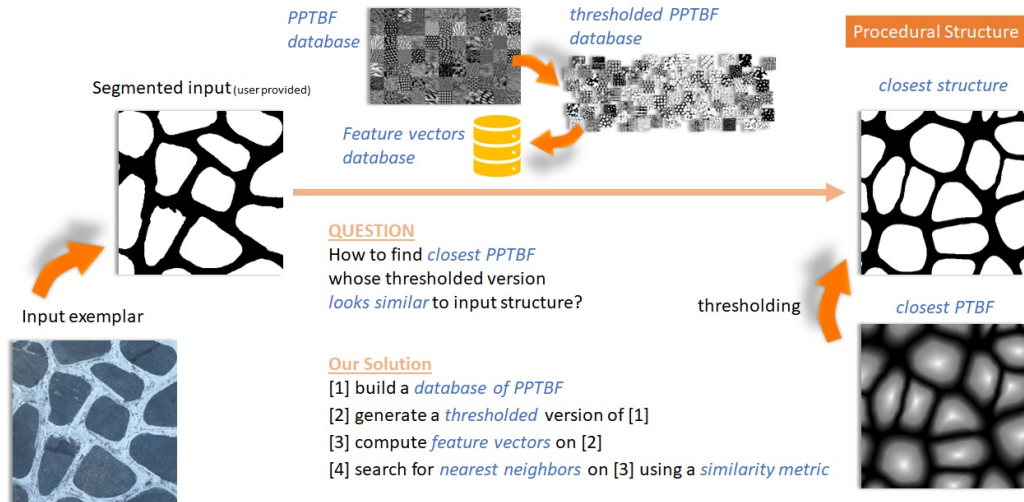


FIGURE 10.1 – Aperçu de notre approche de modélisation procédurale inverse.

### 10.1.2 Problématique

À la base il faut pouvoir comparer des images. Ceci correspond à une approche classique en vision par ordinateur et *pattern recognition*, où l'on cherche à déterminer la ou les plus proches images entre une image en entrée et celles d'une base de données d'images. Comparer des images pixel à pixel n'a pas de sens. On a besoin d'une métrique perceptuelle de similarité. Pour cela, on passe souvent par des comparaisons de distances entre des descripteurs d'images.

Afin de pouvoir estimer les paramètres des structures procédurales, deux éléments sont donc nécessaires :

- un descripteur pour caractériser le contenu des images de structures ;
- une métrique pour quantifier la similarité entre des images de structures via leurs descripteurs.

Le descripteur va servir à la fois à caractériser le contenu des images de structures. La métrique va servir à définir une notion de similarité entre textures via leur descripteur. Il peut également permettre de classifier les textures en catégories.

Nous utilisons un schéma itératif pour récupérer les paramètres optimaux,

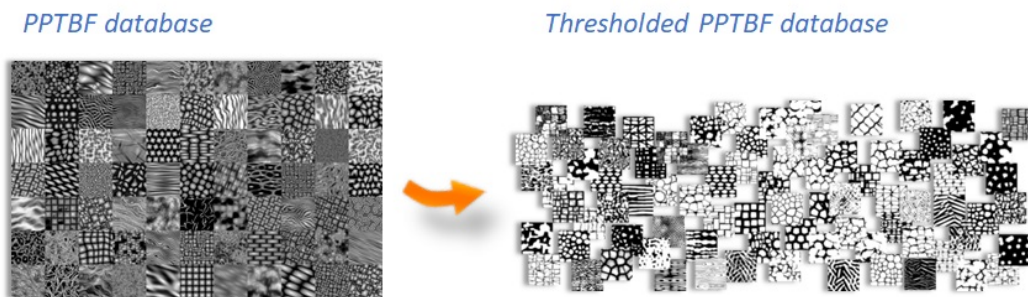
basés sur une mesure de similarité. Une base de données de PPTBF précalculées, échantillonnées et seuillées est utilisée pour initialiser une recherche aléatoire de type Monte Carlo.

Notre solution consiste en 4 étapes :

1. construire une base de données de PPTBF ;
2. générer une version seuillée de 1. ;
3. calculer les vecteurs de caractéristiques sur 2. ;
4. rechercher les voisins les plus proches sur 3. en utilisant une métrique de similarité.

## 10.2 Base de données de structures procédurales

La première étape de notre approche nécessite de générer une base de données de structures procédurales PPTBF (figure 10.2).

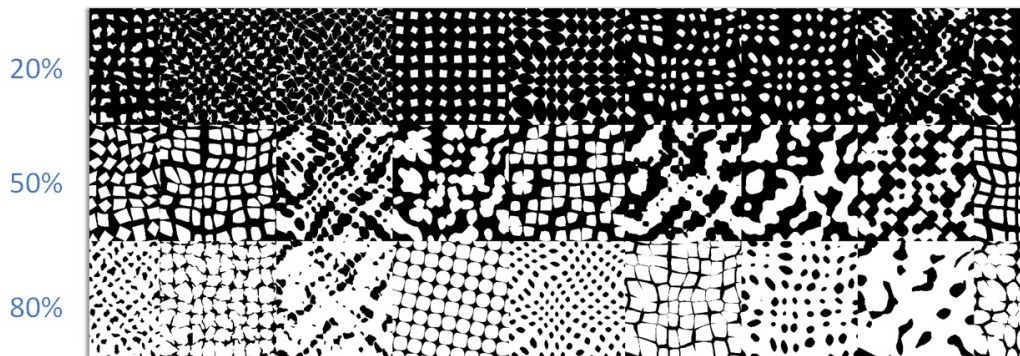


**FIGURE 10.2** – Génération d'une base de données de structures procédurales.

Pour cela, nous avons échantillonné l'espace des paramètres de la PPTBF et sauvegardé chaque image. Le modèle de structures PPTBF comporte environ une trentaine de paramètres. Ainsi, effectuer un échantillonnage régulier, conduirait à des milliards de milliards d'images à générer ! En effet, même avec juste 2 valeurs différentes par paramètre, l'ensemble des combinaisons possibles amènent à :  $2^{30}$  possibilités. Cependant, il est à noter que certains paramètres de la PPTBF sont fortement corrélés à d'autres. Il peut même y avoir un repli de l'espace visuel des structures sur lui-même, alors même

que les paramètres sont différents. Pour pallier ce problème, nous avons exploré de manière interactive l'espace de paramètres pour concevoir un plan d'échantillonnage empirique, en évitant les redondances ainsi que de trop grands écarts d'apparence entre les structures visuelles.

Basé sur ce plan d'échantillonnage non régulier des paramètres, nous avons généré une base de données de 450k images, de dimension 400x400 pixels, en appliquant trois seuils différents à la PPTBF afin de générer les images de structures visuelles binaires associées : 20%, 50% et 80%, correspondant au ratio des pixels blancs (la structure) sur les noirs (le fond), voir figure 10.3. En pratique, d'après nos expérimentations, cela suffit, car les exemples d'entrée peuvent être normalisés selon l'un de ces seuils en appliquant une érosion morphologique ou une dilatation. Afin d'uniformiser la distribution des intensités dans les images de PPTBF, on procède à une égalisation d'histogramme avant de les seuiller.



**FIGURE 10.3** – Génération d'une base de données de structures procédurales (extrait). De haut en bas : exemples de PPTBF seuillées à 20%, 50% et 80%, suivant un échantillonnage empirique. Le pourcentage correspond au rapport du nombre de pixels blancs (c'est-à-dire la structure) sur les noirs (c'est-à-dire le fond).

La génération de la base de données aurait nécessité plusieurs semaines sur un seul PC, que nous avons pu réduire à quelques jours en utilisant une vingtaine de PC, équipés de GPUs pour accélérer les calculs. Notons que cette phase de pré-traitement est assez courante, notamment en vision par ordinateur où l'emploi de l'apprentissage profond (deep learning) nécessite très

souvent des étapes manuelles pour créer des jeux de données d'entraînement (par exemple, labellisation d'images), sur plusieurs jours, voire semaines.

### 10.3 Descripteurs de structures et métrique de similarité

À partir de la base de données d'images binaires de structures générées, on génère une base de données de descripteurs de structures (figure 10.4).

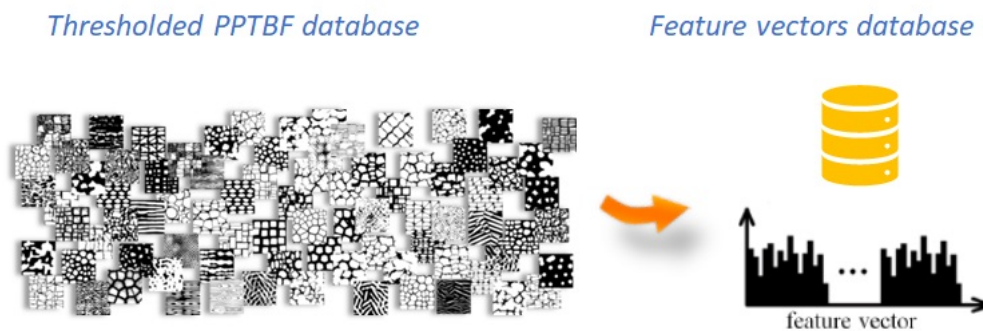
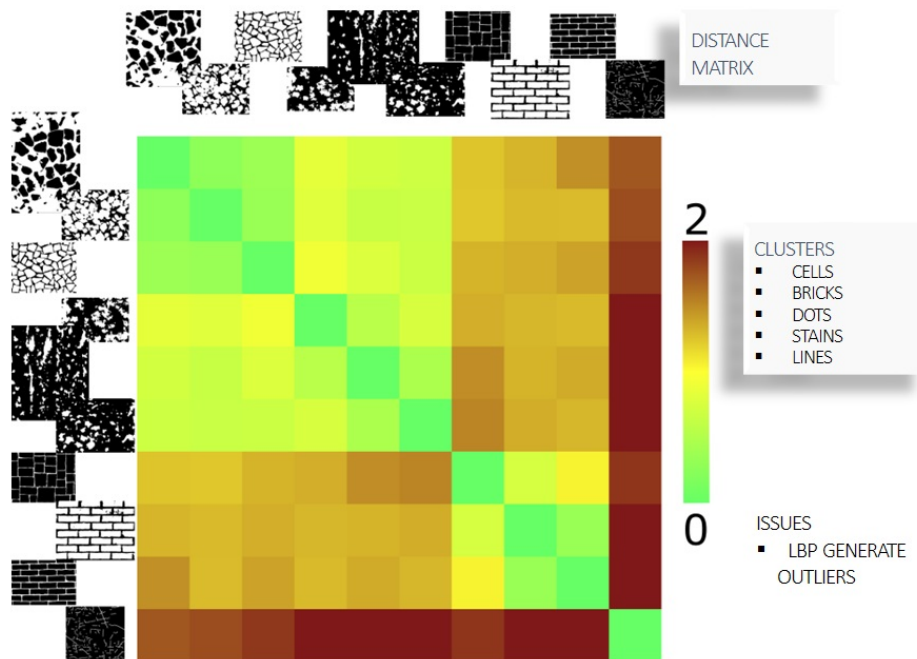


FIGURE 10.4 – Génération d'une base de données de structures procédurales.

Nous utilisons le descripteur pour évaluer les similitudes entre images de structures en calculant des distances sur les vecteurs de caractéristiques (feature vectors). On utilise une norme  $L^2$  sur ces descripteurs, ce qui constitue notre métrique perceptuelle de similarité. La définition de nouveaux descripteurs de textures demanderait une étude approfondie avec des évaluations poussées, ce qui correspond plus à des pistes de travaux futurs et perspectives. De nombreux descripteurs différents ont été proposés au cours des dernières décennies [LCF<sup>+</sup>19] (*From BoW to CNN : Two decades of texture representation for texture classification* de Liu *et al.*), et dans [HH19] (*Texture feature extraction methods : A survey* de Humeau-Heurtier). Nous avons implémenté et testé plusieurs descripteurs classiques, incluant les structurels avec le LBP multi-échelle (Multiscale Local Binary Patterns) et les modèles binaires de Gabor (Gabor Binary Patterns), les fréquentiels (FFT) et spatio-fréquentiels avec les histogrammes des filtres de Gabor, certaines de leurs combinaisons, et ceux basés sur des réseaux de neurones convolutionnels CNN (par exemple,

VGG19). Pour chaque descripteur, nous avons précalculé l'ensemble correspondant de vecteurs de caractéristiques.

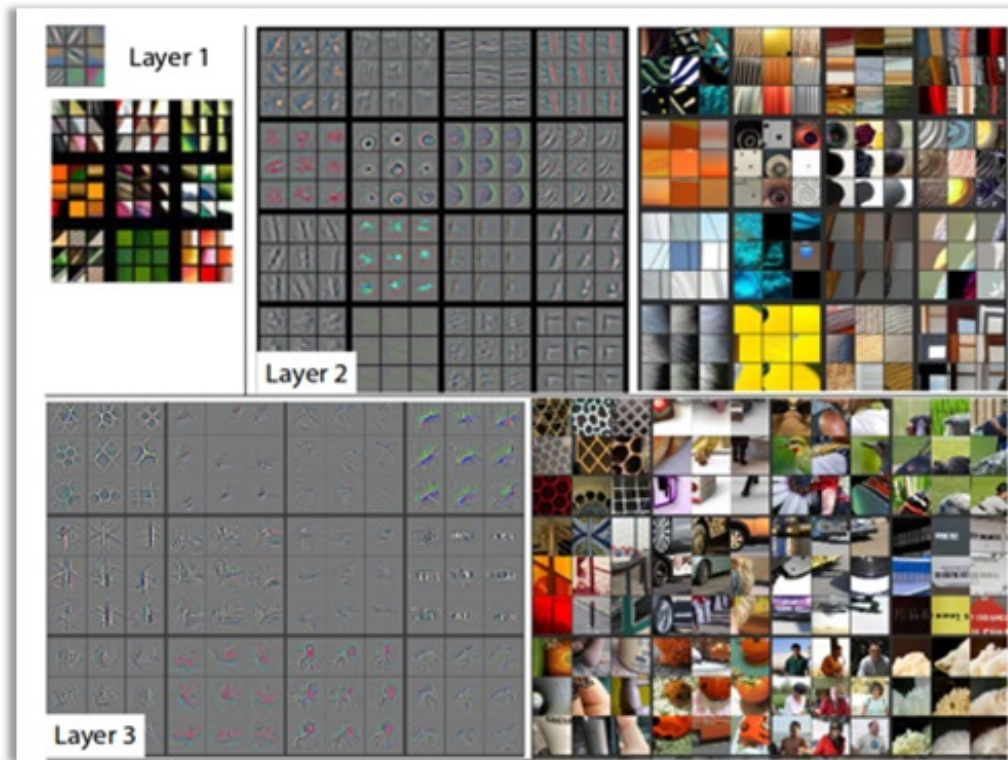
Nous avons expérimenté la capacité de certains descripteurs à classifier les structures à l'aide de calcul de distances et de matrices de dissimilarité (figure 10.5).



**FIGURE 10.5** – Métrique de similarité. Exemple de résultats de métrique de similarité pour évaluer la capacité des descripteurs à reconnaître des structures similaires (ici des premiers tests avec les LBP).

Concernant les descripteurs de réseaux de neurones, nous avons expérimenté ceux issus des réseaux de VGG19, notamment sa dernière couche et également sa 3ème couche. En effet, cette dernière semble intéressante pour l'analyse des structures, car l'observation des réponses de ses filtres montrent une capacité à détecter des formes de base proches de celles présentes dans nos PPTBF (figure 10.6).



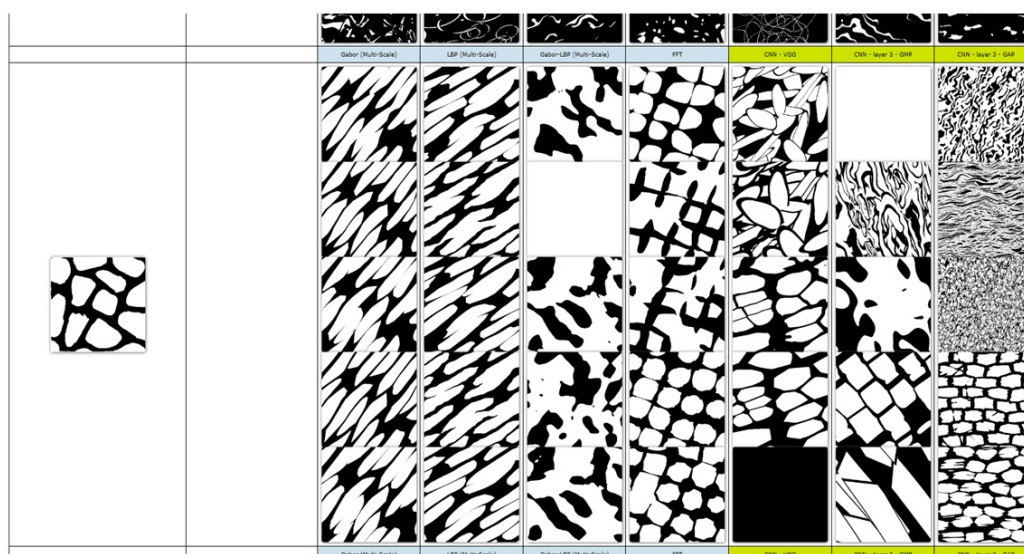


**FIGURE 10.6** – Visualisation des réponses des filtres des premières couches d’un réseau de neurones. Les premières couches semblent contenir des informations similaires aux filtres de bases ainsi que des proto-structures. Nous avons testé des réseaux CNN de type VGG19 [la dernière couche et la 3ème avec des options GMP (global max pooling), GAP (global average pooling)].

## 10.4 Recherche de plus proches voisins

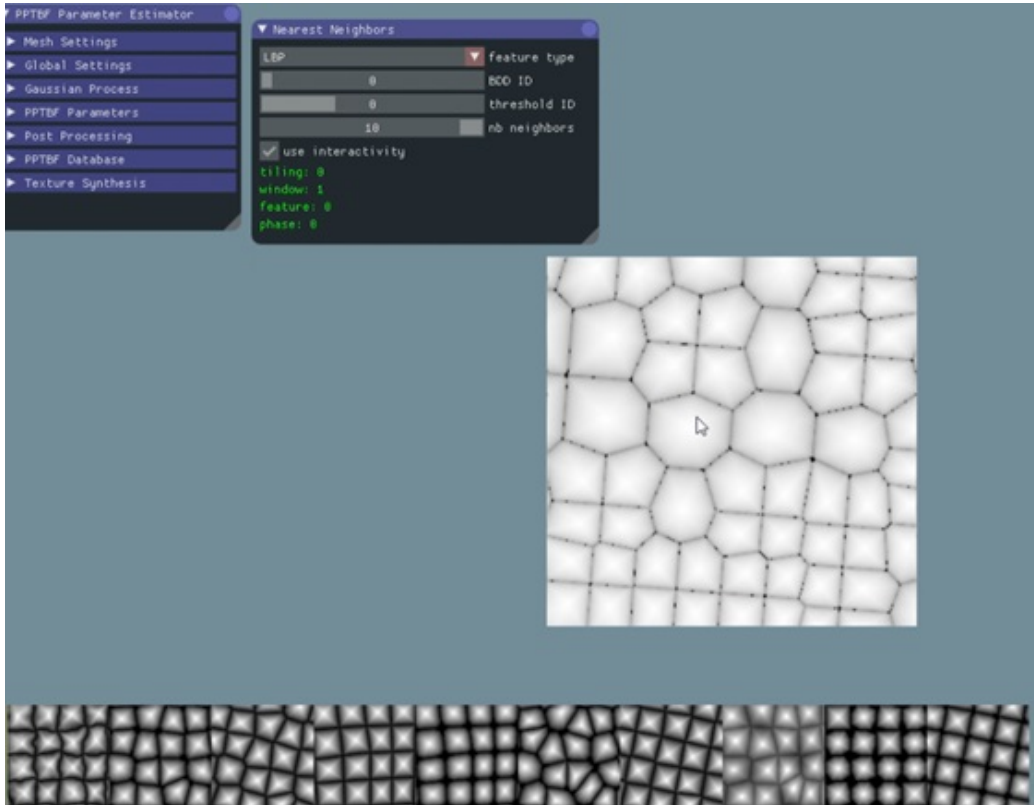
Les descripteurs précédents sont ensuite utilisés pour interroger la base de données et effectuer une recherche de plus proches voisins avec une norme  $L^2$  et la bibliothèque FLANN ([ML09] *Fast approximate nearest neighbors with automatic algorithm configuration*, de Muja et Lowe). La requête fournit une liste de candidats potentiels quasi instantanément. Mais après plusieurs expérimentations, nous avons constaté que la recherche automatique n’est pas satisfaisante. Il n’y a pas réellement de meilleur descripteur qui semble se distinguer. Les résultats dépendent de la forme de l’exemple de structure en

entrée (voir 10.7). Néanmoins, le LBP multi-échelle semble assez intéressant sur un large panel de textures structurées. Les FFT et Gabor semblent mieux capturer les formes comme les tâches éparses et étendues avec des déformations. Les réseaux de neurones nous semblaient les moins précis et moins robustes aux variations, ceci étant peut être lié au fait que les images de structures sont binaires et non pas des images d'intensité, sur lesquelles les réseaux ont été entraînés.



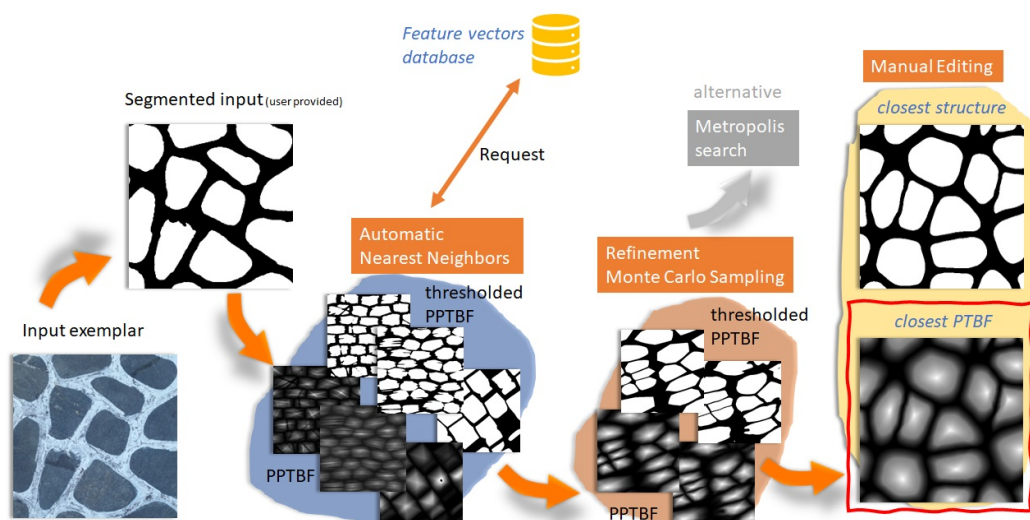
**FIGURE 10.7** – Recherche de plus proches voisins non optimale. Les résultats de requêtes de plus proches voisins sont parfois éloignés de l'image en entrée. Les descripteurs ont du mal à capturer et discriminer nos structures PPTBF. Exemple d'expérimentation sur notre base de structures, recherchant les plus proches candidats par type de descripteurs, pour chacune des images de notre base de données.

Nous avons également développé et expérimenté des outils temps réel pour analyser la robustesse des descripteurs et les métriques aux transformations spatiales (rotation, zoom, translation), voir figure 10.8.

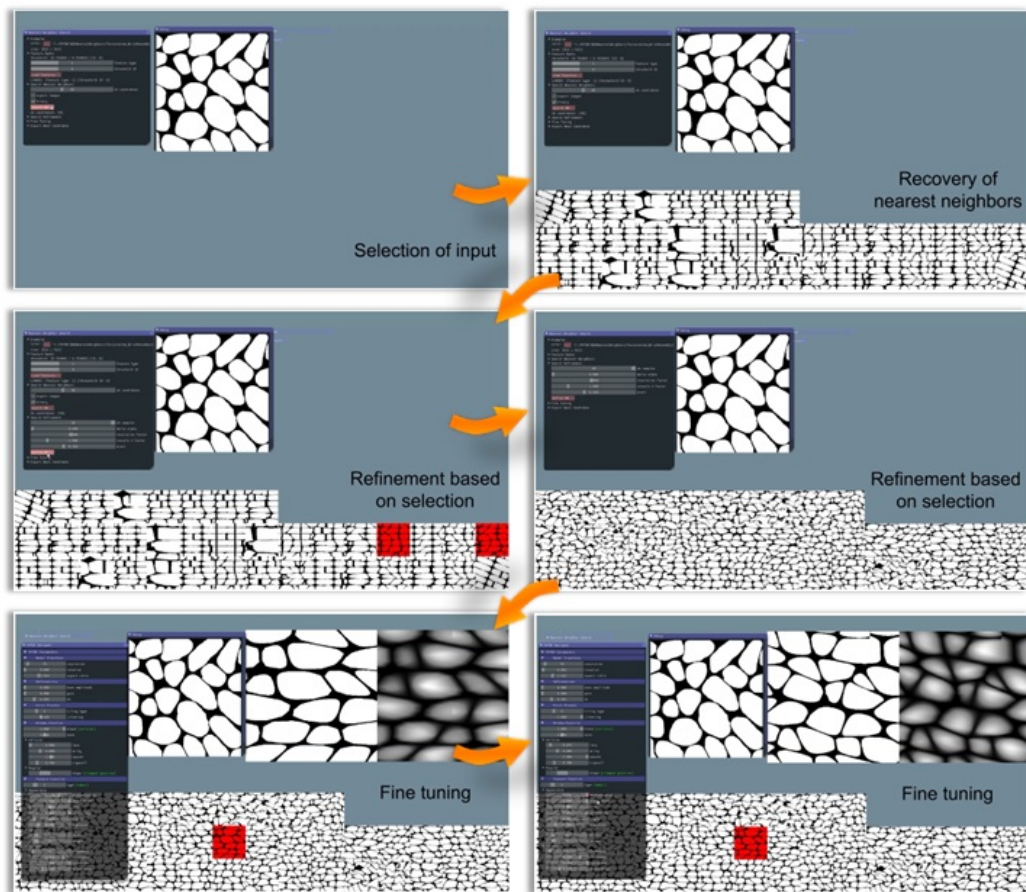


**FIGURE 10.8** – Recherche temps réel de plus proches voisins. Exemple d’une de nos applications, développée pour tester la recherche des plus proches voisins. Ici, affichage temps réel des 10 plus proches candidats, en appliquant des transformations spatiales comme des zooms, translations et rotations, pour tester la robustesse des descripteurs.

Si les requêtes de plus proches voisins avec FLANN fournissent des candidats potentiels très rapidement, mais non optimaux, cela constitue une initialisation intéressante pour un algorithme de recherche, qui nécessite alors une étape de raffinement. Pour cela, nous générons plus de résultats par échantillonnage de Monte Carlo, qui sont ensuite affinés de manière itérative à l'aide d'une recherche automatique de Metropolis (figure 10.9), ou d'une sélection visuelle interactive par l'utilisateur. Il s'avère que pour toutes nos expériences, l'approche interactive semi-automatique, orientée utilisateur, fournissait les meilleurs résultats. L'utilisateur bénéficie de la recherche stochastique avancée de la machine, et conserve le contrôle, soit pour réorienter l'algorithme vers des solutions plus proches visuellement, soit pour éditer interactivement lui-même le modèle PPTBF, lui permettant d'itérer et de raffiner à volonté (figure 10.10).



**FIGURE 10.9** – Recherche automatisée (échantillonnage stochastique et raffinement).



**FIGURE 10.10** – Estimation interactive des paramètres des structures procédurales *PPTBF*. Exemple d'une de nos applications, développée pour tester la recherche des plus proches voisins : recherche initiale, raffinement itératif basée sur la sélection de l'utilisateur, et fine-tuning via la GUI éditeur des paramètres de la PPTBF en temps réel.



## 10.5 Espace de structures

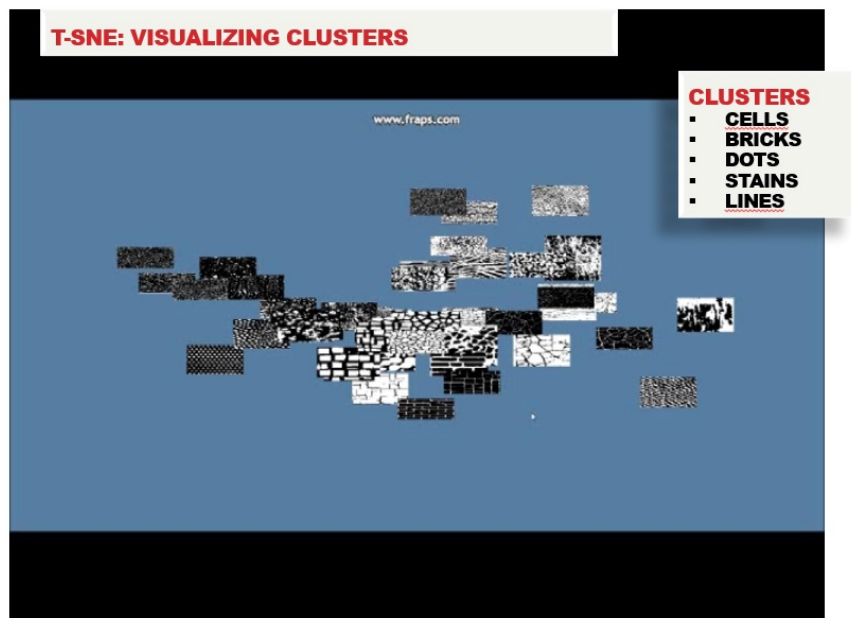
Nous avons également effectué diverses expérimentations sur des techniques de visualisation et de navigation interactives, voire temps réel, dans des espaces de structures et d'apparences de notre modèle PPTBF. Notamment, deux techniques de réduction non linéaire de la dimension :

- t-SNE (t-distributed stochastic neighbor embedding) ;
- les processus gaussiens, par régression (GP) et espaces latents (GPLVM).

### 10.5.1 t-SNE : Visualisation de l'espace de structure

L'algorithme t-SNE (t-distributed stochastic neighbor embedding), ([VdMH08] *Visualizing data using t-SNE* , de Van der Maaten et Hinton), est une technique de réduction non linéaire de la dimension pour la visualisation de données. Nous avons développé des outils 2D et 3D pour analyser la performance de nos descripteurs et la métrique (figure 10.11).





**FIGURE 10.11** – t-SNE : Visualisation 3D temps réel de l'Espace de Structure. Exemple d'application développée pour tester la visualisation 3D temps réel de l'espace de structure et voir si les descripteurs créent des clusters par similarité visuelle. La 3ème dimension donne des informations supplémentaires comme l'organisation intra-clusters.

## 10.5.2 GPLVM : Navigation dans un espace de structure

On s'inspire des travaux de [ZFWW18] (*Gaussian material synthesis* de Zsolnai-Fehér *et al.*), que l'on peut classer dans l'approche Creative AI, pour améliorer la créativité et la productivité des artistes. Ici, l'idée est d'utiliser les espaces latents des processus gaussiens, GPLVM (Gaussian Process Latent Variable Model), pour projeter les candidats PPTBF les plus similaires à une image de structure binaire en entrée, issus de la recherche de plus proches voisins par la librairie FLANN et un descripteur donné, dans un espace navigable de structures (figure 10.12).

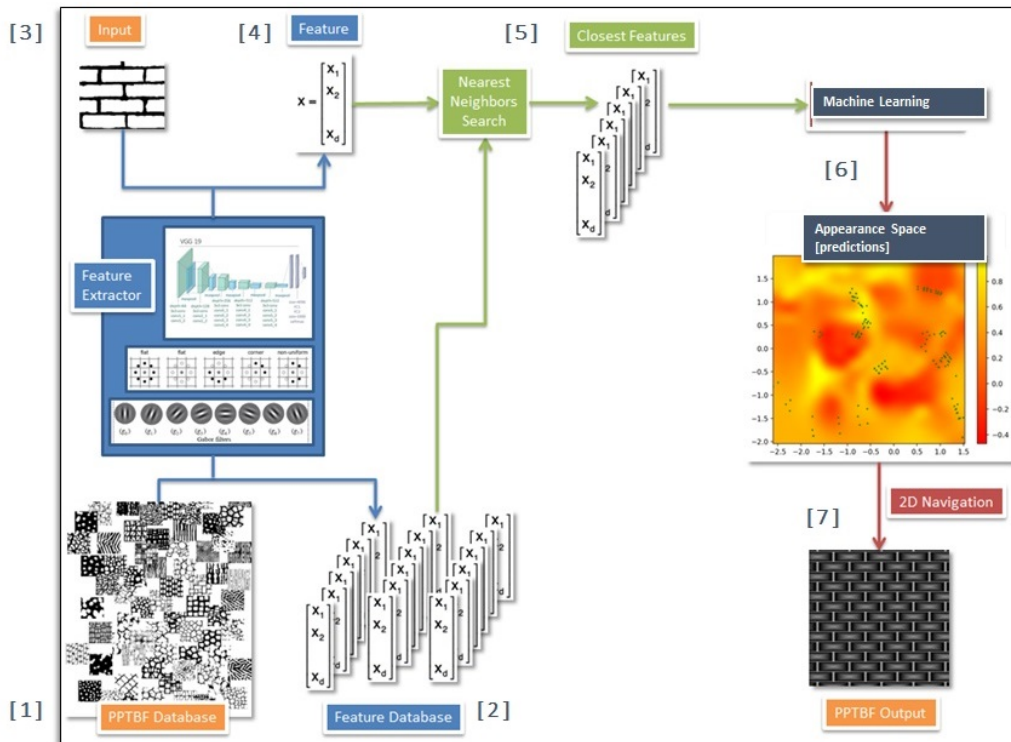
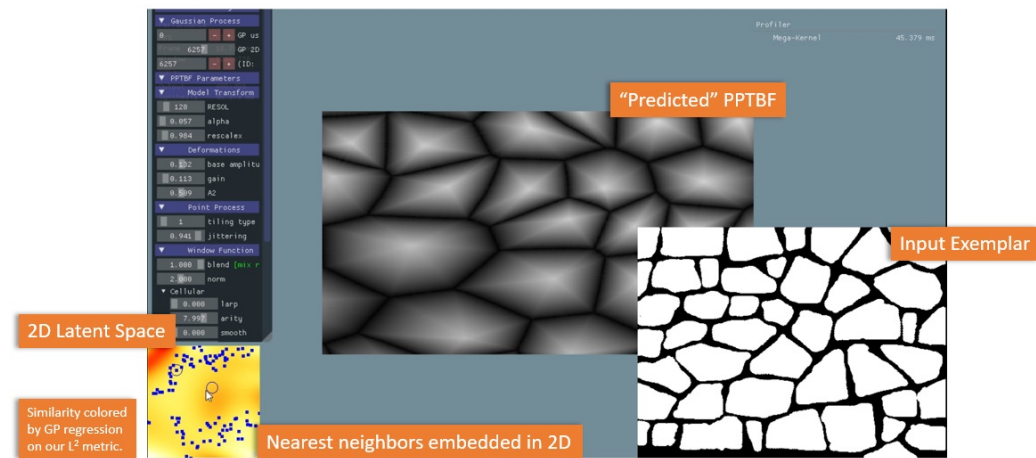


FIGURE 10.12 – Gaussian Process : espace latent de structures

Ensuite, le modèle GPLVM, associé aux régressions GPR (gaussian process regression), est capable de faire des prédictions de nouvelles PPTBF (leurs paramètres procéduraux) lorsque l'on navigue entre les candidats ini-

tiaux. La synthèse est temps réel, ce qui permet à l'utilisateur d'avoir un retour visuel immédiat, de découvrir de nouvelles formes de PPTBF, et de raffiner ses réglages (figure 10.13). Cette approche permet de remplacer l'édition des 30 paramètres de la PPTBF, par une navigation dans un espace 2D.



**FIGURE 10.13** – Gaussian Process : navigation interactive et espace latent. Présentation d’une image de structure en entrée, détermination des plus proches candidats et navigation interactive dans un espace latent 2D, grâce aux prédictions du modèle GPLVM.

## 10.6 Conclusion

Nous avons montré une approche automatique par raffinement stochastique (Metropolis) et semi-automatique pour estimer les paramètres de structures procédurales PPTBF par l’exemple. Il ne s’agit pas d’une contribution à proprement parler, mais plutôt de la mise en oeuvre d’une méthodologie. C’est l’approche centrée sur l’utilisateur, semi-automatique, qui apportait les meilleurs résultats, en bénéficiant d’algorithmes de recherche avancés, tout en donnant à l’utilisateur la main pour éditer et contrôler la structure procédurale souhaitée.

Beaucoup de questions restent ouverte quant à l’estimation de paramètres optimaux, laissant ainsi d’intéressantes perspectives pour des travaux futurs,

comme la création d'un descripteur plus efficace et mieux spécialisé dans la discrimination des structures binaires.

## (iii) SYNTHÈSE DE TEXTURES ET DE MATÉRIAUX

# Chapitre 11

## Synthèse de Textures et de Matériaux *Semi-Procéduraux*

### 11.1 Ajout de détails par l'exemple

#### 11.1.1 Objectif

Les chapitres précédents ont montré que la PPTBF pouvait être utilisée pour générer de manière procédurale la structure d'une texture. À présent, notre objectif est de synthétiser une texture non homogène complète, la structure étant fournie par la PPTBF et les détails de couleurs par un exemple de texture segmentée fourni en entrée. Il s'agit donc de transférer de manière cohérente les détails de couleur de l'exemple d'entrée vers la structure générée par la PPTBF. Cela consiste à appliquer une synthèse de texture contrainte par l'exemple selon une approche de type *Texture-By-Numbers*. Notre approche de synthèse est basée sur une optimisation classique qui consiste à minimiser une énergie composée de deux termes : l'énergie standard de différence quadratique moyenne correspondant à un terme de fidélité aux données, et un terme de régularisation qui va appliquer une contrainte sous forme de pénalité, afin d'équilibrer les détails locaux avec la contrainte donnée :

$$\operatorname{argmin}_{\pi_i} \left\{ \sum_{\pi_i} \|E(\pi_i) - S(\pi_i)\|^2 + \chi \|\bar{E}(\pi_i) - \bar{S}(\pi_i)\|^2 \right\},$$

où  $E$  est l'exemple,  $S$  la texture synthétisée et  $\pi_i$  un ensemble de patches qui se recouvrent.  $\chi$  est une constante qui contrôle la contrainte.  $\bar{\cdot}$  représente

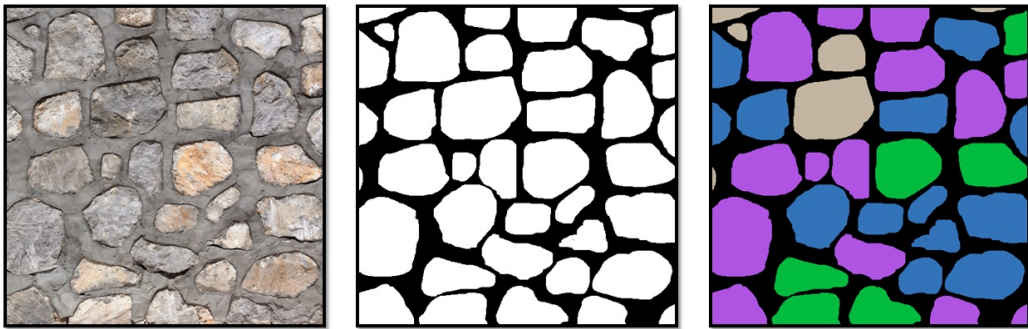


l'information extérieure, c'est-à-dire une carte de guidage ou carte de labels.

Dans notre contexte semi-procédural, une telle approche soulève plusieurs problèmes :

- 1) En échantillonnant la PPTBF, sa nature procédurale infinie est perdue.
- 2) Une structure binaire peut ne pas être suffisante pour capturer certains détails structurels subtils qu'une image de texture peut contenir : plusieurs labels sont généralement utilisés dans les cartes de labels. Les labels permettent de classer et d'identifier les sous-textures (figure 11.1).

- 3) Le mélange de deux énergies avec une contrainte constante peut générer des incohérences visuelles, en particulier au niveau des régions de transition entre les sous-textures. La raison en est que le PPTBF génère de nouvelles variantes qui peuvent ne pas être présentes dans les exemples en entrée.



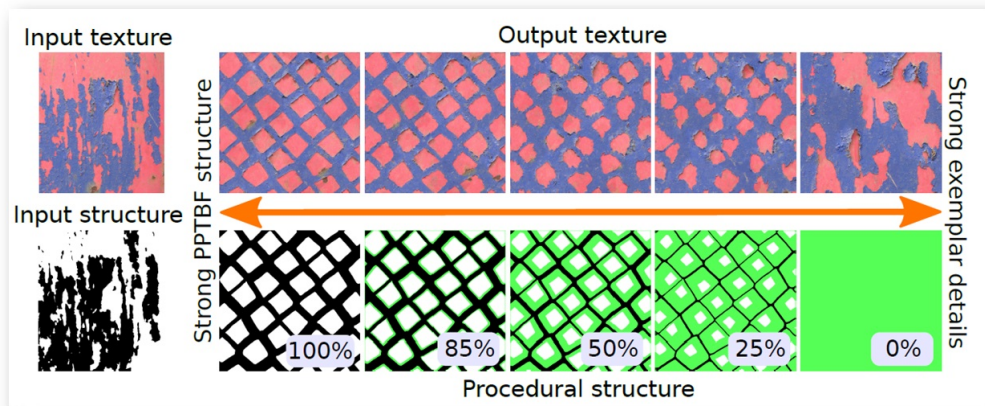
**FIGURE 11.1** – La structure binaire de la PPTBF peut ne pas capturer certains détails d'une texture. Contrairement à une structure binaire (au milieu), les cartes de labels (à droite) permettent de classer et d'identifier les sous-modèles avec plusieurs labels (par exemple, des pierres de différentes apparences claires et foncées).

### 11.1.2 Solution proposée

Notre solution est basée sur un opérateur approprié  $\bar{\tau}$ , à appliquer à la fois à l'exemple d'entrée et à notre PPTBF. En pratique, cet opérateur convertit la PPTBF en une carte de labels. Nous exposons nos choix ci-après.

## Relâchement des contraintes sur les structures

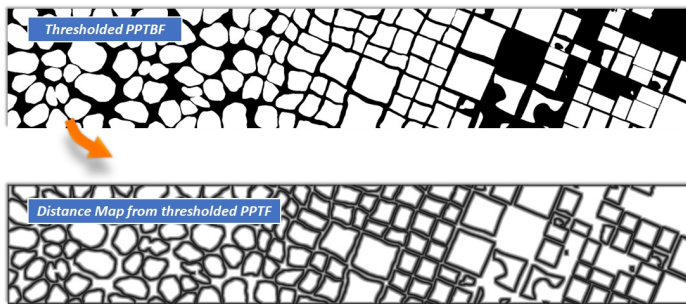
L'utilisateur peut ne pas vouloir que la texture synthétisée corresponde exactement à la structure PPTBF partout. Par conséquent, nous avons mis en place un mécanisme permettant de relâcher les contraintes sur les structures. Pour cela, nous définissons une plage de valeurs  $[\tau - \epsilon; \tau + \epsilon]$ ,  $\tau$  étant le seuil utilisé pour définir la structure binaire, tel que la contrainte  $\chi$  tombe à zéro lorsque la PPTBF a une valeur dans cette plage. Puisque le seuil  $\tau$  définit les contours des structures PPTBF, cela permet de relâcher la contrainte de la PPTBF autour des bords lors de la synthèse (notamment pour de faibles valeurs de  $\epsilon$ ). À l'extrême, pour des grandes valeurs de  $\epsilon$ , la contrainte  $\chi$  peut être nulle partout : dans ce cas, seule une synthèse de texture par l'exemple standard est appliquée. La figure 11.2 illustre un exemple :  $\epsilon$  est progressivement augmenté de gauche à droite.



**FIGURE 11.2** – Relâchement des contraintes sur les structures. La synthèse de textures semi-procédurale combine l'utilisation d'une PPTBF avec un transfert de texture. La PPTBF est utilisée pour contraindre la synthèse. L'utilisateur contrôle la synthèse en définissant une fonction  $\chi$ . Les régions vertes autour des features correspondent aux valeurs 0 de  $\chi$ . Cela relâche la contrainte sur la PPTBF pour préserver les caractéristiques structurelles originales de l'entrée dans ces régions.

## Génération de cartes de labels

**a. Génération automatique de cartes de label à la volée pour la synthèse.** Nous générons un champ de distance à partir des images de la structure binaire pour garantir une synthèse cohérente sur les frontières des entités : il crée des transitions douces sur les frontières (figure 11.3). Le code est exécuté à la volée sur GPU selon l’approche [RT06], *Jump flooding in GPU with applications to Voronoi diagram and distance transform*, de Rong et Tan.



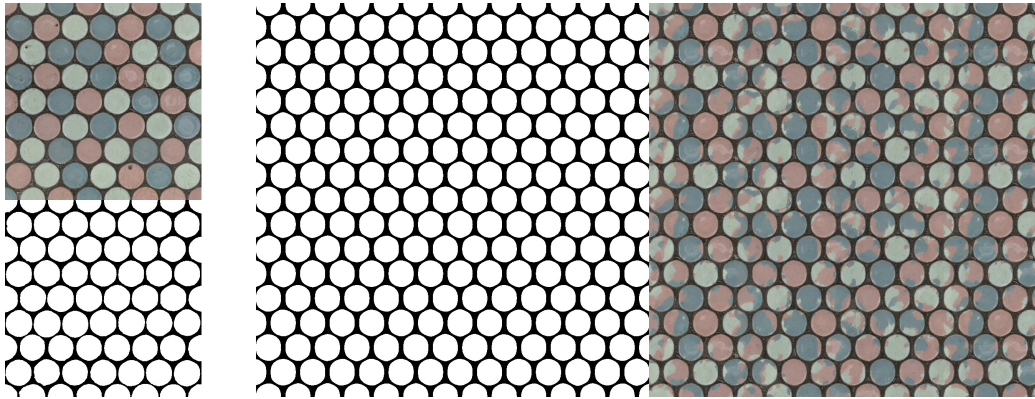
**FIGURE 11.3** – Meilleure prise en compte des bordures à l’aide de champs de distance. Génération de carte de distance à partir de la PPTBF seuillée afin de mieux guider l’algorithme de synthèse lors du choix des meilleurs candidats.

Mais l’utilisation d’un lissage des bordures peut ne pas être suffisante pour contrôler la synthèse (figure 11.4).

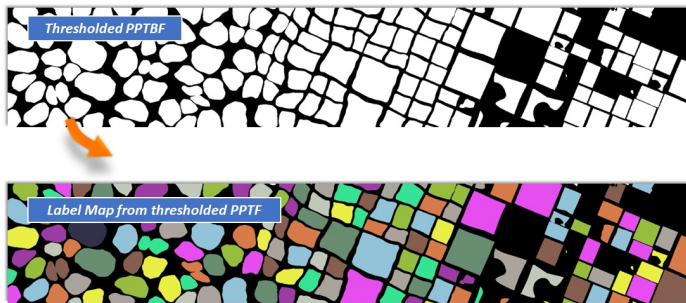
Les cartes de labels sont plus appropriées que les cartes binaires, car elles permettent de distinguer différents types de caractéristiques (*éléments visuels*). Pour en bénéficier, nous avons étendu notre modèle PPTBF, de sorte qu’il génère également un label aléatoire directement aux positions des *feature points*  $\mathbf{x}_i$  de nos cellules  $R_i$ . La figure 11.5 illustre un exemple de génération de carte de labels à partir de la PPTBF : il montre les bordures des éléments visuels lissées dans l’image de la structure binaire recouvertes par des labels affichées en fausses couleurs. La figure 11.6 montre la correction d’une synthèse sans carte de labels (figure 11.4) grâce à l’utilisation d’une carte de labels générée à la volée.

Pour donner plus de contrôle à l’utilisateur, on propose différents modes de génération des cartes de labels (figure 11.7) :

- aléatoire complet ;



**FIGURE 11.4** – Synthèse sans carte de labels. La carte de structure, ne suffit pas à ne pas mélanger les couleurs. Gauche : exemple d’entrée et sa carte de structure binaire. Droite : carte de structure générée (PPTBF seuillée) et synthèse résultante.

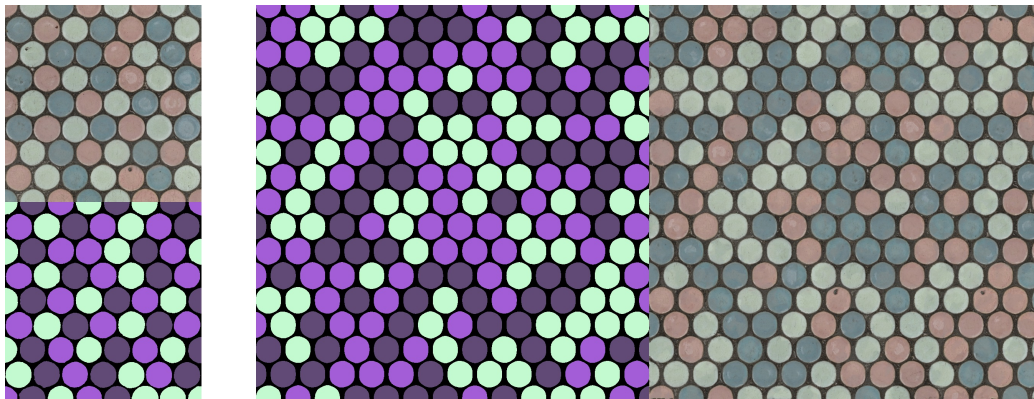


**FIGURE 11.5** – Génération de cartes de labels. Extension de la PPTBF pour générer automatique des cartes de labels. Tirage aléatoire d’un label sur chaque feature point  $\mathbf{x}_i$  des cellules  $R_i$  de la PPTBF.

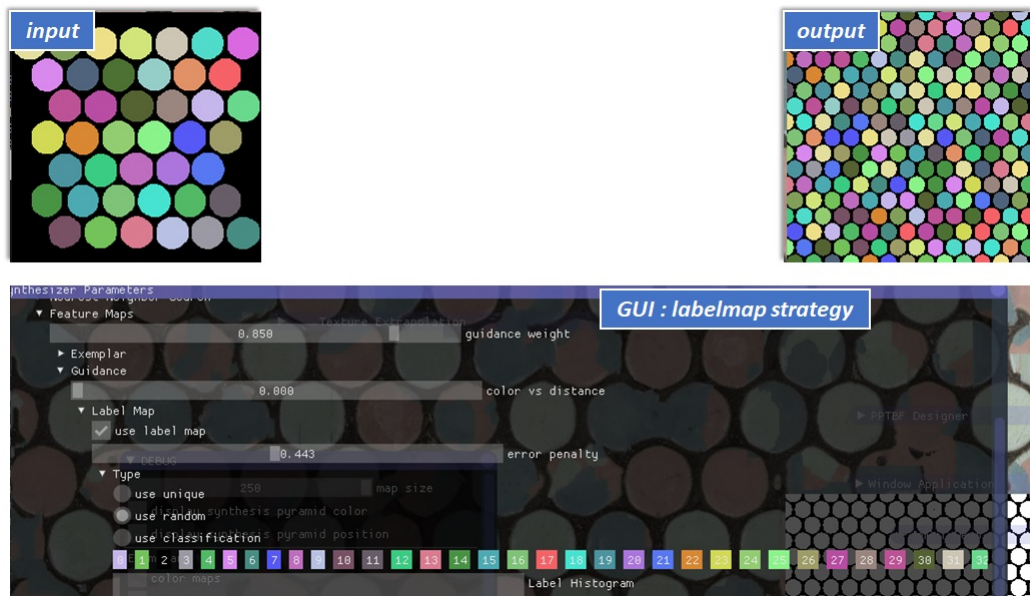
- choix des labels (l’utilisateur dispose d’une liste des labels affichés avec leurs couleurs pour activer et/ou désactiver ceux qu’il souhaite) ;
- préservation des caractéristiques de l’entrée (utilisation d’une LUT, look-up table, pour conserver les statistiques d’ordre 0 de l’entrée, à savoir l’histogramme des labels. L’utilisateur peut modifier l’histogramme des labels en sortie via une interface).

Notons cependant que lorsque le nombre de composantes connexes est trop élevée et qu’à chacun est associé un label propre, cela peut générer des





**FIGURE 11.6** – Synthèse avec carte de labels. Avec plusieurs labels, l’algorithme est capable de ne pas fusionner les couleurs. Gauche : exemple d’entrée et sa carte de label. Droite : carte de labels générée et synthèse résultante.



**FIGURE 11.7** – Différents types de générateurs de labels à la volée. Possibilité de choisir différents types de génération de labels : uniforme, aléatoire, ou par choix de labels dans une liste.

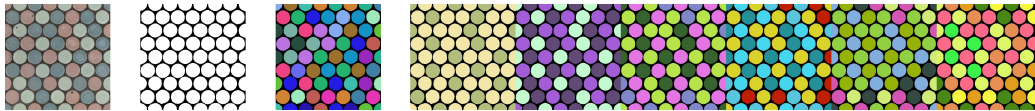
répétitions visibles dans la synthèse.

**b. Génération automatique des labelmaps des images d’entrée.** Lors de la synthèse, l’algorithme par minimisation d’énergie a également besoin des cartes de labels des images d’entrée. Afin d’éviter à l’utilisateur de créer des labelmaps manuellement à partir des images d’entrée, nous proposons plusieurs fonctionnalités pour automatiser ce processus.

Pour les cartes de labels, à au moins 2 labels, on utilise un pré-traitement pour extraire les composantes connexes (CCL : Connected Component Labeling). La classification se base sur les caractéristiques des régions (exemple : statistiques comme les moments, feature vectors, etc.). Ensuite, nous utilisons des méthodes de classification :

- la méthode non supervisée (*Nonnegative Matrix Factorization*) selon l’approche [LSA<sup>+</sup>16] (*Multi-scale Label-map Extraction for Texture Synthesis* de Lockerman *et al.*) pour déterminer un nombre maximum de motifs  $N_m$ , et donc de labels.
- la méthode semi-supervisée *K-means* pour obtenir des classifications intermédiaires comportant 2 à  $N_m$  motifs.

Une fois générées, l’utilisateur peut choisir son type de labelmaps à la demande, lors de la synthèse (figure 11.8).



**FIGURE 11.8** – Cartes de labels calculées à partir d’une texture. De gauche à droite : exemple d’entrée, image de structure binaire, carte de labels avec classification non-supervisée obtenue par NMF, cartes de labels obtenues par  $k$ -means avec  $k$  allant de 2 à 7 labels.

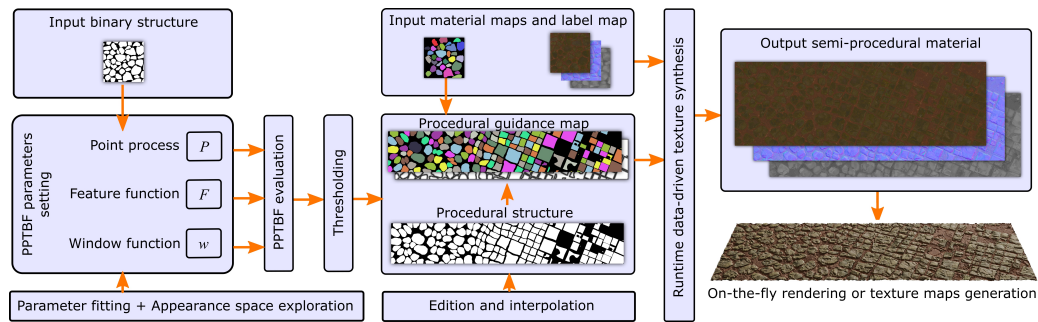
### 11.1.3 Aperçu de la synthèse de textures et de Matériaux semi-procéduraux

#### Chaîne de traitement de la synthèse

La chaîne de traitement complète de notre approche semi-procédurale de synthèse de matériaux est donné par la figure 11.9. La PPTBF est contrôlée



par des paramètres qui peuvent être estimés à partir d’une image entrée, éditée manuellement, ou une exploration interactive d’une base de données et/ou une navigation dans un espace d’apparence latent. La structure est obtenue par seuillage. Elle peut par la suite être éditée et interpolée spatialement avec d’autres structures procédurales. Puis, une synthèse *Texture-by-Numbers* est appliquée afin de générer des textures semi-procédurales.

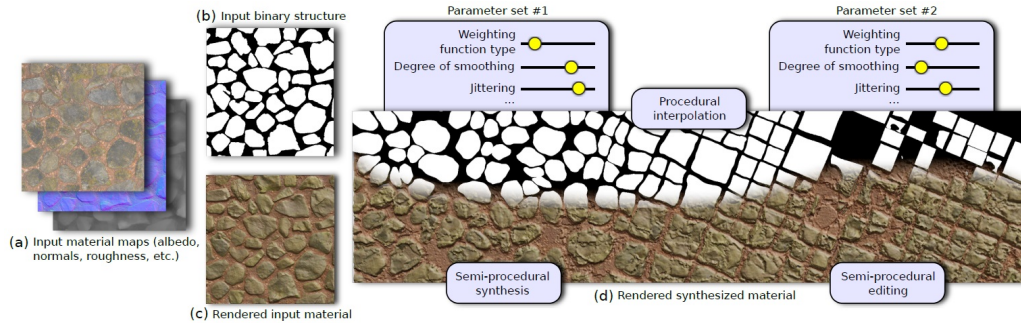


**FIGURE 11.9** – Aperçu de la chaîne de traitement de synthèse de textures et de matériaux semi-procéduraux.

### Exemple d’application

La figure 11.10 illustre un exemple d’application du modèle de textures et matériaux semi-procédural pour la synthèse, l’édition, le contrôle et le passage à l’échelle.

Les données d’entrée, une texture ou plusieurs couches de texture d’un matériau (a) et une structure binaire (b) sont utilisées pour générer une sortie semi-procédurale (d). Cette représentation est hybride dans le sens où la structure est procédurale, infinie et continue (d, haut) et les détails sont discrets et générés à partir de l’exemple (d, bas). Globalement, les textures générées conservent certaines propriétés procédurales : taille infinie, pas de répétition, cohérence et généricité. La structure peut être éditée à l’aide de paramètres (seuls trois d’entre eux sont présentés ici). Le morphing est implicitement obtenu en interpolant ces paramètres. Les détails basés exemple garantissent une bonne correspondance visuelle avec l’exemple d’entrée. Une vue rendue du matériau d’entrée est présentée à des fins de comparaison (c).



**FIGURE 11.10** – Exemple d’application du modèle de textures et matériaux semi-procédural pour la synthèse et l’édit.

### 11.1.4 Algorithme

Pour l’optimisation de l’équation

$$\operatorname{argmin} \left\{ \sum_{\pi_i} \|E(\pi_i) - S(\pi_i)\|^2 + \chi \|\bar{E}(\pi_i) - \bar{S}(\pi_i)\|^2 \right\},$$

nous utilisons une approche parallèle, avec une initialisation par blocs. Nous appliquons également un padding, c’est-à-dire un chevauchement de blocs, pour garantir la continuité sur  $R^2$ . Nous partons d’une résolution grossière. Puis, nous affinons en utilisant des passes d’upscaling et de correction, comme dans [LH05] *Parallel Controllable Texture Synthesis*. Notre passe de correction utilise un algorithme de recherche du plus proche voisin basé sur une marche aléatoire. Il applique une norme  $\ell^2$  sur les texels pour calculer les erreurs par différences quadratiques moyennes sur des patches de taille  $5 \times 5$ . Pour aborder la synthèse des matériaux, nous utilisons plusieurs cartes (albédo, normale, rugosité, hauteur et occlusion ambiante). Ces cartes sont superposées de manière à ce que chaque texel soit composé d’un vecteur de dimension supérieure au lieu de vecteurs RVB uniquement. Les passes de mise à l’échelle et de correction restent inchangées : nous appliquons la norme  $\ell^2$  sur les vecteurs 9-D au lieu des vecteurs 3-D.

## 11.2 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle approche de synthèse de textures et de matériaux, dite *semi-procédurale*, qui évite les inconvénients des textures procédurales et tire parti des avantages de la synthèse de texture de type *data-driven* (par l'exemple). Nous séparons la synthèse en deux étapes : 1) la synthèse de la structure, basée sur un modèle paramétrique procédural, 2) et la synthèse des détails de couleur, étant basées sur les données.

Parce que la PPTBF est basée sur un ensemble unique de paramètres, elle permet des transitions continues entre différentes structures visuelles et un contrôle facilité de ses caractéristiques visuelles.

La couleur est uniformément synthétisée à partir de l'exemple d'entrée à l'aide d'une synthèse de texture parallèle multi-échelle *by-Numbers*, contrainte par la PPTBF.

Les structures générées sont paramétriques, infinies et évitent les répétitions. La synthèse basée sur les données (couleur, paramètres des matériaux) est automatique et garantit une forte ressemblance visuelle avec les entrées. Dans le chapitre suivant, on se propose d'évaluer ce modèle hybride de texture.

# Chapitre 12

## Évaluation du Modèle *Semi-Procédural*

Dans cette partie, on présente une évaluation des résultats de reproduction de textures et de matériaux donnés en entrée. Par la suite, nous comparons nos résultats à des méthodes de l'état de l'art récentes. Enfin, nous présentons une étude des performances.

### 12.1 Introduction

### 12.2 Synthèse de textures

On reprend la même approche et méthodologie qu'au chapitre 9, *Évaluation du Modèle Procédural de Structures* PPTBF, avec la même base de données contenant environ 150 textures et une vingtaine de matériaux avec leurs images de structures.

**Recherche reproductible** De même, on ne présente ici qu'une partie des résultats. L'ensemble des résultats est accessible sur le dépôt *GitHub* associé<sup>1</sup>, dans les suppléments de la publication, sous forme de pages web (HTML), notamment :

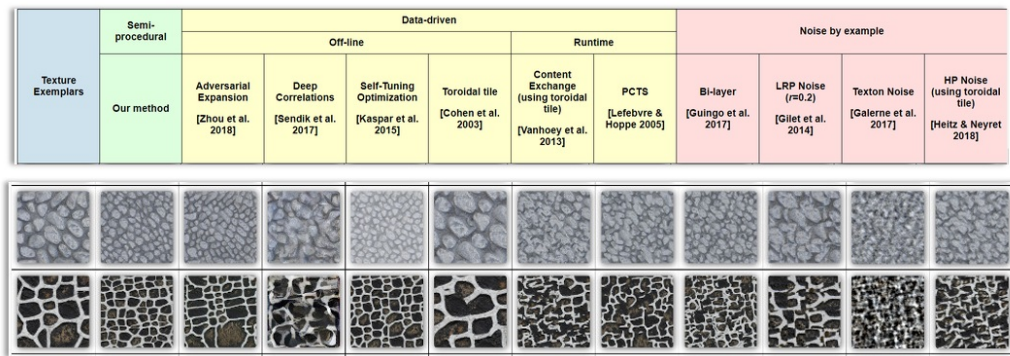
- le "*Supplemental Material 1*" pour une comparaison des méthodes de synthèse de textures ;

---

1. <https://github.com/ASTex-ICube/semiproctex>

- et le "*Supplemental Material 2*" pour des exemples de synthèses de matériaux, ainsi que des exemples de synthèses de textures et de matériaux avec des transitions de structures, dans ses pages "*3. Material synthesis results*" et "*4. Synthesis results with structure transitions*".

Ces données sont également directement accessibles en ligne<sup>2 3</sup>. La figure 12.1 montre un exemple de deux lignes de cette page html.



**FIGURE 12.1** – Comparaisons des méthodes de synthèses de textures. *De gauche à droite : texture d'entrée, notre méthode, méthodes du type data-driven et du type noise-by-example.*

**Aperçu des résultats** Les figures 12.2 et 12.3 présentent quelques exemples de résultats de synthèse de textures.

Notre synthèse semi-procédurale peut être comparée à d'autres méthode de synthèse de texture, à condition que les structures d'entrée correspondent aux exemples d'entrée (typiquement lorsque les exemples ont été segmentés).

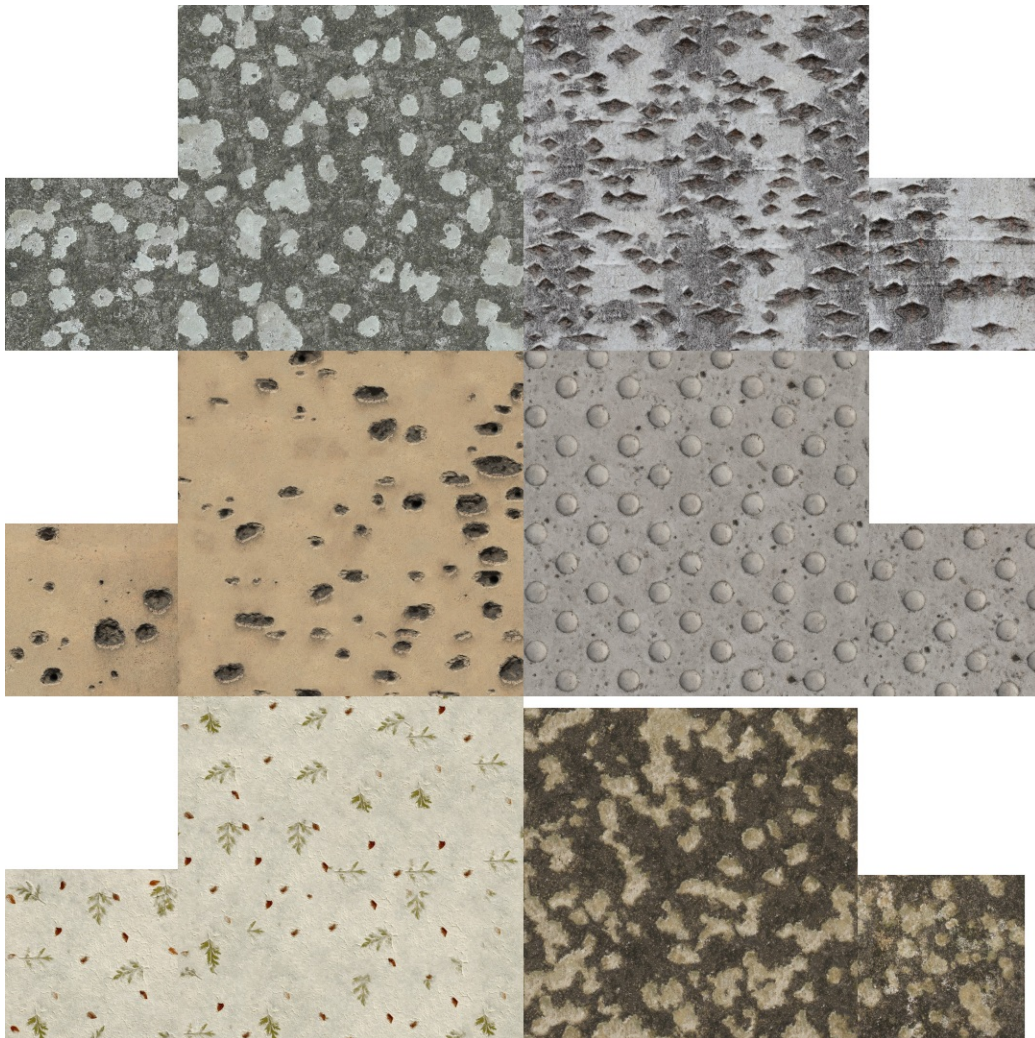
Pour nos comparaisons, nous avons choisi un large ensemble de méthodes (*une dizaine*) en essayant de couvrir plusieurs approches différentes et récentes de l'état de l'art, à savoir :

### Pour les méthodes non procédurales

- [ZZB<sup>+</sup>18], *Non-stationary texture synthesis by adversarial expansion*

2. [http://igg.unistra.fr/People/semiproctex/data/Supplemental1\\_v100.zip](http://igg.unistra.fr/People/semiproctex/data/Supplemental1_v100.zip)  
 3. [http://igg.unistra.fr/People/semiproctex/data/Supplemental2\\_v100.zip](http://igg.unistra.fr/People/semiproctex/data/Supplemental2_v100.zip)





**FIGURE 12.2** – Résultats de synthèses textures. *En petit* : texture d'entrée. *En grand* : texture semi-procédurale synthétisée.

- [SCO17], *Deep Correlations for Texture Synthesis*
- [KNL+15], *Self Tuning Texture Optimization*
- [CSHD03], *Wang tiles for image and texture generation*
- [VSLD13], *On-the-fly multi-scale infinite texturing from example*
- [LH05], *Parallel Controllable Texture Synthesis*



### Pour les méthodes procédurales

- [GSDC17], *Bi-Layer textures : a Model for Synthesis and Deformation of Composite Textures*
- [GSV<sup>+</sup>14], *Local random-phase noise for procedural texturing*
- [GLM17], *Texton noise*
- [HN18], *High-performance by-example noise using a histogram-preserving blending operator*

La figure 12.1 est un extrait d'un très large éventail de textures pour lesquels nous avons fait des comparaisons. Notre observation est la suivante. Ces méthodes se concentrent généralement sur la qualité, mais ne sont pas des approches procédurales génératives : les textures ne peuvent pas être modifiées et éditées en temps réel. Notre approche semi-procédurale transfère la structure de la PPTBF à la texture résultante. Elle génère un contenu varié, c'est-à-dire nouveau et aléatoire, tout en conservant un aspect global cohérent avec l'entrée. Nous sommes capables de mieux traiter certains types de structures régulières, ce qui est lié au paramètre d'aléas global *jitter* (jittering) : lorsqu'il est mis à 0, le caractère aléatoire des positions des points disparaît et la régularité sous-jacente du pavage est garantie, alors que les méthodes par l'exemple souffrent de pouvoir conserver cette cohérence visuelle, y compris pour les méthodes basées sur la recherche de corrélations [SCO17].

Par ailleurs, de manière générale, on constate que les méthodes du type *noise-by-example* ont beaucoup de mal reproduire les structures dans les textures.

Globalement, comparé aux méthodes de synthèse existantes, nous avons constaté :

- une meilleure cohérence de la structure avec notre approche ;
- une bonne correspondance visuelle même lorsque la structure n'est pas parfaitement reproduite ;

Notons par ailleurs que notre synthèse est rapide, pouvant être interactive, voire temps réel, ce qui n'est de loin pas le cas de toutes les méthodes de synthèse.



**FIGURE 12.3** – Résultats de synthèses textures. *En petit* : texture d'entrée. *En grand* : texture semi-procédurale synthétisée.

## 12.3 Synthèse de Matériaux

On présente ici plusieurs exemples de synthèse de matériaux (figure 12.4). Pour chaque matériau, les exemples d'entrée sont en haut et les matériaux semi-procéduraux générés en bas. La colonne de droite affiche les rendus correspondants utilisant les couches d'albédo, de rugosité et de cartes de normales. Les petits encarts présentent des vues rapprochées des rendus pour les matériaux générés.

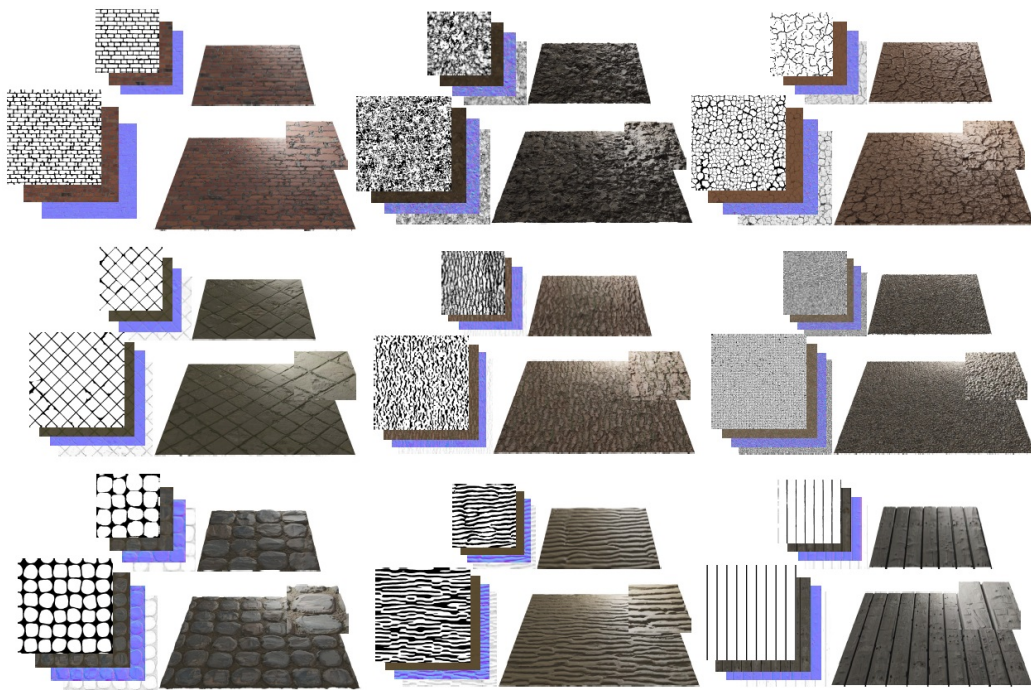
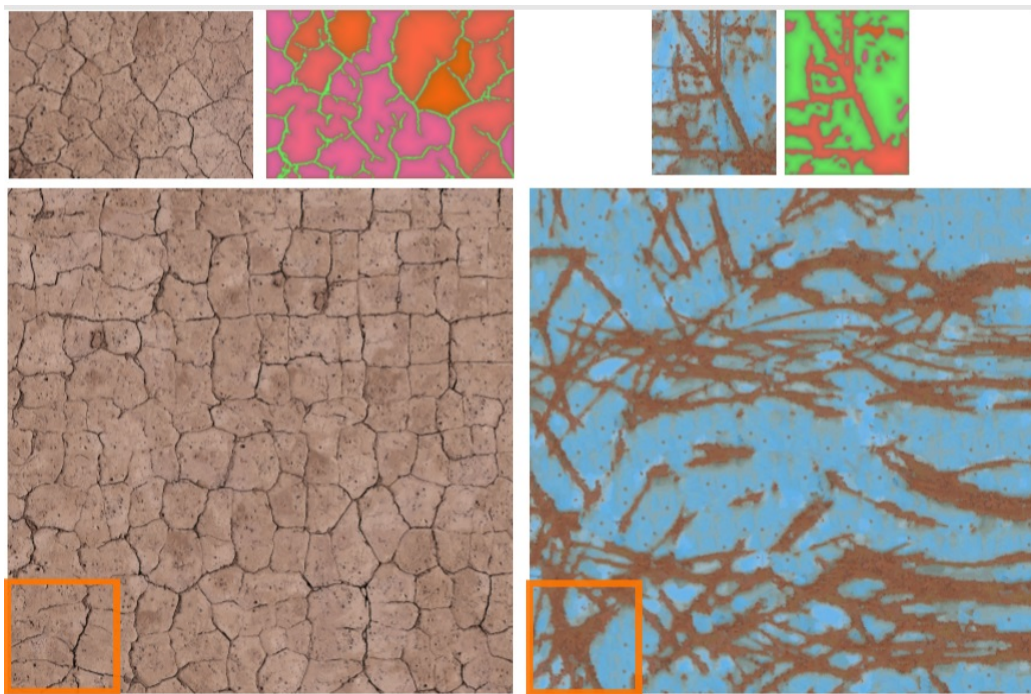


FIGURE 12.4 – Exemples de résultats de synthèse de matériaux











## 12.4 Variations Spatiales de Paramètres

Un des avantages de l'approche procédurale est que ses paramètres peuvent être modifiés, par exemple progressivement. On montre ici un exemple où la structure des textures synthétisée est modifiée progressivement du coin en bas à gauche vers le coin en haut à droite (figure 12.5). Dans les deux cas, la forme et l'arrangement des éléments de structure ont été modifiés à l'aide des paramètres correspondants de la PPTBF.



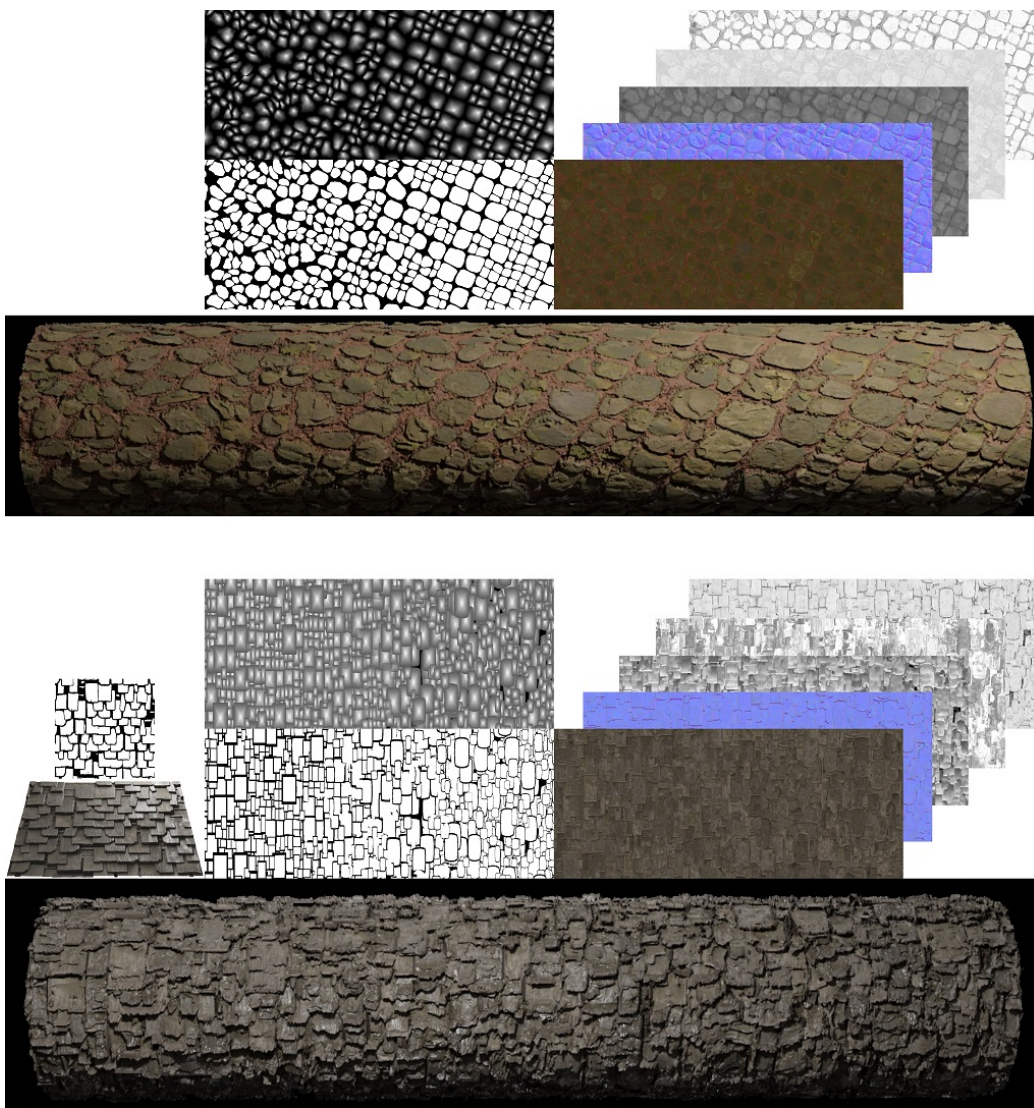
**FIGURE 12.5** – Exemple de variation spatiale de l'aspect visuel des structures. Gauche : modification de la window function afin de modifier la forme des cellules. Droite : modification de l'épaisseur et l'orientation des lignes. La modification est progressive du coin en bas à gauche vers le coin en haut à droite.

La figure 12.6 présente d'autres résultats de synthèse avec des paramètres de PPTBF qui varient spatialement. La structure sous-jacente est également montrée. Enfin, les variations d'aspect peuvent également être appliquées à des matériaux (figures 12.7).

Input texture	Semi-procedural texture synthesis with structure transition
	
	
	
	

**FIGURE 12.6** – Résultats de synthèse semi-procédurale de textures avec des paramètres PPTBF variant spatialement.



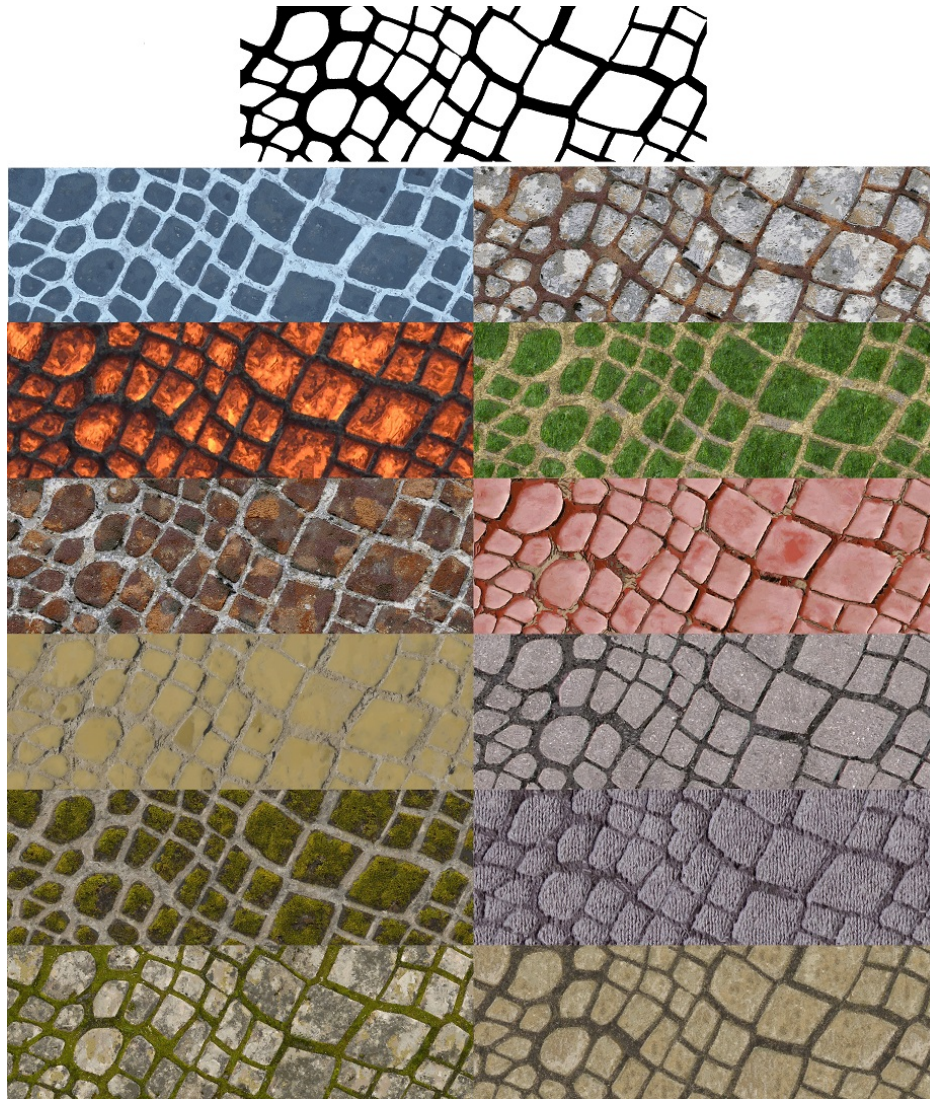


**FIGURE 12.7** – Résultats de synthèses à partir de scans 3D de matériaux. *De gauche à droite : matériau d'entrée, PPTBF synthétisée et seuillée variant spatialement, couches du matériau synthétisé variant spatialement.*



## 12.5 Transfert d'apparence

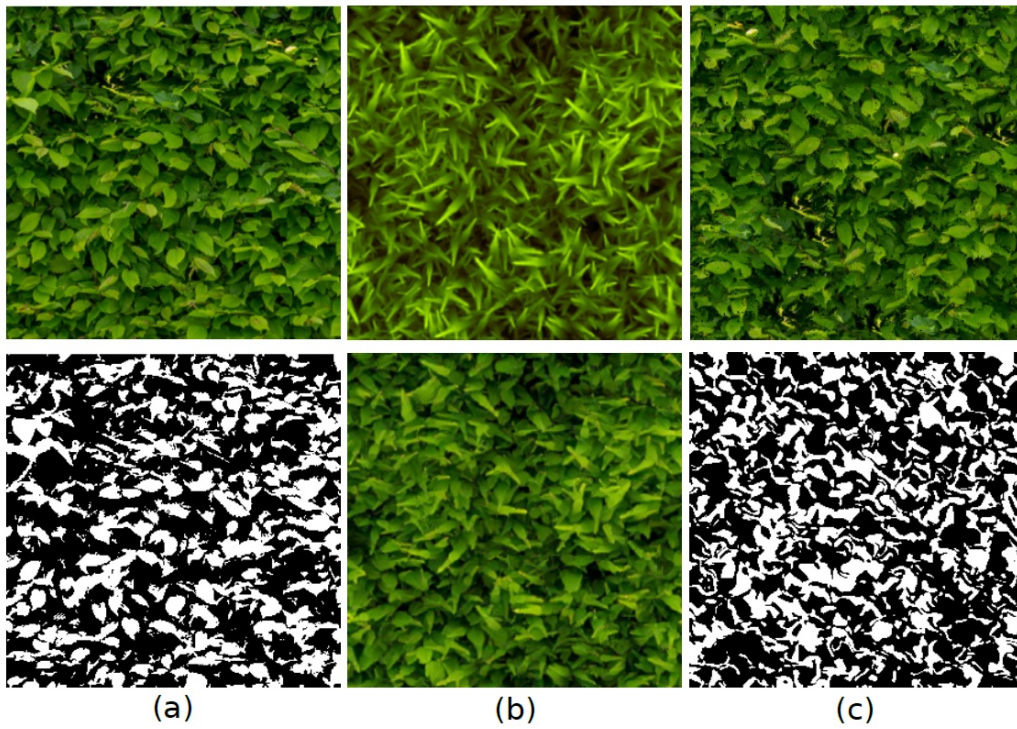
Le fait de découpler la synthèse de la structure de celle des détails permet également de faire aisément du transfert d'apparence : c'est-à-dire que différents détails de couleur sont affectés à la même structure (figure 12.8).



**FIGURE 12.8** – Résultats de synthèses de textures par transferts de matériaux (ici la couleur) (*de notre ancien 1er prototype*).

## 12.6 Comparaisons avec la modélisation de textures procédurales inverse

La modélisation de texture procédurale inverse ([HDR19] *A novel framework for inverse procedural texture modeling*) utilise des bases de données de textures procédurales (*assets*), combinées à une classification et une estimation des paramètres. Malgré des bases de données riches, il n’y a aucune garantie, contrairement aux techniques basées sur les données, qu’une bonne correspondance visuelle ne soit obtenue. Un post-traitement est nécessaire (exemple : transfert de style). Même ce dernier n’est pas en mesure de récupérer tous les détails, car il ne modifie pas les aspects structurels sous-jacents. À l’inverse, notre approche utilise une technique de synthèse par l’exemple : les détails sont transférés à partir de l’exemple d’entrée, ce qui garantit une certaine ressemblance visuelle de détails locaux, même si la structure ne correspond pas parfaitement. De plus, l’utilisateur peut équilibrer la quantité de détails extraits de l’exemple et la quantité de structure conservée de la PPTBF. La figure 12.9 illustre ceci : (a) est l’entrée, (b) est le résultat de [HDR19] (en haut : prédit, en bas : transfert de style), (c) est notre résultat. Bien que de l’aléatoire soit ajouté, les structures locales des feuilles sont mieux préservées.



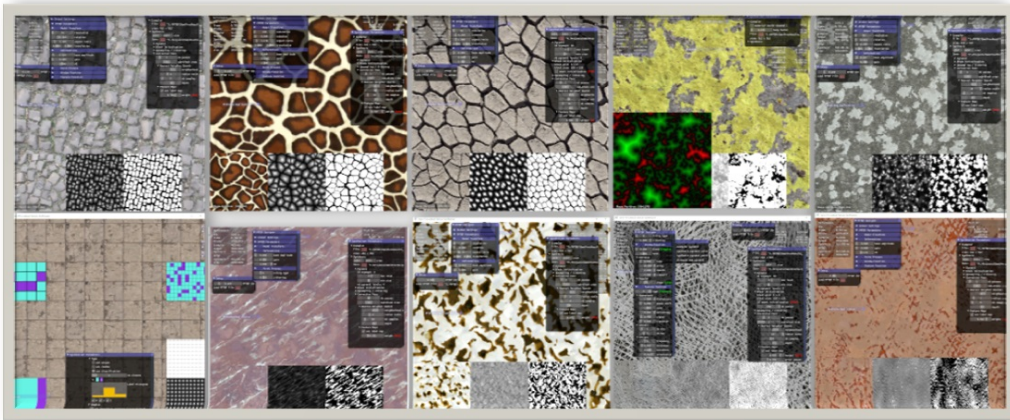
**FIGURE 12.9** – Comparaison avec la modélisation procédurale inverse [HDR19] : (a) est l'entrée, (b) la modélisation procédurale inverse [HDR19] et (c) notre résultat.



## 12.7 Performances

Comme indiqué précédemment (chapitre 9), le modèle de structures procédurales PPTBF peut être éditée de manière interactive, voire temps-réel. La synthèse de textures semi-procédurales, basée sur [LH05] et [BELS10], peut-être interactive, voire temps-réel, mais cela dépend beaucoup du type de structures et de la taille des textures demandées. Notamment, pour des textures très régulières, l’algorithme nécessite une pré-étape d’initialisation (*smart initialization*) plus longue sur CPU (plusieurs secondes), en raison d’une initialisation plus complexe qui n’a pas encore été accélérée sur GPU.

Nous avons développé une version logicielle en C++, et une autre accélérée sur GPU (cartes graphiques) avec la librairie graphique OpenGL des *Compute Shaders GLSL* (figure 12.10) afin de permettre une synthèse interactive. Nous avons utilisé des cartes graphiques GeForce GTX 1060 avec 6 Go de RAM assez anciennes. Malgré cela, nous avons montré dans nos prototypes et démonstrateurs que la synthèse et l’édition peuvent être interactives, voire temps réel.



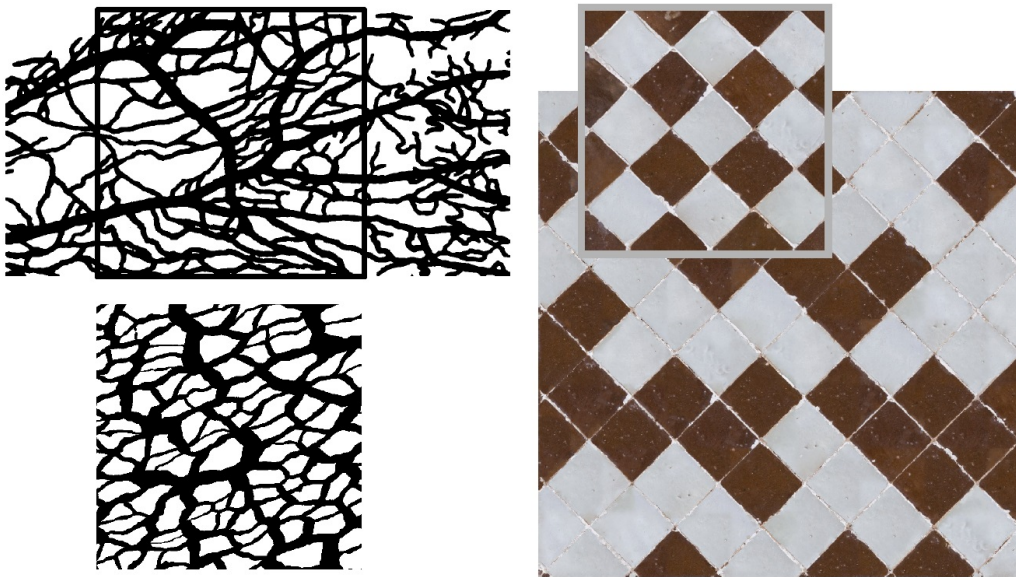
**FIGURE 12.10** – Synthèse semi-procédurale accélérée sur GPU (carte graphique). Copies écran de différents exemples de synthèse avec notre outil interactif, entièrement paramétrable depuis la PPTBF, jusque vers la synthèse.

NOTE : La synthèse de matériaux n’a pas encore été entièrement portée sur GPU, bien que quasiment identique. Nous avons utilisé une version parallélisée sur CPU (multi-threadée avec C++) pour nos résultats.

## 12.8 Discussion et limitations

Notre approche préserve certaines propriétés génératives procédurales, tout en conservant une correspondance visuelle avec un exemple fourni en entrée. Cependant, plusieurs structures naturelles ne sont pas couvertes par notre PPTBF. La figure 12.11 illustre une structure arborescente qui ne peut pas être représentée par les processus ponctuels et les pavages que nous avons utilisés. De telles structures sont généralement modélisées à l'aide de *L-systems* et de grammaires. Par ailleurs, plusieurs labels (affichés à droite) avec des organisations spatiales spécifiques ne peuvent pas non plus être modélisées avec notre PPTBF, car le tirage d'un label est purement aléatoire dans notre cas, sans prise en compte de labels voisins. Enfin, si notre approche semi-procédurale hérite des avantages des techniques *par l'exemple (data-driven)*, elle hérite également de leurs inconvénients : dans certains cas, en raison de la recherche aléatoire des meilleurs pixels voisins, les résultats peuvent devenir bruités. Les détails très basse fréquence (ayant un faible gradient) peuvent également ne pas être bien reproduits.

Remarque : Dans le cas où l'image d'entrée ne contiendrait pas de structure, comme des images de bruit, ou un zoom sur une partie homogène sans véritables contours (exemple : zoom sur un jean), la synthèse se comporte comme la méthode [LH05], *Parallel Controllable Texture Synthesis*.



**FIGURE 12.11** – Notre PPTBF ne couvre pas certains types de structures, telles que les structures de branchement (en haut à gauche : exemple d'entrée ; en bas à gauche : PPTBF estimée après seuillage) ou les textures contenant plusieurs labels avec des arrangements spécifiques (en haut à droite : exemple d'entrée ; en bas à droite : résultat de synthèse semi-procédurale).



# Quatrième partie

## Conclusion

# Chapitre 13

## Bilan

Dans le cadre de ce travail de thèse sur la génération de textures et de matériaux, nous avons proposé :

- un modèle procédural de structures stochastiques ;
- un modèle de synthèse de textures et de matériaux semi-procéduraux.

Nous dressons ici un bilan de ces contributions.

### Modèle procédural de structure

Nous avons mis au point une méthode de synthèse de textures, capable de reproduire de façon automatique ou semi-automatique les structures observées dans des textures stochastiques structurées, à partir de petits échantillons de texture et d'une modélisation procédurale de la structure implémentée sous la forme d'un shader générique sur GPU. Pour cela, nous avons développé une méthode de synthèse de textures dite *semi-procédurale*. Elle consiste à séparer la synthèse de structure, de la synthèse des détails d'apparence (couleur et autres propriétés des matériaux). La structure est procédurale et utilise une nouvelle fonction générique de type bruit par convolution parcimonieuse (*sparse convolution*). Elle est sans limite (infinie), sans répétition, cohérente dans tout le plan 2D, éditable et contrôlable. Les détails sont basés sur des exemples utilisant une approche d'optimisation parallèle automatique engendrant une bonne similitude visuelle (approche *data-driven*). Cette méthode comble ainsi le fossé entre la synthèse procédurale de textures d'une part et la synthèse par l'exemple d'autre part.

Plus précisément, nous avons introduit une nouvelle fonction de base de texture (*texture basis function*) appelée *Point Process Texture Basis Function* (PPTBF), qui étend et d’une certaine manière généralise les modèles de bruit précédents (sparse convolution [Lew84, Lew89, VW91], Gabor [LLDD09, GLLD12], Local Random Phase [GSV<sup>+</sup>14], Phasor [TEZ<sup>+</sup>19], Worley [Wor96] et bombing [SA79]). Cette fonction a été conçue de telle sorte à pouvoir couvrir divers types de structures stochastiques spatiales comme des : cellules, fissures, fibres, rayures, points, tâches, carreaux, vaguelettes et écailles. Elle est construite en utilisant des noyaux de Gabor anisotropes appliquées à des processus ponctuels, générées à partir de pavages récursifs de l’espace.

Contrairement aux approches précédentes qui essayaient de modéliser la structure dans le domaine fréquentiel (Gabor noise [LLDD09], [GLLD12], LRP noise [GSV<sup>+</sup>14]), notre approche s’appuie sur l’observation suivante : les textures semblent contenir un sous-ensemble restreint de motifs visuels distribués spatialement dans l’image comme des points, des lignes et des tâches. Ces structures stochastiques spatiales peuvent être caractérisées par des images binaires dont les éléments ont trois caractéristiques : leur distribution spatiale, leurs interactions mutuelles et leurs formes visuelles locales. Certains motifs se superposent, sont localisés dans des régions de l’espace bien définies, d’autres semblent se mélanger et interagir entre-elles. Des pavages peuvent apparaître comme des cellules et des briques. Enfin, les motifs peuvent être récursifs (fractals) et peuvent contenir des déformations spatiales.

Mathématiquement, ces observations conduisent à représenter la PPTBF par une convolution entre un processus ponctuel  $P$  (somme d’impulsions de Dirac), chargé de la distribution spatiale des éléments visuels, eux-mêmes représentés par une fonction Feature  $F$  vue comme une mixture de noyaux de Gabor anisotropes, contraints ou non spatialement par une fonction de fenêtrage  $W$ .

Concernant le processus ponctuel  $P$ , on se donne la possibilité de gérer des distributions de points non-uniformes de manière procédurale qui sont de type : Poisson (uniformité : complètement aléatoire), Cox (non-uniformité : agrégats de points), Gibbs (interaction entre points voisins), jusqu’à quasi-régulière et régulière (via un paramètre aléatoire pour lequel une faible valeur

modélise des distributions quasi-régulière).

Le fait de moduler l'enveloppe du signal de la mixture des noyaux de la Feature fonction  $F$  par une fenêtre  $W$  dédiée, donne un contrôle supplémentaire sur le type et la forme des structures générées.

Concernant la Feature fonction  $F$ , le contrôle supplémentaire sur l'anisotropie, l'épaisseur et de courbures des bandes permettent une modélisation plus efficace de caractéristiques anisotropes comme des fibres et des rayures.

Plus de réalisme et de contrôle sur les formes des structures sont obtenus en ajoutant des déformations spatiales. Par ailleurs, le modèle donne un contrôle supplémentaire sur l'organisation structurelle de plus haut niveau en contrôlant les corrélations entre les noyaux de la Feature fonction  $F$  et sa fenêtre (tuile sous-jacente).

Nous avons montré que ce modèle permettait de représenter une large variété d'apparences de structures. Enfin, le modèle permet d'interpoler les structures et créer des variations spatiales.

## **Modélisation Procédurale Inverse**

La PPTBF peut être générée automatiquement en estimant ses paramètres à partir d'images exemples de structures par la minimisation d'une mesure de similarité. Cette partie n'est pas encore concluante. L'estimation consiste à interroger une base de données de modèles existants afin de récupérer le plus proche visuellement. Nous avons généré une base de données d'images de PPTBF (images + paramètres) en échantillonnant l'espace des paramètres. et utilisé une "métrique de similarité perceptuelle" en travaillant sur des descripteurs [LCF<sup>+</sup>19] de nos structures (PPTBF seuillées), comme LBP, Gabor, FFT, réseaux de neurone, etc. Aucun n'est assez performant pour donner immédiatement un résultat automatique satisfaisant. Nous avons alors opté pour un raffinement automatique par des méthodes de Metropolis (MCMC), ainsi qu'une interface interactive centrée sur l'utilisateur.

Nous avons également mis en oeuvre des méthodes, dont le but est de fournir des outils d'intelligence artificielle (machine learning, deep learning) permettant aux artistes d'accroître leur créativité et leur productivité. Nous

fournissons un mécanisme permettant de naviguer en temps réel dans un espace de structures, ce qui permet à l’artiste de ne pas avoir à manipuler une trentaine de paramètres mais plutôt de se déplacer dans une carte 2D. Cette approche utilise des processus gaussiens (par espace latent [GPLVM] et régression) inspiré de [ZFWW18].

Nous avons comparé notre approche à celles de l’état de l’art, notamment celles basées sur du bruit ([GLLD12] *Gabor noise by example* et [GSV<sup>+</sup>14] *Local random-phase noise for procedural texturing*), et montré que ces dernières n’arrivent que très difficilement à reproduire les structures de nos images. À l’inverse, notre modèle PPTBF est capable de reproduire de nombreuses structures naturelles, comme celles de notre base de données de 150 exemples de textures que nous avons segmentées.

### **Synthèse de la couleur ou des propriétés des matériaux**

Nous montrons que lorsque des exemples de textures de couleur sont augmentés d’une carte de distance (ou carte binaire), la PPTBF peut être utilisée en combinaison avec la synthèse de texture par l’exemple pour reproduire fidèlement ces textures. Nous nous sommes basés sur les approches *Texture-By-Numbers* et *Constrained Texture Synthesis* en utilisant la PPTBF comme carte de guidage et contrainte. Nous avons ajouté la génération d’une carte de labels afin de contrôler l’organisation spatiale des éléments (et éviter les mélanges de couleurs à l’intérieur des structures). Notre modèle offre la possibilité de définir des contraintes continues, c’est-à-dire pouvant varier spatialement. Avec ce modèle, l’utilisateur dispose de plusieurs leviers de contrôle pour faire varier n’importe quel paramètre de la PPTBF spatialement. De plus, la structure peut être choisie à la fois comme provenant du modèle procédural et/ou de l’exemple : l’utilisateur contrôle avec un seul paramètre le rapport entre la structure procédurale et la structure de l’exemple.

Notre approche est aisément extensible à la synthèse de matériaux sous forme de texture “multi-couches”. À partir d’un matériau sous formes de textures multi-couches et sa carte de structure segmentée, notre algorithme génère toutes les couches du matériau (couleur, hauteur, rugosité, etc.) par combinaison des contraintes provenant de chacun des canaux de textures.

Pour faciliter la reproductibilité de la recherche<sup>1</sup> et améliorer la valorisation<sup>2</sup>, nous avons fourni :

- une banque de données de textures et de matériaux segmentés en cartes de structures binaires ;
- une banque de données de résultats de synthèse avec des comparaisons à des méthodes de l'état de l'art ;
- un code source disponible pour la communauté ;
- demandé et obtenu le label de reproductibilité *Graphics Replicability Stamp Initiative*.

---

1. <http://www.replicabilitystamp.org/>

2. <https://github.com/ASTex-ICube/semiproctex>



# Chapitre 14

## Perspectives

On liste ici des perspectives de travaux. Nous pensons que la génération de texture semi-procédurale peut être un point de départ pour de nombreuses extensions afin d'améliorer encore la synthèse de texture contrôlable et la création de contenu.

### **Ajout d'algorithmes génératifs plus avancés**

Premièrement, le modèle laisse la possibilité d'être étendu à des algorithmes génératifs plus avancés, tels que des grammaires pour définir des pavages, des distributions de points et des structures plus complexes.

### **Ajout de fonctions de densité de probabilités plus avancées pour les paramètres variables de la PPTBF**

L'utilisation de *feature functions*  $F$  avec des PDFs (*Probability Density Functions*) plus élaborées pour certains paramètres peut être imaginée afin d'étendre davantage la gamme de structures représentables.

### **Extension de la PPTBF en 3D**

Les fonctions de base de texture se laissent naturellement bien étendre à des dimensions supérieures, de sorte que la synthèse de texture de type *solid texturing* et volumétrique (*volumetric textures*) à partir d'exemples 2D pourra être abordée dans des travaux futurs. Une difficulté peut être liée au passage en 3D des pavages, dont les schémas se complexifient.

## Augmentation de détails d’objets 3D

Notre approche pourrait être combinée à une modélisation procédurale d’objets 3D augmentée par des données numérisées, la forme de l’objet étant procédurale et les détails de surface générés *par l’exemple*, ce qui pourrait également s’appliquer à la génération de terrains 3D, qui utilise déjà l’augmentation visuelle procédurale. Mais dans notre cas, nous inversons le procédé en ajoutant des détails numérisés, plutôt que des détails procéduraux.

## Amélioration de l’Estimation des paramètres de la PPTBF

La conception d’une méthode robuste pour estimer les paramètres de PPTBFs à partir d’exemples est un autre sujet intéressant pour de futurs travaux de recherche. Actuellement, nous utilisons une technique basée sur une base de données pré-calculée. Elle a des limites, car elle dépend de la stratégie d’échantillonnage de la PPTBF et des descripteurs utilisés. Une approche basée sur l’apprentissage (*learning-based approach*) pourrait conduire à un appariement plus robuste, par exemple via l’utilisation de processus gaussiens (par *régression* GPR et *espace latent* GPLVM) [ZFWW18], *Gaussian material synthesis*, ou la régression des caractéristiques perceptuelles comme proposé dans [LGD<sup>+</sup>18], *Perception-driven procedural texture generation from examples*.

## Différentiabilité en Modélisation Procédurale Inverse

Notre méthode d’estimation des paramètres de PPTBF à partir d’exemples est coûteuse en temps et en mémoire, et est limitée par l’échantillonnage des paramètres. Une piste de recherche pourrait consister à mettre au point une PPTBF *différentiable* pour permettre une approche *basée gradient* de l’estimation des paramètres. En lien avec cette piste de recherche, on peut citer plusieurs travaux récents dans le domaine de la modélisation procédurale inverse de matériaux, comme *MATch* [SLH<sup>+</sup>20], ou encore *DiffProxy* [HGH<sup>+</sup>22].

Il serait également intéressant de conserver le plus possible de nos paramètres comme des réels. Nous avons choisi des entiers pour faciliter l’édition de la PPTBF, mais, en interne, la plupart sont des réels. Seul reste le modèle de mixture des noyaux, le type de profil des fenêtres et le type de pavage (*tiling*). Il pourrait être intéressant de regarder si l’utilisation des variables internes pour subdiviser ou non les cellules et certains de leurs paramètres

d'aléas ne pourrait pas être contrôlé par un ou deux paramètres réels continus plus globaux, comme : complexité (ou anisotropie) et subdivision (ou récursivité).

## **Extraction automatique de Structures à partir de Textures ou de Matériaux**

Au cours des travaux de cette thèse, nous avons constitué une base de données de structures extraites manuellement à partir de textures et de matériaux, en s'aidant d'outils de traitement d'images. Nous avons constaté que ce processus d'extraction de structures binaires est difficile à automatiser avec des outils classiques, et constitue un problème de recherche en soi qui pourrait s'inscrire dans la continuité de cette thèse. Une approche par Deep Learning pourrait être étudiée, à condition d'augmenter sensiblement la taille de notre base de données de structures, puisque plusieurs milliers d'images sont en général nécessaires pour entraîner ce type de modèle.

## **Génération par Contrôle de Processus Ponctuels**

Dans [LDG21], *Determinantal point processes for image processing*, Lounay *et al.* s'intéressent à la possibilité de générer de la géométrie à partir de processus ponctuels déterminantaux (DDP), et estimer ces composants par inférence à partir d'une image (c.-à-d. la fonction spot et son noyau), notamment à partir des ensembles d'excursion de champs aléatoires. Il y a plusieurs points communs avec notre approche de synthèse, notamment notre modèle de structures procédurales PPTBF. C'est une autre piste pour ajouter des comportements plus complexes à notre PPTBF.

## **Nouveaux Types de Tessellations**

Le modèle PPTBF est suffisamment souple pour s'adapter à différents types de pavages et tessellations. Une piste de recherche possible serait d'étudier et d'incorporer de nouveaux types de tessellations dans notre modèle, pour étendre encore davantage les structures représentables. À titre d'exemple, la cristallographie est un domaine où de nombreux types de tessellations sont

décrits <sup>1 2 3 4</sup>. Les tessellations stochastiques font également l'objet d'études en mécanique statistique (par exemple, [DH14], *Self-similarity, Aboav-Weaire's and Lewis' laws in weighted planar stochastic lattice*, Dayeen et Hassan, figure 14.1).

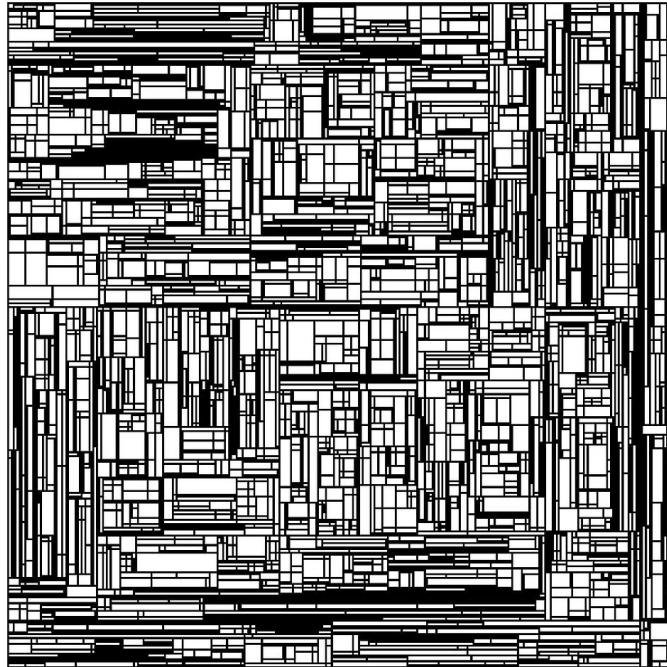


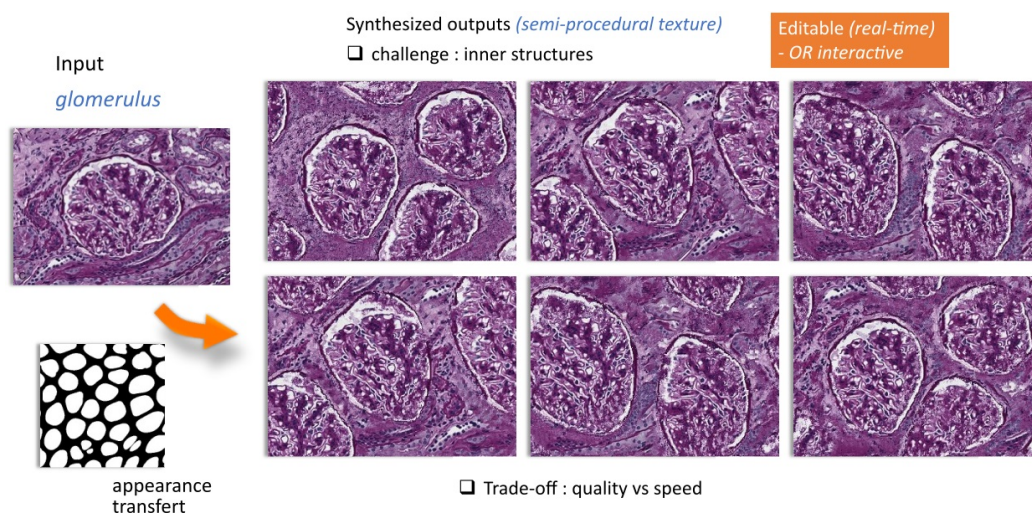
FIGURE 14.1 – Weighted Stochastic Lattice (image extraite de [DH14])

## Gestion de Textures avec Structures Hiérarchiques

Un des problèmes restant à traiter concerne les structures hiérarchiques, c'est-à-dire les structures comportant des motifs imbriqués. Nous montrons ci-dessous un exemple qui a attiré notre attention, dans le cadre de travaux sur l'augmentation de données pour l'analyse d'images histopathologiques rénales par Deep Learning (figure 14.2). Les coupes de tissus biologiques

1. <https://en.wikipedia.org/wiki/Tessellation>
2. <https://en.wikipedia.org/wiki/Mosaic>
3. <https://en.wikipedia.org/wiki/Tile>
4. [https://en.wikipedia.org/wiki/Wang\\_tile](https://en.wikipedia.org/wiki/Wang_tile)

sont des exemples typiques de structures à différents niveaux d'échelle. Il s'agit non seulement d'être capable de représenter ce type de structures, mais également de savoir les extraire à partir d'exemples, ce qui représente une opération qui n'est pas entièrement automatisée. La synthèse de ce type d'images, qui s'apparente à de la texture, pourrait également faire l'objet de travaux futurs.

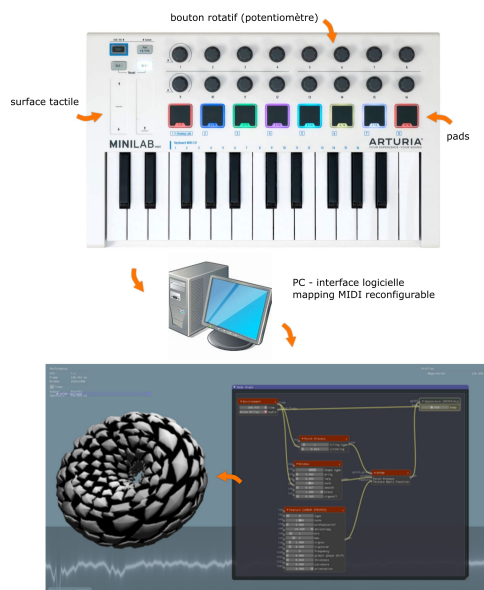


**FIGURE 14.2** – Exemple de génération d'images histopathologiques synthétiques (coupes de tissus rénaux).

## Nouvelles modalités de Contrôle des Paramètres

Enfin, en matière d'interaction homme-machine, une autre piste de recherche pourrait consister à étudier de nouvelles modalités de contrôle des paramètres de la PPTBF. Nous proposons ici une possibilité de contrôle à l'aide de signaux audio, que l'on pourrait appeler *Textura-tronique*, ou *Matéria-tronique*. Il est alors possible de contrôler la structure PPTBF avec un contrôleur MIDI de type clavier maître d'un synthétiseur, possédant de nombreux contrôles de types variés comme : des pads, des touches, des potentiomètres, des surfaces tactiles. Il est très simple d'associer chaque paramètre de PPTBF à n'importe quel élément du contrôleur MIDI. Ceci pourrait être utilisé pour la création de matériaux Substance Designer, en prévisualisation,

pour interagir avec le directeur artistique, ou dans des spectacles multimédia, etc.<sup>5 6</sup>.



**FIGURE 14.3** – Exemple de contrôleur MIDI proposant différentes modalités d’interaction

Les contrôleurs MIDI sont des dispositifs paramétrables. Il est aisé de créer un mapping (entièrement reconfigurable) entre chaque manipulateur et une action vers une texture semi-procédurable. Manipulateurs disponibles : potars (boutons rotatifs), pads, surfaces tactiles, touches de clavier, boutons de contrôle (changement d’octave, etc...). Chaque élément est configurable (vitesse, valeur, min, max...)

Dès lors, il est envisageable de réagir à l’environnement comme la température, l’humidité, l’ensoleillement, etc. pour faire évoluer l’apparence des textures et des matériaux. Cela constituerait un moyen d’avoir à disposition un substitut de l’outil Substance Designer (ou de graphes de textures et matériaux procéduraux) en temps-réel, et d’effectuer des requêtes de génération et modification à la demande, sans avoir à exporter les textures et matériaux à une résolution donnée.

5. <https://fr.wikipedia.org/wiki/Animatronique>

6. [https://fr.wikipedia.org/wiki/Musical\\_Instrument\\_Digital\\_Interface](https://fr.wikipedia.org/wiki/Musical_Instrument_Digital_Interface)



# Cinquième partie

## Annexes

# Annexe A

## Algorithme de la PPTBF

Cet appendice décrit l'algorithme de génération de structures procédurales stochastiques du modèle PPTBF (Point Process Texture Basis Functions).

### A.1 Pseudo-Code

Ci-dessous, le pseudo-code détaillé de la PPTBF :

```

// POINT PROCESS TEXTURE BASIS FUNCTION PSEUDO-CODE

// This pseudo code slightly differs from the actual GPU implementation:
// some parameters being tuned on GPU to improve performance
// PPTBF Parameters provided in the supplementals match the GPU implementation
// that will be made publicly available

float compute_pptbf ( vec2 x, // where to compute PPTBF

    // deformation and normalization parameters
    float zoom, float rotation_angle, float rescalex,
    float ampli[3], // Brownian distorsion parameters

    // point set parameters
    int tile_type,
    float jitter,

    // window function parameters
    float normblend, //  $\omega$ 
    float arity, //  $n_v$  control points for Bezier
    float larp, //  $\lambda$  window anisotropy
    float wsmooth, //  $l_c$  window smoothing
    float norm, float sigw1, float sigw2, //  $||\cdot||$  and  $\sigma_j$ 

    // feature function parameters
    int mixture, // mixture model
    float winfeatcorrel, // correlation with window
    float feataniso, //  $\eta$  anisotropy
    int Jmin, int Jmax,
    float freq, //  $\phi$ 
    float thickness, float curvature, float deltaorient //  $\tau, \kappa, \theta$ 
    float normfeat, //  $||\cdot||_f$ 
    float sigcos, float sigcosvar, //  $\sigma_f$  and  $\sigma_{fvar}$ 
)
{
    // storing tessellation cells and point distributions
    vec2 c[MAX_NEIGH_CELLS]; // cells lower left corner coords
    vec2 d[MAX_NEIGH_CELLS]; // cells size
    vec2 p[MAX_NEIGH_CELLS]; // random points
    int mink[MAX_NEIGH_CELLS];

    //-----
    // [1] Brownian Deformation
    //-----

    x = x + amp[0] * brownnoise(amp[1]*x*zoom, amp[2]) ;

    //-----
    // [2] Model Transform: scale x, rotation and zoom
    //-----

    x = x * mat2(cos(rotation_angle),-sin(rotation_angle),
                sin(rotation_angle),cos(rotation_angle))*zoom;

    //-----
    // [3] Point Process
    //-----

    int npp = genPointSet(x, tile_type, jitter, p, c, d);
    // order according to nth closest: result is in table mink[]
    nthclosest(mink, npp, x, p, c, d, larp, norm);

    //-----
    // [4] PPTBF = PP x ( W F )
    //-----

    float pptbfv = 0.0f; // final value, to be computed and returned

```

```

float priomax = -1.0f; // initial lowest priority for mixture
float minval = -1000.0f; // initial value for max mixture

for (int k = 0; k < npp; k++) // for each tessellation cell
{
    // init PRNG at cell center
    seeding(p[mink[k]]);
    float bezierangularstep = 2.0 * M_PI / arity;
    float bezierstartangle = bezierangularstep * rand();
    int J = Jmin + (int)((float)(Jmax - Jmin)*rand());

        //-----
        // [5] Window Function: W
        //-----

    // window_1: cellular basis window
    float cval = 0.0;
    if (k == 0) // only inside Voronoi cell
    {
        float smoothdist = beziercell(mink[0],x,c,d,p,
                                      bezierangularstep, bezierstartangle);
        float cdist = cellborder(mink[0],x,c,d,p);
        cval = mix(smoothdist,cdist,wsmooth);
    }
    float w1 = normblend * (exp((cv - 1.0)*sigw1) - exp(-1.0*sigw1));
    if (w1<0) w1=0;

    // window_2: overlapping basis window
    float sddno = p-norm(x - p[mink[k]]);
    ; // empirical constant for clamping gaussian, depending on tile type
    float footprint = 1.5
    if (tile_type >= 10) footprint *= 0.4;
    // compute w2
    float w2 = (1.0 - normblend) * exp(-sigw2 * sddno) - exp(-sigw2* footprint)
    if (w2<0) w2=0;

        //-----
        // [7] Feature Function
        //-----

    float feat = 0.0; // feature function value to be computed

    // stringed Gaussian parameters
    float mu[MAX_G], dif[MAX_G];
    float theta[MAX_G], prior[MAX_G], sigb[MAX_G];
    float valb[MAX_G]; // for amplitude

    // init PRNG, decorelated from window seed
    seeding2(p[mink[k]]);
    for (int i = 0; i < J; i++)
    {
        prior[i] = rand();
        valb[i] = rand();
        mu[i] = c[mink[k]] + (0.5+0.5*rand()) * d[mink[k]];
        // shift mu according to correlation
        mu[i] = mix(p[mink[k]],mu[i], winfeatcorrel);
        // orient Gabor stripes
        dif[i] = (x - mu[i]) / d[mink[k]];
        theta[i] = deltaorient * rand();
        sigb[i] = sigcos * (1.0 + sigcosvar*rand());
        // apply rotation, anisotropy and curliness
        vec2 dd = mat2(cos(theta[i]),-sin(theta[i]),
                      sin(theta[i]),cos(theta[i])) * dif[i];
        dd.y /= feataniso;
        float xfeat = sqrt(dd.x * dd.x * curvature * curvature + dd.y * dd.y);
        // compute stringed gaussian value
        float ff = 0.5 + 0.5 * cos(pi * freq * xfeat);
        ff = pow(ff, 1.0 / (0.0001 + thickness)); // avoids division by zero

```

```

float fdist = p-norm(dd,normfeat) / (footprint / sigb[i]);
// apply mixture
switch (mixture) {
  case 1:
    float amp = valb[i] < 0.0 ? -0.25 + 0.75 * valb[i] : 0.25 + 0.75 * valb[i];
    feat += ff * amp * exp(-fdist);
    break;
  case 2:
  case 3:
    feat += ff * exp(-fdist);
    break;
  case 4:
    if (priomax < prior[i] && fdist < 1.0 && ff>0.5)
    {
      priomax = prior[i];
      pptbfvv = 2.0*(ff - 0.5) * exp(-fdist);
    }
    break;
  case 5:
    float ww = ff* exp(-fdist);
    if (minval < ww) { pptbfvv = ww; minval = ww; }
    break;
  default: feat = 1.0;
}
// normalization according to mixture model
if (mixture == 1) feat = 0.5 * feat + 0.5;
if (mixture == 2) feat /= float(J);
if (mixture == 3) feat = 1.0-feat;

// add contribution except for max operators
if (mixture < 4) pptbfvv += (w1 + w2) * feat;
}
return pptbfvv;
}

```

# Annexe B

## Format de Fichier de Structures PPTBF

Cet appendice décrit le format des fichiers PPTBF, qui sert à stocker les paramètres de notre modèle de structures stochastiques procédurales PPTBF (Point Process Texture Basis Functions).

### B.1 Description

```
---- PPTBF file format ----
```

```
A PPTBF file store parameters of our PPTBF model (Point Process Texture  
Basis Functions) of procedural stochastic structures.
```

```
Example:
```

```
0.892735  
5 1 55 0.359 0.850971 0.034 0.872 0.547 1 10 0 2 0 0.812 0.01 0 2 0 5 1 1  
0.855 0.855 1 1 0.0844974 1 0
```

```
---- 1st line ----
```

```
The first line contains the threshold to apply to the PPTBF procedural model  
. This threshold HAS TO be applied after a "histogram_egalization" of  
the PPTBF values to renormalize its values in [0;1]. So, threshold value  
lies in [0;1.0].
```

```
- [0] PPTBF threshold value
```

```
---- 2nd line ----
```



The second line contains the concatenated list of PPTBF procedural parameters. They describe 5 types of PPTBF parameters :

- code PPTBF model: point process, window function and feature function.
- additional parameters: transformations and deformations

NOTE: Below, in "values", you will find some values we used in our GUI ( graphical user interface). Some could be changed, such as minNbKernels and maxNbKernels, if you want more kernels to add in your feature function.

[ Point process ]

Point set parameters:

- [0] tiling type: tiling type (e.g., regular, irregular...) - values: {0, 17}
- [1] jittering: randomness - values: [0, 1]

[ Transformation ]

NOTE: also used for normalization parameters during parameter estimation.

- [2] resolution: uniform scale (zoom/unzoom) - values: {1, 1000}
- [3] rotation: rotation around world coordinate frame at origin (0.0;0.0) - values: [0, 2]
- [4] aspect ratio: shear on X axis (scale) - values: [0.01, 10]

[ Turbulence ]

Brownian distorsion parameters:

- [5] distorsion base amplitude: values: [0, 0.25]
- [6] distorsion amplitude gain: values: [0, 4]
- [7] distorsion frequency: values: [0, 1]

[ Window function ]

Window function parameters:

- [8] window shape: window function type - values: {0, 3}
- [9] window arity: number of control points for smoothing - values: [2, 10]
- [10] window larp: anisotropy of cellular basis window - values: [0, 1]
- [11] window norm: cellular basis window norm - values: [0, 3]
- [12] window smoothness: degree of smoothing - values: [0, 2]
- [13] window blend: linear combination of the two basis windows - values: [0, 1]
- [14] window sigwcell: sigma of window - values: [0.01, 4]

[ Feature function ]

Feature function parameters:

- [15] feature bombing: mixture model - values: {0, 4}
- [16] feature norm: feature norm - values: [0, 3]
- [17] feature winfeatcorrel: correlation with centroids - values: [0, 1]
- [18] feature aniso: anisotropy - values: [0, 5]
- [19] feature minNbKernels: min number of kernels - values: {0, 8}
- [20] feature maxNbKernels: max number of kernels - values: {0, 8}

- [21] feature sigcos: sigma of Gabor kernel - values: [0, 10]
- [22] feature sigcosvar: random variation of [21] (sigma of Gabor kernel) - values: [0, 1]
- [23] feature frequency: frequency of stringed Gabor stripes - values: {0, 16}
- [24] feature phaseShift: - values: [0, 1]
- [25] feature thickness: thickness of stringed Gabor stripes - values: [0.001, 1]
- [26] feature curvature: curvature of stringed Gabor stripes - values: [0, 1]
- [27] feature orientation: orientation of stringed Gabor stripes - values: [0, 0.5]

Sixième partie

Bibliographie

# Bibliographie

- [AAL16] Miika Aittala, Timo Aila, and Jaakko Lehtinen. Reflectance modeling by neural texture synthesis. *ACM Transactions on Graphics (ToG)*, 35(4) :1–13, 2016.
- [AR81] Narendra Ahuja and Azriel Rosenfeld. Mosaic models for textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(1) :1 – 11, 1981.
- [AV12] Susana Alvarez and Maria Vanrell. Texton theory revisited : A bag-of-words approach to combine textons. *Pattern Recognition*, 45(12) :4312–4325, 2012.
- [BD01] E. Bourque and G. Dudek. Image-driven procedural texture specification. *Research report*, 2001.
- [BD04] Eric Bourque and Gregory Dudek. Procedural texture matching and transformation. *Computer Graphics Forum*, 23(3) :461–468, 2004.
- [BELS10] Pau Panareda Busto, Christian Eisenacher, Sylvain Lefebvre, and Marc Stamminger. Instant Texture Synthesis by Numbers. In Reinhard Koch, Andreas Kolb, and Christof Rezk-Salama, editors, *Vision, Modeling, and Visualization (2010)*. The Eurographics Association, 2010.
- [Bro66] Phil Brodatz. *Textures : a photographic album for artists and designers*. Dover Pubns, 1966.
- [BS12] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7. vol. 2012, 2012.
- [BSFG09] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch : A randomized correspondence al-

- gorithm for structural image editing. *ACM Trans. Graph.*, 28(3) :24, 2009.
- [CGG19] Arthur Cavalier, Guillaume Gilet, and Djamchid Ghazanfar-pour. Local spot noise for procedural surface details synthesis. *Computers & Graphics*, 85 :92–99, 2019.
- [Coo84] Robert L Cook. Shade trees. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231, 1984.
- [CSHD03] Michael F Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics (TOG)*, 22(3) :287–294, 2003.
- [DH14] FR Dayeen and MK Hassan. Self-similarity, aboav-weaire’s and lewis’ laws in weighted planar stochastic lattice. *arXiv preprint arXiv :1409.7928*, 2014.
- [DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society : Series B (Methodological)*, 39(1) :1–22, 1977.
- [DLY<sup>+</sup>20] Junyu Dong, Jun Liu, Kang Yao, Mike Chantler, Lin Qi, Hui Yu, and Muwei Jian. Survey of procedural methods for two-dimensional texture generation. *Sensors*, 20(4) :1135, 2020.
- [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, 1999.
- [DZ06] Jean-Michel Dischler and Florence Zara. Real-time structured texture synthesis and editing using image-mesh analogies. *The Visual Computer*, 22(9-11) :926–935, 2006.
- [EF01] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH ’01, pages 341–346, New York, NY, USA, 2001. ACM.
- [EL99] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1033–1038. IEEE, 1999.

- [EVC<sup>+</sup>15] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. WorldBrush : Interactive Example-based Synthesis of Procedural Virtual Worlds. *ACM Transactions on Graphics*, 34(4) :11, August 2015.
- [F<sup>+</sup>04] Randima Fernando et al. *GPU gems : programming techniques, tips, and tricks for real-time graphics*, volume 590. Addison-Wesley Reading, 2004.
- [FAW19] Anna Frühstück, Ibraheem Alhashim, and Peter Wonka. Tilegan : synthesis of large-scale non-homogeneous textures. *ACM Transactions on Graphics (ToG)*, 38(4) :1–11, 2019.
- [Fra05] Marie-Claude Frasson. *Decoupling structure and appearance for example-driven detail synthesis*. PhD thesis, Université Nice Sophia Antipolis, 2005.
- [GAD<sup>+</sup>20] Pascal Guehl, Remi Allègre, Jean-Michel Dischler, Bedrich Benes, and Eric Galin. Semi-Procedural Textures Using Point Process Texture Basis Functions. *Computer Graphics Forum*, 2020.
- [GBG<sup>+</sup>19] Mathieu Gaillard, Bedrich Benes, Eric Guérin, Eric Galin, Damien Rohmer, and Marie-Paule Cani. Dendry : A procedural model for dendritic patterns. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 1–9, 2019.
- [GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture Synthesis Using Convolutional Neural Networks. *arXiv :1505.07376 [cs, q-bio]*, May 2015. arXiv : 1505.07376.
- [GKHF14] Lena Gieseke, Sebastian Koch, J-U Hahn, and Martin Fuchs. Interactive parameter retrieval for two-tone procedural textures. *Computer Graphics Forum*, 33(4) :71–79, 2014.
- [GLCC17] James Gain, Harry Long, Guillaume Cordonnier, and M-P Cani. Ecobrush : Interactive control of visually consistent large-scale ecosystems. In *Computer Graphics Forum*, volume 36, pages 63–73. Wiley Online Library, 2017.
- [GLLD12] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Dretakis. Gabor noise by example. *ACM Transactions on Graphics (ToG)*, 31(4) :1–9, 2012.



- [GLM17] Bruno Galerne, Arthur Leclaire, and Lionel Moisan. Texton noise. *Computer Graphics Forum*, 36(8) :205–218, 2017.
- [GSDC17] Geoffrey Guingo, Basile Sauvage, Jean-Michel Dischler, and Marie-Paule Cani. Bi-layer textures : a model for synthesis and deformation of composite textures. *Computer Graphics Forum*, 36(4) :111–122, 2017.
- [GSV<sup>+</sup>14] Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Local random-phase noise for procedural texturing. *ACM Transactions on Graphics (ToG)*, 33(6) :1–11, 2014.
- [HB95] David J. Heeger and James R. Bergen. Pyramid-based Texture Analysis/Synthesis. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '95*, pages 229–238, New York, NY, USA, 1995. ACM.
- [HDR19] Yiwei Hu, Julie Dorsey, and Holly Rushmeier. A novel framework for inverse procedural texture modeling. *ACM Transactions on Graphics (TOG)*, 38(6) :1–14, 2019.
- [HGH<sup>+</sup>22] Yiwei Hu, Paul Guerrero, Milos Hasan, Holly Rushmeier, and Valentin Deschaintre. Node graph optimization using differentiable proxies. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022.
- [HH19] Anne Humeau-Heurtier. Texture feature extraction methods : A survey. *IEEE access*, 7 :8975–9000, 2019.
- [HJO<sup>+</sup>01] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image Analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01*, pages 327–340, New York, NY, USA, 2001. ACM.
- [HKYM16] Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE transactions on visualization and computer graphics*, 23(8) :2003–2013, 2016.
- [HLT<sup>+</sup>09] Thomas Hurtut, P-E Landes, Joëlle Thollot, Yann Gousseau, Remy Drouillhet, and J-F Coeurjolly. Appearance-guided synthesis of element arrangements by example. In *Proceedings of*

- the 7th International Symposium on Non-photorealistic Animation and Rendering*, pages 51–60, 2009.
- [HN18] Eric Heitz and Fabrice Neyret. High-performance by-example noise using a histogram-preserving blending operator. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2) :1–25, 2018.
- [IPSS08] Janine Illian, Antti Penttinen, Helga Stoyan, and Dietrich Stoyan. *Statistical analysis and modelling of spatial point patterns*. John Wiley & Sons, 2008.
- [Jul81] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802) :91–97, 1981.
- [KEBK05] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture Optimization for Example-based Synthesis. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH ’05, pages 795–802, New York, NY, USA, 2005. ACM.
- [KNL<sup>+</sup>15] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. Self Tuning Texture Optimization. *Comput. Graph. Forum*, 34(2) :349–359, May 2015.
- [KS17] Isaac Karth and Adam M Smith. Wavefunctioncollapse is constraint solving in the wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pages 1–10, 2017.
- [KSE<sup>+</sup>03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut Textures : Image and Video Synthesis Using Graph Cuts. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH ’03, pages 277–286, New York, NY, USA, 2003. ACM.
- [LBAD<sup>+</sup>06] Jason Lawrence, Aner Ben-Artzi, Christopher DeCoro, Wojciech Matusik, Hanspeter Pfister, Ravi Ramamoorthi, and Szymon Rusinkiewicz. Inverse shade trees for non-parametric material representation and editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3), July 2006.
- [LCF<sup>+</sup>19] Li Liu, Jie Chen, Paul Fieguth, Guoying Zhao, Rama Chelappa, and Matti Pietikäinen. From bow to cnn : Two decades of texture representation for texture classification. *International Journal of Computer Vision*, 127(1) :74–109, 2019.

- [LD05] Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM transactions on graphics (TOG)*, 24(4) :1442–1461, 2005.
- [LDG21] Claire Launay, Agnès Desolneux, and Bruno Galerne. Determinantal point processes for image processing. *SIAM Journal on Imaging Sciences*, 14(1) :304–348, 2021.
- [Lef14] Sylvain Lefebvre. *Synthèse de textures par l'exemple pour les applications interactives*. Habilitation à diriger des recherches, Université de Lorraine (Nancy), June 2014.
- [Lew84] John-Peter Lewis. Texture synthesis for digital painting. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 245–252, 1984.
- [Lew89] John-Peter Lewis. Algorithms for solid noise synthesis. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 263–270, 1989.
- [LGD<sup>+</sup>18] Jun Liu, Yanhai Gan, Junyu Dong, Lin Qi, Xin Sun, Muwei Jian, Lina Wang, and Hui Yu. Perception-driven procedural texture generation from examples. *Neurocomputing*, 291 :21–34, 2018.
- [LH05] Sylvain Lefebvre and Hugues Hoppe. Parallel Controllable Texture Synthesis. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 777–786, New York, NY, USA, 2005. ACM.
- [LH06] Sylvain Lefebvre and Hugues Hoppe. Appearance-space Texture Synthesis. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 541–548, New York, NY, USA, 2006. ACM.
- [LHVT17] Hugo Loi, Thomas Hurtut, Romain Vergne, and Joelle Thollot. Programmable 2d arrangements for element texture design. *ACM Transactions on Graphics (TOG)*, 36(4) :1, 2017.
- [Lin04] Wen-Chieh Lin. A comparison study of four texture synthesis algorithms on regular and near-regular textures. *Research report*, 2004.
- [LLC<sup>+</sup>10] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony Derosé, George Drettakis, David S. Ebert, J. P. Lewis, Ken Perlin, and Matthias Zwicker. A Survey of Procedural Noise Functions. *Computer Graphics Forum*, 29(8) :2579–2600, 2010.

- [LLDD09] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. *ACM Transactions on Graphics (TOG)*, 28(3) :1–10, 2009.
- [Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2) :129–137, 1982.
- [LLX<sup>+</sup>01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3) :127–150, 2001.
- [LP00] Laurent Lefebvre and Pierre Poulin. Analysis and synthesis of structural textures. In *Graphics Interface*, volume 2000, pages 77–86, 2000.
- [LSA<sup>+</sup>16] Yitzhak David Lockerman, Basile Sauvage, Rémi Allègre, Jean-Michel Dischler, Julie Dorsey, and Holly Rushmeier. Multi-scale Label-map Extraction for Texture Synthesis. *ACM Trans. Graph.*, 35(4) :140 :1–140 :12, July 2016.
- [LSC18] Zhengqin Li, Kalyan Sunkavalli, and Manmohan Chandraker. Materials for masses : Svbrdf acquisition with a single mobile phone image. In *Proceedings of the European conference on computer vision (ECCV)*, pages 72–87, 2018.
- [Mer07] Paul Merrell. Example-based model synthesis. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 105–112, 2007.
- [Mer21] Paul Merrell. Comparing model synthesis and wave function collapse. <https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf>, 2021.
- [ML09] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340) :2, 2009.
- [MM08] Paul Merrell and Dinesh Manocha. Continuous Model Synthesis. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, pages 158 :1–158 :7, New York, NY, USA, 2008. ACM.
- [MM11] Paul Merrell and Dinesh Manocha. Model Synthesis : A General Procedural Modeling Algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 17(6) :715–728, June 2011.

- [MZD05] Wojciech Matusik, Matthias Zwicker, and Frédo Durand. Texture Design Using a Simplicial Complex of Morphable Textures. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 787–794, New York, NY, USA, 2005. ACM.
- [ÖG12] A Cengiz Öztireli and Markus Gross. Analysis and synthesis of point distributions based on pair correlation. *ACM Transactions on Graphics (TOG)*, 31(6) :1–10, 2012.
- [OL81] Alan V Oppenheim and Jae S Lim. The importance of phase in signals. *Proceedings of the IEEE*, 69(5) :529–541, 1981.
- [Ots79] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1) :62–66, 1979.
- [Per85] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3) :287–296, 1985.
- [PGDG16] Nicolas Pavie, Guillaume Gilet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Procedural texture synthesis by locally controlled spot noise. *Proc. WSCG 2016*, 2016.
- [PS99] Javier Portilla and Eero P Simoncelli. Texture modeling and synthesis using joint statistics of complex wavelet coefficients. In *IEEE workshop on statistical and computational theories of vision*, 1999.
- [QY01] Xuejie Qin and Yee-Hong Yang. A generalized cellular texture basis function. *Department of Computer Science, University of Saskatchewan*, 2001.
- [QY05] Xuejie Qin and Yee-Hong Yang. Basic gray level aura matrices : theory and its application to texture synthesis. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 128–135 Vol. 1, 2005.
- [RB06] Ganesh Ramanarayanan and Kavita Bala. Constrained texture synthesis via energy minimization. *IEEE Transactions on Visualization and Computer Graphics*, 13(1) :167–178, 2006.
- [RCOL09] Amir Rosenberger, Daniel Cohen-Or, and Dani Lischinski. Layered Shape Synthesis : Automatic Generation of Control Maps for Non-stationary Textures. In *ACM SIGGRAPH Asia 2009 Papers*, SIGGRAPH Asia '09, pages 107 :1–107 :9, New York, NY, USA, 2009. ACM.

- [RT06] Guodong Rong and Tiow-Seng Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116, 2006.
- [SA79] Bruce Schachter and Narendra Ahuja. Random pattern generation processes. *Computer Graphics and Image Processing*, 10(2) :95 – 114, 1979.
- [SCO17] Omry Sendik and Daniel Cohen-Or. Deep Correlations for Texture Synthesis. *ACM Trans. Graph.*, 36(5) :161 :1–161 :15, July 2017.
- [SLH<sup>+</sup>20] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. Match : Differentiable material graphs for procedural material capture. *ACM Trans. Graph.*, 39(6) :1–15, December 2020.
- [SSGS11] Johannes Schmid, Martin Sebastian Senn, Markus Gross, and Robert W. Sumner. Overcoat : An implicit canvas for 3d painting. *ACM Trans. Graph.*, 30(4), jul 2011.
- [TEZ<sup>+</sup>19] Thibault Tricard, Semyon Efremov, Cédric Zanni, Fabrice Neyret, Jonàs Martínez, and Sylvain Lefebvre. Procedural phasor noise. *ACM Transactions on Graphics (TOG)*, 38(4) :1–13, 2019.
- [VdMH08] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [VSLD13] Kenneth Vanhoey, Basile Sauvage, Frédéric Larue, and Jean-Michel Dischler. On-the-fly multi-scale infinite texturing from example. *ACM Transactions on Graphics (TOG)*, 32(6) :1–10, 2013.
- [VW91] Jarke J Van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 309–318, 1991.
- [WL00] Li-Yi Wei and Marc Levoy. Fast Texture Synthesis Using Tree-structured Vector Quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.



- [WL02] Li-Yi Wei and Marc Levoy. Order-independent texture synthesis. *arXiv preprint arXiv :1406.7338*, 2002.
- [WLKT09] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117, 2009.
- [WMLT07] Bruce Walter, Stephen R. Marschner, Hongsong Li, and Kenneth E. Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques, EGSR’07*, page 195–206, Goslar, DEU, 2007. Eurographics Association.
- [Wor96] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, 1996.
- [WQC<sup>+</sup>14] Lili Wang, Qinglin Qi, Yi Chen, Wei Ke, and Aimin Hao. Interactive texture design and synthesis from mesh sketches. *Frontiers of Computer Science*, 8(2) :330–341, Apr 2014.
- [YLT19] Cem Yuksel, Sylvain Lefebvre, and Marco Tarini. Rethinking texture mapping. *Computer Graphics Forum*, 38(2) :535–551, 2019.
- [ZFWW18] Károly Zsolnai-Fehér, Peter Wonka, and Michael Wimmer. Gaussian material synthesis. *arXiv preprint arXiv :1804.08369*, 2018.
- [ZSL<sup>+</sup>17] Yang Zhou, Huajie Shi, Dani Lischinski, Minglun Gong, Johannes Kopf, and Hui Huang. Analysis and controlled synthesis of inhomogeneous textures. *Computer Graphics Forum*, 36(2) :199–212, 2017.
- [ZZB<sup>+</sup>18] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *arXiv preprint arXiv :1805.04487*, 2018.
- [ZZV<sup>+</sup>03] Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of Progressively-variant Textures on Arbitrary Surfaces. In *ACM SIGGRAPH 2003 Papers, SIGGRAPH ’03*, pages 295–302, New York, NY, USA, 2003. ACM.