Contributions à la modélisation procédurale de structures cellulaires stochastoques 2D et à leur génération par l'exemple

Guillaume Baldi

Université de Strasbourg

École doctorale : **Mathématiques, Sciences de l'Information et de l'Ingénieur**

Unité de recherche : **ICube – UMR 7357**

THÈSE

Présentée par : **Guillaume Baldi**

Soutenue le : **19 mars 2024**

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

Discipline/Spécialité : **Informatique**

Contributions à la modélisation procédurale de structures cellulaires stochastiques 2D et à leur génération par l'exemple

Thèse dirigée par :

M. DISCHLER Jean-Michel	Professeur des Universités, Université de Strasbourg

Rapporteurs :

M. BOUBEKEUR Tamy	Professeur des Universités, École Polytechnique, Directeur de Recherche à Adobe Research
M. GUÉRIN Éric	Maître de Conférences HDR, Institut National des Sciences Appliquées de Lyon

Examinateurs :

Mme CANI Marie-Paule	Professeure des Universités, École Polytechnique
M. ALLÈGRE Rémi	Maître de Conférences, Université de Strasbourg

Résumé

Cette thèse s'inscrit dans les travaux de l'équipe Informatique Géométrique et Graphique du laboratoire ICube portant sur la synthèse de textures par l'exemple, en particulier pour des applications dans l'industrie graphique. La création de matériaux et de textures procéduraux demande une grande expertise et constitue un travail long, fastidieux et coûteux, c'est pourquoi on cherche à développer des outils permettant leur génération automatique à partir d'exemples en entrée fournis sous la forme d'images : on parle de *modélisation procédurale inverse*. Les méthodes récentes de l'état de l'art nécessitent de grandes collections de matériaux procéduraux conçus par des experts pour entraîner des réseaux de neurones profonds et se confrontent à la grande complexité combinatoire des graphes. Une autre approche consiste à développer des modèles procéduraux plus simples d'un point de vue combinatoire, et capables de couvrir une grande variété d'apparences tout en gardant un calcul en temps réel. Le problème concerne alors l'estimation des paramètres de ces modèles à partir d'exemples.

Dans cette thèse, nous proposons un modèle procédural de type Point Process Texture Basis Function (PPTBF) permettant de représenter des structures cellulaires stochastiques 2D, très répandues dans la nature, que nous appelons Cellular-PPTBF (C-PPTBF). L'originalité de notre modèle est d'impliquer des fonctions différentiables par rapport à la plupart de leurs paramètres, ce qui rend possible l'estimation de ces paramètres à partir d'exemples sans recourir entièrement à des réseaux de neurones profonds. Nous avons mis en place une chaîne de traitement permettant d'estimer les paramètres de notre modèle à partir d'exemples de structures fournis sous la forme d'images binaires. Notre chaîne de traitement combine une estimation réalisée à l'aide d'un réseau de neurones convolutif entraîné sur des images produites avec notre modèle de C-PPTBF et une phase d'estimation par descente de gradient directement sur les paramètres du modèle procédural. Notre approche est automatique, robuste, et moins dépendante du jeu de données d'entraînement que les méthodes précédentes. Elle permet d'obtenir des résultats visuellement proches même pour des exemples en entrée éloignés des données d'entraînement.

Abstract

This thesis is part of the work carried out by the ICube laboratory's Computer Graphics and Geometry research group on texture synthesis by example, especially for applications in the Computer Graphics industry. The creation of procedural materials and textures requires considerable expertise, and is time-consuming, tedious and costly. We are therefore looking to develop tools for the automatic generation of procedural textures and materials from input exemplars provided in the form of images : This is known as *inverse procedural modeling*. Recent state-of-the-art methods require large collections of procedural materials designed by experts to train deep neural networks, and come up against the high combinatorial complexity of graphs. Another approach consists in developing procedural models that are simpler from a combinatorial point of view, and capable of covering a wide variety of appearances while maintaining real-time computation. The problem then concerns the estimation of the parameters of these models from image examples.

In this thesis, we propose a procedural Point Process Texture Basis Function (PPTBF) model for representing 2D stochastic cellular structures, which are widespread in nature, and which we call Cellular-PPTBF (C-PPTBF). The originality of our model is that it involves functions that are differentiable with respect to most of their parameters, making it possible to estimate these parameters from examples without resorting entirely to deep neural networks. We have set up a processing pipeline to estimate the parameters of our model from structural examples provided in the form of binary images. Our processing pipeline combines an estimation performed using a convolutional neural network trained on images produced with our C-PPTBF model and an estimation phase using gradient descent directly on the parameters of the procedural model. Our approach is automatic, robust, and less dependent on the training dataset than previous methods, providing visually close results even for input examples that visually are far from the training data.

Remerciements

Je tiens tout d'abord à remercier Tamy Boubekeur et Éric Guérin pour avoir accepté d'être les rapporteurs de ces travaux de thèse, ainsi que Marie-Paule Cani pour avoir accepté de faire partie de mon jury de thèse. Je remercie également Éric Galin et Franck Hétroy-Wheeler pour avoir participé à mes comités de suivi de thèse.

Je tiens ensuite à remercier Jean-Michel Dischler, mon directeur de thèse, pour m'avoir fait confiance pour entreprendre ces travaux de recherche. Je remercie grandement Rémi Allègre, mon encadrant, pour m'avoir accompagné et aidé au quotidien tout au long de ces années de thèse.

Je tiens à remercier tous les membres de l'équipe IGG du laboratoire ICube pour leur accueil et leur accompagnement, ainsi que les équipes de l'IUT de Haguenau et de l'IUT Robert Schuman.

Je tiens à remercier mes amis d'Atlas pour avoir égayé mon quotidien pendant toutes ces années.

Je tiens finalement à remercier ma famille pour m'avoir soutenu depuis le début tout au long de mon parcours.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte et problématique

Dans cette partie nous présentons le contexte dans lequel s'inscrit cette thèse, dans le domaine de l'Informatique Graphique et plus particulièrement celui de la synthèse de textures, ainsi que les problématiques abordées dans ce manuscrit.

1.1.1 Informatique Graphique

Cette thèse s'inscrit dans le domaine de l'Informatique Graphique, discipline de l'Informatique se concentrant sur l'élaboration d'algorithmes de synthèse d'images par ordinateur pour répondre à plusieurs problématiques comme la génération de textures et de matériaux, la modélisation d'objets en 3 dimensions, l'animation de modèles, ainsi que le rendu de scènes virtuelles. Les méthodes développées dans ces différentes thématiques ont contribué à de multiples champs d'application, du prototypage industriel aux effets spéciaux pour le cinéma, en passant par les jeux vidéo, la réalité virtuelle et augmentée, ou encore l'imagerie médicale.

Le pipeline standard de l'Informatique Graphique commence par une étape de modélisation d'une scène en trois dimensions, pour se terminer par l'étape de rendu. L'étape de modélisation 3D consiste à créer des modèles géométriques permettant de représenter les différents objets présents dans la scène. Cette étape peut passer soit par l'utilisation de primitives (formes de base simples), soit par celles de modèles issus de la numérisation d'objets physiques grâce aux technologies d'acquisition 3D. Les objets sont généralement représentés à l'aide de maillages surfaciques constitués de triangles, qui offrent des facilités pour l'édition, et qui sont utilisés de façon standard par les algorithmes de rendu. Pour obtenir une apparence visuelle réaliste, les objets

sont habillés de matériaux et de textures qui définissent les propriétés de couleur, et plus généralement le comportement des surfaces vis-à-vis de la lumière. Le rendu consiste à produire des images d'une scène depuis une ou plusieurs caméras, en simulant le parcours de la lumière et ses interactions avec les surfaces et/ou les volumes des objets, selon différents algorithmes possibles comme le *lancer de rayons* et différents modèles d'ombrages. Finalement, les scènes peuvent être animées en s'occupant du mouvement des objets et de personnages, des interactions entre un utilisateur et les objets, ou encore du déroulement de phénomènes physiques.

Dans le cadre de nos travaux, nous nous intéressons à l'étape d'habillage des objets à l'aide de matériaux et de textures, pour en définir l'apparence visuelle.

1.1.2 Matériaux et textures

Afin d'habiller une scène en 3 dimensions, la technique la plus courante consiste à plaquer des images en 2 dimensions sur la surface des objets, au moyen de techniques de paramétrisation [YLT19]. Ces images sont appelées *textures*[1]. Les textures permettent de donner une apparence à une surface en modifiant ses propriétés comme sa couleur intrinsèque (ou *albédo*), mais aussi ses autres propriétés de réflexion de la lumière qui varient spatialement et en fonction du point de vue. Dans le cadre des méthodes de rendu physique réaliste (*Physically-Based Rendering*, PBR), les matériaux sont représentés à l'aide de multiples couches de textures, utilisées pour stocker les valeurs des paramètres qui interviennent dans les modèles d'ombrage avec leurs variations spatiales, ou modèles de SVBRDF (*Spatially Varying Bidirectional Reflectance Distribution Function*) : albédo, normales, rugosité, métallicité, déplacement. La figure 1.1 montre l'exemple d'une image de textures de briques qui est plaquée sur un objet 3D de la forme d'un mur.

Qu'est-ce qu'une texture ? D'un point de vue plus formel, les textures se distinguent d'autres types d'images par le fait qu'elles correspondent à des distributions spatiales de motifs (ou primitives, ou éléments structurés) élémentaires répétitifs dans le plan. Différentes classifications des textures ont été proposées, en lien étroit avec des modèles de représentation ou des algorithmes d'analyse ou de synthèse. Suivant la proposition de Lin et al. [Lin+04], on peut classifier les textures des plus régulières aux plus aléatoires, ou stochastiques. Dans son HDR, Sylvain Lefebvre [Lef14], propose une classification faisant intervenir les notion de *structure* et d'*organisation*

1. Dans le présent manuscrit, nous nous intéressons seulement aux textures 2D, bien que la notion de texture puisse être étendue à des dimensions supérieures, notamment en 3D.

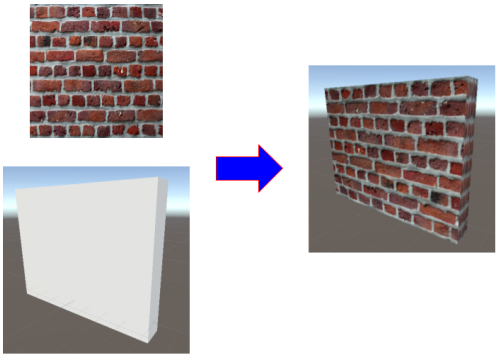

Figure 1.1 – Application d'une image de texture représentant des briques sur un objet en forme de mur.

spatiale : régulières et quasi-régulières, structurées organisées, stochastiques structurées, stochastiques non structurées. Des exemples de textures de ces différents types sont présentés dans la figure 1.2.

Figure 1.2 – Exemples d'images de textures naturelles. De gauche à droite, les textures présentées sont d'abord régulières ou quasi-régulières (1ère colonne), puis stochastiques structurées (2ème et 3ème colonne), puis stochastiques non structurées (4ème colonne).

Textures stochastiques structurées Dans cette thèse, nous nous intéressons plus particulièrement aux textures *stochastiques structurées*, qui possèdent un aspect global stochastique, mais qui comportent aussi des éléments structurés à une échelle plus fine. Considérons par exemple le cas d'une texture représentant un mur de pierres : les images des pierres individuelles sont des éléments structurés, qui se répètent avec des variations aléatoires : position, forme et motif de la roche. Les textures de ce type sont particulièrement répandues [Bro66] et nécessitent des algorithmes complexes pour être reproduites. Dans la suite, nous appellerons *structure* l'organisation spatiale des éléments structurés composant une texture. Dans ce travail, nous nous intéressons plus particulièrement à un sous-ensemble des textures stochastiques structurées comportant un unique élément structuré se répétant avec des variations spatiales aléatoires, sans empilement, sur un arrière-plan stochastique non structuré à l'échelle d'observation. Nous qualifierons ces structures de *cellulaires.* Des exemples d'images de textures comportant ces structures cellulaires sur lesquelles nos travaux se portent sont présentées dans la figure 1.3.

Figure 1.3 – Exemples de textures de structures cellulaires, comme des arrangements de galets, des murs de briques, des écailles de peau de serpent ou des craquelures.

1.1.3 Synthèse de textures et génération par l'exemple

Dans un contexte de demande croissante d'environnements 3D virtuels riches et complexes, il est aujourd'hui courant pour les artistes de l'industrie graphique de concevoir des *graphes de matériaux procéduraux* à l'aide d'outils comme Adobe Substance 3D Designer [22] pour créer des matériaux complexes. Les graphes de matériaux procéduraux sont constitués de nœuds correspondant à l'implémentation d'algorithmes de génération de motifs (par exemple, un mur de briques) ou de bruits (par exemple, des motifs de nuages), ou à des opérations de traitement d'image appliquées sur les motifs générés (par exemple, un filtre gaussien). Ces graphes peuvent être évalués par pixel, à résolution quelconque, et permettent un rendu en temps réel grâce aux GPUs. Un exemple de graphe conçu sur Adobe Substance 3D Designer pour la génération de cartes de matériaux est présenté dans la figure 1.4.

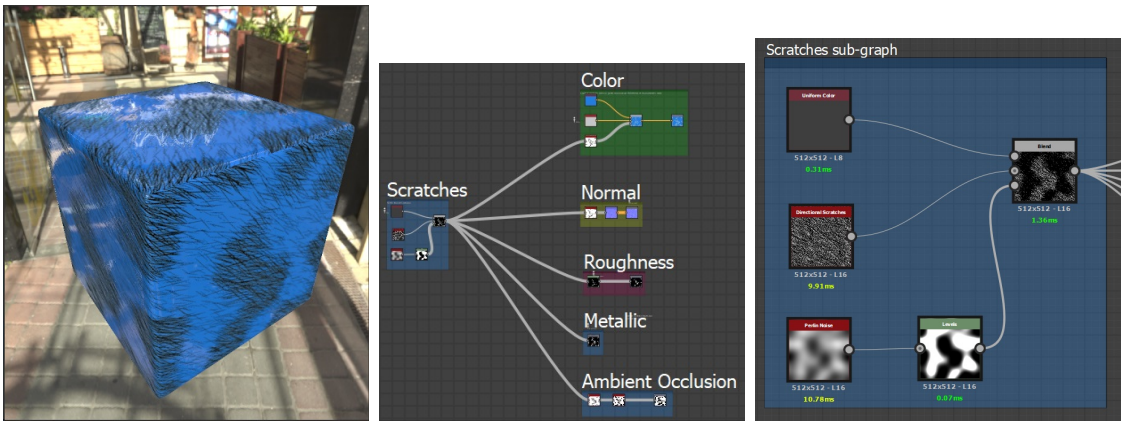

Figure 1.4 – Exemple de graphe de nœuds issu d'Adobe Substance 3D Designer extrait de [Gue22]. *À gauche* : rendu physiquement réaliste d'un matériau PBR avec une carte d'environnement. *Au milieu* : génération des couches de textures d'un matériau à partir de nœuds procéduraux. *À droite* : création de motifs de bruits procéduraux (rayures stochastiques) et mélange avec un masque procédural pour remplir des régions d'intérêts spécifiques avec le motif reposant sur du bruit de Perlin.

Cette approche offre un haut niveau de contrôle aux artistes. Cependant, le fait de construire manuellement des graphes de matériaux procéduraux est un travail long, coûteux, et qui nécessite une grande expertise, ce qui limite le passage à l'échelle de cette approche. Une difficulté particulière réside dans le caractère peu intuitif de l'assemblage de nœuds qui obéit à des règles mathématiques.

La synthèse de textures *par l'exemple* constitue un champ de recherche de l'Informatique Graphique dont le but est de développer des algorithmes pour générer, à

partir d'exemples fournis en entrée sous la forme d'images, des textures représentant de manière fidèle les caractéristiques visuelles des exemples, à une résolution généralement plus grande. La synthèse par l'exemple peut s'appliquer aussi bien à des textures ne comportant que des données de couleur, qu'aux multiples couches de textures composant un matériau. De nombreuses approches ont été proposées [Wei+09 ; Akl+18 ; ADM18], parmi lesquelles on peut distinguer les approches *statistiques*, les approches *géométriques*, et les approches *procédurales*. Dans cette dernière famille, qui concentre beaucoup de travaux récents, on parle également de *modélisation procédurale inverse* (*Inverse Procedural Modeling*), ou de capture procédurale inverse (*Inverse Procedural Material Capture*). Dans la suite de cette section, nous passons brièvement en revue les méthodes de l'état de l'art, avec une attention particulière portée au cas de la synthèse de textures stochastiques structurées de taille arbitraire, et en temps réel. Nous détaillons plus spécifiquement l'approche *semi-procédurale* proposée par Guehl et al. [Gue+20 ; Gue22], sur laquelle s'appuient les travaux de cette thèse, dans la section suivante.

Approches statistiques Les approches se reposant sur les statistiques utilisent des ensembles de statistiques extraites de l'exemple en entrée et les appliquent lors de la synthèse en transformant progressivement une image initiale de bruit blanc. Les techniques les plus récentes utilisent des réseaux de neurones profonds, soit pré-entraînés (par exemple, un réseau de neurones convolutif (CNN) d'architecture VGG19 pré-entraîné sur ImageNet), soit entraînés spécifiquement à synthétiser une texture particulière.

Dans cette famille, les méthodes de l'état de l'art permettant de générer des textures de taille arbitraire utilisent l'exemple d'entrée lui-même comme guide spatial à plusieurs niveaux de résolution, comme *Non-stationary texture synthesis by adversarial expansion* [Zho+18], ou *TileGan : Synthesis of large-scale non-homogeneous textures* [FAW19]. Ces approches permettent de reproduire fidèlement des textures stochastiques structurées. Cependant, leurs performances sont très dépendantes des résolutions des images d'entrée et de sortie, et ne permettent pas une synthèse en temps réel pour des résolutions élevées.

Approches géométriques Les approches géométriques sont des algorithmes cherchant à recopier des ensembles de pixels des exemples d'entrée pour produire les textures en sortie. Ces méthodes s'inspirent à l'origine des modèles de MRF (*Markov Random Fields*), en formulant des hypothèses de stationnarité et de localité. Un algorithme de référence est *Parallel Controllable Texture Synthesis* (PCTS) [LH05], qui est à la fois multi-échelle, parallèle, et temps réel.

Dans le cas de textures structurées, afin d'apporter du contrôle sur la distribution spatiale des motifs ou leurs variations, l'utilisation de *cartes de labels* (ou *cartes de textons*) a été introduite. Pour synthétiser une texture, il est nécessaire de fournir trois images : 1) un échantillon de texture, 2) une carte de labels correspondant à une partition spatiale de l'échantillon de texture en régions correspondant aux différents motifs, et 3) une carte de labels correspondant à la répartition spatiale souhaitée des motifs dans la texture en sortie. L'algorithme PCTS a été adapté pour prendre en compte des cartes de labels [Bus+10], la principale limite de cette méthode étant que la création des cartes de labels pour les textures à synthétiser est entièrement manuelle.

Approches procédurales Les approches procédurales sont des méthodes qui impliquent des fonctions mathématiques et des algorithmes dont les paramètres permettent de contrôler le résultat de la synthèse, offrant des représentations compactes en mémoire avec une génération possible en parallèle et en temps réel. De nombreuses techniques à base de modèles de bruit ont été proposées [Ebe+02 ; Lag+10]. Les méthodes à contrôle spectral ne sont pas adaptées aux textures structurées. Le modèle de bruit cellulaire de Worley [Wor96], permettant de représenter certaines textures stochastiques structurées, n'a pas fait l'objet d'un algorithme de synthèse par l'exemple.

Plusieurs approches permettant d'estimer les paramètres de modèles procéduraux plus complexes, pouvant être construits comme des graphes de matériaux procéduraux, ont été proposées à partir de 2019 [HDR19 ; Shi+20 ; Guo+20a ; Hu+22c ; Hu+22c ; Gue+22 ; LSM23]. À partir de rendus des matériaux, les paramètres continus des modèles peuvent être estimés directement à partir d'exemples en entrée par descente de gradient, ou bien en utilisant des réseaux de neurones profonds entraînés à « mimer » les modèles procéduraux. Ces approches sont très flexibles et sont aujourd'hui capables de couvrir une large diversité de textures et de matériaux, avec cependant une complexité qui reste limitée. Par exemple, des motifs hiérarchiques (différents motifs imbriqués) ne sont pas facilement pris en charge, ce qui justifie la poursuite de travaux autour de modèles procéduraux avec des pouvoirs de représentation supérieurs.

1.1.4 Approche semi-procédurale de [Gue+20]

Guehl et al. [Gue+20 ; Gue22] ont introduit une approche *semi-procédurale* combinant une méthode procédurale et une méthode géométrique pour synthétiser des textures stochastiques structurées par l'exemple, incluant différents types de struc-

tures : pavages, cellules, craquelures, tâches, lignes, ondes, empilements, etc. L'approche proposée a pour objectif de tirer parti des avantages complémentaires des deux familles de méthodes. La méthode, telle qu'elle a été proposée par les auteurs, est limitée à des textures comportant un unique élément structuré se répétant avec des variations spatiales aléatoires sur un arrière-plan non structuré, mais pourrait être étendue à ces organisation spatiales plus complexes, notamment hiérarchiques. La méthode prend en entrée un exemple de texture accompagné d'une carte de labels *binaire* représentant les éléments structurés avec leurs variations géométriques et leur distribution spatiale, appelée *carte de structures* (figure 1.5). L'étape d'extraction d'une carte de labels associée à un exemple de texture donné n'est pas automatisée. La méthode se déroule alors en deux étapes successives illustrées par la figure 1.6 :

1. l'estimation des paramètres du modèle procédural de PPTBF (*Point Process Texture Basis Function*) proposée par les auteurs à partir de la carte de structures en entrée ;
2. la synthèse des motifs de texture à l'aide de la méthode PCTS [LH05], guidée par une carte de structures générée à la volée à l'aide du modèle de PPTBF.

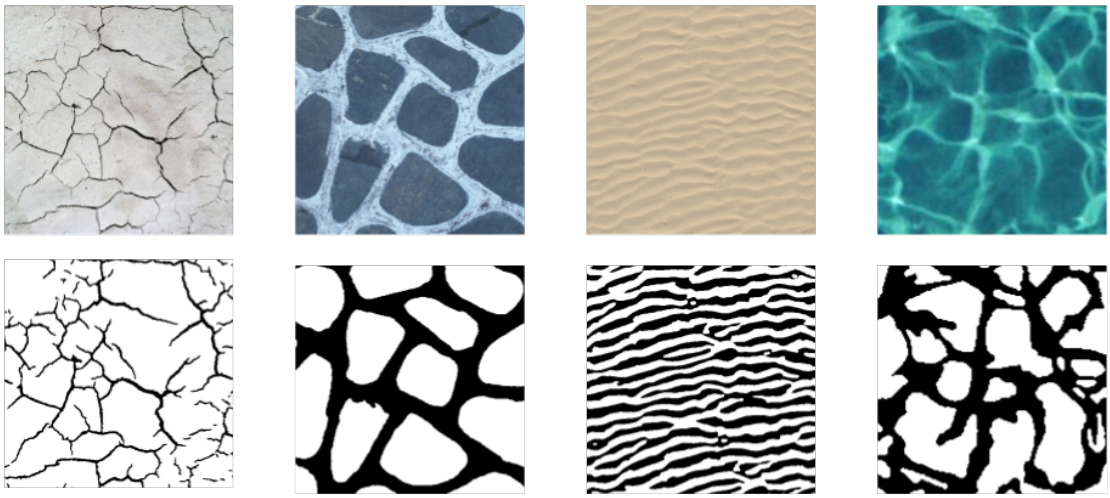

Figure 1.5 – Exemples d'images de textures naturelles (en haut) et leurs cartes de structures segmentées manuellement (en bas).

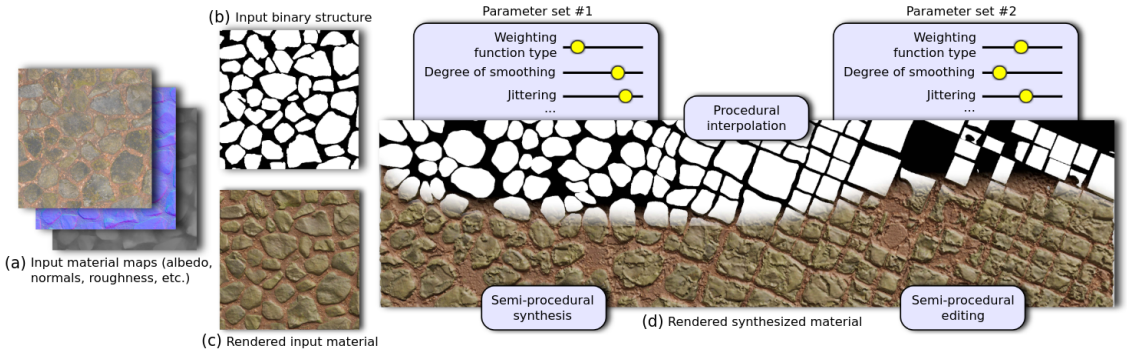

Figure 1.6 – Synthèse de matériaux à partir de cartes de matériaux et de cartes de structures binaires en entrée, en estimant une trentaine de paramètres qui permettent de générer des images binaires utilisées pour la synthèse [Gue+20 ; Gue22].

Modèle de PPTBF Le modèle de PPTBF introduit par Guehl et al. [Gue+20 ; Gue22] permet de représenter une large classe de structures stochastiques. Une PPTBF consiste en la réalisation d'un processus spatial ponctuel (P) convolué par le produit d'une fonction *Window* (W) qui contrôle les interactions mutuelles entre des éléments voisins et d'une fonction *Feature*[2] (F) contrôlant les formes locales (figure 1.7). La sortie d'une PPTBF est une image d'intensité avec des variations continues, qui devient une carte de structures après un seuillage binaire.

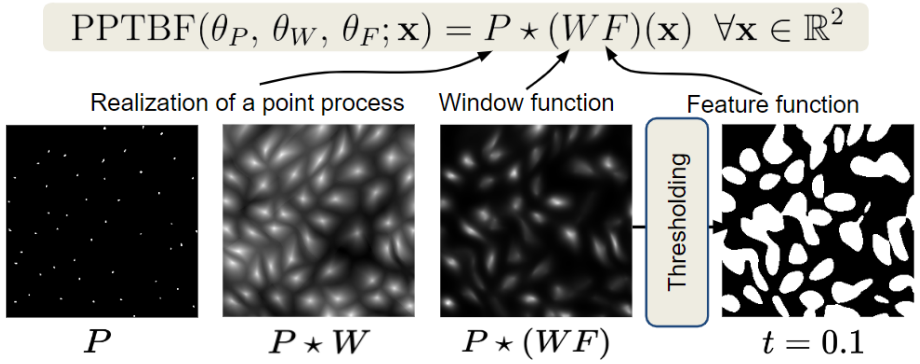

Figure 1.7 – Formule générale du modèle de PPTBF [Gue+20 ; Gue22] et exemple de carte de labels binaire produite par le modèle procédural.

Le modèle de PPTBF ainsi présenté correspond à une formulation générale pou-

2. Les termes en anglais sont conservés par souci de cohérence avec le vocabulaire utilisé dans les publications associées.

vant être implémentée à l'aide de différents P, W et F. Guehl et al. [Gue+20 ; Gue22] ont montré que cette formulation, inspirée des bruits à convolutions parcimonieuses [Lag+09] et du bruit cellulaire de Worley [Wor96], inclut non seulement plusieurs modèles proposés précédemment dans la littérature, mais permet aussi d'élargir la gamme de structures pouvant être représentées, notamment grâce à l'utilisation de distributions de points aléatoires non uniformes et de distances anisotropes.

Guehl et al. [Gue+20 ; Gue22] proposent des P, W et F menant à un modèle comprenant environ 30 paramètres, rendant un réglage manuel particulièrement long et fastidieux, ce qui soulève la question d'une estimation automatisée à partir d'exemples. Face à la non-différentiabilité du modèle en raison de plusieurs paramètres discrets, l'approche originale proposée par les auteurs pour estimer les paramètres à partir de cartes de structures en entrée consiste à effectuer des requêtes de plus proches voisins sur une collection de cartes de structures synthétiques obtenues en échantillonnant les paramètres du modèle de manière uniforme. Les résultats obtenus sont ensuite raffinés manuellement, entraînant un processus d'une durée de l'ordre de 5 minutes par carte de structures.

Hu et al. [Hu+22c] ont proposé plus tard d'entraîner un réseau de neurones génératif appelé *DiffProxy* permettant de reproduire de façon approchée des modèles procéduraux, incluant le modèle de PPTBF, à l'aide d'une fonction mathématique différentiable, et de l'utiliser pour estimer les paramètres moyennant la définition d'une fonction d'erreur et un algorithme de descente de gradient. Dans les deux cas, les méthodes dépendent entièrement de l'échantillonnage de l'espace des paramètres dans des intervalles choisis a priori, et peuvent échouer si l'exemple est trop éloigné visuellement des données utilisées pour l'entraînement.

1.2 Objectifs et contributions

Dans ce travail de thèse, nous avons souhaité améliorer les performances d'estimation des paramètres du modèle procédural de PPTBF proposé par Guehl et al. [Gue+20 ; Gue22]. Notre étude se concentre sur une implémentation du modèle de PPTBF permettant de couvrir une large gamme de structures stochastiques *cellulaires*, pour lesquelles nous avons observé que les fonctions Window et Feature de la PPTBF peuvent être définies avec un nombre réduit de paramètres, et de façon différentiable par rapport à ces paramètres. En s'appuyant sur ces observations, nous proposons une chaîne de traitement entièrement différentiable pour estimer les paramètres par descente de gradient, qui implique une PPTBF définie avec des fonctions Window et Feature différentiables. Nous appelons notre modèle *C-PPTBF*, pour *Cellular-PPTBF*, dans la mesure où il s'inscrit dans la formulation de la PPTBF

originale en étant dédié aux structures stochastiques cellulaires.

Nous contributions sont les suivantes :

— Une C-PPTBF avec des fonctions Window et Feature différentiables par rapport à leurs paramètres, couvrant des structures stochastiques cellulaires ;

— Une fonction d'erreur appropriée et une procédure d'optimisation pour l'estimation des paramètres à partir d'exemples de cartes de structures ;

— Une évaluation en profondeur de la méthode, et une étude comparative avec l'approche DiffProxy [Hu+22c].

— Un code mis à disposition de la communauté via un dépôt GitHub [3].

Nous montrons que notre méthode pour estimer les paramètres est moins dépendante des données d'entraînement que les méthodes précédemment proposées par Guehl et al. [Gue+20 ; Gue22] et par Hu et al. [Hu+22c] (DiffProxy), résulte en une robustesse accrue, et permet une estimation avec un temps de l'ordre de 2 minutes, ce qui représente un gain de 60% par rapport à la méthode proposée par Guehl et al., et un temps similaire à celui de la méthode de Hu et al.

1.2.1 Publications

Les travaux présentés dans ce manuscrit ont fait l'objet des publications suivantes :

Communication dans une conférence nationale sans comité de lecture, avec actes

[BAD22] *Differentiable Point Process Texture Basis Functions for inverse procedural modeling of cellular stochastic structures*, Journées Françaises d'Informatique Graphique 2022, 23 au 25 Novembre 2022, Bordeaux, France.

Article dans une revue internationale avec comité de lecture

[BAD23] *Differentiable Point Process Texture Basis Functions for inverse procedural modeling of cellular stochastic structures*, Computer & Graphics, Volume 112, 2023, Pages 116-131, ISSN 0097-8493.

3. https://github.com/ASTex-ICube/DiffPPTBF

1.3 Plan

Dans ce chapitre d'introduction, nous avons présenté le contexte dans lequel s'inscrit cette thèse, la problématique, ainsi que les objectifs du travail et les contributions que nous apportons. Dans le deuxième chapitre nous proposons un état de l'art consacré aux méthodes de modélisation procédurale de structures stochastiques et de modélisation procédurale inverse. Dans le troisième chapitre nous détaillons notre modèle procédural de C-PPTBF (*Cellular Point Process Texture Basis Functions*). Dans le quatrième chapitre, nous présentons la chaîne de traitement de notre méthode de modélisation procédurale inverse en se focalisant sur l'estimation des paramètres du modèle de C-PPTBF. Dans le cinquième chapitre, nous montrons les résultats de notre méthode en nous comparant avec d'autres méthodes de l'état de l'art, notamment DiffProxy [Hu+22c]. Dans le sixième et dernier chapitre nous concluons sur les travaux présentés dans ce manuscrit en proposant des perspectives possibles de poursuites sur ces travaux.

Chapitre 2

État de l'art

Dans cette partie nous passons en revue l'état de l'art portant tout d'abord sur la modélisation procédurale de structures stochastiques cellulaires, puis sur la modélisation procédurale inverse de textures et de matériaux.

2.1 Modélisation procédurale de structures stochastiques cellulaires

Un modèle procédural pouvant représenter des structures stochastiques cellulaires doit être capable de représenter à la fois les éléments structurés *et* leurs variations aléatoires. Les méthodes existantes s'appuient pour cela sur des fonctions de bruit et des méthodes de distribution et de contrôle spatial.

Bruits à contrôle spectral Une des premières fonctions de bruit développée et qui est toujours très utilisée aujourd'hui est le bruit de Perlin [Per85]. Cette fonction utilise une grille régulière divisée en cellules et composée de nœuds auxquels sont assignés pour chacun un vecteur gradient de valeur aléatoire en utilisant un générateur de nombres aléatoires. Pour calculer la valeur du bruit en un point donné, une interpolation lisse est effectuée selon les gradients des nœuds voisins au point considéré et les distances entre eux. Ce bruit, contrôlé au travers de ses propriétés spectrales (amplitude, fréquence) permet de présenter une large gamme de textures stochastiques non structurées ou de générer des terrains et reliefs ce qui le rend très populaire dans les domaines des jeux vidéo ou des effets spéciaux (figure 2.1). Une limite des fonctions de bruit à contrôle spectral est cependant l'impossibilité de définir des formes locales et de les contrôler sans introduire des outils supplémentaires.

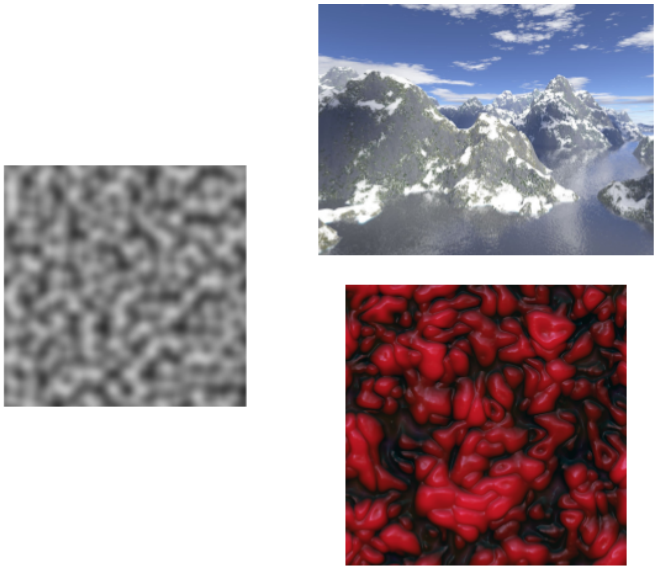

Figure 2.1 – À gauche : bruit de Perlin [Per85] en deux dimensions ; à droite : exemples d'applications avec la création de paysages virtuels et de textures organiques (images extraites de l'article Perlin Noise de Wikipédia).

Bruits à convolutions parcimonieuses La limite précédente est dépassée par les *bruits à convolutions parcimonieuses* (*Sparse Convolution Noises* [Lew84 ; Lag+10]), qui génèrent des bruits selon des sommes pondérées de noyaux, notamment gaussiens ou de Gabor [Lag+09]. Ces noyaux sont distribués spatialement de façon aléatoire selon des distributions de points, les *feature points*[1], par exemple selon un processus de Poisson (figure 2.2). Les noyaux de Gabor permettent d'obtenir des motifs d'oscillation avec contrôle de l'orientation. Le *Phasor Noise* [Tri+19] modifie le Gabor Noise en introduisant un champ de phases sous la forme d'une onde sinusoïdale, chaque oscillation de l'onde étant contrôlable par plusieurs paramètres pour modifier les profils, les fréquences ou encore les orientations. Ce bruit permet alors de représenter des motifs comme des rayures ou des fissures (figure 2.3).

Cependant, le contrôle sur les formes locales produites est limité, en raison du mélange qui se produit entre les noyaux, permettant difficilement d'obtenir des éléments structurés de forme contrôlable.

1. Nous conservons dans ce manuscrit le terme *feature point* en anglais qu'on pourrait traduire par « point caractéristique ».

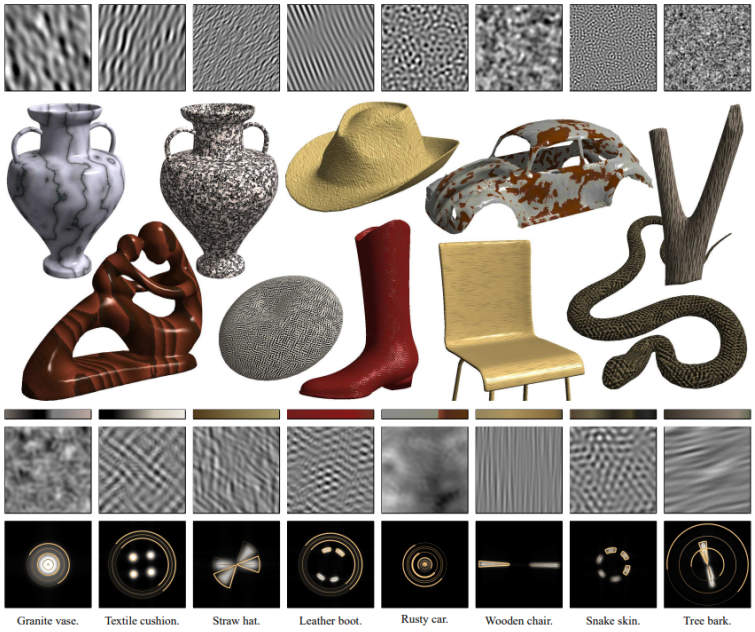

Figure 2.2 – Synthèse de multiples motifs de bruits avec du *Gabor Noise* en modifiant les paramètres de bandes et de fréquence [Lag+09].

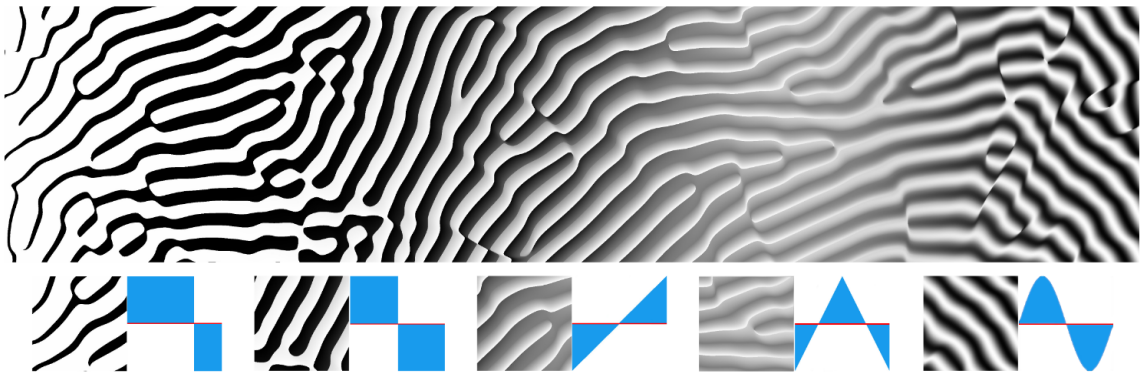

Figure 2.3 – Génération de motifs sous formes de rayures avec le *Phasor Noise* selon les profils utilisés pendant le contrôle [Tri+19].

Bruit cellulaire Worley [Wor96] a proposé le modèle de *Cellular Texture Basis Function* qui est une fonction de bruit capable de partitionner le plan en cellules aléatoires afin de produire des textures comme des sols pavés, de la peau, du papier froissé, de la glace, des rochers ou des montagnes (figure 2.4). Comme pour le bruit à convolutions parcimonieuses, le modèle s'appuie sur une distribution spatiale aléa-

toire de points. La fonction de bruit consiste alors à calculer les distances entre un point $\mathbf{x}$ donné du plan et les N feature points les plus proches de ce point et à les utiliser dans une combinaison linéaire pour générer les structures.

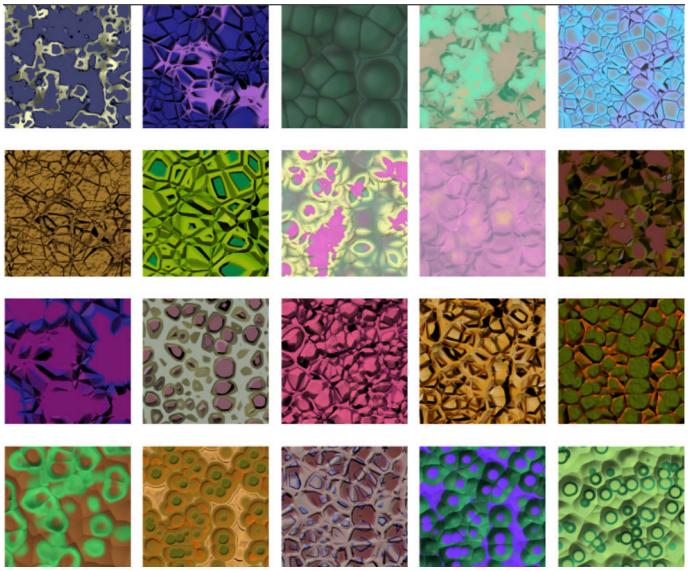

Figure 2.4 – Synthèse de motifs cellulaires par utilisation de feature points [Wor96].

Motifs dendritiques Gaillard et al. [Gai+19] ont proposé Dendry, une fonction procédurale capable de générer des motifs dendritiques et qui comme les autres fonctions de bruit est localement calculable. La fonction est contrôlée par des paramètres tels que le niveau de ramification et le degré de lissage local, ainsi que l'intervalle des angles de ramification. Elle peut également être contrôlée par une fonction de contrôle globale qui définit la structure générale des ramifications. Le principe consiste à calculer la distance à une structure arborescente qui est implicitement construite à la volée, tout en nécessitant un faible coût en mémoire. Comme pour le bruit cellulaire, l'évaluation peut être effectuée en parallèle pour plusieurs positions spatiales. Les auteurs présentent une application à la génération de champs de hauteurs utilisés pour créer des terrains contenant par ailleurs des réseaux de rivières.

Edge-Based Procedural Texture Kim et al. [Kim+21] ont mis en place une méthode de modélisation procédurale reposant sur les bords des structures présentes dans une image d'exemple en entrée. Ainsi dans une première étape d'analyse de

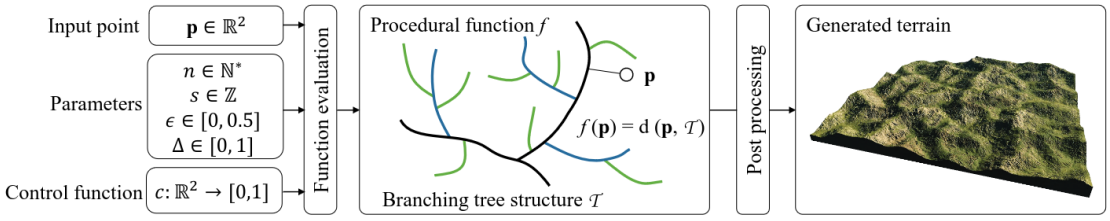

FIGURE 2.5 – Génération de terrains à partir de motifs dendritiques selon une structure arborescente [Gai+19].

l'exemple tous les bords sont extraits puis regroupés selon leur position, leur orientation ou encore leur longueur en un nombre de groupes défini par l'utilisateur, chaque groupe formé comprend alors des éléments dont les statistiques locales sont similaires. Une texture peut ensuite être synthétisée avec une étape de génération procédurale de groupes de bords, suivie d'une étape de synthèse de type géométrique par patch, guidée par les groupes de bords (figure 2.6). Dans la mesure où la méthode détecte des bords indépendants et non des formes, le niveau de contrôle sur les formes générées est limité.

FIGURE 2.6 – Synthèse de textures reposant sur les bords des structures des exemples en entrée [Kim+21].

Point Process Texture Basis Function Le modèle de PPTBF proposé par [Gue+20 ; Gue22], inspiré du bruit cellulaire [Wor96], cherche à offrir une formulation mathématique et algorithmique suffisamment générale pour inclure les différents modèles précédents, avec comme objectif la génération de structures stochastiques sous la forme d'images binaires. Une PPTBF consiste en la réalisation d'un processus spatial ponctuel (P) convolué par le produit d'une fonction Window (W) contrôlant les interactions mutuelles entre éléments voisins et d'une fonction Feature

(F) contrôlant les formes locales. La sortie d'une PPTBF est une image d'intensité avec des variations continues, qui devient une carte de structures après un seuillage binaire. Les choix multiples possibles pour P, F et W permettent à ce modèle de couvrir une large gamme de structures régulières à stochastiques. Si P peut être la réalisation d'un processus de Poisson, il peut aussi s'agir d'une distribution plus complexe comme la réalisation d'un processus de Cox. Les auteurs de la méthode proposent jusqu'à 17 types de pavages à partir desquels calculer les distributions de points. La fonction Window W peut être définie de façon à contrôler le mélange entre des éléments structurés voisins, avec la possibilité d'utiliser des distances anisotropes plutôt que la distance euclidienne, de façon à varier la forme des éléments. La fonction Feature F peut être un simple noyau gaussien, un noyau de Gabor, ou encore une somme de noyaux. Le grand nombre de paramètres (environ 30) rend ce modèle difficile à contrôler, ce qui rend nécessaire des méthodes d'estimation de paramètres à partir d'exemples de structures.

MatFormer Guerrero et al. [Gue+22] ont développé le modèle génératif MatFormer qui permet de générer des matériaux procéduraux divers comprenant des motifs et des apparences complexes, dont l'architecture se base sur des transformeurs et des mécanismes d'attention. Les graphes de nœuds utilisés dans le réseau sont découpés entre nœuds, arêtes et paramètres, un transformeur étant dédié à chacune des trois parties. Le réseau génère alors en sortie des graphes de matériaux complets, mais l'utilisateur peut aussi fournir des nœuds générateurs de départ pour former un graphe partiel qui sera automatiquement complété par le réseau en proposant des variantes différentes (figure 2.7). Cette méthode a beaucoup de potentiel pour représenter des structures stochastiques complexes, mais les auteurs ne fournissent pas de méthode pour générer des matériaux à partir d'exemples fournis sous la forme d'images.

2.2 Modélisation procédurale inverse

Dans le cadre de nos travaux nous nous intéressons plus précisément à la modélisation procédurale inverse. L'objectif de la modélisation procédurale inverse est de retrouver les paramètres d'un modèle procédural qui s'approche d'une texture, d'un matériau, ou d'une structure donnée en entrée. Afin de remplir cet objectif, deux problématiques sont à considérer : 1) sélectionner une méthode d'estimation des paramètres efficace, et 2) choisir une métrique d'évaluation de similarité pour comparer un exemple en entrée et l'image que nous obtenons. Dans cet état de l'art,

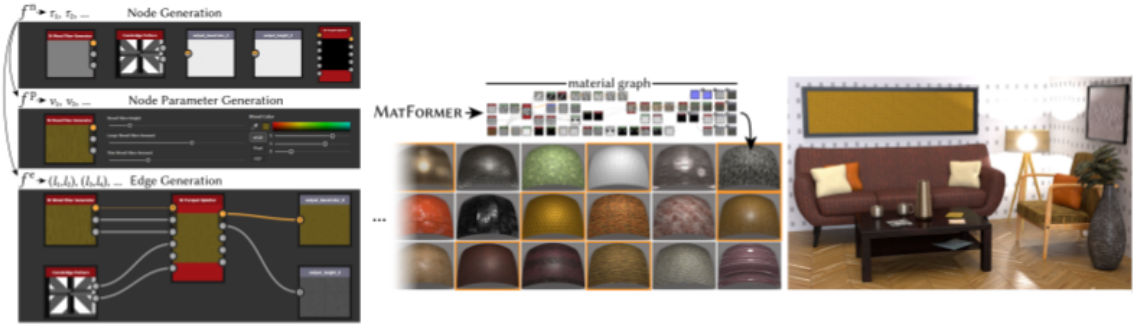

Figure 2.7 – Synthèse de matériaux à partir du réseau génératif MatFormer qui permet de découper des graphes de nœuds entre nœuds, arêtes et paramètres [Gue+22].

nous passons d'abord en revue les premiers travaux sur ce domaine, puis nous présentons les méthodes récentes traitant de modèles procéduraux plus complexes et utilisant principalement l'apprentissage profond.

2.2.1 Premiers travaux

Dischler et Ghazanfarpour [DG97] proposent tout d'abord une technique d'analyse hybride de profils en combinant les domaines spatial et spectral afin de générer des textures procédurales. Cette méthode utilise pour le domaine spectral des transformées de Fourier qui permettent de décomposer des signaux pour isoler les fréquences, amplitudes et phases. Comme une texture possède des propriétés statistiques invariables dans l'espace, il est alors possible de capturer ces propriétés et de les reproduire pour générer une nouvelle texture similaire à un exemple en entrée. Pour le domaine spatial ce sont des histogrammes de niveaux de gris qui sont utilisés. Une nouvelle image de texture peut être générée à partir d'un exemple en entrée en environ deux minutes. Cette approche est cependant limitée aux signaux stationnaires et échouent à représenter des modèles caractérisés par des statistiques d'ordre élevé.

Lefebvre et Poulin [LP00] ont proposé une première approche afin d'estimer les paramètres de modèles procéduraux, et particulièrement pour des pavages rectangulaires et des écorces de bois. Leur objectif est alors de pouvoir extraire automatiquement depuis des photographies les valeurs des paramètres pouvant être utilisées dans des modèles procéduraux afin de générer des nouvelles images synthétiques. Leur ap-

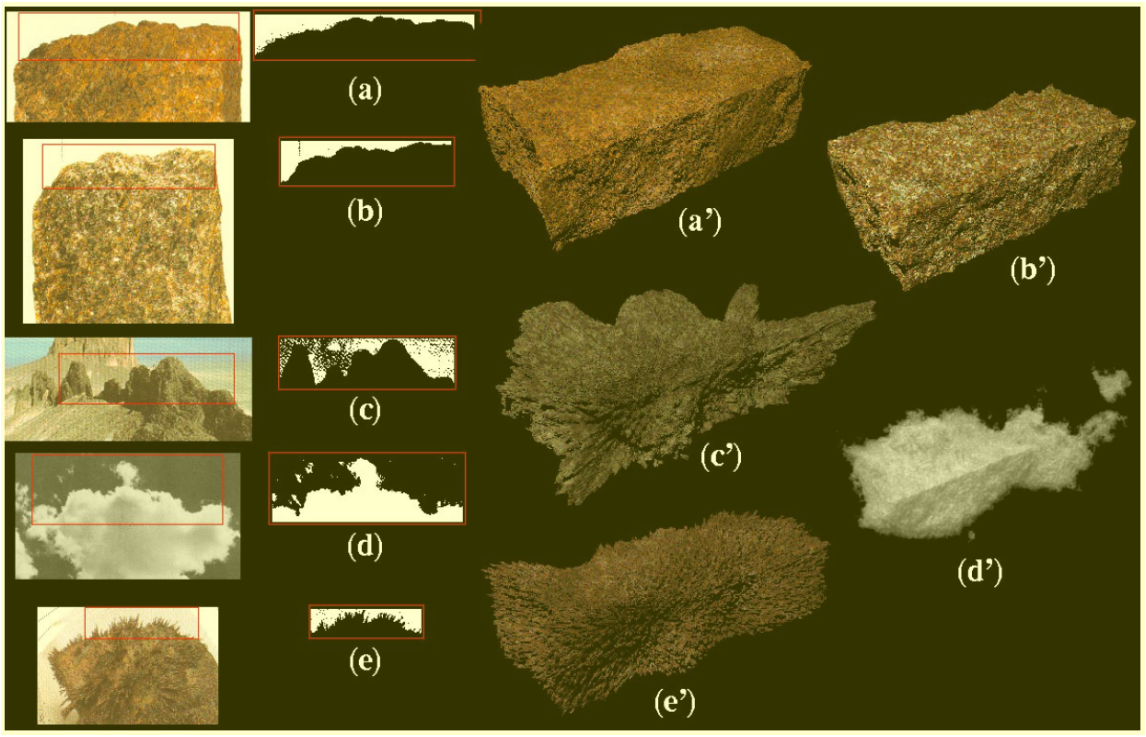

Figure 2.8 – Synthèse de textures à partir de profils extraits après une analyse hybride sur des photographies naturelles [DG97].

proche est interactive, l'utilisateur doit alors choisir le type de texture et construire un masque binaire de l'image en entrée qui sera la base des opérations effectuées dans leur méthode. Ainsi les différents paramètres vont être estimés en utilisant des outils d'analyse d'image ainsi qu'une transformée de Fourier rapide (FFT) sur le masque binaire. Finalement l'utilisateur peut à tout moment modifier les valeurs manuellement pour guider l'algorithme et appliquer des cartes de déplacement ou de *bump* ainsi qu'agir sur les paramètres de réflexion et de rugosité pour améliorer le réalisme de l'image générée (figure 2.9). Bien que cette approche se concentre sur deux types de structures, elle pourrait être généralisée selon les auteurs à d'autres modèles procéduraux grâce à l'utilisation des fréquences des images. Cette méthode prendrait entre quelques minutes jusqu'à une heure pour synthétiser une nouvelle image après avoir estimé les paramètres du modèle procédural selon le type de structures et sa complexité.

Bourque et Dudek [BD04] cherchent quant à eux à obtenir le modèle procédu-

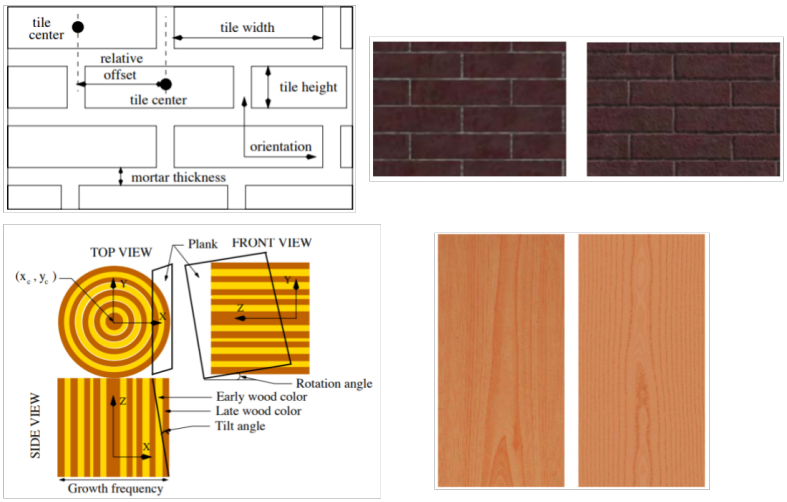

Figure 2.9 – Estimation de paramètres de modèles procéduraux et synthèse de textures. En haut : modélisation procédurale de pavages rectangulaires et de textures générées avec une carte de bosses et une carte de déplacement ; en bas : modélisation procédurale de bois avec un exemple naturel en entrée et la texture correspondante générée [LP00].

ral le plus approprié permettant de présenter à l'image d'exemple en entrée, ainsi que ses paramètres associés. Ainsi une base de données de modèles procéduraux est mise en place et une recherche du plus proche voisin est effectuée pour trouver les modèles permettant d'obtenir les apparences les plus similaires à l'image en entrée, cette recherche se repose sur une fonction de similarité perceptuelle construite à partir des transformées de Fourier et des histogrammes laplaciens des images. Une fois la sélection des modèles les plus performants effectuée, une phase d'optimisation est appliquée sur les paramètres en utilisant soit l'algorithme du simplexe soit une descente de gradient dans le but de maximiser une fonction de similarité perceptuelle (figure 2.10). Une dizaine de minutes est nécessaire pour obtenir le modèle procédural et ses paramètres. De plus si l'exemple en entrée est trop éloigné des structures représentables par les modèles procéduraux contenus dans la base de données, les résultats ne seront pas concluants.

Gilet et al. [GD10] proposent une méthode pour générer des détails procéduraux volumétriques à partir d'un exemple de photographie en entrée et une approximation de la forme de l'objet représenté pour les appliquer sur une nouvelle image. Une somme de fonctions de bruits est utilisée pour générer ces détails, les paramètres

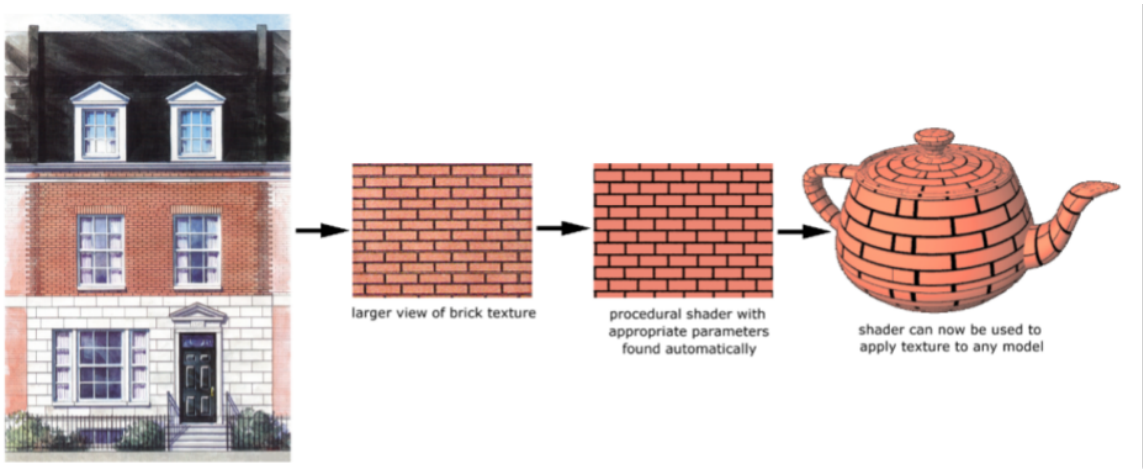

Figure 2.10 – Estimation de paramètres de *shaders* pour générer une image ressemblant à l'image d'exemple en entrée [BD04].

de ces fonctions sont calculés par une optimisation par descente de gradient. Cette phase d'optimisation cherche à maximiser une fonction de similarité entre l'image d'entrée et la nouvelle image générée, cette fonction étant fondée sur un mélange de filtres de Gabor et de transformées de Fourier fenêtrées car ils permettent d'unifier les domaines spatial et spectral (figure 2.11).

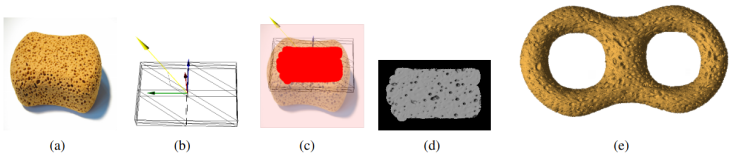

Figure 2.11 – Génération de détails procéduraux volumétriques à appliquer sur un nouvel objet, à partir d'un exemple de photographie (a), d'une forme approximative de l'objet (b) et de la zone de l'objet à considérer (c) pour donner une représentation procédurale selon les paramètres des fonctions de bruits estimés (d) à appliquer sur un autre objet (e) [GD10].

Galerne et al. [Gal+12] mettent ensuite en place une méthode pour générer des textures stochastiques à partir d'une image d'exemple en entrée en utilisant du bruit de Gabor. Le spectre de puissance de l'exemple en entrée est décomposé en une somme de gaussiennes, puis un algorithme d'optimisation convexe est utilisé pour estimer les paramètres des noyaux de Gabor d'un bruit afin que son spectre de

puissance s'approche de celui de l'exemple, cette estimation de paramètres prendrait environ deux minutes. L'utilisation du bruit de Gabor permet une plus grande diversité de structures représentables qu'avec le bruit de Perlin pour des textures stochastiques (figure 2.12).

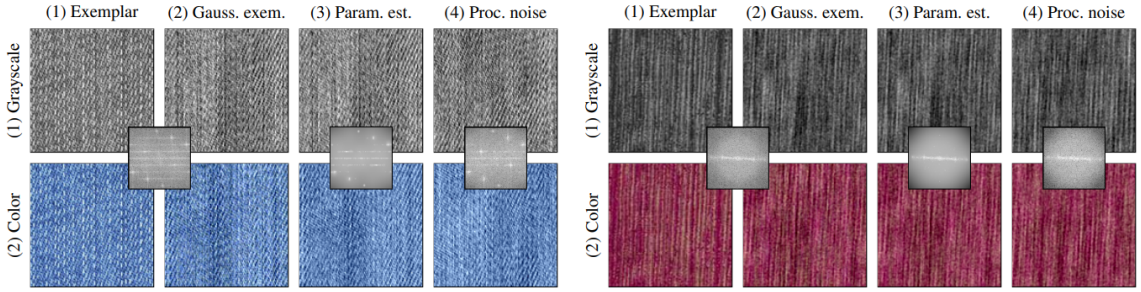

Figure 2.12 – Synthèse de bruit par l'exemple avec du *Gabor Noise* avec les exemples en entrée (1), leurs versions gaussiennes (2), les textures correspondant aux spectres de puissances estimés par estimation des paramètres (3) et les bruits procéduraux obtenus par synthèse (4) [Gal+12].

Gilet et al. [Gil+14] proposent le *Local Random-Phase Noise* pour générer des textures structurées à partir d'exemples en entrée avec un modèle fondé sur des séries de Fourier. La structure de l'exemple en entrée est récupérée grâce à un échantillonnage séparé des domaines spectral et spatial, permettant alors d'appliquer cette structure à une nouvelle image tout en faisant varier l'apparence finale, pour ne plus être restreint uniquement à des textures possédant des motifs aléatoires (figure 2.13). Cette méthode peut avoir des difficultés à récupérer la structure de l'exemple en entrée si elle s'étend sur l'entièreté du domaine spectral et non dans une région précise.

(a) Noise based on a continuous target spectrum.

(b) Structure-preserving procedural texture deduced from an example.

Figure 2.13 – Génération de textures par l'exemple avec le *Local random-phase noise* permettant d'intégrer la structure de l'exemple en entrée [Gil+14].

Par la suite, Pavie et al. [Pav+16] proposent d'améliorer ce *Local Random-Phase Noise* en présentant le *Locally Controlled Spot Noise* qui modifie les bruits locaux

utilisés pour introduire des noyaux gaussiens, ce qui permet de réduire le coût de calcul et augmente le contrôle de l'utilisateur sur les paramètres. Ainsi une plus grande étendue de motifs représentables est possible, le modèle ne cherchant plus à faire correspondre des spectres de puissance mais plutôt à faire correspondre la structure de l'image générée avec celle de l'image d'exemple par interaction avec l'utilisateur et segmentation automatique pour isoler la structure, ce qui représente une tâche difficile à implémenter pour obtenir des résultats satisfaisants (figure 2.14).

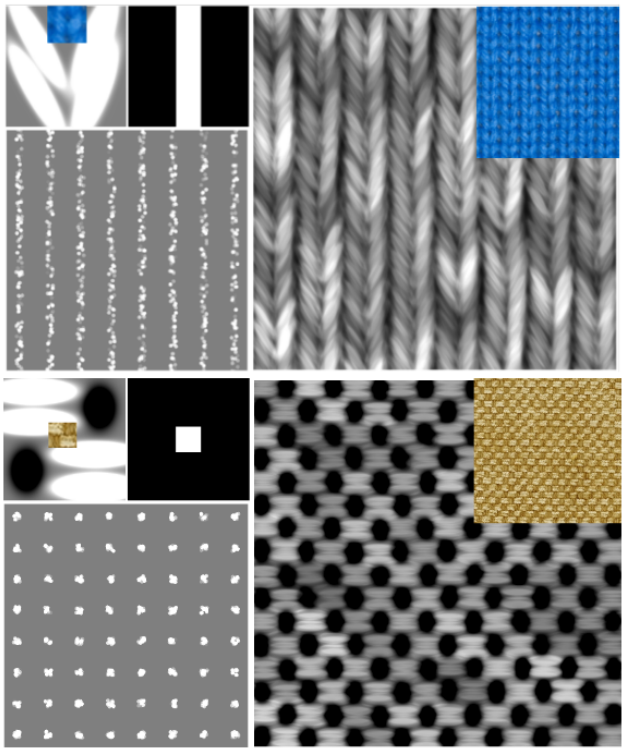

Figure 2.14 – Génération de textures par l'exemple avec le *Locally Controlled Spot Noise* utilisant des noyaux gaussiens, avec le profil du noyau et l'exemple en entrée (colonne de gauche, en haut à gauche), le profil de la distribution (colonne de gauche, en haut à droite), la distribution des impulsions (gauche), le bruit généré (droite) et l'image de texture générée (en haut à droite) [Pav+16].

2.2.2 Méthodes pour les matériaux procéduraux

L'émergence de l'apprentissage profond apporte de nouvelles possibilités pour la modélisation procédurale inverse, avec des avancées à la fois pour les textures et les matériaux procéduraux.

Zsolnai-Fehér et al. [ZWW18] utilisent le *machine learning* pour estimer les paramètres d'un modèle procédural de matériau, à travers une Régression par Processus gaussien (*gaussien Process Regression*, GPR). Cette approche s'appuie sur une sélection de préférences d'apparences du matériau par l'utilisateur, correspondant à différents vecteurs de paramètres. Sur la base de ces vecteurs de paramètres, la GPR permet de construire un espace latent interpolant ces vecteurs pour proposer à l'utilisateur une exploration interactive permettant d'affiner la sélection du matériau (figure 2.15). Les matériaux sont rendus en illumination globale en utilisant une approximation obtenue à l'aide d'un CNN pour accélérer le rendu. Il ne s'agit pas d'une méthode automatique qui s'appuie sur un exemple sous la forme d'une image, mais d'une méthode interactive qui permet de régler les paramètres du matériau procédural de façon plus intuitive et efficace qu'un réglage manuel.

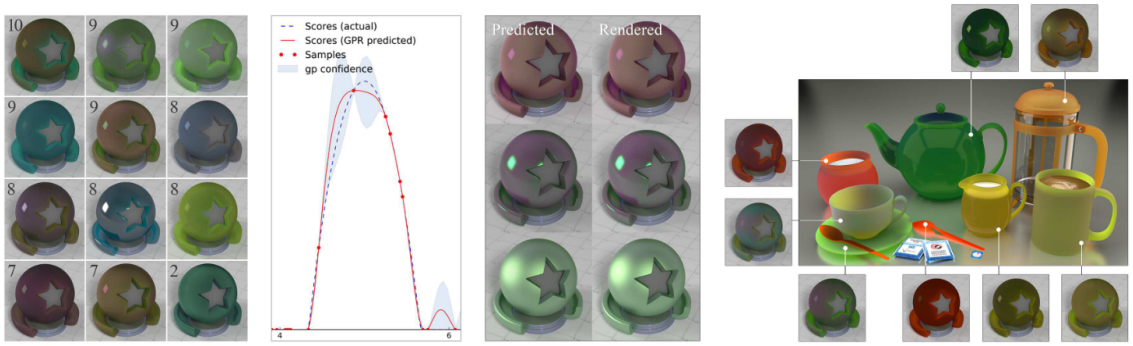

Figure 2.15 – Synthèse de matériaux à partir de processus gaussiens reposant sur les préférences de l'utilisateur [ZWW18].

Hu et al. [HDR19] proposent, à notre connaissance, la première méthode permettant d'obtenir un modèle procédural de texture à partir d'un exemple sous la forme d'une image, sans connaissance a priori sur les caractéristiques de la texture. La méthode utilise une base de données de modèles de textures procéduraux, à partir desquels des classes sont définies. La classification est obtenue par *clustering*, sur

la base de rendus des textures, en créant des *feature vectors*[2] à l'aide d'un CNN d'architecture VGG19. Pour chaque modèle associé aux classes définies, un CNN d'architecture AlexNet est entraîné pour estimer par régression les paramètres correspondants à une image donnée en entrée. Afin de combler l'écart pouvant exister entre l'apparence de la texture fournie en exemple et l'apparence du modèle procédural le plus proche, une phase optionnelle de transfert de style [GEB16] est appliquée (figure 2.16). Cette méthode nécessite l'entraînement d'un CNN par modèle procédural afin d'estimer les paramètres appropriés ce qui demande une grande quantité de mémoire pour stocker les réseaux pré-entraînés, et rend alors la méthode dépendante des modèles procéduraux qui ont été implémentés et de l'échantillonnage des paramètres.

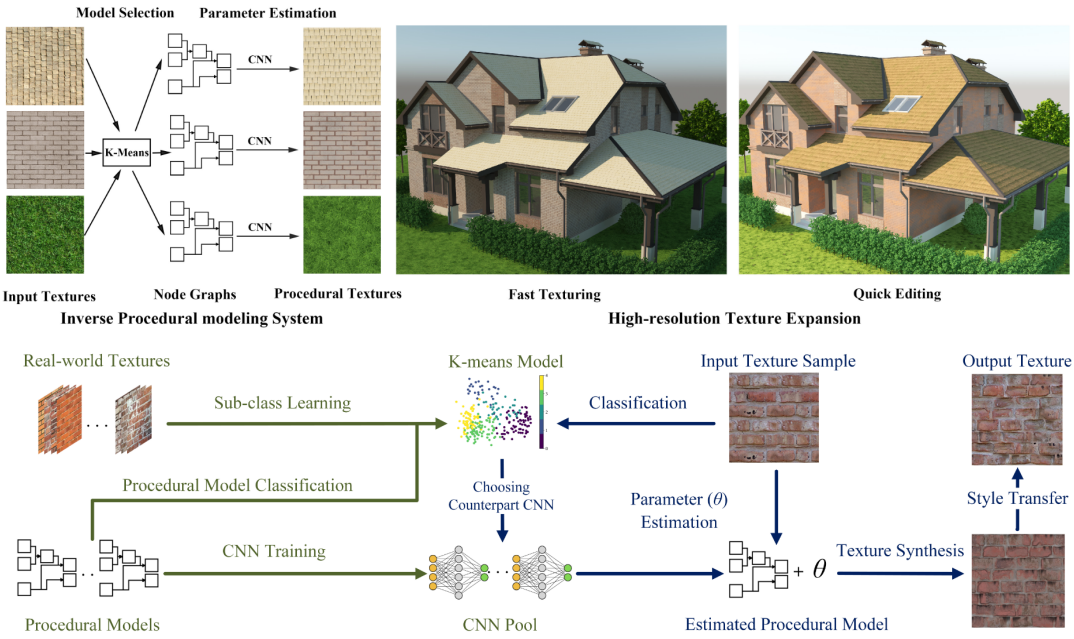

Figure 2.16 – Modélisation procédurale inverse de textures en utilisant une classification puis des CNN pour estimer les paramètres [HDR19].

De manière différente, Guo et al. [Guo+20a] utilisent une approche bayésienne pour estimer les paramètres d'un modèle procédural de matériau à partir d'une image d'exemple. Cette méthode met en œuvre une optimisation fondée sur un échantillonnage de l'espace des paramètres du modèle procédural avec une méthode de Monte-

2. L'expression *feature vector* est volontairement conservée en anglais. Elle peut être traduite par « vecteur de caractéristiques », ou bien « vecteur caractéristique ».

Carlo par chaînes de Markov. Une distribution de probabilité des paramètres est définie et utilisée avec les erreurs sur les distributions de probabilités des paramètres entre les exemples en entrée et les images rendues. L'optimisation est alors effectuée sur les paramètres continus et discrets à partir de plusieurs méthodes, comme l'algorithme de Metropolis-Hastings, le Monte-Carlo hamiltonien et le Langevin Metropolis-ajusté, avec une fonction d'erreur fondée sur les matrices de Gram construites à partir des feature vectors extraits depuis un réseau VGG19, une matrice de Gram étant formée par l'ensemble des produits scalaires calculés entre les feature vectors. Finalement une phase d'interaction avec l'utilisateur est possible pour sélectionner parmi une liste de sorties, celle qui correspond le mieux à l'exemple en entrée (figure 2.17). La méthode n'implique pas d'entraînement, et nécessiterait au moins 3 minutes pour estimer le matériau et ses paramètres à partir d'un exemple en entrée.

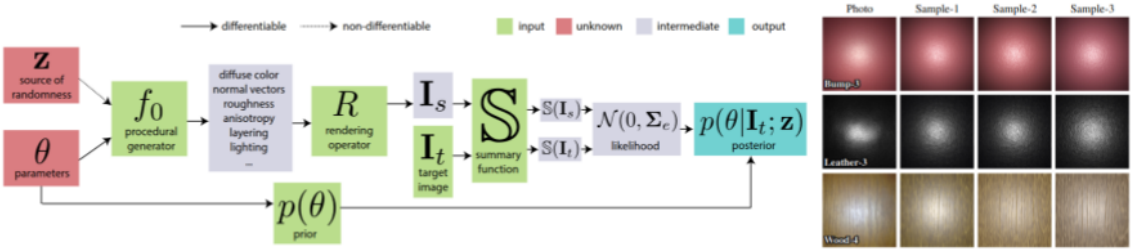

Figure 2.17 – Approche bayésienne fondée sur la méthode de Monte-Carlo par chaînes de Markov [Guo+20a].

Shi et al. [Shi+20] introduisent la notion de différentiabilité dans la modélisation procédurale inverse. Ils cherchent à pouvoir retrouver à partir de photos de matériaux les modèles de matériaux procéduraux appropriés. Ainsi une base de données est construite à partir d'une grande quantité de modèles procéduraux qui sont transformés en graphes de nœuds différentiables, les opérations effectuées par les nœuds dans des graphes pouvant être représentées de manière différentiable grâce à des méthodes d'apprentissage profond. Leur approche est composée de deux étapes, dans une première étape une sélection de plusieurs graphes de la base de données qui sont les plus appropriés pour reproduire l'image en entrée est effectuée. Pour ce faire les caractéristiques de l'image en entrée sont extraites avec un réseau VGG19 pré-entraîné et des matrices de Gram sont formées, qui sont ensuite passées en entrée d'un réseau à trois couches entièrement connectées pour obtenir une estimation des paramètres continus de chaque graphe de la base de données. Les images générées avec chaque graphe sont ensuite comparées avec l'image en entrée à l'aide d'une erreur fondée sur le transfert de style [GEB16] et les graphes dont les images donnent les erreurs

les plus basses sont conservées. La deuxième étape consiste en une optimisation des valeurs de ces paramètres continus grâce à une descente de gradient stochastique en minimisant l'erreur fondée sur le transfert de style. Cette étape est permise grâce à la différentiabilité par rapport à la totalité des paramètres des graphes de nœuds (figure 2.18). Cette approche est dépendante des graphes présents dans la base de données, ainsi si l'exemple en entrée n'est pas représentable entièrement par les graphes utilisés les résultats peuvent être mauvais.

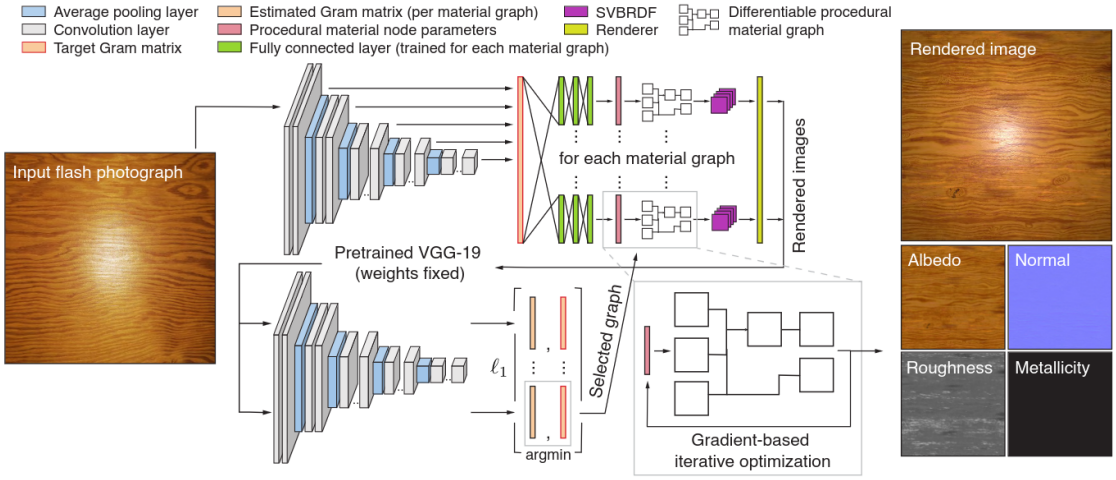

Figure 2.18 – Synthèse de matériaux procéduraux à partir de graphes de nœuds de matériaux traduits en graphes de nœuds différentiables afin de pouvoir effectuer une optimisation des paramètres [Shi+20].

Hu et al. [Hu+22c] ont proposé *DiffProxy*, un réseau de neurones génératif qui permet d'obtenir un modèle entièrement différentiable à partir de fonctions non-différentiables comme certaines fonctions impliquées dans les nœuds des graphes de matériaux procéduraux. DiffProxy repose sur un réseau génératif d'architecture StyleGAN2 qui est entraîné pour apprendre les correspondances entre des images générées par un modèle procédural et les paramètres de ce modèle. Le réseau entraîné est ensuite utilisé dans une phase d'optimisation par une descente de gradient stochastique pour estimer les valeurs de tous les paramètres, continus et discrets, du modèle procédural afin de générer une image approchant l'exemple en entrée. La fonction d'erreur de cette optimisation est fondée sur les matrices de Gram construites à partir des feature vectors extraits d'un réseau VGG19 pré-entraîné. Un temps d'exécution d'entre 3 à 5 minutes est nécessaire pour un exemple sur un graphe de matériau procédural donné, ou un modèle procédural comme la PPTBF (figure 2.19). Cette

méthode présente le même inconvénient que les méthodes précédentes de dépendance aux images présentes dans le jeu de données d'entraînement et à l'échantillonnage des paramètres utilisés pour les générer.

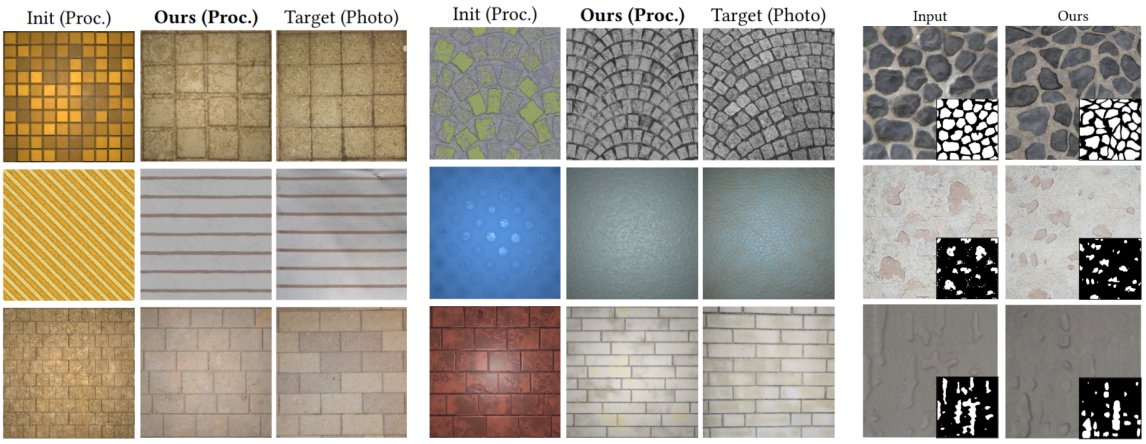

Figure 2.19 – Synthèse de textures à partir du réseau génératif DiffProxy permettant un modèle entièrement différentiable et une optimisation des paramètres par descente de gradient stochastique [Hu+22c].

Zhou et al. [Zho+23] proposent une approche semi-procédurale et différentiable représentant un matériau par un ensemble de bruits et de structures procédurales simples définis par des fonctions différentiables par rapport à leur paramètres. Parmi l'ensemble des fonctions de bruit et de structures présents dans une base de données, l'utilisateur sélectionne deux structures qui s'approchent de l'image d'exemple en entrée, puis trois bruits sont choisis aléatoirement dans la même base de données. Une phase d'optimisation par descente de gradient stochastique est ensuite effectuée entre l'image générée par les motifs et bruits choisis et l'exemple en entrée reposant sur une erreur portant sur les matrices de Gram des caractéristiques extraites des premières couches de chacun des cinq blocs d'un réseau VGG19 pré-entraîné. Il y a finalement une phase d'édition durant laquelle l'utilisateur peut choisir de modifier les motifs et bruits définis initialement ou de fournir une carte de guidage pour aider l'optimisation, ce qui entraîne des nouvelles itérations de descente de gradient (figure 2.20). L'optimisation ne nécessite que quelques minutes et les modifications suite à une édition par l'utilisateur sont effectuées en quelques secondes. Néanmoins il est nécessaire de choisir des motifs initiaux appropriés pour obtenir de bons résultats à partir de l'exemple en entrée, et cette étape ne peut être faite que manuellement par l'utilisateur.

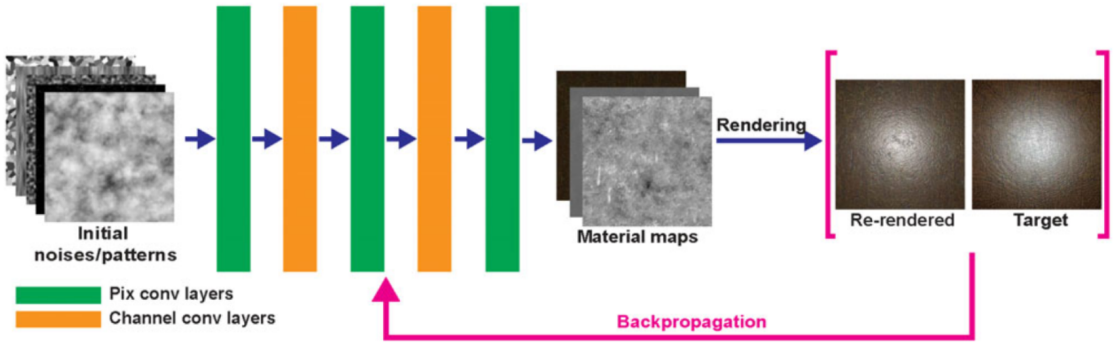

Figure 2.20 – Synthèse de matériaux à partir de motifs et bruits traduits en fonctions différentiables grâce à des couches de convolution pour permettre une optimisation pour un rendu différentiable [Zho+23].

Li et al. [LSM23] étendent la base de données construite par Shi et al. [Shi+20] en ajoutant davantage de nœuds générateurs et d'opérations issus des graphes de nœuds existants dans Substance 3D Designer. Ils utilisent ensuite une procédure d'optimisation en trois phases afin de générer des matériaux procéduraux approchant l'apparence de photographies, en optimisant tous les paramètres qu'ils soient discrets ou continus. Dans la première phase ils cherchent à optimiser une première fois les valeurs des paramètres continus pour réduire l'écart d'apparence entre l'exemple d'entrée et le matériau généré avec une descente de gradient fondée sur une erreur portant sur les matrices de Gram construites à partir des feature vectors extraits d'un réseau VGG19 pré-entraîné. Dans la deuxième phase une optimisation mixte des paramètres discrets et de certains paramètres continus est effectuée sans gradient avec une approche à directions de descente combinant une optimisation bayésienne, du recuit simulé et un algorithme génétique. Cette phase cherche à minimiser une erreur portant sur les matrices de Gram et sur les transformées de Fourier rapides. Finalement une dernière étape de raffinement est appliquée par descente de gradient pour optimiser à nouveau les paramètres continus en ayant maintenant une meilleure estimation des valeurs des paramètres discrets. Les auteurs indiquent alors des meilleurs résultats que ceux obtenus par Shi et al. [Shi+20] et Hu et al. [Hu+22c] comme illustré dans la figure 2.21.

Récemment, Hu et al. [Hu+23] ont développé le premier modèle capable de directement générer des graphes de nœuds de matériaux à partir de plusieurs types d'entrées, comme des images, du texte ou des graphes de nœuds partiels. Ils utilisent

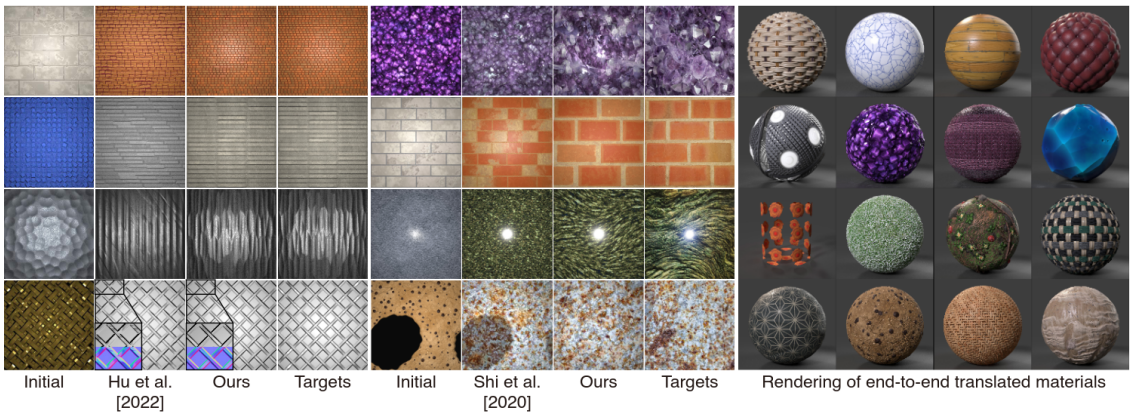

Figure 2.21 – Synthèse de matériaux procéduraux à partir de graphes de nœuds de matériaux étendus par rapport à [Shi+20] avec une optimisation des paramètres discrets et continus en trois phases [LSM23].

un modèle génératif conditionnel multi-modal inspiré de MatFormer [Gue+22] et composé de transformeurs adaptés selon la nature des entrées. Le modèle est alors entraîné sur un ensemble de graphes de nœuds appairés avec leurs cartes de matériaux 2D correspondantes, et échantillonnés avec une multitude de jeux de paramètres différents pour chacun des graphes (figure 2.22). Une phase d'optimisation peut ensuite être effectuée pour affiner les résultats obtenus, les auteurs préconisant une erreur fondée sur une similarité cosinus CLIP, c'est-à-dire une erreur fondée sur la similarité entre une image et une légende, ou une erreur fondée sur une distance de Wasserstein tranchée pour comparer deux images. L'entraînement des différentes parties du modèle prend environ une semaine. Cette approche est limitée par la quantité de données utilisée pour l'entraînement des réseaux, étant donné que chaque graphe de nœuds présent dans la base de données d'entraînement est créé manuellement par un expert.

2.3 Méthodes d'estimation de paramètres du modèle de PPTBF

Dans le chapitre d'introduction, nous avons introduit le modèle procédural de PPTBF de Guehl et al. [Gue+20 ; Gue22]. Nous allons ici détailler les différentes méthodes qui ont été mises en place pour estimer les paramètres de ce modèle.

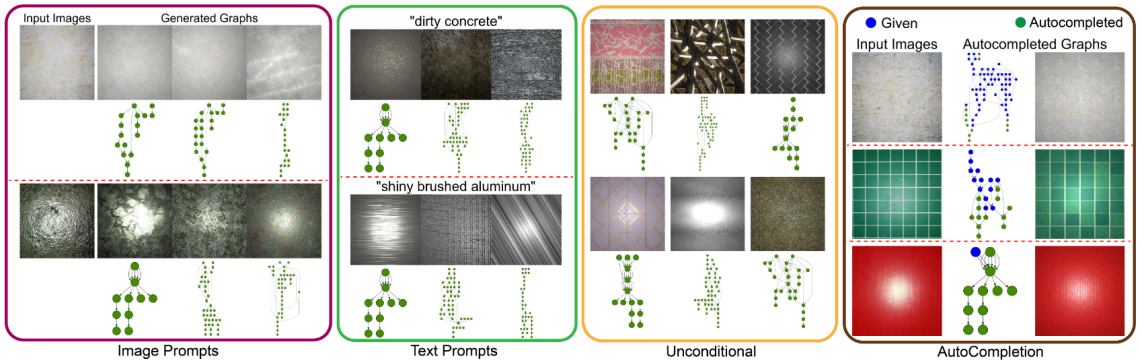

FIGURE 2.22 – Génération de graphes de nœuds de matériaux grâce à un réseau génératif à partir d'images, de textes ou de graphes de nœuds partiels [Hu+23].

Afin d'estimer les paramètres de leur modèle, Guehl et al. [Gue+20 ; Gue22] ont proposé une méthode reposant sur la création d'une base de données de 450 000 images de structures binaires avec trois valeurs de seuils différentes, à partir d'images de PPTBF dont l'espace des paramètres est échantillonné pour obtenir une grande variété de structures différentes tout en excluant les images qui seraient trop similaires. Puis des feature vectors sont calculés pour chaque image afin de servir de descripteur du contenu des images. Plusieurs types de descripteurs sont calculés : des *Local Binary Pattern*, des *Gabor Binary Pattern*, des Transformées de Fourier rapides, des histogrammes de filtres de Gabor, et des feature vectors extraits à partir d'un réseau de neurones VGG19 pré-entraîné. Une fois que cette base de données d'images et de feature vectors est construite une recherche des plus proches voisins fondée sur une norme L2 est effectuée entre les feature vectors de l'exemple en entrée et ceux de l'intégralité des images de la base de données afin de trouver les structures les plus ressemblantes, et donc leurs paramètres associés. Finalement l'utilisateur peut sélectionner la structure qu'il souhaite et la transformer en modifiant ses paramètres associés manuellement pour essayer d'approcher davantage l'exemple en entrée. Quelques résultats de cette méthode sont disponibles dans la figure 2.23. Cette méthode n'est pas entièrement automatique car elle nécessite de l'interaction avec l'utilisateur pour obtenir des résultats satisfaisants. Elle est coûteuse en mémoire car elle nécessite de construire une grande base de données d'images et de feature vectors, et en temps car il faut jusqu'à cinq minutes d'interaction pour un seul exemple. Finalement elle est dépendante de la précision de l'échantillonnage des paramètres des images de la base de données.

Hu et al. [Hu+22a] utilisent le modèle de PPTBF pour générer des masques bi-

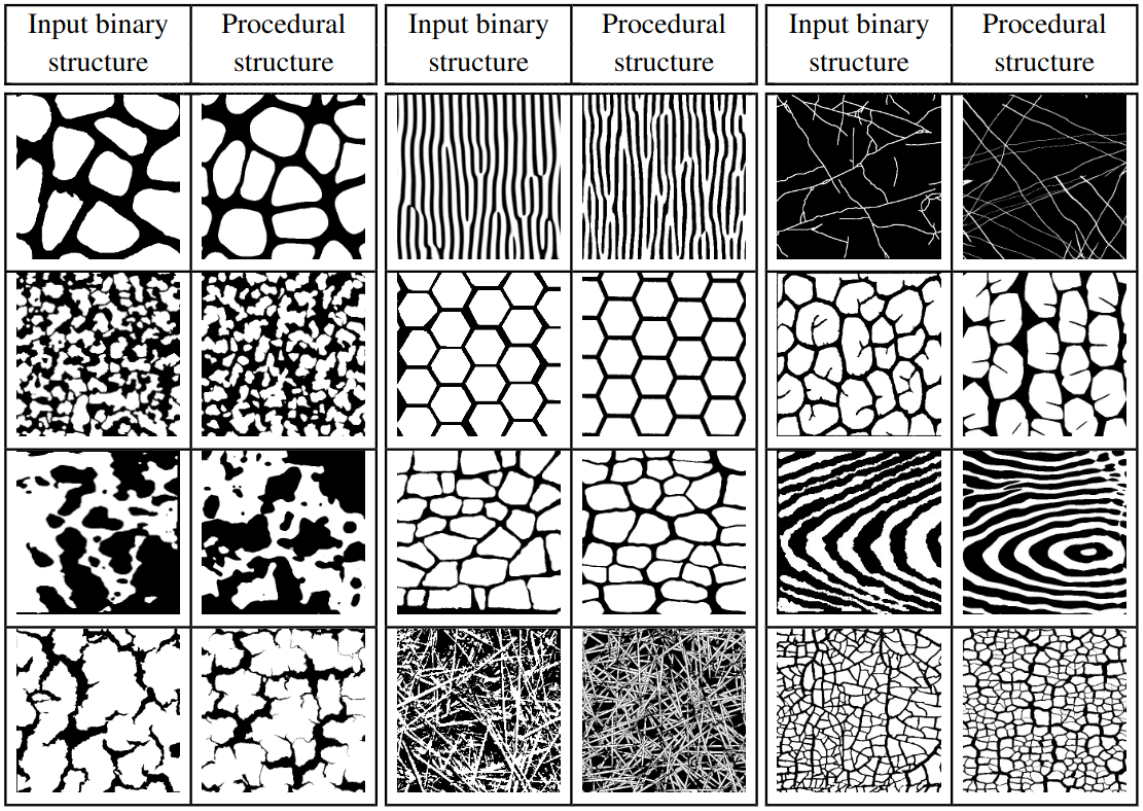

Figure 2.23 – Résultats de la méthode d'estimation proposée par [Gue+20] pour obtenir à partir du modèle de PPTBF des structures binaires ressemblant à des cartes de structures binaires en entrée.

naires dont les paramètres sont estimés par une méthode similaire avec une base de données de masques puis une recherche du plus proche voisin reposant sur une distance L_2, en utilisant cette fois-ci des feature vectors concaténés à partir de *Local Binary Pattern*, de matrices de Gram issues des caractéristiques extraites à partir d'un réseau VGG19 pré-entraîné et de spectres de puissances de Fourier. Une analyse en composantes principales (PCA) est ensuite appliquée sur tous les vecteurs calculés pour chaque masque de la base de données afin de réduire leur dimensionnalité. Les auteurs indiquent que la construction de la base de données prendrait environ 36 heures mais qu'une recherche peut être effectuée en moins d'une seconde. Une phase d'optimisation sur CPU est ensuite effectuée sur les paramètres continus et discrets, le modèle n'étant alors pas différentiable, et prendrait alors environ 20 minutes. De plus, la base de données construite pour la recherche pèserait à peu près 7,8 Go en

mémoire.

Dans leur modèle DiffProxy, Hu et al. [Hu+22c] entraînent leur GAN de type StyleGAN2 pour apprendre les correspondances entre les paramètres du modèle de PPTBF et l'apparence des images de PPTBF. Par la suite leur optimisation par descente de gradient est possible car les paramètres discrets du modèle procédural sont estimés comme des paramètres continus, donc avec des valeurs flottantes. Les auteurs notent qu'il faut entre deux à trois jours pour entraîner leur réseau génératif, puis entre trois et cinq minutes pour que leur optimisation par le gradient s'effectue sur un seul exemple, ce qui donne un gain de temps considérable par rapport à la méthode précédente pour des résultats plutôt similaires comme illustrés dans la figure 2.24. De plus le coût en mémoire est réduit car seuls les paramètres du réseau entraîné sont à stocker. Cependant le fait d'estimer les paramètres discrets comme des paramètres continus peut être problématique, surtout si les valeurs qui se suivent de ces paramètres n'ont pas forcément de lien entre elles.

Les méthodes présentées dépendent entièrement de l'échantillonnage de l'espace des paramètres dans des intervalles choisis a priori, et peuvent échouer si l'exemple est trop éloigné visuellement des données utilisées pour l'entraînement.

2.4 Bilan

Les travaux sur les modèles procéduraux pour représenter des textures, matériaux, ou structures, ont évolué de deux façons. D'une part, les graphes de matériaux sont construits à partir d'un grand nombre de nœuds simples par des experts ou bien générés par des modèles génératifs. L'accès aux collections de graphes procéduraux, comme ceux produits à l'aide de Substance 3D Designer, n'est pas public. D'autre part, des travaux s'orientent vers des modèles plus généralistes comme le modèle de PPTBF pour représenter des structures stochastiques.

Nous pouvons constater dans cet état de l'art que les premières méthodes de modélisation procédurale inverse à partir d'exemples en entrée se concentraient essentiellement sur l'estimation de paramètres de modèles procéduraux reposant sur des fonctions de bruit. Ces approches d'estimation de paramètres utilisaient des méthodes reposant sur le domaine spectral en analysant les fréquences et en utilisant des métriques fondées sur des transformées de Fourier ou des filtres de convolution.

Le développement de l'apprentissage profond a apporté de nouvelles perspectives pour la modélisation procédurale inverse sur différents aspects. Tout d'abord sur les outils utilisés pour l'estimation des paramètres, des CNN utilisant des architectures AlexNet ou VGG19 peuvent être utilisés pour estimer directement les paramètres de

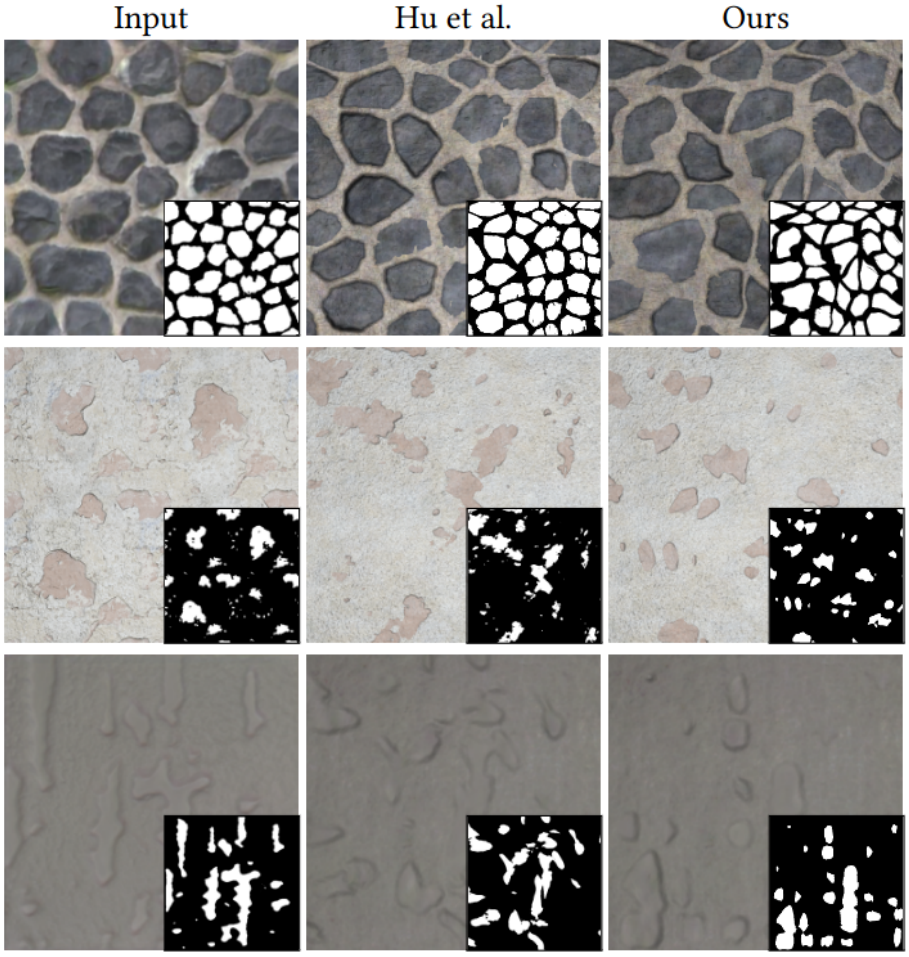

Figure 2.24 – Comparaison des résultats obtenus par [Hu+22a] (2ème colonne) en utilisant une recherche du plus proche voisin sur une base de données puis une optimisation simple des paramètres en 20 minutes et [Hu+22c] (3ème colonne) en utilisant un réseau génératif puis une optimisation par le gradient en 3 à 5 minutes [Hu+22c].

modèles procéduraux, mais aussi des réseaux génératifs pour générer des images en apprenant les correspondances entre apparences et paramètres des modèles procéduraux. L'entraînement de ces réseaux est néanmoins coûteux en temps, de quelques heures pour les CNN jusqu'à plusieurs jours pour les GAN, et en mémoire pour stocker les modèles entraînés. Des phases d'optimisation par le gradient sont ensuite effectuées grâce à la différentiabilité des modèles permise par l'utilisation des réseaux

de neurones, permettant d'estimer les valeurs de paramètres discrets. Les métriques d'erreur utilisées par ces phases d'optimisation influencent les résultats. Le domaine spectral n'est plus nécessairement analysé car des descripteurs d'images plus performants peuvent être récupérés et utilisés dans ces optimisations par le gradient. La métrique d'erreur la plus répandue étant alors la fonction d'erreur fondée sur les matrices de Gram construites à partir de feature vectors extraits depuis un réseau de neurones pré-entraîné, héritée des travaux de Gatys et al. [GEB15b]. Les réseaux de neurones utilisés pour extraire ces feature vectors sont des CNN comme AlexNet pour Huang et al. [Hua+17], mais surtout le réseau VGG19 qui contient davantage de couches et permet alors de mieux caractériser les images. Plus récemment la fonction d'erreur fondée sur la distance de Wasserstein tranchée est utilisée, par Hu et al. ([Hu+22b] et [Hu+23]), elle serait plus robuste que la métrique d'erreur fondée sur les matrices de Gram pour capturer toutes les statistiques stationnaires des images.

Dans cette thèse, nous nous intéressons plus particulièrement à l'estimation des paramètres du modèle de PPTBF, qui comporte des paramètres discrets et continus. Il n'est pas différentiable par rapport à ses paramètres discrets, ce qui rend nécessaire la construction d'une grande base de données d'images et de feature vectors coûteuse en temps et en mémoire, et rend impossible une optimisation par le gradient à partir d'une fonction d'erreur à minimiser. À notre connaissance, seule l'approche basée sur le modèle DiffProxy introduit par Hu et al. [Hu+22c] a été proposée pour résoudre ce problème, avec les limites exposées précédemment. Nous proposons une autre approche, spécialisée sur les structures stochastiques cellulaires, permettant de lever un certain nombre de limites des méthodes proposées précédemment pour estimer les paramètres du modèle de PPTBF.

Chapitre 3

Modèle de C-PPTBF

Dans cette partie nous présentons notre modèle procédural C-PPTBF (*Cellular Point Process Texture Basis Function*), qui correspond à une implémentation du modèle de PPTBF avec un processus ponctuel P permettant de réaliser des types de pavages adaptés pour couvrir une large gamme de structures stochastiques cellulaires, ainsi que des fonctions Window W et Feature F appropriées, tout en ayant des propriétés favorisant la mise en place d'une estimation des paramètres selon une approche par gradient à partir d'une fonction d'erreur.

3.1 Formulation générale

Pour chaque point $\mathbf{x}$ d'une image, la fonction C-PPTBF génère un ensemble de feature points à l'aide d'un processus ponctuel P. Puis cet ensemble est convolué à un produit de deux fonctions, les fonctions Window W et Feature F en prenant en compte les k feature points les plus proches voisins de $\mathbf{x}$. Ainsi la fonction C-PPTBF en un point $\mathbf{x}$ consiste en la somme de tous ces produits, telle que :

$$\text{C-PPTBF}(\theta_P,\, \theta_W,\, \theta_F; \mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})} w(\overrightarrow{\mathbf{x}_i\mathbf{x}})\, f(\overrightarrow{\mathbf{x}_i\mathbf{x}}), \tag{3.1}$$

où θ_P, θ_W et θ_F représentent respectivement les paramètres de P, W et F, $\mathbf{x}_i$ le voisin de $\mathbf{x}$ considéré et $\mathcal{N}_k$ l'ensemble des k feature points les plus proches voisins de $\mathbf{x}$. La valeur de k est dépendante du type de processus ponctuel utilisé.

3.2 Réalisation du processus ponctuel

Dans les images naturelles nous retrouvons principalement des motifs distribués de façon uniforme, agrégés ou selon une distribution en disques de Poisson ([DH06]). En s'appuyant sur cette observation nous pouvons conclure qu'on peut représenter une grande partie des structures stochastiques seulement avec ces trois distributions. Celles-ci ont l'avantage de pouvoir être représentées à l'aide d'un petit nombre d'algorithmes de pavages. Ainsi, dans le modèle de PPTBF initial, trois types de processus ponctuels stochastiques correspondant aux distributions précédentes étaient proposés pour donner un total de 17 types de pavages différents contrôlant la distribution des feature points dans l'image.

Dans le modèle de C-PPTBF nous nous concentrons sur des structures cellulaires, ainsi nous conservons deux des trois types de processus ponctuels stochastiques, les processus de Poisson et de Cox, correspondant alors à 14 types de pavage présentés dans la figure 3.1. Les types de pavages définis par les processus ponctuels de Poisson ont des feature points indépendants et distribués uniformément dans l'image car l'intensité est constante sur $\mathbb{R}^2$. Les types de pavage définis par les processus ponctuels de Cox ont quant eux des feature points qui ne sont pas distribués de manière uniforme et peuvent former des concentrations de points, l'intensité variant spatialement selon un bruit blanc.

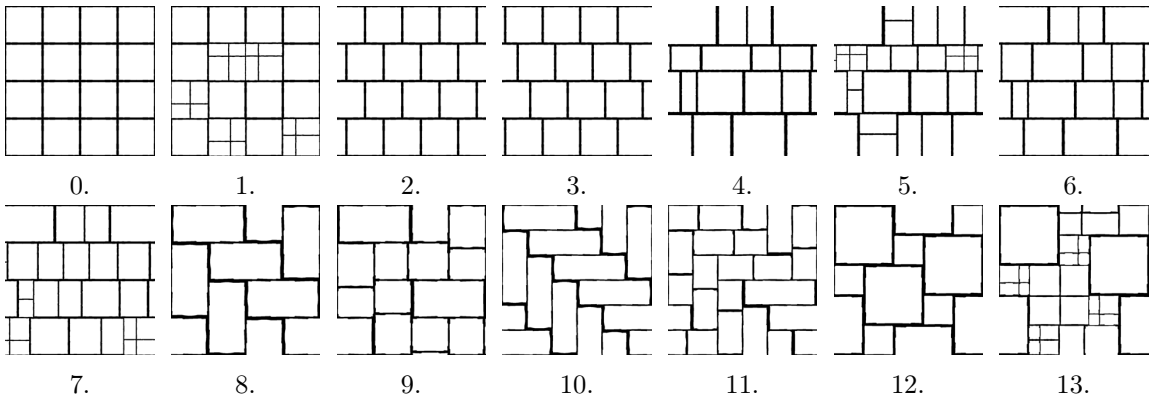

Figure 3.1 – Les 14 types de pavages implémentés dans le modèle de C-PPTBF.

Les feature points sont situés au centre des cellules générées par le type de pavage comme montré dans la figure 3.2. L'ensemble des paramètres du processus ponctuel θ_P comporte alors le type de pavage, qui est le seul paramètre discret du modèle de C-PPTBF, et le paramètre de perturbation β qui ajoute de l'aléatoire sur la position des feature points à l'intérieur des cellules, illustré dans la figure 3.3. Ce paramètre

influe alors sur la position des feature points comme une interpolation linéaire entre les positions des centres des cellules C définies par le type de pavage, et un ensemble de positions aléatoires calculées dans les cellules C_A^{jit} tel que :

$$P = (1 - \beta)\, C + \beta\, C_A^{jit} \tag{3.2}$$

où P représente l'ensemble des feature points.

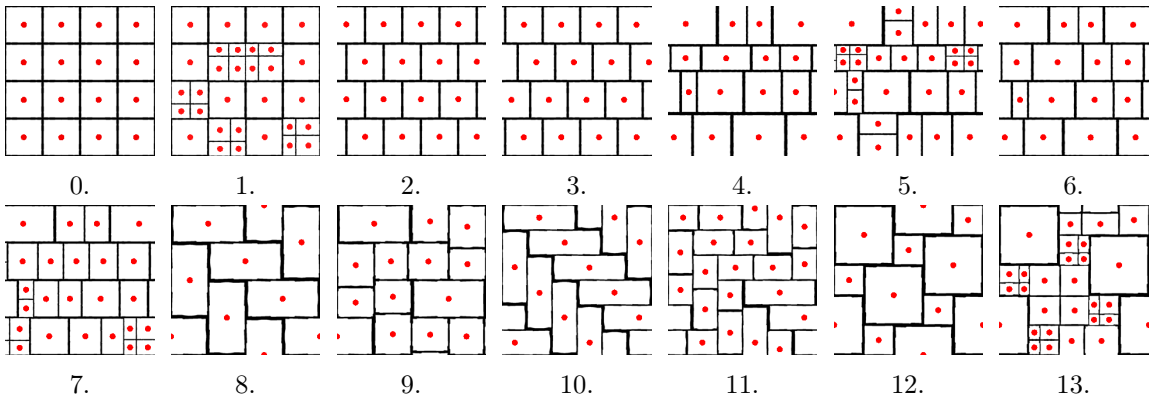

Figure 3.2 – Positions des centroïdes des cellules des 14 types de pavage.

L'ensemble des paramètres de l'étape du processus ponctuel est nommé θ_P, tel que $\theta_P = (tt, \beta)$.

Les 14 types de pavage sont prédéfinis à l'avance selon plusieurs paramètres qui permettent de contrôler la distribution des points qui en découle. Le premier paramètre utilisé est le paramètre *decalx* qui permet d'effectuer un décalage horizontal sur les cellules. L'influence de ce paramètre est visible dans les figures 3.1 et 3.2 pour les types de pavage 2 et 3 qui sont des modifications du type de pavage 0 avec du décalage.

On peut aussi effectuer des subdivisions horizontales et verticales sur les cellules avec les paramètres *subx* et *suby* qui sont des probabilités régissant ces subdivisions. Deux autres paramètres, *jitx* et *jity*, permettent de contrôler la position horizontalement et verticalement de ces subdivisions par rapport aux centres des cellules en interpolant linéairement entre les centroïdes et les positions aléatoires C_A^{jit}. L'influence de ces paramètres est visible dans les figures 3.1 et 3.2 par exemple pour les types de pavage 1 et 13, qui sont respectivement des modifications des types de pavage 0 et 12 avec des probabilités de 50% pour les subdivisions horizontales et verticales.

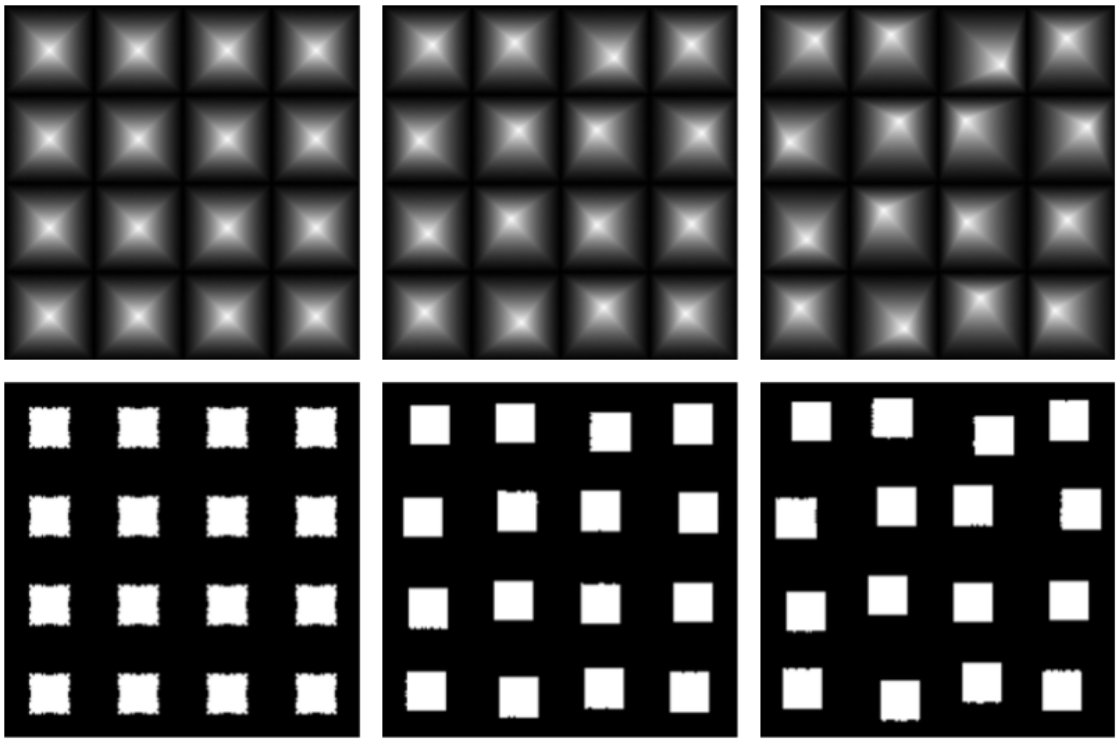

Figure 3.3 – Influence du paramètre de perturbation β sur la position des centres des cellules de pavage. De gauche à droite les valeurs de β sont respectivement 0, 0,25 et 0,5.

Le déroulement du processus ponctuel est résumé dans la figure 3.4 et l'algorithme 1.

3.3 Fonction Window

Une fois que la position des feature points a été définie nous pouvons convoluer cet ensemble de points à un produit de deux fonctions, la fonction Window et la fonction Feature. La fonction Window tout d'abord permet de déterminer les interactions entre les motifs locaux pour construire des cellules autour des points. Elle est définie comme une interpolation linéaire entre une fenêtre cellulaire w_c et une fenêtre de mélange w_m :

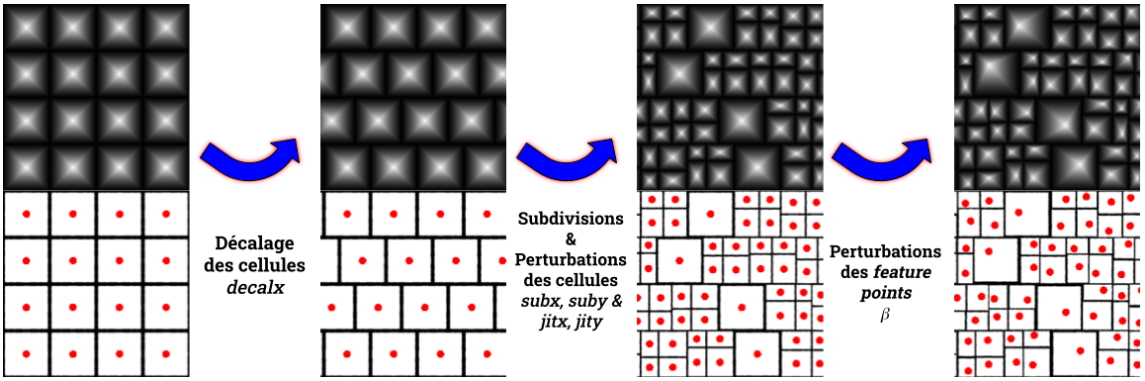

FIGURE 3.4 – Illustration du processus ponctuel avec application du décalage *decalx*, des subdivisions des cellules avec les paramètres *subx* et *suby* d'une probabilité de 75%, des perturbations des subdivisions *jitx* et *jity* d'une valeur de 0,9, et de perturbations des feature points d'une valeur de 0,5.

$$\text{C-PPTBF}(\theta_P,\, \theta_W,\, \theta_F; \mathbf{x}) = \omega\, w_c(\overrightarrow{\mathbf{x}_1\mathbf{x}}) f(\overrightarrow{\mathbf{x}_1\mathbf{x}}) + (1-\omega) \sum_{\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})} w_m(\overrightarrow{\mathbf{x}_i\mathbf{x}}) f(\overrightarrow{\mathbf{x}_i\mathbf{x}}) \quad (3.3)$$

avec ω le coefficient d'interpolation entre les deux fenêtres, qui permet alors de contrôler l'influence des deux fenêtres et donc de déterminer si le contenu des cellules est plus ou moins fusionné, comme illustré dans la figure 3.5.

La **fenêtre cellulaire** w_c est définie en calculant une distance aux bords des cellules définies par le pavage et une distance aux bords de cellules de Voronoï construites autour des feature points. Elle définit alors si les cellules vont plutôt avoir une forme rectangulaire ou ressembler à des cellules de Voronoï. Elle est notée de la sorte :

$$w_c(\overrightarrow{\mathbf{x}_1\mathbf{x}}) = \frac{\exp(-d_{nrc}(\overrightarrow{\mathbf{x}_1\mathbf{x}})) - \exp(-1)}{1 - \exp(-1)} \quad (3.4)$$

d_{nrc} représente une fonction de distance cellulaire radiale normalisée. Elle se calcule à partir de $\mathbf{r}(\mathbf{x})$ qui est l'intersection du segment partant de $\mathbf{x}_1$, le feature point le plus proche de $\mathbf{x}$, et passant par $\mathbf{x}$, avec le bord de la cellule de Voronoï construite autour de $\mathbf{x}_1$. $\mathbf{r}(\mathbf{x})$ se notant alors de la sorte :

$$\mathbf{r}(\mathbf{x}) = \left\{ \mathbf{q} \in \mathbb{R}^2 \,\middle|\, \frac{\overrightarrow{\mathbf{x}_1\mathbf{x}}}{\|\overrightarrow{\mathbf{x}_1\mathbf{x}}\|} \cdot \frac{\overrightarrow{\mathbf{x}_1\mathbf{q}}}{\|\overrightarrow{\mathbf{x}_1\mathbf{q}}\|} = 1 \right\} \cap \partial \text{Vor}_P(\mathbf{x}_1) \quad (3.5)$$

Algorithme 1 Algorithme du Processus ponctuel

1: Application des transformations géométriques et des déformations sur le pixel $\mathbf{x}$ considéré ;
2: Placement des feature points selon le type de pavage influant sur la position et le paramètre *decalx* décalant la position des feature points sur l'axe x. Les positions des feature points et de leurs versions perturbées sont alors obtenues ;
3: Construction des cellules de pavage autour des feature points selon les paramètres *subx* et *suby* pour créer des subdivisions dans les cellules, et les paramètres *jitx* et *jity* pour contrôler la position des subdivisions dans les cellules. Les distances des feature points aux bords des cellules sont alors obtenues ;
4: Interpolation linéaire entre les positions des feature points et les positions perturbées aléatoirement selon le paramètre de perturbation β pour obtenir les positions finales des feature points ;

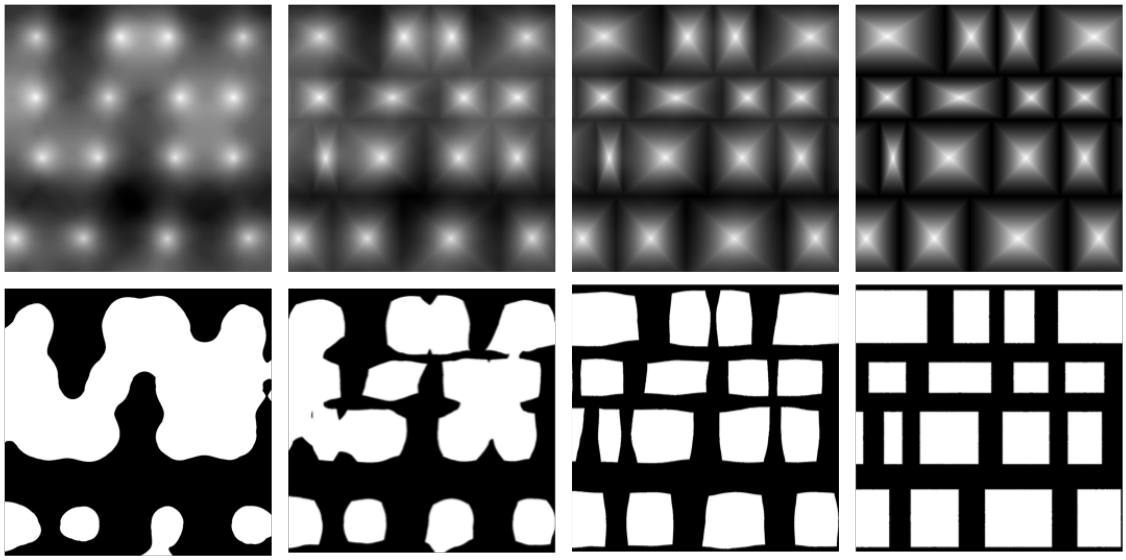

Figure 3.5 – Influence du paramètre d'interpolation ω entre la fenêtre de mélange et la fenêtre cellulaire. De gauche à droite les valeurs de ω sont respectivement 0, 0,33, 0,66 et 1.

avec :

$$\mathrm{Vor}_P(\mathbf{x}_1) = \left\{ \mathbf{q} \in \mathbb{R}^2 \middle| \forall \mathbf{p} \in P, d_c(\overrightarrow{\mathbf{x}_1\mathbf{q}}) \leq d_c(\overrightarrow{\mathbf{p}\mathbf{q}}) \right\}. \tag{3.6}$$

Ainsi $d_{nrc}(\overrightarrow{\mathbf{x}_1\mathbf{x}})$, illustrée dans la figure 3.6, se calcule comme une distance radiale

de $\mathbf{x}$ au bord de la cellule de Voronoï de $\mathbf{x}_1$:

$$d_{nrc}(\overrightarrow{\mathbf{x}_1\mathbf{x}}) = 1 - \frac{\|\overrightarrow{\mathbf{x}_1\mathbf{x}}\|}{\|\overrightarrow{\mathbf{x}_1\mathbf{r}(\mathbf{x})}\|} \tag{3.7}$$

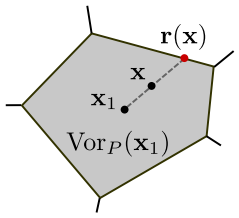

FIGURE 3.6 – Illustration de la distance radiale $d_{nrc}(\overrightarrow{\mathbf{x}_1\mathbf{x}})$ [BAD23].

Pour construire les cellules de Voronoï dans le plan une fonction de distance anisotrope d_c est calculée :

$$d_c(\overrightarrow{\mathbf{x}_1\mathbf{x}}) = (1-\lambda)\,\|\overrightarrow{\mathbf{x}_1\mathbf{x}}\| + \lambda\, d_{\text{Til}}(\overrightarrow{\mathbf{x}_1\mathbf{x}}) \tag{3.8}$$

où $d_{\text{Til}}(\overrightarrow{\mathbf{x}_1\mathbf{x}})$ représente une fonction de distance lisse de pavage anisotrope entre $\mathbf{x}$ et $\mathbf{x}_1$ selon la distance aux bords de la cellule de pavage de $\mathbf{x}_1$ et λ un facteur d'interpolation linéaire entre les deux distances $d_{\text{Til}}(\overrightarrow{\mathbf{x}_1\mathbf{x}})$ et $\|\overrightarrow{\mathbf{x}_1\mathbf{x}}\|$ pour définir si les cellules seront rectangulaires ou proches de cellules de Voronoï, comme illustré dans la figure 3.7.

La distance d_{Til}, illustrée dans la figure 3.8, se note de la façon suivante :

$$d_{\text{Til}}(\overrightarrow{\mathbf{x}_1\mathbf{x}}) = \max_\epsilon(d_x, d_y) \tag{3.9}$$

avec $d_x = \max_\epsilon(d_x^{\text{left}}, d_x^{\text{right}})$ et $d_y = \max_\epsilon(d_y^{\text{top}}, d_y^{\text{bottom}})$ définis par :

$$\begin{aligned} d_x^{\text{left}} &= \frac{x_1 - x}{x_1 - x_c + c_w} \\ d_x^{\text{right}} &= \frac{x - x_1}{x_c + c_w - x_1} \\ d_y^{\text{top}} &= \frac{y - y_1}{y_c + c_h - y_1} \\ d_y^{\text{bottom}} &= \frac{y_1 - y}{y_1 - y_c + c_h} \end{aligned} \tag{3.10}$$

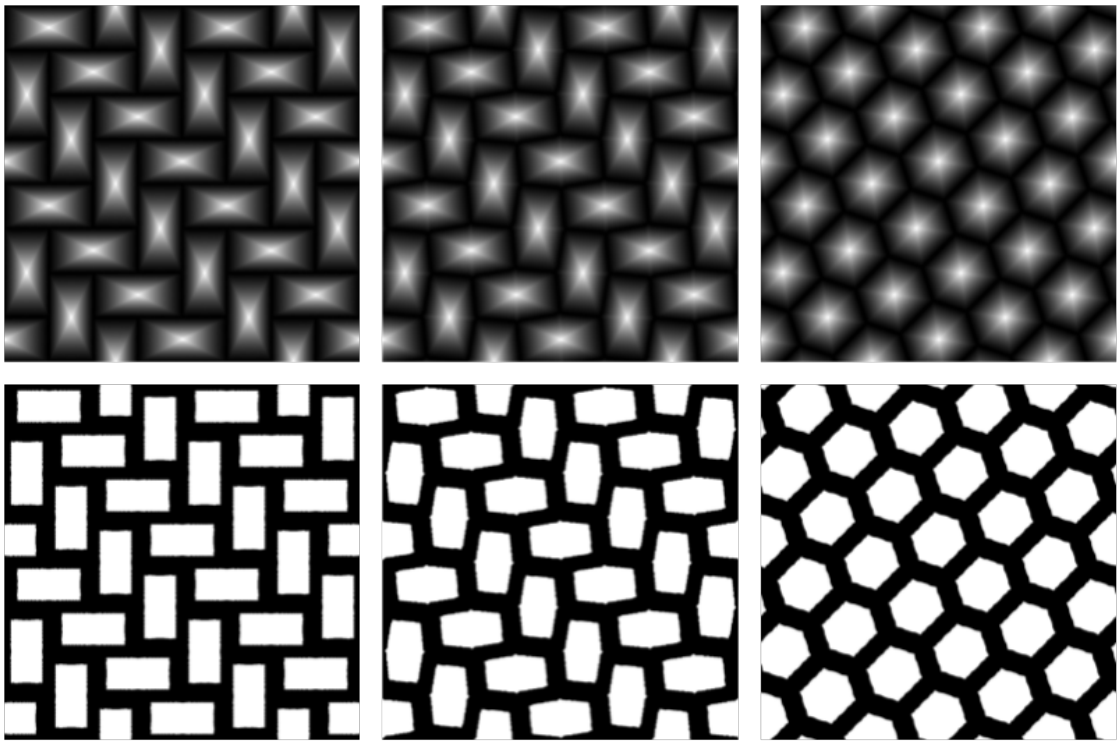

Figure 3.7 – Influence du paramètre d'interpolation λ entre des cellules rectangulaires et des cellules proches de cellules de Voronoï. De gauche à droite les valeurs de λ sont respectivement 1, 0,3 et 0.

où (x_c, y_c) correspond au centre de la cellule contenant $\mathbf{x}_1$ et (c_w, c_h) à la moitié de la taille de cette cellule. La fonction $\max_\epsilon$ se note telle que :

$$\max_\epsilon(a, b) = \frac{1}{2}\left(a + b + \sqrt{(a-b)^2 + \epsilon}\right) \tag{3.11}$$

avec ϵ fixé à 0,001 pour permettre une approximation lissée de la fonction max.

Grâce à cette fenêtre cellulaire le modèle peut générer une grande variété de cellules ayant des formes polygonales, ce qui peut poser problème pour représenter des formes arrondies qui sont aussi fréquentes dans la nature. Le modèle de C-PPTBF offre alors la possibilité de lisser les cellules obtenues par la fenêtre cellulaire en divisant les cellules en secteurs angulaires d'angle $\pi/4$ selon un certain nombre de points de contrôle en utilisant une interpolation par courbes de Bézier. Ainsi un paramètre s_c est utilisé comme interpolation linéaire de lissage pour déterminer si

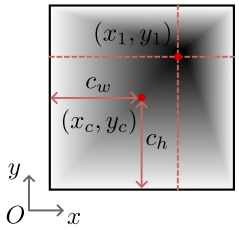

FIGURE 3.8 – Illustration de la distance de pavage $d_{\text{Til}}(\overrightarrow{\mathbf{x}_1\mathbf{x}})$ [BAD23].

les bords des cellules sont droits ou courbés, comme illustré dans la figure 3.9.

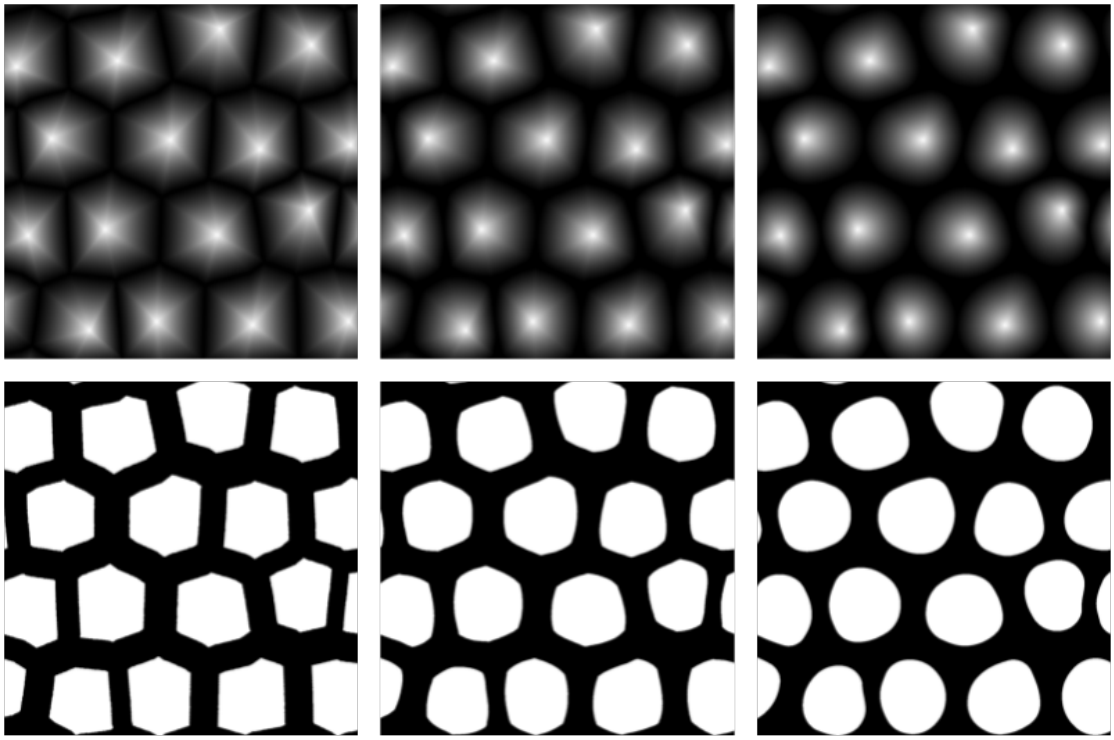

FIGURE 3.9 – Influence du paramètre d'interpolation de lissage s_c entre des bords droits et courbés. De gauche à droite les valeurs de s_c sont respectivement 0, 0,5 et 1.

La **fenêtre de mélange** w_m est définie par la distance de $\mathbf{x}$ à ses feature points $\mathbf{x}_i$ les plus proches selon une fonction gaussienne afin de mélanger des cellules voisines. Elle se note telle que :

$$w_m(\overrightarrow{\mathbf{x}_i\mathbf{x}}) = \frac{\exp(-2\,\|\overrightarrow{\mathbf{x}_i\mathbf{x}}\|) - \exp(-2\,\psi)}{1 - \exp(-2\,\psi)} \tag{3.12}$$

avec ψ une empreinte spatiale dont la valeur dépend du type de pavage. Pour les types de pavage de 0 à 7 cette empreinte est égale à 1, et pour les types de 8 à 13 cette empreinte est égale à 0,4.

Trois paramètres sont donc utilisés dans la fonction Window du modèle de C-PPTBF. Tout d'abord le facteur d'interpolation entre les deux fenêtres ω, puis le facteur d'interpolation entre cellules rectangulaires et cellules de Voronoï λ et enfin le facteur d'interpolation sur le lissage des cellules s_c. L'ensemble de ces paramètres est nommé θ_W, tel que $\theta_W = (\omega, \lambda, s_c)$. La fonction Window est différentiable en fonction de tous les paramètres de θ_W. Comme le calcul de la fonction dépend des feature points elle doit aussi être différentiable en fonction de β. Pour vérifier cela, nous devons démontrer que la fonction de distance d_{nrc} est différentiable en fonction de $\mathbf{x}_1$ qui dépend de β. Ceci est prouvé en considérant que $\partial \mathrm{Vor}_P$ varie continuellement selon β avec $P = (1-\beta)\,C + \beta\,C_A^{jit}$. La fonction de Voronoï d_c étant différentiable en fonction de $\mathbf{x}_1$, ainsi d_{nrc} est différentiable en fonction de β. De plus, la différentiabilité de la fenêtre cellulaire w_c est préservée par le lissage des cellules.

Différences avec le modèle de PPTBF

Le modèle de PPTBF original comporte un paramètre discret permettant d'obtenir des profils de fenêtres différents, comme un profil gaussien, triangulaire ou encore de cosinus effilé. Nous avons fait le choix de ne conserver que le profil gaussien. Pour le lissage, un paramètre permet de contrôler le nombre de points de contrôle utilisés pour l'interpolation par courbes de Bézier, que nous avons fixé à 5. Ces deux choix permettent de supprimer deux paramètres discrets dans notre modèle. Finalement, nous avons fixé le paramètre continu représentant la norme $\|\cdot\|$ utilisée dans les équations 3.7 et 3.8 à 2.

3.4 Fonction Feature

Le résultat de la fonction Window est ensuite multiplié à la fonction Feature qui définit les formes locales des cellules. Elle permet d'enrichir les motifs des cellules et d'apporter des variations spatiales avec un caractère aléatoire. Différents choix sont possibles pour cette fonction comme discuté par Guehl et al. [Gue+20]. Dans notre cas, nous utilisons un noyau gaussien anisotrope qui permet d'apporter des

variations dans la forme des cellules sans introduire de motifs à l'intérieur des cellules. La fonction Feature est ainsi définie de la façon suivante :

$$\text{C-PPTBF}(\theta_P,\, \theta_W,\, \theta_F; \mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})} w(\overrightarrow{\mathbf{x}_i\mathbf{x}}) \, \exp\Big(- \sigma \, \|\tau \overrightarrow{\boldsymbol{\mu}_i\mathbf{x}}\| \Big) \tag{3.13}$$

avec σ un coefficient appliqué sur le noyau et τ une transformation composée d'une rotation et d'une mise à l'échelle telle que :

$$\tau = \begin{pmatrix} \cos(\phi) & -\rho \sin(\phi) \\ \sin(\phi) & \rho \cos(\phi) \end{pmatrix} \tag{3.14}$$

dont ϕ représente l'angle de la rotation et ρ le facteur de mise à l'échelle.

On calcule un ensemble de positions $\boldsymbol{\mu}_i$ qui correspondent à des positions perturbées des feature points calculées par interpolation linéaire entre les feature points $\mathbf{x}_i$ et des positions perturbées des centres des cellules C_B^{jit} tels que $\mathbf{x}_i^{jit} \in C_B^{jit}$:

$$\boldsymbol{\mu}_i = (1 - \gamma)\, \mathbf{x}_i + \gamma\, \mathbf{x}_i^{jit} \tag{3.15}$$

avec γ le facteur d'interpolation entre les deux positions.

La fonction Feature comprend alors quatre paramètres avec σ le coefficient appliqué au noyau gaussien dont l'influence est illustrée dans la figure 3.11, ϕ l'angle de rotation de la transformation appliquée au noyau comme montré dans la figure 3.10, ρ le facteur de mise à l'échelle de la transformation présenté dans la figure 3.12 et γ le facteur d'interpolation entre les positions des feature points et les positions perturbées des centres des cellules illustré par la figure 3.13. L'ensemble de ces paramètres est nommé θ_F, tel que $\theta_F = (\sigma, \phi, \rho, \gamma)$. La fonction Feature est différentiable en fonction de tous ses paramètres.

Différences avec le modèle de PPTBF

Dans la fonction Feature initiale du modèle de PPTBF, une somme de noyaux de Gabor est utilisée. Nous avons décidé dans le modèle de C-PPTBF de n'utiliser qu'un unique noyau gaussien car nous considérons qu'il est suffisant pour représenter des structures cellulaires. Cela nous permet également de supprimer deux paramètres discrets représentant les nombres minimum et maximum de noyaux en les fixant à 1, et les trois paramètres continus liés aux bandes des noyaux de Gabor. De la même façon, un paramètre discret permet de définir le modèle de mélange contrôlant

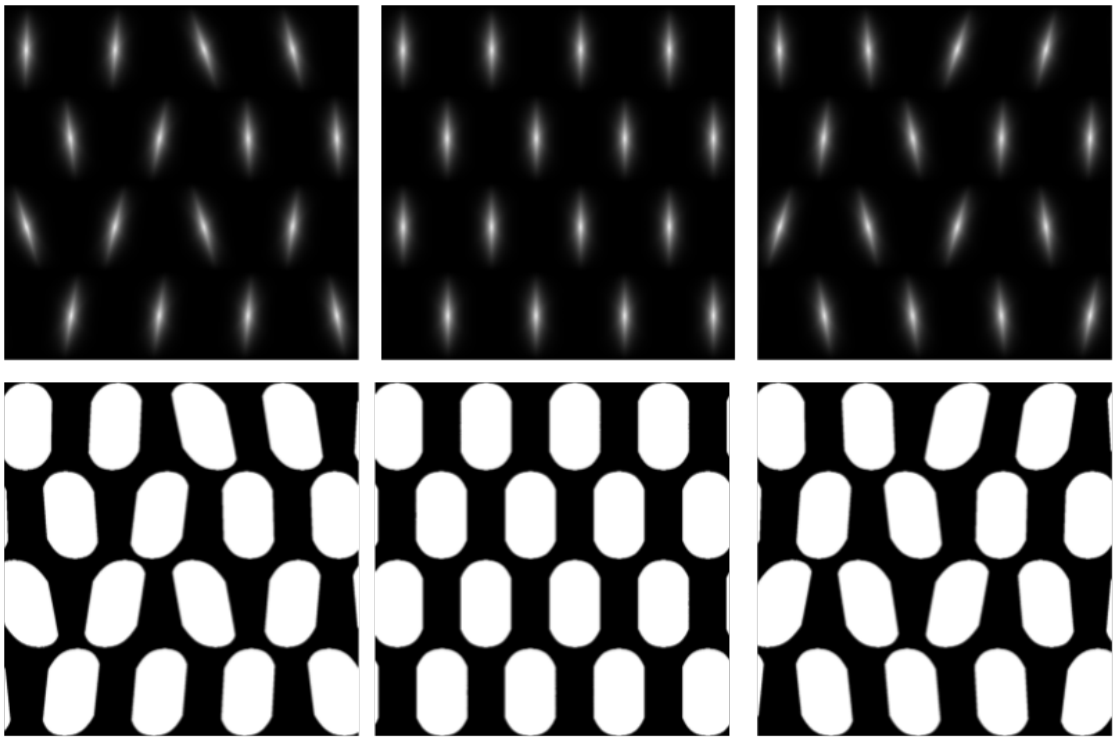

Figure 3.10 – Influence de l'angle d'orientation ϕ du noyau gaussien. De gauche à droite les valeurs de ϕ sont respectivement -0,1π, 0 et 0,1π.

l'influence de chaque noyau de Gabor, par une somme, une somme sans valeurs négatives, une somme inversée ou encore en prenant la plus grande valeur de chaque noyau. Ce paramètre discret est ainsi supprimé comme nous n'avons qu'un seul noyau. Finalement, nous avons fixé le paramètre continu représentant la norme $\|\cdot\|$ utilisée dans l'équation 3.13 à 2, et supprimé le paramètre permettant d'apporter de la variation sur le paramètre σ.

Étude sur les noyaux de Gabor et gaussien

Afin de comparer l'influence du noyau gaussien utilisé dans le modèle de C-PPTBF et les noyaux de Gabor utilisés dans le modèle de PPTBF, nous proposons d'étudier les différences que nous observons quand nous remplaçons le noyau gaussien par un noyau de Gabor anisotrope G. Dans ce cas, la fonction Feature serait définie de la sorte :

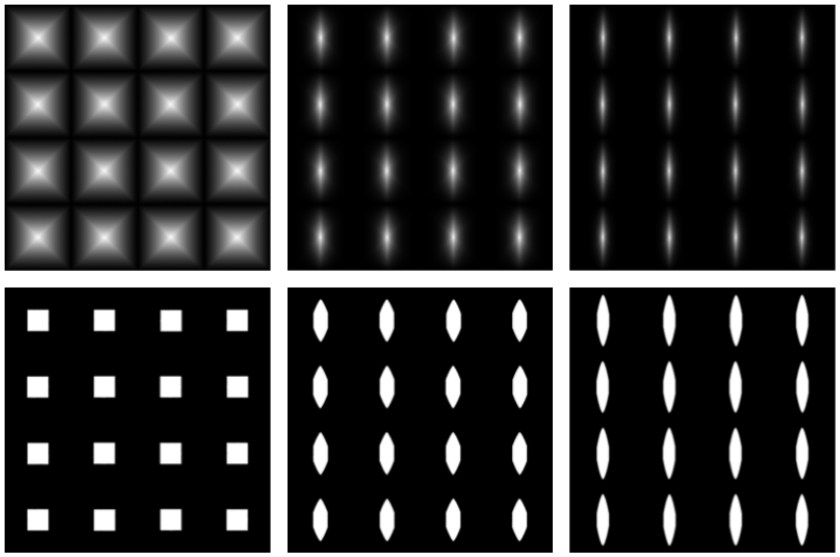

Figure 3.11 – Influence du coefficient σ appliqué au noyau gaussien. De gauche à droite les valeurs de σ sont respectivement 0, 5 et 10.

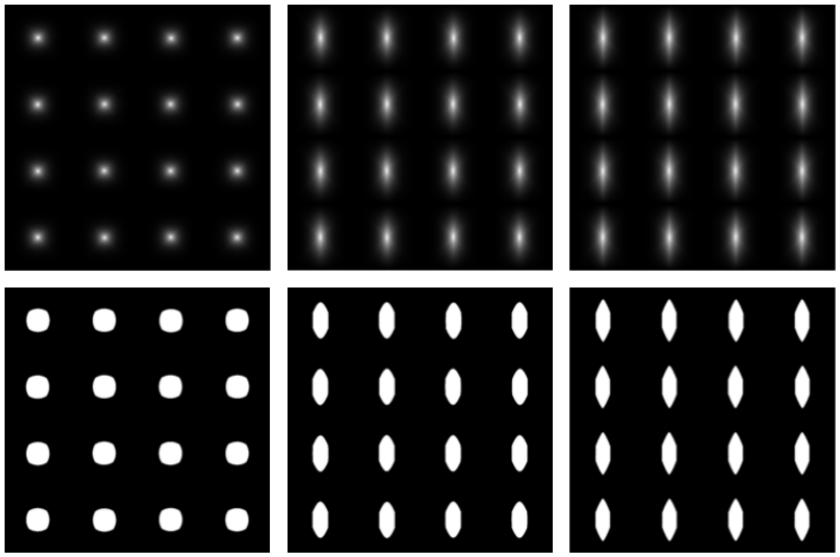

Figure 3.12 – Influence du facteur de mise à l'échelle ρ de la transformation appliquée au noyau gaussien. De gauche à droite les valeurs de ρ sont respectivement 0, 2,5 et 5.

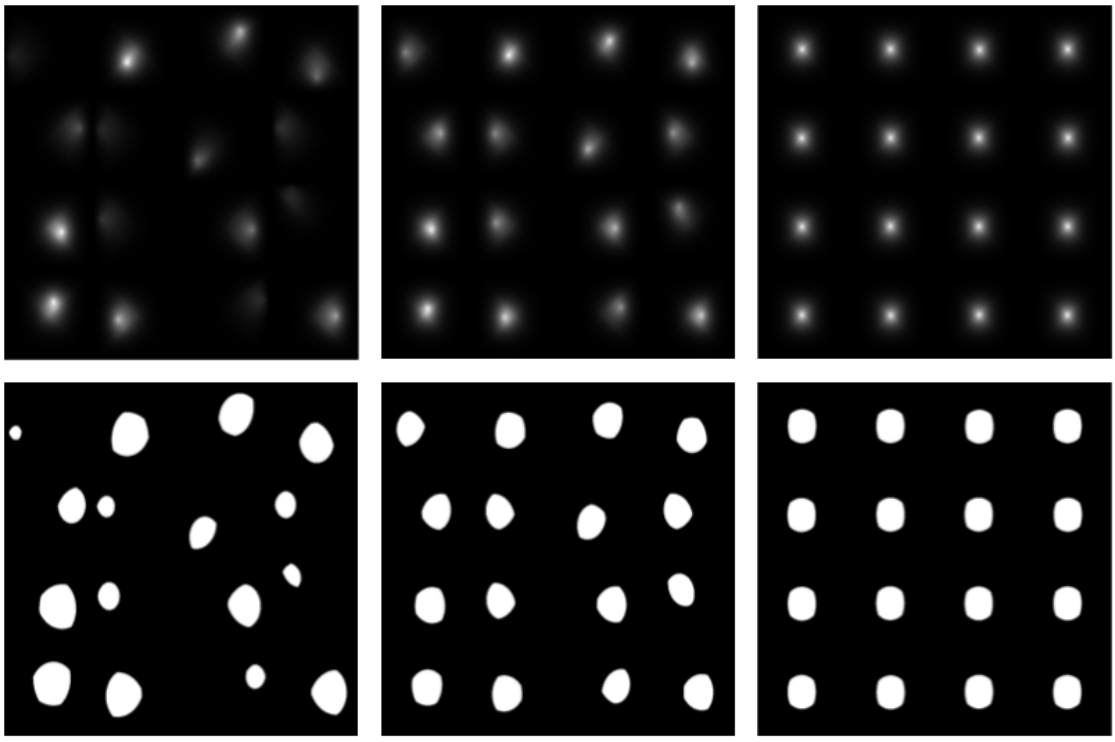

Figure 3.13 – Influence du facteur d'interpolation γ entre les positions des feature points et les positions perturbées des centres des cellules. De gauche à droite les valeurs de γ sont respectivement 0, 0,5 et 1.

$$\text{C-PPTBF}(\theta_P,\, \theta_W,\, \theta_F; \mathbf{x}) = \sum_{\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})} w(\overrightarrow{\mathbf{x}_i\mathbf{x}})\, G(\overrightarrow{\mathbf{x}_i\mathbf{x}}) \tag{3.16}$$

Trois nouveaux paramètres sont ajoutés afin d'agir sur les bandes du noyau de Gabor. Le premier paramètre χ porte sur la fréquence des bandes du noyau de Gabor dans les cellules de pavage, qui a pour effet aussi de réduire l'épaisseur des bandes. Le deuxième paramètre κ agit sur la courbure des bandes du noyau de Gabor, entre bandes rectilignes et cercles concentriques. Finalement le dernier paramètre δ contrôle l'épaisseur des bandes du noyau. La formulation du noyau G est donc la suivante :

$$G(\overrightarrow{\mathbf{x}}) = \exp(-\sigma \, \|\tau\overrightarrow{\boldsymbol{\mu}_i\mathbf{x}})\|_f)\Big(0,5 + 0,5\cos\left(\chi\|\tau\overrightarrow{\boldsymbol{\mu}_i\mathbf{x}}\|_f\right)\Big)^{\delta} \tag{3.17}$$

avec la distance $\|.\|_f$ telle que :

$$\|.\|_f = \sqrt{(x')^2\kappa^2 + (y')^2} \tag{3.18}$$

L'utilisation d'un noyau de Gabor permet de représenter des structures en formes de rayures ou de plis grâce à ses bandes, qui s'éloignent de structures cellulaires dans leur apparence. Quelques exemples d'images pouvant être générées avec un noyau de Gabor dans le modèle de C-PPTBF sont présentés dans la figure 3.14, ces structures n'étant pas représentables avec notre modèle de C-PPTBF utilisant un noyau gaussien.

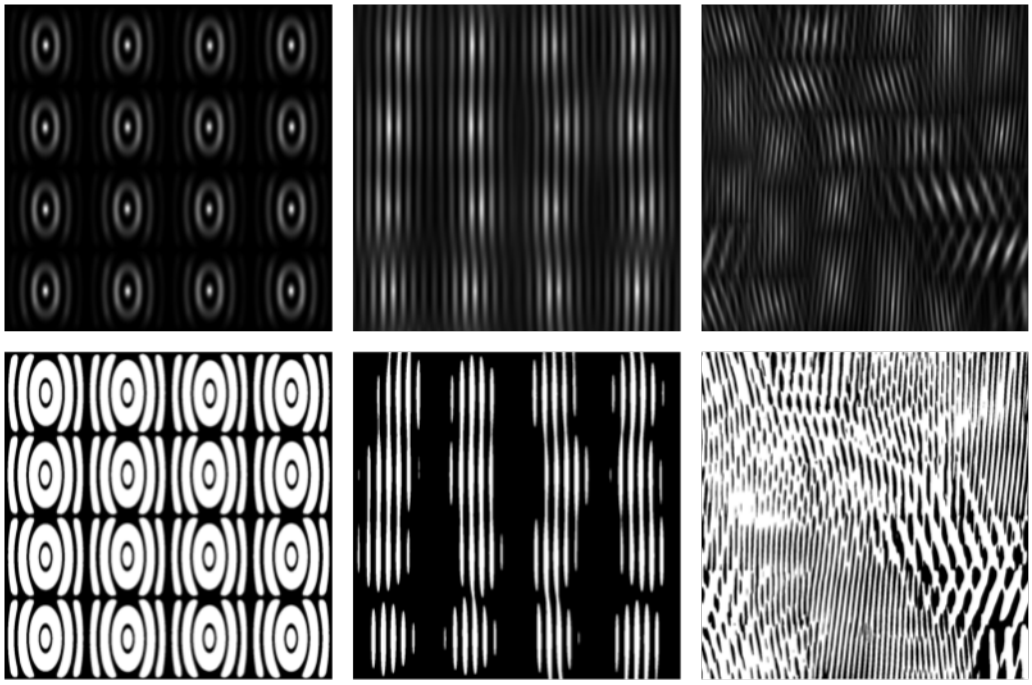

Figure 3.14 – Exemples d'images de C-PPTBF et leurs structures binaires obtenues en utilisant un noyau de Gabor dans la fonction Feature du modèle de C-PPTBF.

3.5 Transformations spatiales et déformations

Le modèle de C-PPTBF inclut différents paramètres de transformations spatiales et de déformations permettant d'obtenir de la variété dans les structures représentables et d'ajouter un peu d'aléatoire pour approcher les cartes de structures extraites

de textures naturelles. Ces paramètres sont continus et le modèle est différentiable par rapport à eux.

Ainsi des paramètres de translation sur les deux axes tx et ty, de rotation d'angle α et un facteur de mise à l'échelle s sont proposés pour appliquer des transformations spatiales affines à l'image de C-PPTBF. La calcul de la fonction C-PPTBF en un point de l'image commence alors par l'application de ces transformations.

De plus, des déformations spatiales aléatoires se reposant sur le bruit sont appliquées ensuite sur les points en utilisant une somme fractale de bruits de Perlin contrôlée par un coefficient d'amplitude a. Cette somme s'apparente à un mouvement Brownien fractionnaire qui va se noter tel que :

$$\mathrm{fBm}_{a,n}(\mathbf{x}) = a\,\mathrm{fBm}_n(\mathbf{x}) = a\sum_{j=0}^{n}\frac{1}{2^j}\eta(2^j\,\mathbf{x}) \tag{3.19}$$

avec η la fonction de bruit de Perlin renvoyant une valeur dans l'intervalle $[-1, 1]$ pour tout point $\mathbf{x} = (x, y)$ dans le plan. En un point la déformation appliquée est définie comme suit :

$$\mathcal{D}(\mathbf{x}_i) = \mathbf{x}_i + a\begin{bmatrix}\mathrm{fBm}_n(x_i + o_x, y_i)\\ \mathrm{fBm}_n(x_i, y_i + o_y)\end{bmatrix} \tag{3.20}$$

avec o_x et o_y deux paramètres permettant d'effectuer un décalage afin que η soit échantillonnée dans différentes régions du champ de déformation. Dans la pratique nous fixons n à 4, ce qui donne une somme de quatre bruits de Perlin appliquée sur chaque point. L'influence de ce paramètre de déformation sur les structures représentables par le modèle est illustrée dans la figure 3.15.

L'ensemble des paramètres de transformations et déformations est nommé θ_T, tel que $\theta_T = (tx, ty, \alpha, s, a)$.

3.6 Seuillage

Une fois que l'image d'intensité de C-PPTBF est générée après application de la fonction C-PPTBF, on souhaite obtenir une carte de structures binaires M à partir de cette image. Cette carte de structures est obtenue en appliquant une fonction de seuillage binaire th selon la valeur de seuillage ν, qui contrôle la proportions de pixels noirs et blancs dans la carte de structures, de cette sorte :

$$M = th\big(\mathrm{R}\big(\theta_T; \text{C-PPTBF}(\theta_P, \theta_W, \theta_F)\big), \nu\big) \tag{3.21}$$

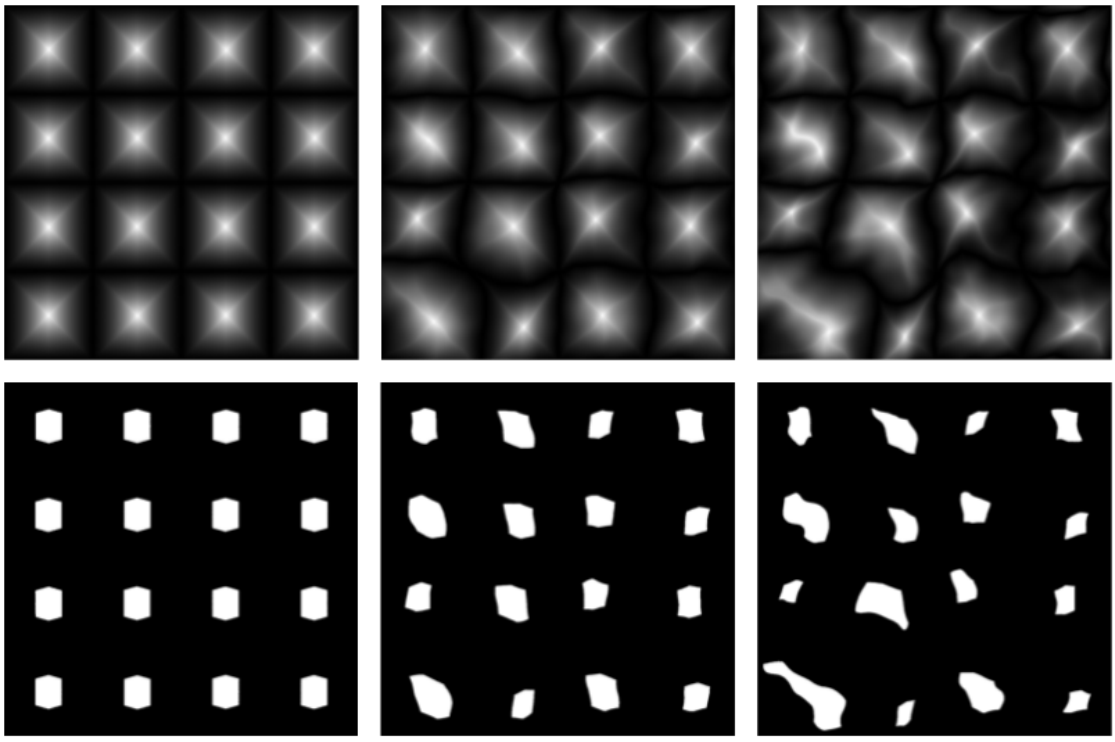

Figure 3.15 – Influence du coefficient d'amplitude α de la somme fractale de bruits de Perlin. De gauche à droite les valeurs de α sont respectivement 0, 0,075 et 0,15.

avec R la fonction de rendu permettant d'obtenir l'image d'intensité de C-PPTBF. Ainsi selon la valeur de ν il est possible d'obtenir une multitude de cartes de structures binaires différentes pour une seule image d'intensité comme illustré dans la figure 3.16.

3.7 Différences avec le modèle de PPTBF

Comme détaillé dans chaque partie du modèle de C-PPTBF, nous avons fait le choix de supprimer tous les paramètres discrets du modèle de PPTBF original, sauf le type de pavage, en fixant les valeurs, afin d'obtenir plus facilement un modèle différentiable. Nous avons aussi fixé les valeurs de plusieurs paramètres continus qui n'étaient pas essentiels pour la génération de structures cellulaires stochastiques afin de limiter le nombre de paramètres pour simplifier leur estimation. Tous ces

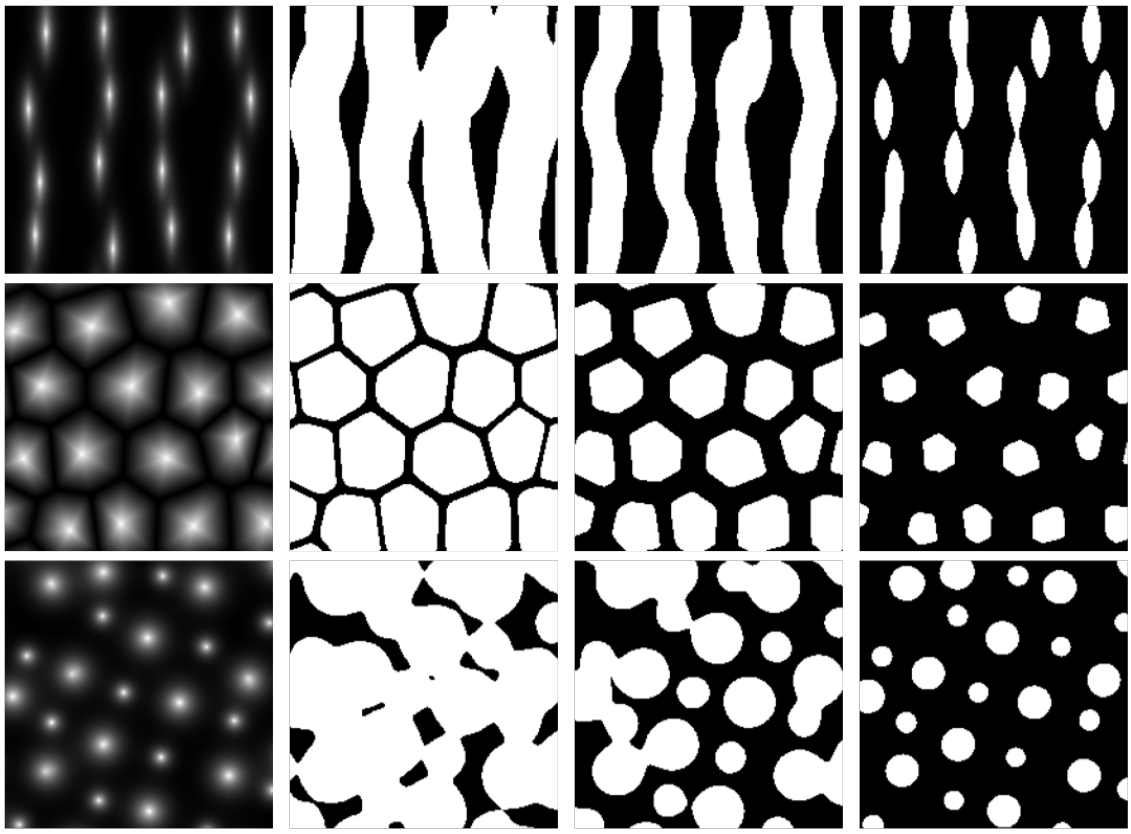

Figure 3.16 – Exemples d'images de C-PPTBF avec leurs cartes de structures binaires obtenues après seuillage. De gauche à droite les valeurs du paramètre de seuillage ν sont respectivement 0,2, 0,5 et 0,8, correspondant à la proportion de pixels noirs.

changements sont résumés dans la figure 3.17.

3.8 Résumé des paramètres du modèle

Nous résumons dans le tableau 3.1 les différents paramètres utilisés par le modèle de C-PPTBF. Dans la pratique, nous n'allons estimer les valeurs que de 12 des paramètres suivants, ce sont les 12 paramètres dont la notation est en rouge. Les différents paramètres permettant de construire les types de pavage, $decalx$, $subx$, $suby$, $jitx$ et $jity$, ne sont pas estimés car ils sont fixes selon le type de pavage concerné.

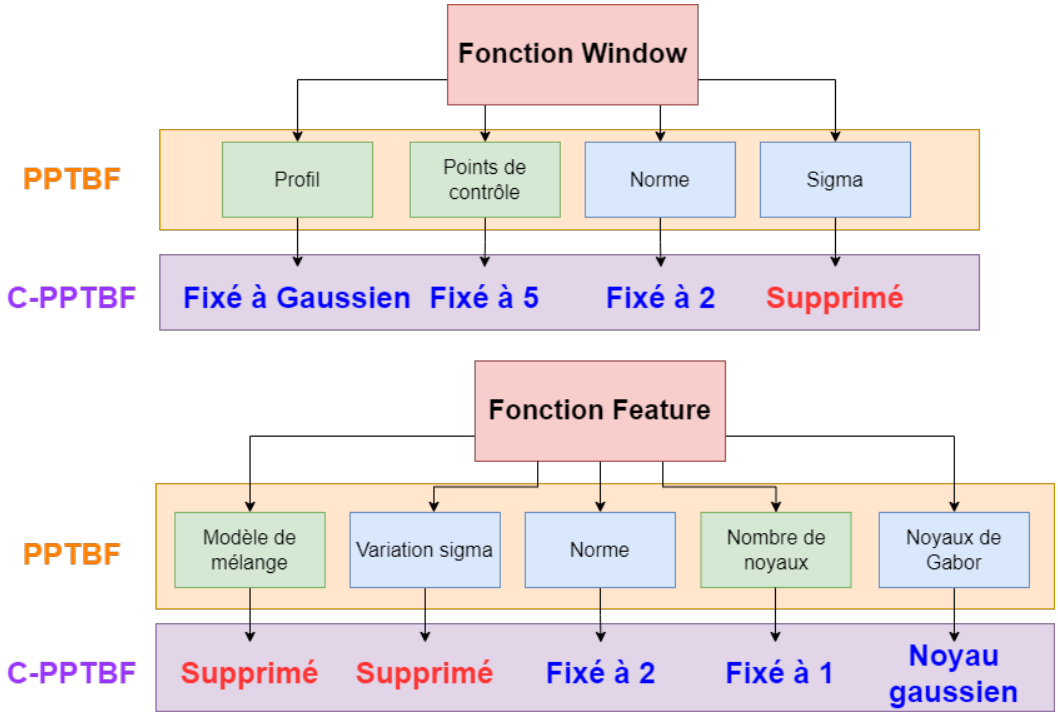

Figure 3.17 – Comparaison des paramètres utilisés entre les modèles de PPTBF et de C-PPTBF.

Dans le cadre de notre travail nous n'estimons pas non plus les deux paramètres de translation tx et ty.

Notation	Description	Intervalle de valeurs
	PROCESSUS PONCTUEL	
tt	Type de pavage	{0,13}
β	Perturbation de la position des feature points	[0,01, 0,5]
$decalx$	Décalage horizontal des cellules	$\mathbb{N}$
$subx$	Probabilité de subdivision horizontale d'une cellule	[0, 1]
$suby$	Probabilité de subdivision verticale d'une cellule	[0, 1]
$jitx$	Perturbation de la position de la subdivision horizontale	[0, 1]
$jitx$	Perturbation de la position de la subdivision verticale	[0, 1]
	FONCTION *WINDOW*	
ω	Facteur d'interpolation entre la fenêtre cellulaire et la fenêtre de mélange	[0, 1]
λ	Facteur d'interpolation entre cellules rectangulaires et de Voronoï	[0, 1]
s_c	Facteur d'interpolation sur le lissage des cellules	[0, 1]
	FONCTION *FEATURE*	
σ	Coefficient appliqué au noyau Gaussien	[0, 10]
ϕ	Angle de rotation du noyau Gaussien	[-π/10, π/10]
ρ	Facteur de mise à l'échelle du noyau Gaussien	[0, 5]
γ	Facteur d'interpolation entre les positions des feature points et les positions $\boldsymbol{\mu}_i$	[0, 1]
	TRANSFORMATIONS SPATIALES ET DÉFORMATIONS	
tx	Translation sur l'axe horizontal	[-20, 20]
ty	Translation sur l'axe vertical	[-20, 20]
α	Angle de rotation	[0, 2]
s	Facteur de mise à l'échelle	[1, 20]
a	Coefficient d'amplitude sur les déformations	[0, 0,15]

Table 3.1 – Résumé des paramètres du modèle de C-PPTBF.

Chapitre 4

Estimation des paramètres du modèle de C-PPTBF

Dans ce chapitre, nous présentons notre approche du problème d'estimation des paramètres du modèle de C-PPTBF.

4.1 Problème d'optimisation

Notre objectif avec le modèle de C-PPTBF est de mettre en place une chaîne de traitement automatique permettant l'estimation de paramètres discrets et continus du modèle pour obtenir une carte de structures binaires ressemblant le plus possible à la carte de structures extraite d'une image de texture naturelle en entrée. Afin d'effectuer une estimation de paramètres performante et de pallier les problèmes de la procédure d'estimation proposée par Guehl et al. [Gue+20], nous voulons créer un modèle entièrement différentiable pour pouvoir effectuer une optimisation par le gradient. Il nous faut alors minimiser l'erreur d'une fonction de dissimilarité entre la carte de structures en entrée et la structure binaire procédurale en sortie. Nous souhaitons alors résoudre un problème d'optimisation de cette sorte :

$$(\hat{\theta}_T, \hat{\theta}_P, \hat{\theta}_W, \hat{\theta}_F) = \underset{(\theta_T, \theta_P, \theta_W, \theta_F)}{\operatorname{argmin}} \mathcal{L}\Big(I, th\big(\mathrm{R}\big(\theta_T; \text{C-PPTBF}(\theta_P, \theta_W, \theta_F)\big), \nu\big)\Big) \quad (4.1)$$

Plusieurs problématiques se posent afin d'obtenir un modèle entièrement différentiable. Tout d'abord nous avons le type de pavage qui est un paramètre discret et donc empêche le modèle d'être entièrement différentiable en fonction de tous ses

paramètres. De la même façon la fonction de seuillage n'est pas différentiable. Finalement il faut déterminer comment initialiser les valeurs de l'intégralité des paramètres continus.

Nous résolvons ces problématiques en trois étapes :

(1) Tout d'abord nous utilisons un réseau antagoniste génératif conditionnel afin de reconstruire une image de C-PPTBF plausible $U(I)$ à partir d'une carte de structures I en entrée.

(2) Puis nous estimons le type de pavage ainsi que des valeurs initiales pour l'intégralité des paramètres continus avec un CNN entraîné pour la prédiction des paramètres.

(3) Finalement nous effectuons une optimisation des valeurs des paramètres continus avec une descente de gradient stochastique.

Notre chaîne de traitement complète est présentée dans la figure 4.1. Toutes les étapes vont être détaillées dans les sections suivantes.

4.2 Implémentation du modèle de C-PPTBF

Nous avons implémenté notre modèle de C-PPTBF de deux façons différentes. Afin de générer des images de C-PPTBF et de les seuiller, nous avons implémenté le modèle avec l'API OpenCL et nous utilisons PyOpenCL pour faire l'interface avec notre code en Python et le modèle. Cette implémentation est utilisée pour générer les jeux de données d'images pour notre étape de reconstruction d'une image de C-PPTBF plausible et pour la première phase d'estimation des paramètres, et pour la génération des feature points de la deuxième phase d'estimation. Pour la deuxième phase d'estimation en revanche le modèle est implémenté en utilisant le *framework* JAX en Python, qui est spécialisé pour le machine learning et la différentiation automatique [1] et qui est optimisé pour une exécution sur GPU, les fonctions du modèle sont donc implémentées pour être différentiables et ainsi utilisables dans une optimisation par descente de gradient stochastique.

L'implémentation de notre chaîne de traitement est entièrement mise à disposition de la communauté via un dépôt GitHub [2].

1. Du terme anglais *automatic differentiation*, peut être traduit aussi par « dérivation automatique ».

2. https://github.com/ASTex-ICube/DiffPPTBF

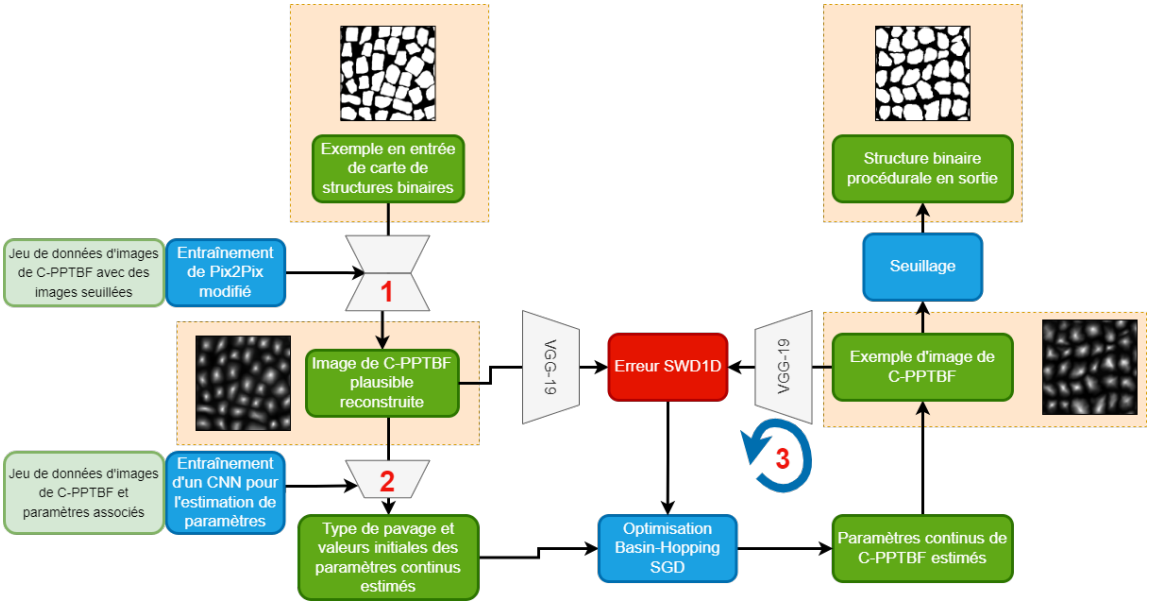

Figure 4.1 – Chaîne de traitement de notre méthode. (1) : Reconstruction d'une image de C-PPTBF plausible à partir d'une carte de structures binaires en entrée en utilisant un réseau Pix2Pix modifié ; (2) : Estimation du type de pavage et des valeurs initiales des paramètres continus avec un CNN ; (3) : Optimisation par descente de gradient stochastique des valeurs des paramètres continus.

4.3 Estimation des paramètres discrets

Notre modèle de C-PPTBF ne contient qu'un seul paramètre discret qui est le type de pavage. Ce paramètre est primordial car il définit la distribution des points et donc l'apparence globale de l'image. Ainsi une attention particulière est donnée à l'estimation de ce paramètre car une mauvaise estimation peut mener à une image en sortie trop éloignée de l'exemple en entrée. De plus ce paramètre discret rend le modèle non différentiable par rapport à tous ses paramètres et ne permet alors pas une optimisation par descente de gradient stochastique. Il est possible de pallier ce problème en transformant les paramètres discrets en paramètres continus comme c'est le cas dans DiffProxy [Hu+22c] par exemple, mais les paramètres discrets seront alors estimés en même temps que les paramètres continus avec les mêmes paramètres de descente de gradient, en particulier le taux d'apprentissage, ce qui peut poser problème en cas de paramètre discret ayant une grande étendue de valeurs possibles par rapport à des paramètres continus ayant des petits intervalles. La question de l'initialisation des paramètres discrets se pose alors, car si la valeur initiale est trop

éloignée de la valeur souhaitable alors il est possible que cette valeur ne soit pas atteinte lors de la descente de gradient en raison d'un taux d'apprentissage trop bas. De plus, le paramètre discret étant devenu continu, la valeur obtenue ne sera pas entière, ce qui pose problème pour des paramètres comme le type de pavage pour lequel les valeurs qui se suivent n'ont pas forcément de rapport entre elles, le choix de l'arrondi pouvant donner alors deux apparences complètement différentes.

Nous décidons de décomposer l'estimation des paramètres en deux phases. Dans la première phase, la valeur du type de pavage est estimée ainsi que des valeurs initiales pour l'ensemble des paramètres continus du modèle. Dans la deuxième phase, seules les valeurs des paramètres continus sont affinées par une optimisation à partir d'une descente de gradient stochastique qui est permise car le modèle devient entièrement différentiable par rapport à tous ses paramètres comme le type de pavage est déjà estimé et seuls les paramètres continus subsistent.

Pour la première phase, nous profitons de l'efficacité des réseaux de neurones et de leur capacité à pouvoir prédire les valeurs de paramètres discrets et continus. Nous avons effectué des expérimentations avec différentes architectures de réseaux de neurones implémentées dans l'API Keras. Les différentes architectures essayées sont :

1. VGG16 et VGG19 [SZ15] : 5 blocs de couches convolutionnelles, au total 13 ou 16 couches, séparés par des couches de *Max Pooling* et suivies de 3 couches entièrement connectées ;
2. ResNet [He+16a] : multiples blocs de 3 couches convolutionnelles, pour un total de 50, 101 ou 152 couches, séparés par des couches de *Max Pooling*. Les blocs sont appelés blocs résiduels car le réseau utilise des *sauts de connexion* pour faire de l'entrée d'un bloc la combinaison des sorties des deux blocs précédents ;
3. ResNetV2 [He+16b] : modification de ResNet en modifiant l'ordre des couches dans les blocs pour avoir la couche de *Batch Normalization* en entrée de bloc et non plus en sortie ;
4. InceptionV3 [Sze+16] : le réseau Inception utilise plusieurs couches convolutionnelles de tailles de filtres différentes au même niveau pour la même entrée, les sorties de chaque couche étant concaténées afin de former la sortie du bloc ;
5. InceptionResNetV2 [Sze+17] : le principe des couches appliquées en parallèle du réseau Inception est repris en y ajoutant les *sauts de connexion* du réseau ResNet ;
6. Xception [Cho17] : même architecture que le réseau Inception mais en modifiant l'ordre des couches convolutionnelles dans les blocs et en supprimant certaines

activations ;

7. DenseNet [Hua+17] : toutes les couches convolutionnelles sont connectées entre elles afin de maximiser les informations qui transitent entre les couches et de donner accès au gradient à tout moment ;
8. NASNet [Zop+18] : architecture qui n'est pas prédéfinie mais qui est construite à partir d'une méthode d'apprentissage par renforcement profond. Ainsi l'architecture s'adapte aux entrées pour définir un nombre de blocs et de filtres approprié ;
9. EfficientNet [TL19] : multiples blocs dans lesquels le nombre de couches en profondeur et en largeur, et la résolution des filtres, augmentent graduellement au fil des blocs.

Les réseaux sont pré-entraînés sur ImageNet et du transfert d'apprentissage est appliqué en affinant sur un jeu de données de 50 000 images de C-PPTBF pour 75 époques. Les résultats en termes de précision sur la validation pour le type de pavage sont présentés dans le tableau 4.1 et montrent que la meilleure précision atteinte est de 82,52% en utilisant une architecture ResNet152V2.

Afin de comprendre pourquoi les réseaux échouent à prédire le type de pavage sur certaines images, une analyse est effectuée en utilisant le réseau de type ResNet152V2 sur un jeu de données de 14 000 images synthétiques de C-PPTBF dont 1 000 pour chaque type de pavage. Aucune déformation ou perturbation ne sont appliquées sur les images mais tous les autres paramètres sont échantillonnés pour obtenir des images diverses pour chaque type de pavage.

On observe sur la figure 4.2 que le réseau a le plus de difficultés à estimer correctement le type de pavage pour les types 4, 5, 6 et 7. Ce qu'on constate dans les figures 4.3 et 4.4 c'est que le type de pavage 4 est confondu avec le type 6 dans environ 21% des cas et que le type de pavage 5 est confondu avec le type 7 dans environ 23% des cas. Le phénomène inverse est vrai dans une moindre mesure, dans 12% des cas le réseau donne le type 4 pour des images de type 6 et dans 8% des cas le type 5 pour des images de type 7. Ceci s'explique par le fait que certains types de pavage peuvent se ressembler et donner des structures similaires selon les configurations de paramètres. En effet, les types de pavage 4 et 6, ainsi que les types de pavage 5 et 7 sont très proches, leur seule différence étant que les types 4 et 5 ajoutent de l'aléatoire sur la position des bords des cellules sur les axes x et y, alors que les types 6 et 7 ne le font que sur l'axe x.

Dans les figures 4.5 et 4.6 on peut voir deux exemples d'images ayant un type de pavage 4 et 5 pour lesquelles le réseau a respectivement estimé un type 6 et 7. Si on

Architecture	Précision sur la validation pour le type de pavage
Xception	80,13%
VGG16	81,12%
VGG19	81,51%
ResNet50	72,73%
ResNet101	74,85%
ResNet152	75,90%
ResNet50V2	80,16%
ResNet101V2	81,99%
ResNet152V2	82,52%
InceptionV3	78,12%
InceptionResNetV2	77,74%
DenseNet121	81,15%
DenseNet169	81,93%
DenseNet201	82,18%
NASNetMobile	76,29%
NASNetLarge	80,49%
EfficientNetB0	21,11%
EfficientNetB1	37,76%
EfficientNetB2	56,29%
EfficientNetB3	66,15%
EfficientNetB4	68,43%
EfficientNetB5	73,55%
EfficientNetB6	71,94%
EfficientNetB7	73,29%

Table 4.1 – Précision sur la validation pour le type de pavage selon l'architecture du CNN avec un jeu de données d'images de C-PPTBF.

regarde les images générées avec les mêmes paramètres continus mais avec les types de pavage estimés par le réseau, on remarque que les images sont très ressemblantes, ainsi l'erreur du réseau est compréhensible et les confusions sur certains types de pavage peuvent alors s'expliquer. Cette erreur de type de pavage n'est cependant pas pénalisante pour la suite car c'est la similarité visuelle qui est importante et non la justesse des paramètres.

Sur ce jeu de données d'images sans perturbation et déformation, nous obtenons une précision globale sur le type de pavage de 90,44%, ce qui est supérieur aux 82,52%

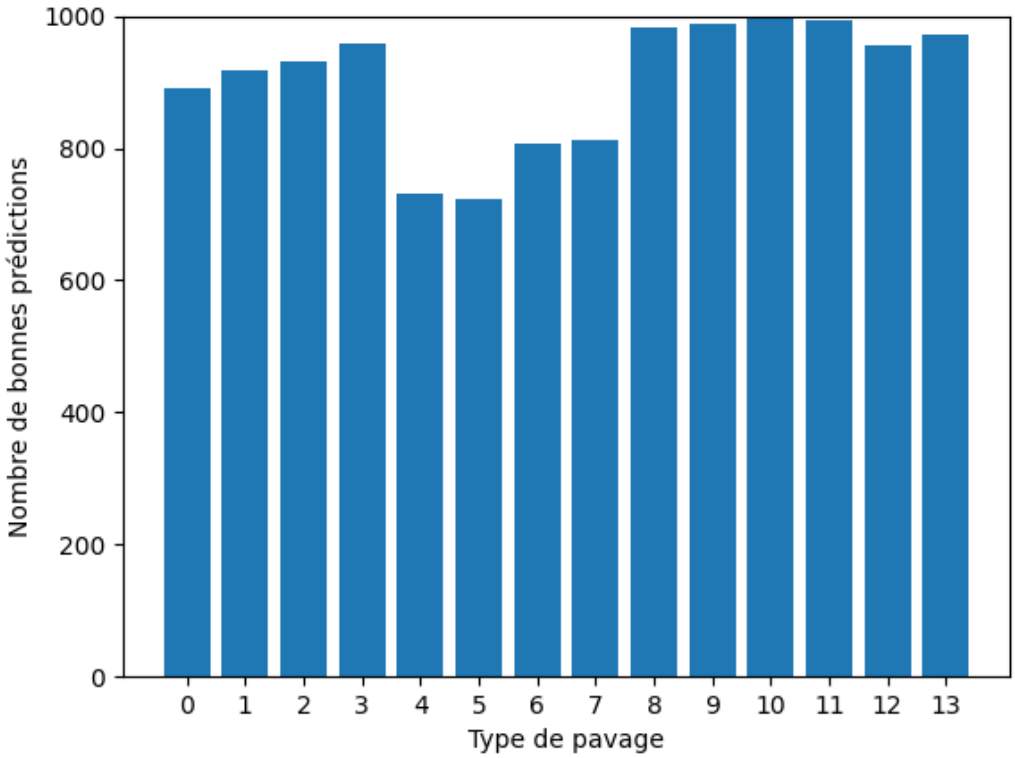

Figure 4.2 – Nombre de bonnes prédictions du réseau ResNet152V2 sur 1 000 images de C-PPTBF de chaque type de pavage.

que nous obtenions lors de l'entraînement du réseau sur des images comportant des perturbations et déformations. Cela montre alors que ces éléments influent sur l'estimation du bon type de pavage.

Pour aller plus loin, nous proposons plusieurs métriques permettant d'analyser ces résultats sur les différents types de pavage, à savoir la précision, le rappel et l'indice de Dice. Ces résultats sont présentés dans le tableau 4.2. Nous pouvons constater que l'on retrouve les mêmes tendances que précédemment avec les types de pavage 4, 5, 6 et 7 qui ont des valeurs de métriques inférieures à celles des autres types de pavage, ces types ayant les nombres de vrais positifs les moins élevés et les nombres de faux positifs les plus élevés. Le nombre de faux positifs du type de pavage 5 est en revanche inférieur à ceux des trois autres types considérés, ce qui lui permet d'avoir

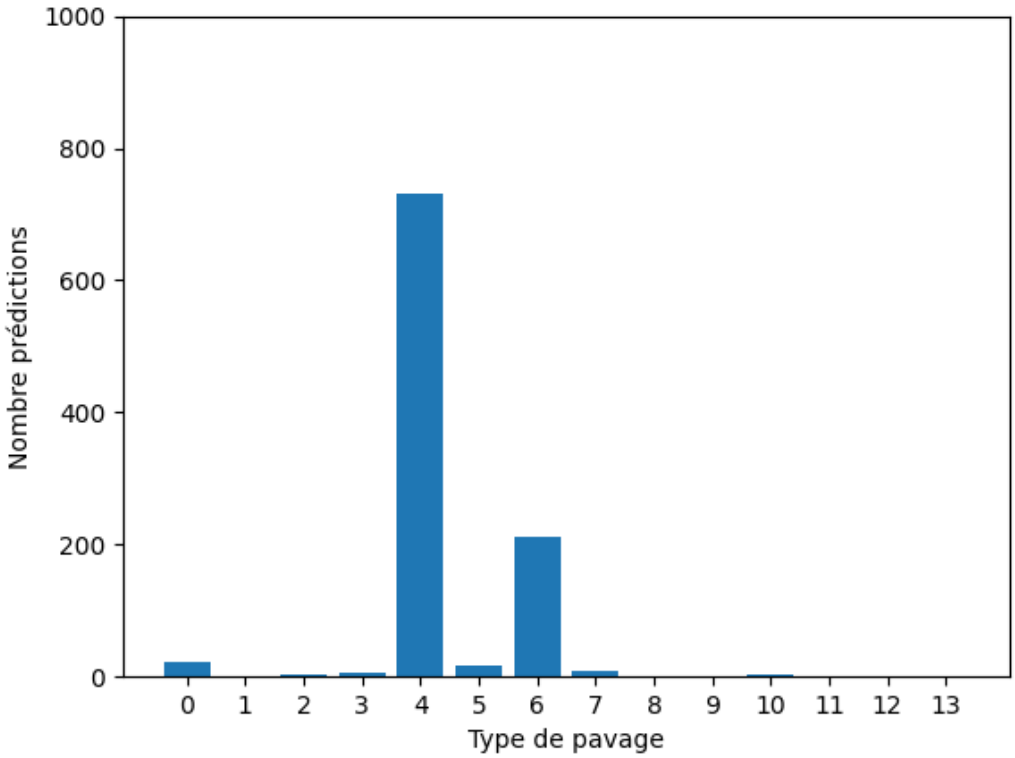

Figure 4.3 – Nombre de prédictions pour chaque type de pavage du réseau ResNet152V2 sur 1 000 images de C-PPTBF du type de pavage 4.

une précision supérieure alors que c'est le type de pavage avec le nombre de vrais positifs le plus bas. Nous observons aussi que c'est pour les types de pavage 8 à 13 que l'estimation est la meilleure, c'est-à-dire les types disposant des plus grands nombres de feature points.

Nous décidons de comparer ces résultats avec ceux que nous obtenons si nous utilisions des images de structures binaires obtenues après seuillage d'images de C-PPTBF. Ainsi un réseau d'architecture ResNet152V2 est entraîné de la même manière que précédemment, mais cette fois-ci sur un jeu de données d'images binaires avec une proportion de pixels noirs et blancs fixée à 50%. Nous obtenons une précision de validation sur l'estimation du type de pavage de 70,37%, ce qui est inférieur à la précision obtenue en utilisant des images de C-PPTBF, justifiant leur utilisation

plutôt que les images binaires.

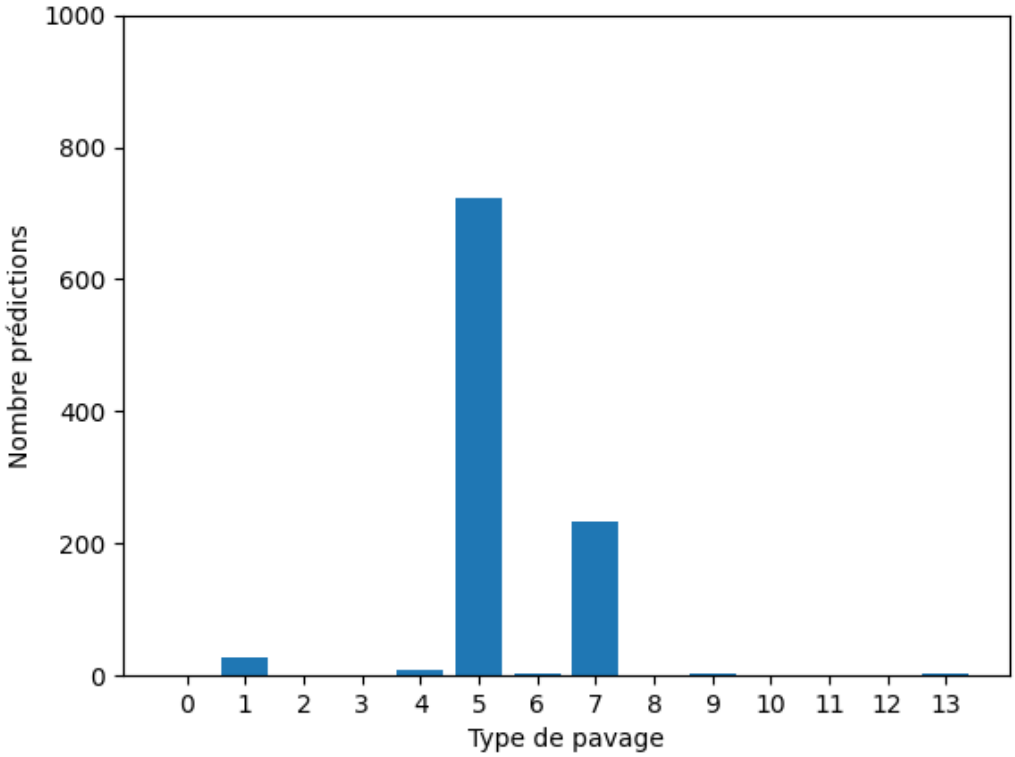

Figure 4.4 – Nombre de prédictions pour chaque type de pavage du réseau ResNet152V2 sur 1 000 images de C-PPTBF du type de pavage 5.

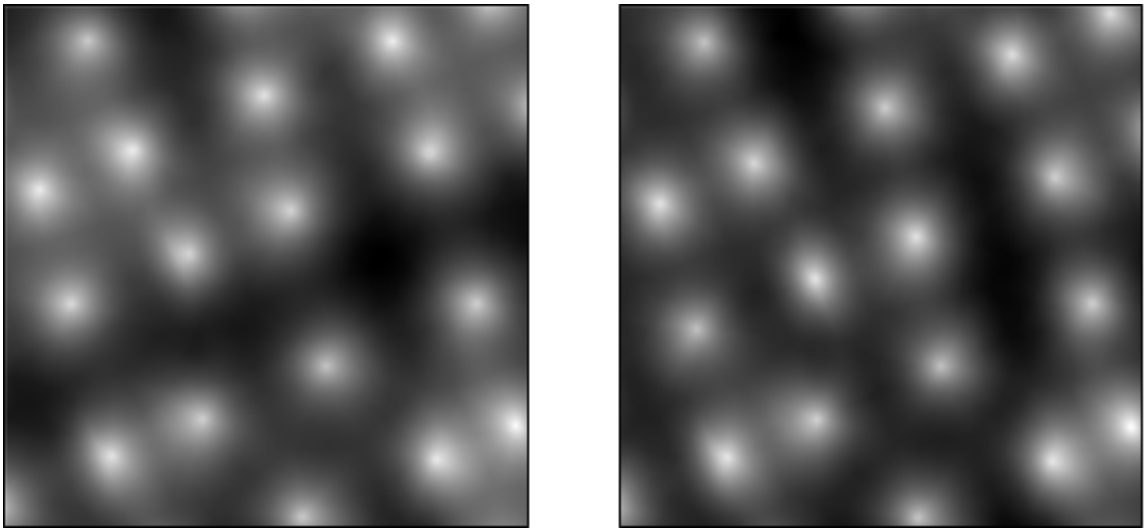

Figure 4.5 – Exemple d'image de C-PPTBF de type de pavage 4 estimé avec un type de pavage 6 (gauche) et image avec les mêmes paramètres continus mais un type de pavage 6 (droite).

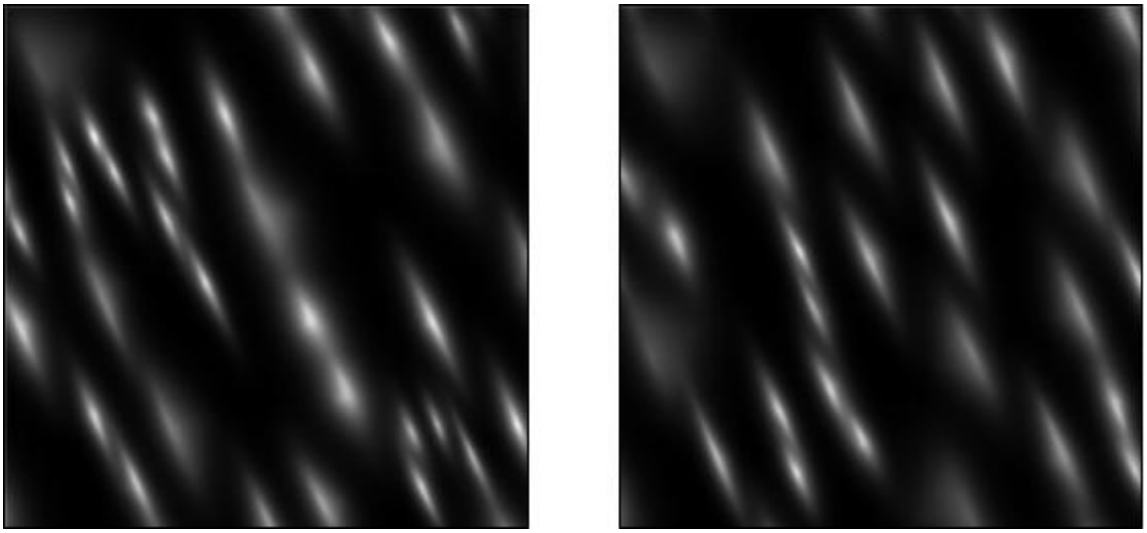

Figure 4.6 – Exemple d'image de C-PPTBF de type de pavage 5 estimé avec un type de pavage 7 (gauche) et image avec les mêmes paramètres continus mais un type de pavage 7 (droite).

Type de pavage	Précision	Rappel	Indice de Dice
0	0,922	0,891	0,906
1	0,871	0,917	0,894
2	0,955	0,931	0,943
3	0,893	0,959	0,925
4	0,780	0,730	0,754
5	0,842	0,722	0,777
6	0,742	0,808	0,774
7	0,719	0,813	0,763
8	0,995	0,983	0,989
9	0,985	0,989	0,987
10	0,995	0,998	0,996
11	0,999	0,994	0,996
12	0,997	0,956	0,976
13	0,998	0,971	0,984

Table 4.2 – Métriques de précision, rappel et indice de Dice selon le type de pavage sur un jeu de données de 14 000 images de C-PPTBF synthétiques.

4.4 Estimation des paramètres continus

Nous présentons maintenant les résultats que nous obtenons en utilisant les mêmes réseaux mais pour estimer cette fois-ci les valeurs de l'intégralité des paramètres continus du modèle de C-PPTBF. Les réseaux sont à nouveau pré-entraînés sur ImageNet et du transfert d'apprentissage est appliqué en affinant sur un jeu de données de 50 000 images de C-PPTBF pour 75 époques. Les valeurs de tous les paramètres sont normalisées dans l'intervalle [0,1] pour que tous les paramètres aient la même influence sur l'erreur quelque soit leur intervalle initial. Les résultats en termes d'erreur MSE (*Mean Squared Error*) sur la validation pour l'intégralité des paramètres continus sont présentés dans le tableau 4.3 et montrent que les meilleurs résultats sont obtenus avec le réseau d'architecture DenseNet169 avec une erreur MSE sur la validation de l'ensemble des paramètres continus de 0,292. On observe néanmoins que le réseau d'architecture ResNet152V2 avec lequel nous obtenions la meilleure précision sur le type de pavage n'est pas très loin derrière avec une erreur de 0,311, ce qui en fait une architecture de réseau efficace pour notre problématique.

De la même façon que pour le type de pavage, nous comparons ces résultats avec ceux que nous obtenons sur des images de structures binaires obtenues après seuillage

Architecture	Erreur MSE de validation sur les paramètres continus
Xception	0,314
VGG16	0,297
VGG19	0,295
ResNet50	0,387
ResNet101	0,388
ResNet152	0,383
ResNet50V2	0,323
ResNet101V2	0,309
ResNet152V2	0,311
InceptionV3	0,333
InceptionResNetV2	0,330
DenseNet121	0,295
DenseNet169	0,292
DenseNet201	0,297
NASNetMobile	0,335
NASNetLarge	0,314
EfficientNetB0	0,565
EfficientNetB1	0,486
EfficientNetB2	0,499
EfficientNetB3	0,436
EfficientNetB4	0,442
EfficientNetB5	0,396
EfficientNetB6	0,396
EfficientNetB7	0,412

Table 4.3 – Erreur MSE de validation sur les paramètres continus selon l'architecture du CNN avec un jeu de données d'images de C-PPTBF.

d'images de C-PPTBF. Ainsi un réseau d'architecture ResNet152V2 est entraîné sur un jeu de données d'images binaires avec une proportion de pixels noirs et blancs fixée à 50%. Nous obtenons une erreur MSE sur la validation de l'ensemble des paramètres continus de 0,431, ce qui est supérieur à ce que nous obtenions avec les images de C-PPTBF et justifiant à nouveau qu'il est préférable d'utiliser des images de C-PPTBF pour estimer les paramètres.

Ainsi, sur les deux types de paramètres, l'utilisation d'images de C-PPTBF est plus efficace pour estimer les paramètres qu'en utilisant des images binaires, comme

résumé dans le tableau 4.4. Ceci justifie le besoin d'une phase de reconstruction d'une image de C-PPTBF plausible, présentée dans la partie suivante.

	Précision du type de pavage	**Erreur MSE sur les paramètres continus**
Images de C-PPTBF	82,52%	0,311
Images binaires	70,37%	0,431

Table 4.4 – Précision sur le type de pavage et erreur MSE sur les paramètres continus selon le type d'images avec un réseau d'architecture ResNet152V2.

4.5 Reconstruction d'une image de C-PPTBF plausible

Nos expérimentations d'estimation des paramètres sur des images de structures binaires et sur des images de C-PPTBF montrent que les résultats de prédiction sont meilleurs sur des images d'intensité, ainsi il ne paraît pas judicieux d'estimer directement les paramètres des cartes de structures binaires en entrée. Nous commençons alors notre chaîne de traitement par une étape de reconstruction d'une image de C-PPTBF plausible $U(I)$ à partir de la carte de structures binaires I en entrée permettant de faciliter par la suite l'estimation des paramètres, et de pallier aussi la non différentiabilité de la fonction de seuillage. Afin de reconstruire cette image nous décidons d'utiliser un réseau antagoniste génératif conditionnel ($cGAN$) pour effectuer de la translation image-à-image entre la carte de structures binaires en entrée et une image de C-PPTBF plausible. Nous choisissons particulièrement le cGAN Pix2Pix [Iso+17] qui est entraîné pour apprendre une correspondance entre une carte de labels X et un vecteur de bruit aléatoire Z, et une image générée Y. Des exemples d'applications du modèle Pix2Pix sont présentés dans la figure 4.7. Ainsi dans notre cas X va représenter une carte de structures binaires et Y une image de C-PPTBF.

Le principe d'un réseau antagoniste génératif (GAN) est d'entraîner en concurrence deux réseaux de neurones différents : un réseau générateur G qui utilise un vecteur de bruit pour générer une image et un réseau discriminateur D qui doit déterminer si l'image qu'il reçoit est une image réelle provenant du jeu de données d'entraînement, ou une fausse image produite par G. Ainsi l'objectif de G est de

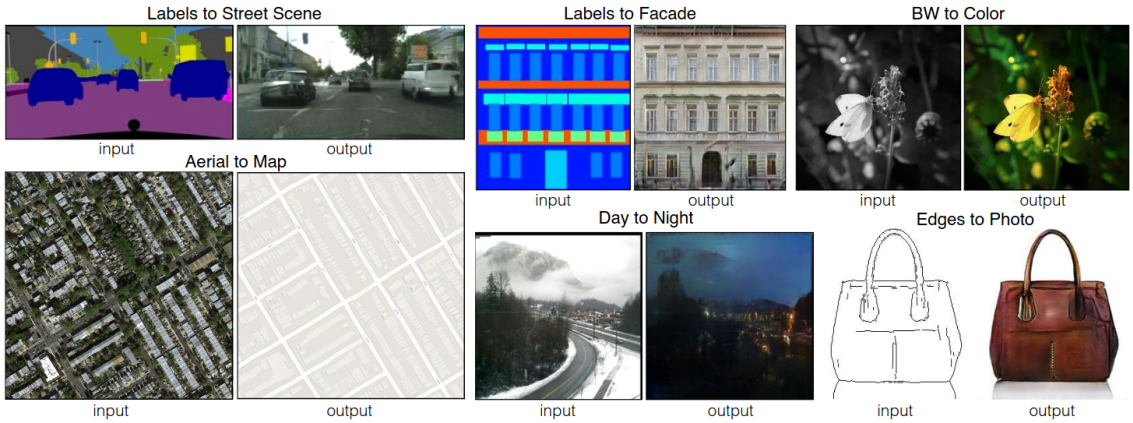

FIGURE 4.7 – Exemples d'applications du modèle Pix2Pix pour de la translation image-à-image [Iso+17].

tromper D et D doit apprendre à différencier des images réelles et fausses. Une fonction objectif est alors définie avec G cherchant à la minimiser alors que D cherche à la maximiser. Le modèle Pix2Pix utilise pour son générateur une architecture de type U-Net composé d'un encodeur et d'un décodeur. Le discriminateur quant à lui est un *PatchGAN* qui décompose en patchs l'image reçue, puis applique des couches de convolution pour classifier les patchs entre réels ou faux.

Un réseau antagoniste génératif conditionnel (cGAN) spécifiquement est une forme de GAN qui ajoute une condition en entrée du générateur. Dans le cas de Pix2Pix cette condition est la carte X, la fonction objectif d'un réseau cGAN est donc définie de cette sorte :

$$\mathcal{L}_{cGAN}(\mathrm{G},\mathrm{D}) = \mathbb{E}_{X,Y}[\log \mathrm{D}(X,Y)] + \mathbb{E}_{X,Z}[\log(1-\mathrm{D}(X,\mathrm{G}(X,Z)))] \tag{4.2}$$

Pix2Pix ajoute un autre terme à la fonction d'erreur globale du modèle, avec une norme L_1 entre l'image générée et l'image réelle de vérité terrain telle que :

$$\mathcal{L}_{L1}(\mathrm{G}) = \mathbb{E}_{X,Y,Z}[\|Y - \mathrm{G}(X,Z)\|_1]. \tag{4.3}$$

La fonction objectif de Pix2Pix est alors :

$$\hat{\mathrm{G}} = \arg\min_{\mathrm{G}} \max_{\mathrm{D}} \mathcal{L}_{cGAN}(\mathrm{G},\mathrm{D}) + \zeta_1 \mathcal{L}_{L1}(\mathrm{G})). \tag{4.4}$$

avec ζ_1 le facteur de régularisation appliqué sur le terme de norme L_1.

Avec cette fonction objectif de base de Pix2Pix, nous constatons que le modèle peut manquer de précision dans la reconstruction de structures qui peuvent être très proches entre elles ou proches des bords de l'image. Ainsi, afin de pallier ces problèmes et d'améliorer la ressemblance entre les cartes de structures en entrée et les cartes de structures obtenues après seuillage de l'image de sortie du réseau, nous décidons d'ajouter un nouveau terme à la fonction objectif initiale de Pix2Pix. Ce nouveau terme est une norme L_2 entre la carte de structures binaires en entrée, dont on connaît la valeur de seuillage comme elle est définie par la proportions de pixels noirs et blancs, et l'image générée par le réseau après seuillage notée $th(\mathrm{G}(X, Z))$, telle que :

$$\mathcal{L}_{bin}(\mathrm{G}) = \mathbb{E}_{X,Z}[\|X - th(\mathrm{G}(X, Z))\|_2]. \tag{4.5}$$

Pour conclure, la fonction objectif de notre version modifiée du réseau Pix2Pix se note alors de la façon suivante :

$$\hat{\mathrm{G}} = \arg\min_{\mathrm{G}} \max_{\mathrm{D}} \mathcal{L}_{cGAN}(\mathrm{G}, \mathrm{D}) + \zeta_1 \mathcal{L}_{L1}(\mathrm{G}) + \zeta_2 \mathcal{L}_{bin}(\mathrm{G}). \tag{4.6}$$

avec ζ_2 le facteur de régularisation appliqué sur le terme de norme L_2.

Nous entraînons ce réseau Pix2Pix modifié sur un jeu de données de paires d'images de C-PPTBF et structures binaires. Pour chaque image de C-PPTBF du jeu d'entraînement, huit images binaires sont générées avec des valeurs de seuil différentes : 0,05, 0,1, 0,15, 0,2, 0,3, 0,4, 0,6, et 0,8. Ainsi dans le jeu de données d'entraînement chaque image de C-PPTBF est présente dans huit paires différentes, avec à chaque fois une image binaire différente associée. Les valeurs de seuil choisies sont concentrées sur des petites valeurs car ce sont ces valeurs qui sont les plus présentes dans les cartes de structures extraites de textures naturelles, ce qui permet alors de meilleures reconstructions sur ces exemples.

Nous présentons dans la figure 4.8 différents résultats de reconstructions d'images de C-PPTBF plausibles que nous obtenons avec le modèle Pix2Pix en comparant plusieurs configurations, c'est-à-dire en utilisant la fonction objectif originale du modèle, ou la fonction modifiée ajoutant le nouveau terme portant sur les images binaires, et en modifiant les poids des différents termes de la fonction. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF plausible reconstruite par Pix2Pix et sa version seuillée pour trois configurations différentes.

Dans la première configuration le facteur de régularisation ζ_2 est fixé à 0, c'est-à-dire que nous utilisons la fonction d'erreur initiale de Pix2Pix, (b) et (c). Dans la deuxième configuration ζ_2 est fixé à 25, (d) et (e), et dans la troisième ζ_2 est fixé à 50, (f) et (g), ainsi nous utilisons la fonction d'erreur modifiée avec notre nouveau terme d'erreur. Dans tous les cas ζ_1 est fixé à 100. En comparant la carte de structures en entrée avec les images seuillées, nous remarquons que l'ajout du nouveau terme d'erreur permet de reconstruire plus fidèlement les détails des structures.

4.5.1 Utilisation de Pix2PixHD [Wan+18]

Bien que Pix2Pix soit un modèle performant et toujours très utilisé, d'autres GAN ont été créés dans les années suivantes et présentent des résultats de synthèse d'images supérieurs. Ici nous comparons les résultats obtenus par Pix2Pix avec ceux obtenus par le modèle Pix2PixHD, un modèle conçu par Wang et al. [Wan+18]. Pix2PixHD cherche à améliorer Pix2Pix en permettant de générer des images de taille beaucoup plus importantes, les auteurs mentionnent par exemple une taille de 4096×2048, contre 256×256 en général pour Pix2Pix. Cela permet alors d'améliorer le niveau de détail des images générées. Le générateur de Pix2PixHD est composé de deux réseaux utilisant des blocs résiduels qui sont associés pour permettre de générer une image d'une résolution importante. Le discriminateur du modèle est composé de trois réseaux qui vont chacun analyser l'image reçue à des échelles différentes afin de pouvoir capturer tous les détails de l'image, ce qui est nécessaire étant donné sa très grande résolution. L'autre modification importante porte sur la fonction objectif à minimiser du modèle, un nouveau terme d'erreur est ajouté, nommé *feature matching*, portant sur les feature vectors extraits depuis plusieurs couches du discriminateur entre l'image réelle et l'image générée. À l'inverse, la norme L_1 existant dans la fonction objectif de Pix2Pix entre l'image réelle et l'image générée disparaît de la nouvelle fonction. Ainsi, la fonction objectif de Pix2PixHD se note de la façon suivante :

$$\hat{\mathrm{G}} = \arg\min_{\mathrm{G}} \Big(\big(\max_{D_1, D_2, D_3} \sum_{k=1,2,3} \mathcal{L}_{GAN}(\mathrm{G}, D_k) \big) + \zeta_3 \sum_{k=1,2,3} \mathcal{L}_{FM}(\mathrm{G}, D_k) \Big). \tag{4.7}$$

avec $\mathcal{L}_{FM}$ le terme d'erreur de *feature matching*, D_1, D_2 et D_3 les trois réseaux composant le discriminateur du modèle et D_k le réseau considéré du discriminateur.

Nous présentons dans la figure 4.9 quelques résultats de reconstructions d'images de C-PPTBF plausibles que nous obtenons avec le modèle Pix2PixHD, générées en

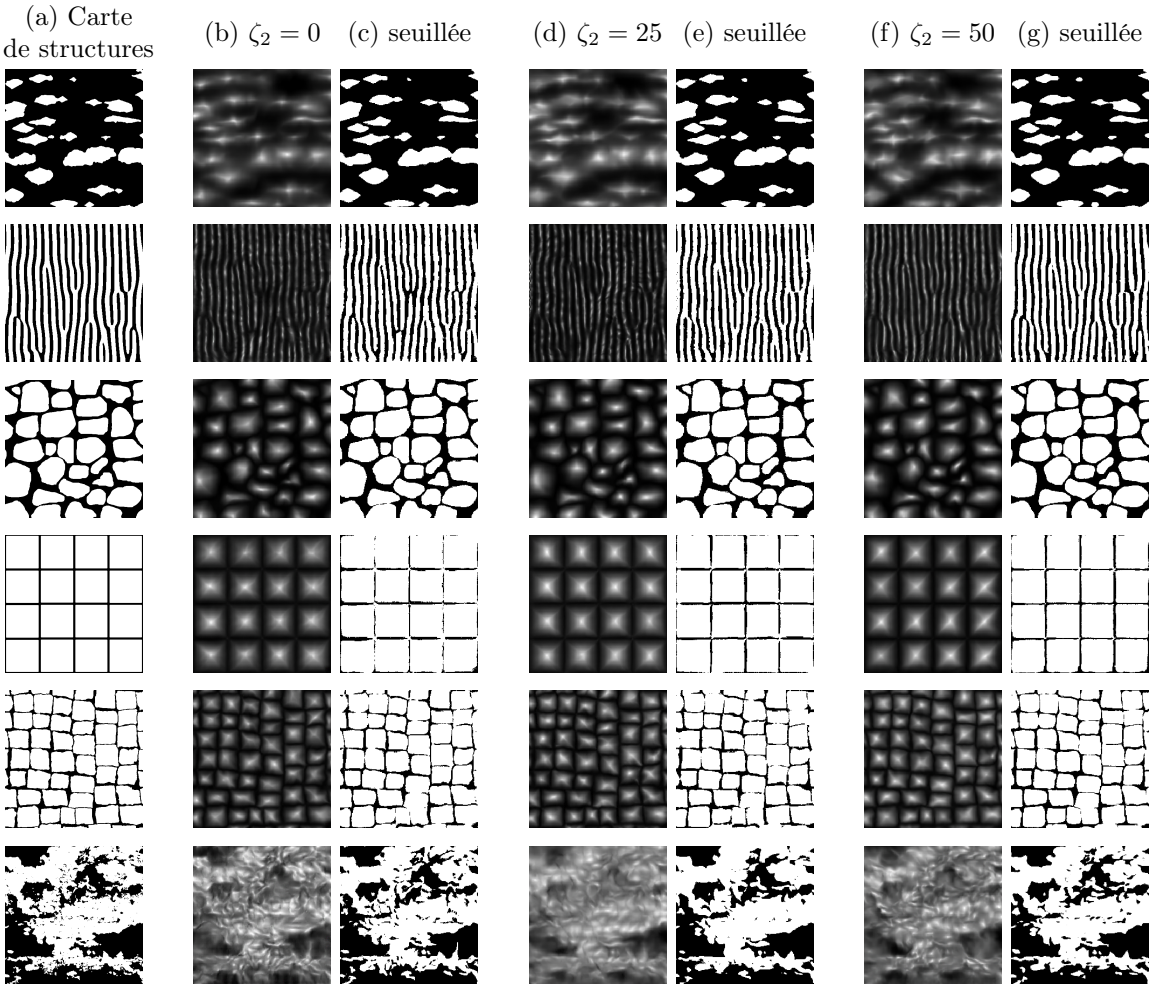

Figure 4.8 – Comparaison de résultats de Pix2Pix avec trois configurations de paramètres différentes. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF plausible reconstruite avec Pix2Pix avec $\zeta_2 = 0$ et (c) : sa version seuillée ; (d) : Image de C-PPTBF plausible reconstruite avec Pix2Pix avec $\zeta_2 = 25$ et (e) : sa version seuillée ; (f) : Image de C-PPTBF plausible reconstruite avec Pix2Pix avec $\zeta_2 = 50$ et (g) : sa version seuillée.

résolution 1024 × 1024 et redimensionnées en résolution 256 × 256 pour la figure et pour la suite de la chaîne de traitement, pour les comparer avec les résultats obtenus avec Pix2Pix. Nous fixons le facteur de régularisation ζ_3 portant sur le terme d'erreur de *feature matching* à 10. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF plausible reconstruite par Pix2Pix et sa version seuillée avec le facteur de régularisation ζ_2 fixé à 50, (b) et (c). Ensuite c'est l' image de C-PPTBF plausible reconstruite par Pix2PixHD et sa version seuillée qui sont données, avec ζ_2 fixé à 0, donc avec la fonction d'erreur initiale de Pix2PixHD, (d) et (e). Finalement nous intégrons notre terme d'erreur portant sur la norme L_2 entre la carte de structures en entrée et l'image seuillée de l'image générée au modèle Pix2PixHD, et nous présentons l'image de C-PPTBF plausible reconstruite par Pix2PixHD et sa version seuillée avec ζ_2 fixé à 50, (f) et (g). En comparant la carte de structures en entrée avec les différentes images binaires, nous pouvons constater que les détails sont plus fins sur les reconstructions obtenues avec Pix2PixHD qu'avec Pix2Pix, ce qui est particulièrement visible sur les bords des structures. De plus, les images de C-PPTBF plausibles reconstruites semblent ressembler davantage à de vraies images de C-PPTBF que celles générées avec Pix2Pix.

4.6 Première phase de l'estimation des paramètres

Une fois que nous avons reconstruit des images de C-PPTBF plausibles à partir de cartes de structures binaires nous pouvons procéder à l'estimation des paramètres. Cette estimation est divisée en deux phases, dans cette première phase nous estimons le type de pavage ainsi que des valeurs initiales pour l'intégralité des paramètres continus du modèle. Selon les expérimentations présentées dans la partie 4.3 pour le paramètre discret et dans la partie 4.4 pour les paramètres continus, nous faisons le choix d'utiliser un réseau CNN s'appuyant sur une architecture de type ResNet152V2. Nous utilisons TensorFlow 2 pour implémenter le réseau en prenant l'architecture entière sauf les couches entièrement connectées du haut du réseau, c'est-à-dire la base du modèle, puis nous ajoutons une couche de *GlobalAveragePooling*, une couche de *Dropout* avec une probabilité de 80% pour limiter le surentraînement et enfin les différentes couches entièrement connectées en sorties. Les paramètres étant tous estimés en même temps, il y a alors autant de couches en sortie que de paramètres à estimer. Pour le type de pavage, qui est le seul paramètre discret, la fonction d'activation est le *Softmax* afin d'obtenir des probabilités pour chaque type de pavage, et pour les paramètres continus c'est une fonction linéaire qui est utilisée pour l'activation. L'architecture utilisée est illustrée dans la figure 4.10.

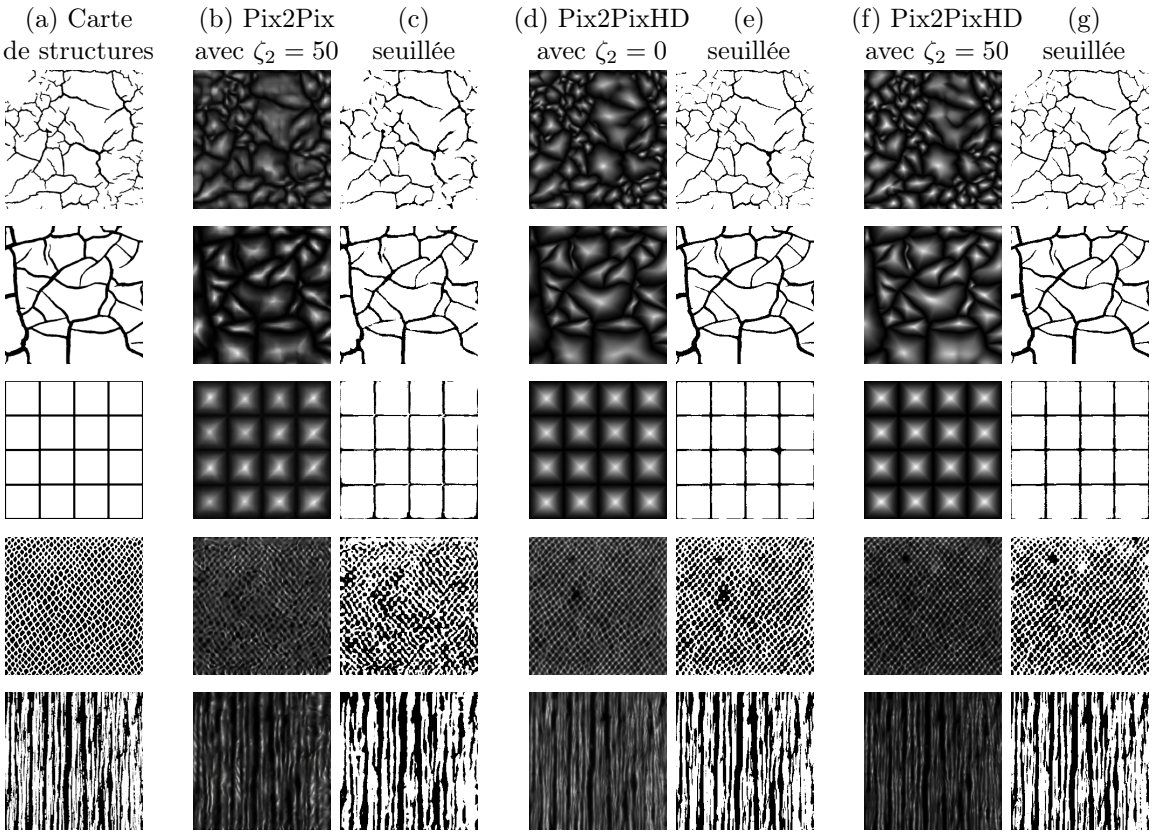

Figure 4.9 – Comparaison de résultats entre Pix2Pix et Pix2PixHD. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF plausible reconstruite avec Pix2Pix avec $\zeta_2 = 50$ et (c) : sa version seuillée ; (d) : Image de C-PPTBF plausible reconstruite avec Pix2PixHD avec $\zeta_2 = 0$ et (e) : sa version seuillée ; (f) : Image de C-PPTBF plausible reconstruite avec Pix2PixHD avec $\zeta_2 = 50$ et (g) : sa version seuillée.

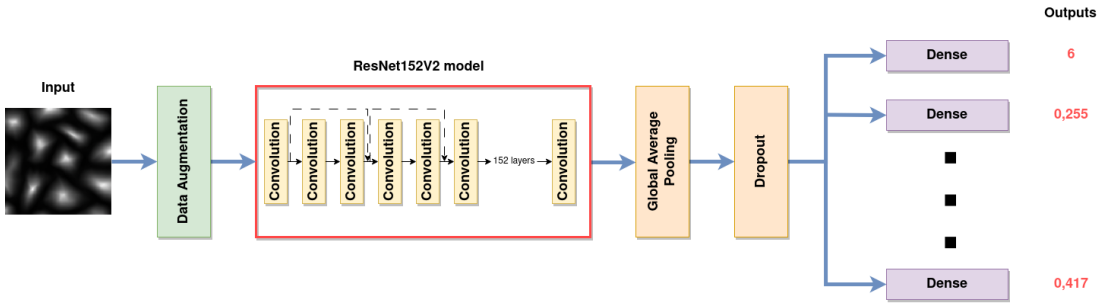

FIGURE 4.10 – Architecture utilisant le réseau ResNet152V2 utilisée pour la première phase de l'estimation des paramètres.

Les poids d'un réseau pré-entraîné sur le jeu de données ImageNet sont utilisés pour initialiser le réseau. L'entraînement se fait en deux parties, dans la première toutes les couches de la base du modèle sont gelées pour que les poids ne se modifient pas pour un entraînement de 50 époques sur un jeu de données de 50 000 images de C-PPTBF de résolution 200 × 200. Ces images sont générées à la volée avec un échantillonnage uniforme des paramètres de C-PPTBF. Le taux d'apprentissage est fixé à 1e-3. La deuxième partie est une phase de raffinement dans laquelle toutes les couches sont dégelées, le modèle est alors entraîné pendant 75 époques avec un taux d'apprentissage plus bas fixé à 1e-5. Pendant l'entraînement nous obtenons pour le type de pavage un pourcentage de précision sur l'estimation du bon type de pavage, et pour les paramètres continus une erreur MSE entre les valeurs estimées et les valeurs réelles.

Ainsi une fois le réseau entraîné il peut être utilisé pour prédire les valeurs à partir d'une image de C-PPTBF plausible reconstruite. On obtient en sortie des probabilités en pourcentage pour chaque type de pavage, et des valeurs initiales pour les paramètres continus.

L'intégralité des paramètres du modèle est estimée dans le même réseau à l'exception du paramètre de rotation α pour lequel on obtient les meilleures estimations quand il est estimé seul avec un autre réseau. Ainsi un deuxième réseau de type ResNet152V2 est entraîné de la même façon pour estimer uniquement le paramètre α. Avec ce réseau séparé, l'erreur quadratique moyenne sur le paramètre α est de 0,0076 contre 0,0155 quand il est estimé en même temps que les autres paramètres. C'est d'autant plus important d'avoir une bonne estimation initiale de ce paramètre car il peut être difficile d'affiner sa valeur lors de l'optimisation de la deuxième phase. En effet une valeur de rotation qui donnerait une apparence trop différente de celle qu'on

recherche peut mener à des structures trop éloignées de ce que l'on souhaite obtenir, pouvant compliquer l'optimisation qui risque de vouloir affiner les autres paramètres sans parvenir à avoir une rotation appropriée.

De plus nous constatons que le paramètre de mise à l'échelle s pose problème dans l'estimation, principalement car selon les types de pavage le nombre de cellules visibles n'est pas le même. Par exemple pour les types complexes, donc ceux des types 8 à 13, un faible niveau de mise à l'échelle va déjà donner un grand nombre de cellules contrairement aux types de 0 à 7, ainsi ce paramètre a une influence différente selon les types. Pour pallier ce problème, au lieu d'estimer le paramètre s nous allons plutôt estimer à la place le nombre de feature points K. Nous pouvons connaître le nombre de feature points à partir du type de pavage et du facteur de mise à l'échelle, ainsi si le réseau estime le type de pavage et le nombre de feature points, nous pouvons déduire le facteur de mise à l'échelle à utiliser. Pour une valeur de facteur de mise à l'échelle donnée, le nombre de feature points générés est différent selon le type de pavage, comme illustré dans la figure 4.11. On peut constater que le nombre de feature points croît linéairement pour les types de pavage 0, 2, 3, 4 et 6, c'est-à-dire les types n'ayant pas de possible subdivision horizontale ou verticale des cellules. À l'inverse le nombre de feature points des types de pavage 1, 5, 7, 8, 9, 10, 11, 12 et 13 croît de manière polynomiale voire exponentielle en raison du plus grand nombre de cellules dans ces types dû aux subdivisions et de leurs motifs de pavages plus complexes.

4.7 Deuxième phase de l'estimation des paramètres

Une fois que le type de pavage est établi, et que les valeurs initiales pour chaque paramètre continu du modèle sont estimées, nous passons à une deuxième phase de l'estimation dans laquelle nous optimisons les valeurs des paramètres continus en utilisant une descente de gradient stochastique. Cette optimisation par le gradient est possible car le modèle de C-PPTBF est différentiable en fonction de tous ses paramètres continus.

L'objectif de cette descente de gradient stochastique est de résoudre le problème d'optimisation posé dans l'équation 4.1. Nous avons alors l'image de C-PPTBF plausible $U(I)$ reconstruite par un réseau génératif pour laquelle les paramètres estimés de type de pavage et de rotation, ainsi que le facteur de mise à l'échelle déduit à partir du nombre de feature points, sont fixés au début de la descente de gradient. Ainsi les 10 paramètres continus restants dont nous avons obtenu des valeurs initiales

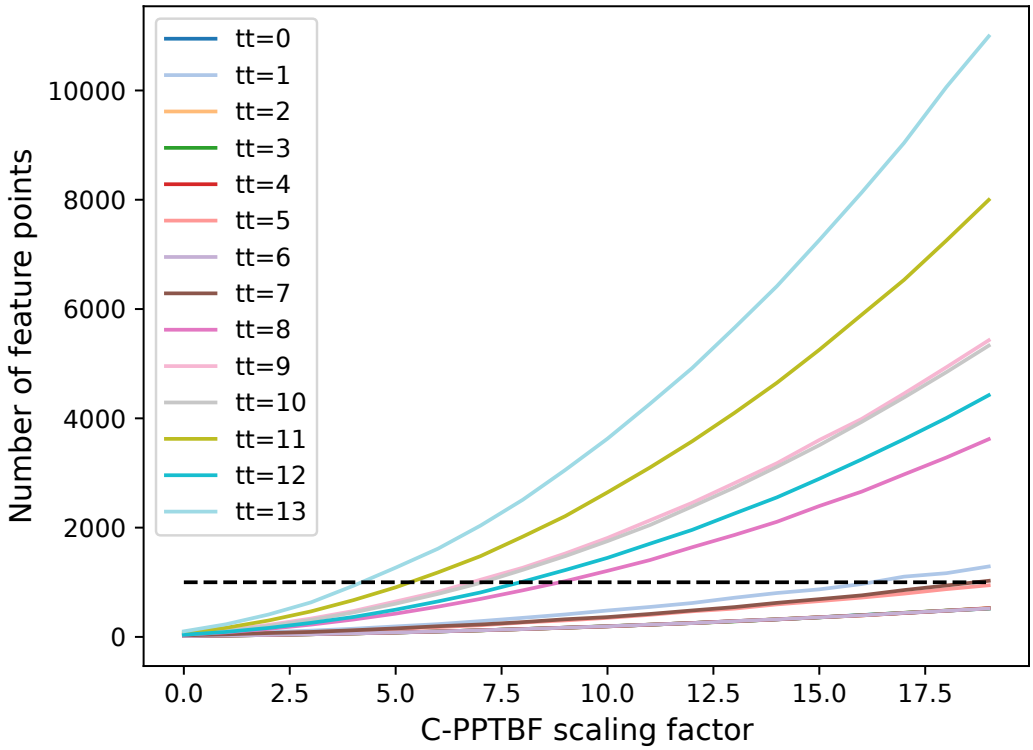

Figure 4.11 – Nombre de feature points pour une valeur de facteur de mise à l'échelle donnée selon le type de pavage [BAD23].

lors de la phase précédente, ainsi que le paramètre de déformation a, peuvent être optimisés. L'image de C-PPTBF générée avec les paramètres estimés durant la descente de gradient se note R. Nous cherchons alors à minimiser une fonction d'erreur entre la version seuillée de R et la carte de structures binaires en entrée I pendant cette descente de gradient.

4.7.1 Fonction objectif

Il apparaît qu'utiliser une simple erreur fondée sur les pixels entre les deux images n'est pas appropriée, en effet nous cherchons surtout à ce que les formes des structures soient les plus ressemblantes possibles, sans qu'elles ne soient forcément situées aux mêmes endroits dans les deux images. De plus le modèle de C-PPTBF ne pouvant pas

couvrir toutes les structures naturelles possibles, une erreur fondée sur les pixels peut poser problème si l'exemple en entrée est trop différent de la gamme de structures représentables par le modèle.

Nous souhaitons alors utiliser des descripteurs d'images pour la fonction d'erreur à minimiser, qui seraient plus appropriés dans notre cas pour caractériser les images.

Nous décidons d'utiliser la distance de Wasserstein tranchée 1D (*SWD1D*), inspirés par Heitz et al. [Hei+21], qui a pour avantage de pouvoir capturer les statistiques stationnaires dans une image, ce qui lui permet alors de ne pas dépendre de la position des structures dans l'image. La SWD1D va consister à utiliser des feature vectors des deux images extraits à partir d'un réseau VGG19 pré-entraîné qui sont ensuite projetés dans des directions aléatoires, les vecteurs projetés sont alors triés et finalement comparés avec une distance L2. Ainsi, la fonction d'erreur se note de la façon suivante :

$$\mathcal{L}^k_{SW}(\mathrm{U}(I), R) = \mathbb{E}_V[\mathcal{L}^k_{SW1D}(p_V, \hat{p}_V)] \tag{4.8}$$

avec k correspondant à une couche de convolution du réseau VGG19, et $\mathcal{L}^k_{SW1D}$ se notant :

$$\mathcal{L}^k_{SW1D}(p_V, \hat{p}_V) = \frac{1}{|p_V|}\|\mathrm{sort}(p_V) - \mathrm{sort}(\hat{p}_V)\|^2 \tag{4.9}$$

où p_V correspond aux feature vectors projetés de $U(I)$ et $\hat{p}_V$ aux feature vectors projetés de l'image de C-PPTBF R. Ainsi la fonction d'erreur finale sur l'intégralité des couches de convolution concernées du réseau VGG19 est :

$$\mathcal{L}_{SW}(\mathrm{U}(I), R) = \mathbb{E}_k[\mathcal{L}^k_{SW}(\mathrm{U}(I), R)]. \tag{4.10}$$

Le processus de calcul de la SWD1D pour la sortie d'une couche convolutionnelle k du réseau VGG19 est illustré dans la figure 4.12.

Nous devons déterminer maintenant quelles couches de convolution du réseau VGG19 utiliser. Dans leur méthode de synthèse de textures, Heitz et al. [Hei+21] utilisent la SWD1D en prenant les 12 premières couches de convolution du réseau, soit toutes les couches des 4 premiers blocs. De précédents travaux utilisant des matrices de Gram ont pu utiliser toutes les couches de convolution du réseau VGG19, soit 16 couches, comme par exemple Gatys et al. [GEB15a] pour de la synthèse de textures. D'autres travaux toujours avec des matrices de Gram ont utilisé les premières couches de chaque bloc du réseau, soit 5 couches au total, comme Gatys et al. [GEB16] pour

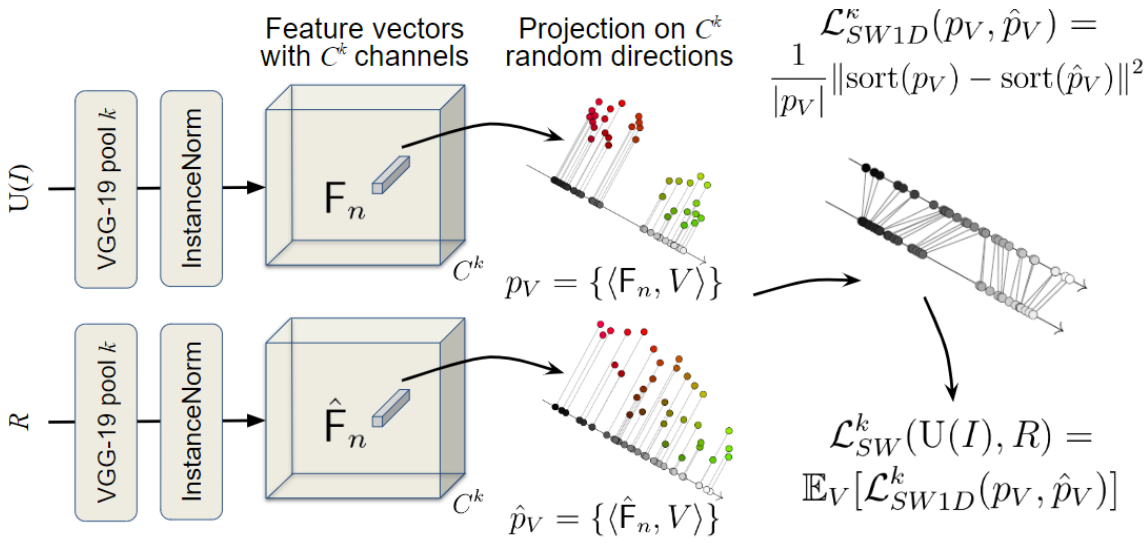

Figure 4.12 – Calcul de la distance de Wasserstein tranchée 1D pour la sortie de la couche convolutionnelle k du réseau VGG19 [BAD23].

du transfert de style, ce qui a donné la fonction d'erreur de style aussi utilisée par exemple par Hu et al. [Hu+22a] et [Hu+22c] ou par Guo et al. [Guo+20a]. Un autre exemple est celui de Guo et al. [Guo+20b] utilisant une fonction d'erreur perceptuelle issue de Johnson et al. [JAF16], portant sur les feature vectors de 4 couches du réseau, les deux premières couches du premier bloc, et les deuxièmes couches des blocs 3 et 4. Nous avons proposé dans nos travaux [BAD23] d'utiliser toutes les couches des trois premiers blocs du réseau, en partant de l'hypothèse que le réseau VGG19 étant pré-entraîné sur le jeu de données ImageNet, qui est un jeu de données comprenant des images naturelles de toutes sortes, il pouvait être ainsi pertinent d'exclure les blocs 4 et 5 qui représentent des features propres à ces images naturelles ne correspondant pas à nos images de C-PPTBF.

Nous proposons une étude détaillée sur les couches de convolution à utiliser dans la partie 5.3.1.

4.7.2 Canaux des images

Les images de C-PPTBF que nous utilisons sont en niveaux de gris et ne possèdent alors qu'un seul canal. Le réseau VGG19 prend en entrée des images RGB donc à trois canaux, ainsi habituellement il faudrait dupliquer sur les trois canaux pour chaque pixel la valeur de nuances de gris de l'image de C-PPTBF. Néanmoins, nous

avons remarqué dans nos expérimentations que d'autres constructions de l'image à trois canaux sont possibles et seraient meilleures que cette construction dupliquée habituelle. Ainsi, il est possible d'intégrer les images de structures binaires à cette construction, et donc de former une combinaison en entrée de $U(I)$ et I qui serait alors comparée lors de la descente de gradient à une construction reposant sur R et sa version seuillée.

Nous proposons une étude détaillée sur les différentes constructions possibles des trois canaux dans la partie 5.3.2.

4.7.3 Procédure d'optimisation

Pour optimiser les valeurs des paramètres continus nous utilisons une descente de gradient stochastique avec l'optimiseur Adam [KB15].

La SWD1D est une fonction d'erreur non convexe qui peut aboutir à de multiples minima locaux, c'est pourquoi nous utilisons l'algorithme du *Basin-Hopping* [IO04], inspiré des travaux de Gaillard et al. [Gai+22] sur le rendu différentiable. Cette méthode, inspirée par l'algorithme de Monte Carlo, alterne entre descente de gradient pour trouver un minimum local et des sauts aléatoires vers d'autres régions de l'espace de recherche afin de potentiellement trouver un meilleur minimum local, tout en excluant les régions possédant des minima trop élevés.

Nous définissons θ l'ensemble des paramètres estimés pendant la descente de gradient associés à R, tel que $\theta = (\theta_T, \beta, \theta_W, \theta_F)$. Les valeurs initiales de θ sont les valeurs obtenues lors de la première phase d'estimation de paramètres. Ainsi la fonction d'erreur minimisée par l'optimisation se note :

$$\hat{\mathcal{L}}_{SW}(\theta) = \min \mathcal{L}_{SW}\big(\mathrm{U(I)}, R\big). \tag{4.11}$$

Le principe de l'algorithme de Basin-Hopping est de perturber les paramètres estimés aléatoirement à la fin d'un certain nombre d'itérations de descente de gradient, selon une probabilité calculée par l'erreur et une température T définie. Nous définissons un nombre d'itérations de l'algorithme et le nombre d'itérations de descente de gradient stochastique qui seront effectuées à chacune des itérations du Basin-Hopping. Une première itération de l'algorithme est effectuée durant laquelle la descente de gradient stochastique va tenter d'obtenir des valeurs de θ donnant une erreur inférieure à celle obtenue avec les valeurs initiales, la meilleure configuration sera notée θ_1 et l'erreur associée err_1. À la fin de cette première itération, une perturbation des valeurs de θ_1 est effectuée. Nous définissons la perturbation par une valeur aléatoire qui est ajoutée à chaque valeur de paramètre, l'intervalle de cette valeur étant fixé à $[-0,8, 0,8]$ afin de permettre d'effectuer des grandes variations

sur les paramètres. La boucle sur le nombre d'itérations du Basin-Hopping débute alors, pour chercher à obtenir une meilleure configuration de θ et ainsi minimiser l'erreur obtenue lors de la première itération. À la fin de chaque itération un test d'acceptation est effectué pour déterminer quelles valeurs de θ seront perturbées. Le test d'acceptation se calcule de la façon suivante :

$$\text{accept}(\theta_1 \to \theta_i) = \min \exp\left(-\left(\hat{\mathcal{L}}_{SW}(\theta_i) - \hat{\mathcal{L}}_{SW}(\theta_1)\right)/T\right) \tag{4.12}$$

avec θ_i les valeurs des paramètres qui minimisent l'erreur lors de l'itération de Basin-Hopping i actuellement considérée, et err_i cette erreur. Le test d'acceptation est réussi si err_i est inférieure à l'erreur la plus basse mesurée jusqu'à présent lors de l'optimisation, ou si le résultat de l'équation 4.12 est supérieur à une valeur aléatoire entre 0 et 1. Si le test d'acceptation est réussi, alors la prochaine itération commencera avec un θ égal à une version de θ_i perturbée, sinon le θ sera égal à une version de θ_1 perturbée.

Afin de détecter rapidement qu'un minimum local a été atteint lors d'une itération de Basin-Hopping, nous utilisons une Moyenne mobile exponentielle sur l'erreur calculée à chaque itération de descente de gradient stochastique afin d'arrêter l'itération de l'algorithme si l'erreur moyenne augmente trop fortement, ce qui indiquerait qu'il est très improbable de trouver un meilleur minimum local lors de cette itération de Basin-Hopping. Cela permet alors de passer plus rapidement à l'itération suivante et donc de gagner un peu de temps d'exécution.

La procédure d'optimisation avec l'algorithme de Basin-Hopping est présentée de manière détaillée dans l'algorithme 2. err_{best} représente l'erreur minimale obtenue lors de l'optimisation et θ_{best} ses paramètres associés, bh_steps le nombre d'itérations de Basin-Hopping et sgd_steps le nombre d'itérations de descente de gradient par itération de Basin-Hopping. La fonction $PERTURBATION$ représente la perturbation telle que définie précédemment, la fonction SGD la descente de gradient effectuée à partir des paramètres donnés en entrée, la fonction $SWD1D$ le calcul de l'erreur par la SWD1D à partir des paramètres donnés en entrée et la fonction $ACCEPT\_TEST$ le test d'acceptation comme défini dans l'équation 4.12. Finalement la fonction EMA représente le test sur la Moyenne mobile exponentielle qui arrête la boucle de descente de gradient si la valeur de err_j, l'erreur calculée sur les paramètres θ_j, tend à augmenter pendant un certain nombre d'itérations de descente de gradient.

Algorithme 2 Procédure d'optimisation par algorithme de Basin-Hopping

$\theta_{best} \leftarrow \theta_1$, $err_{best} \leftarrow err_1$;
$\theta_i = PERTURBATION(\theta_1)$;
pour $i = 0$ à bh_steps **faire**
    $err_i \leftarrow SWD1D(\theta_i)$;
    $\theta_j \leftarrow \theta_i$, $err_j \leftarrow err_i$;
    pour $j = 0$ à sgd_steps **faire**
        $SGD(\theta_j)$;
        $err_j \leftarrow SWD1D(\theta_j)$;
        si $err_j < err_i$ **alors**
            $err_i \leftarrow err_j$, $\theta_i \leftarrow \theta_j$;
        fin si
        si EMA **alors**
            break
        fin si
    fin pour
    si $err_i < err_{best}$ **alors**
        $err_{best} \leftarrow err_i$, $\theta_{best} \leftarrow \theta_i$;
    fin si
    si $ACCEPT\_TEST$ **alors**
        $\theta_i = PERTURBATION(\theta_i)$;
    sinon
        $\theta_i = PERTURBATION(\theta_{best})$;
    fin si
fin pour

Chapitre 5

Résultats et Discussion

Dans cette partie nous présentons les résultats que nous obtenons avec notre chaîne de traitement pour les différentes étapes détaillées dans le chapitre précédent.

5.1 Reconstruction d'images de C-PPTBF plausibles

Comme présenté dans la partie 4.5, la première étape de notre chaîne de traitement consiste en la reconstruction d'une image de C-PPTBF plausible à partir de cartes de structures binaires. Nous avons démontré l'efficacité des GAN pour réaliser cette reconstruction, avec les réseaux Pix2Pix et Pix2PixHD, ce dernier permettant d'obtenir davantage de détails. Nous présentons ainsi ici les résultats obtenus avec Pix2PixHD en utilisant la fonction d'erreur présentée dans l'équation 4.7 à laquelle nous ajoutons le nouveau terme portant sur la norme L_2 entre la carte de structures en entrée et la version seuillée de l'image générée par le réseau, présenté dans l'équation 4.5. Nous fixons le paramètre de régularisation ζ_3 appliqué sur le terme d'erreur de *feature matching* à 10 et le paramètre de régularisation ζ_2 appliqué sur le terme de norme L_2 à 50. Le jeu de données utilisé pour l'entraînement comporte 240 000 paires d'images de C-PPTBF et binaires au total, il n'y a en fait que 30 000 images de C-PPTBF différentes car pour chaque image de C-PPTBF huit images binaires obtenues avec des valeurs de seuil différentes sont utilisées. Les images de C-PPTBF, d'une résolution de 256 $\times$ 256, sont générées avec un échantillonnage uniforme des différents paramètres du modèle, selon les intervalles présentés dans le tableau 3.1. Nous utilisons l'implémentation PyTorch officielle de Pix2PixHD modifiée pour y ajouter le nouveau terme d'erreur et la prise en compte des valeurs de seuil.

Nous avons entraîné le réseau Pix2PixHD jusqu'à 33 époques, pendant environ 1

mois sur un NVidia P100/16GB SXM2, avec des images en entrée de résolution 256 $\times$ 256 et des images en sortie de résolution 1024 $\times$ 1024, ensuite redimensionnées en 256 $\times$ 256. Nous avons produit nos résultats uniquement en considérant la dernière époque. Nous n'avons pas réalisé d'étude pour déterminer si un réseau entraîné avec un nombre d'époque inférieur, et donc un temps d'entraînement plus court, pourrait donner des résultats similaires. À titre de comparaison, nous arrivons au même nombre d'époques avec le réseau Pix2Pix en seulement 62 heures, pour des images d'une résolution de 256 $\times$ 256. De plus, en utilisant la fonction d'erreur d'origine de Pix2PixHD, c'est-à-dire en n'incluant pas notre nouveau terme d'erreur, 38 époques d'entraînement peuvent être effectuées en 1 mois. Ainsi, l'utilisation de Pix2PixHD à la place de Pix2Pix et l'inclusion de notre nouveau terme d'erreur dans la fonction d'erreur du réseau augmentent le temps d'entraînement du réseau. Le fait de prendre Pix2PixHD plutôt que Pix2Pix améliore ainsi les résultats obtenus pour cette reconstruction et ainsi pour les phases suivantes, mais au prix d'un temps d'entraînement beaucoup plus long par époque d'un facteur 12. La reconstruction d'une image à partir d'un exemple prend ensuite moins d'une seconde.

Les résultats sont présentés dans la figure 5.1. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF plausible reconstruite par Pix2PixHD (b) et sa version seuillée (c). Les images seuillées sont obtenues de manière à respecter la proportion de pixels noirs et blancs dans l'exemple en entrée. On peut constater visuellement que les images seuillées se rapprochent des cartes de structures en entrée. Pour le vérifier numériquement, nous avons calculé une erreur L2 pixel à pixel entre la carte de structures en entrée et la version seuillée de l'image de C-PPTBF plausible reconstruite, et ce pour l'intégralité d'un jeu de données de 84 cartes de structures binaires extraites de textures naturelles, et nous faisons la somme des erreurs. Nous obtenons une somme d'erreurs de 2,026 en utilisant Pix2PixHD, à titre de comparaison nous obtenons une somme d'erreurs de 2,843 en utilisant Pix2Pix à la place, ce qui montre la supériorité de la reconstruction de Pix2PixHD sur Pix2Pix visuellement et numériquement.

Le réseau peut néanmoins avoir des difficultés à reconstruire les détails très fins. On peut le remarquer sur le cinquième exemple dans lequel les rayures blanches sont très fines dans la carte de structures, la version seuillée de l'image générée par le réseau va mélanger des rayures proches car l'écart entre les rayures sur l'image de C-PPTBF plausible est trop mince, résultant alors en des tâches plus épaisses que les rayures initiales. De plus, quand il y a une trop grande étendue de pixels blancs dans la carte de structures, le réseau peut donner une reconstruction incohérente. On le

constate dans le quatrième exemple, dans la partie supérieure gauche de l'image reconstruite le réseau ne sait pas comment remplir cette zone entièrement blanche dans la carte de structures qui n'est pas délimitée. Contrairement au premier exemple dans lequel les craquelures représentées par les pixels noirs délimitent les zones blanches, permettant alors au réseau de former des cellules.

5.2 Première phase de l'estimation

Une fois les images de C-PPTBF plausibles obtenues par le réseau Pix2PixHD, nous pouvons passer à l'estimation des paramètres de ces images. Comme présenté dans la partie 4.6, pour la première phase de l'estimation, nous utilisons deux réseaux de neurones d'architecture ResNet152V2. Dans le premier réseau, 10 paramètres sont estimés : (tt, β, ω, λ, s_c, σ, ϕ, η, γ et K). Dans le deuxième réseau, le seul paramètre estimé est α. Les deux réseaux sont entraînés avec un jeu de données de 50 000 d'images de C-PPTBF générées à la volée de résolution 200 $\times$ 200 générées avec un échantillonnage uniforme des différents paramètres du modèle, selon les intervalles présentés dans le tableau 3.1, et implémentés avec TensorFlow 2. Les réseaux sont entraînés sur 50 époques d'entraînement durant lesquelles les différentes couches sont gelées avec un taux d'apprentissage de 1e-3, suivies de 75 époques de raffinement où les couches sont dégelées avec un taux d'apprentissage de 1e-5. Cet entraînement prend environ 4 heures sur un NVidia P100/16GB SXM2, et l'estimation en elle-même prend ensuite moins d'une seconde.

Cette première phase ne donne normalement que des paramètres qui sont ensuite utilisés comme initialisation pour la deuxième phase, mais pour visualiser l'influence de ces paramètres sur la chaîne de traitement nous présentons dans la figure 5.2 les images de C-PPTBF générées avec ces paramètres. Pour chaque exemple, la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés lors de cette première phase (b) et sa version seuillée (c). Nous pouvons constater que les images seuillées obtenues sont assez éloignées des cartes de structures binaires en entrée, ce qui montre que cette première phase seule n'est pas suffisante pour obtenir des résultats satisfaisants. Cependant l'utilité principale de cette phase dans notre chaîne de traitement est d'estimer un type de pavage qui sera fixe pour le reste de la chaîne, et des valeurs initiales pour les paramètres continus pour servir de base à l'optimisation par le gradient de la deuxième phase. Ainsi, on observe ici que les types de pavage estimés semblent cohérents et permettraient d'obtenir des images seuillées proches des exemples d'entrée si les valeurs des paramètres continus sont ensuite optimisées.

(a) Carte de structures (b) Image de C-PPTBF plausible reconstruite (c) Image seuillée

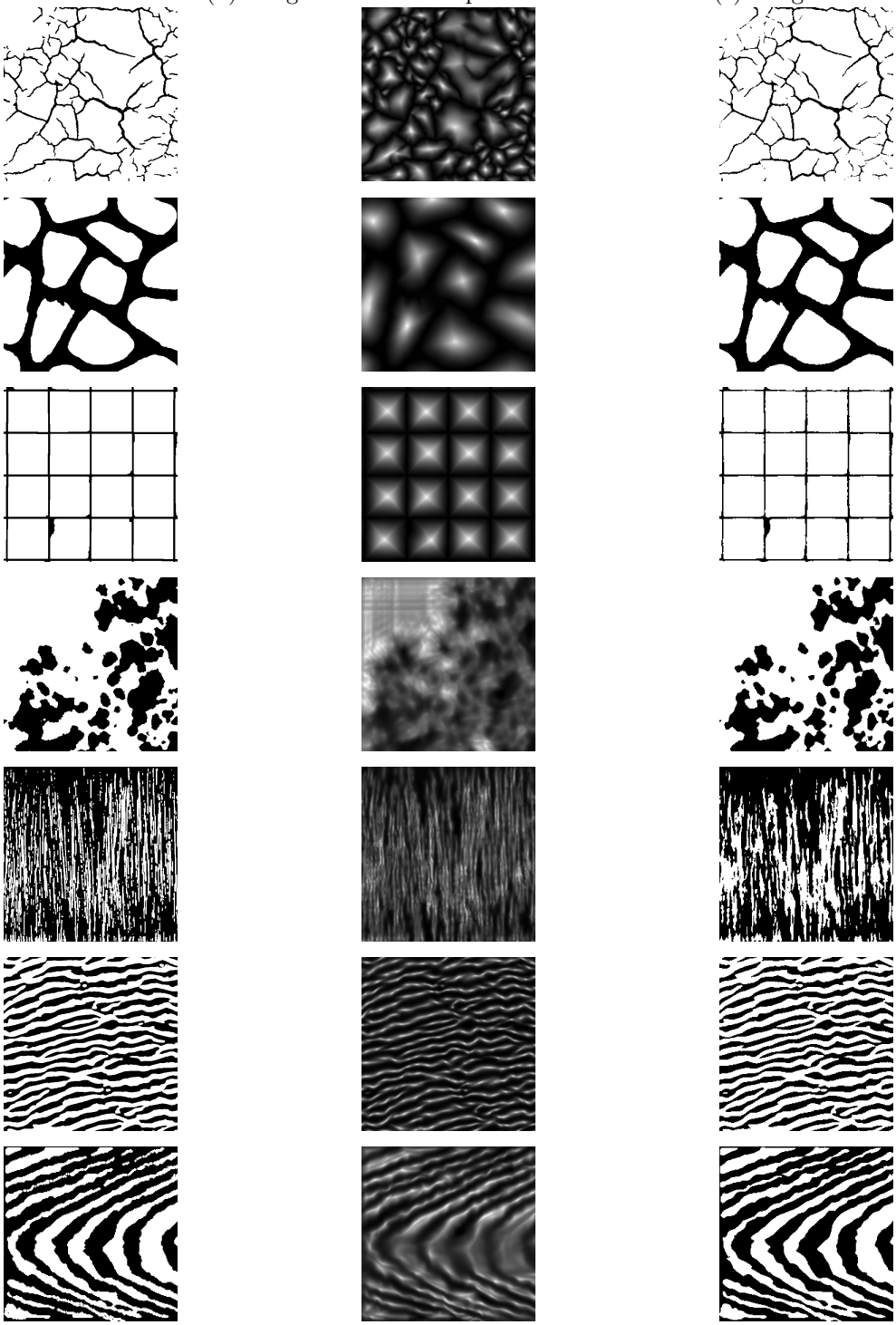

Figure 5.1 – Résultats de reconstructions d'images de C-PPTBF plausibles obtenues avec le réseau Pix2PixHD. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF plausible reconstruite avec Pix2PixHD ; (c) : Version seuillée de (b).

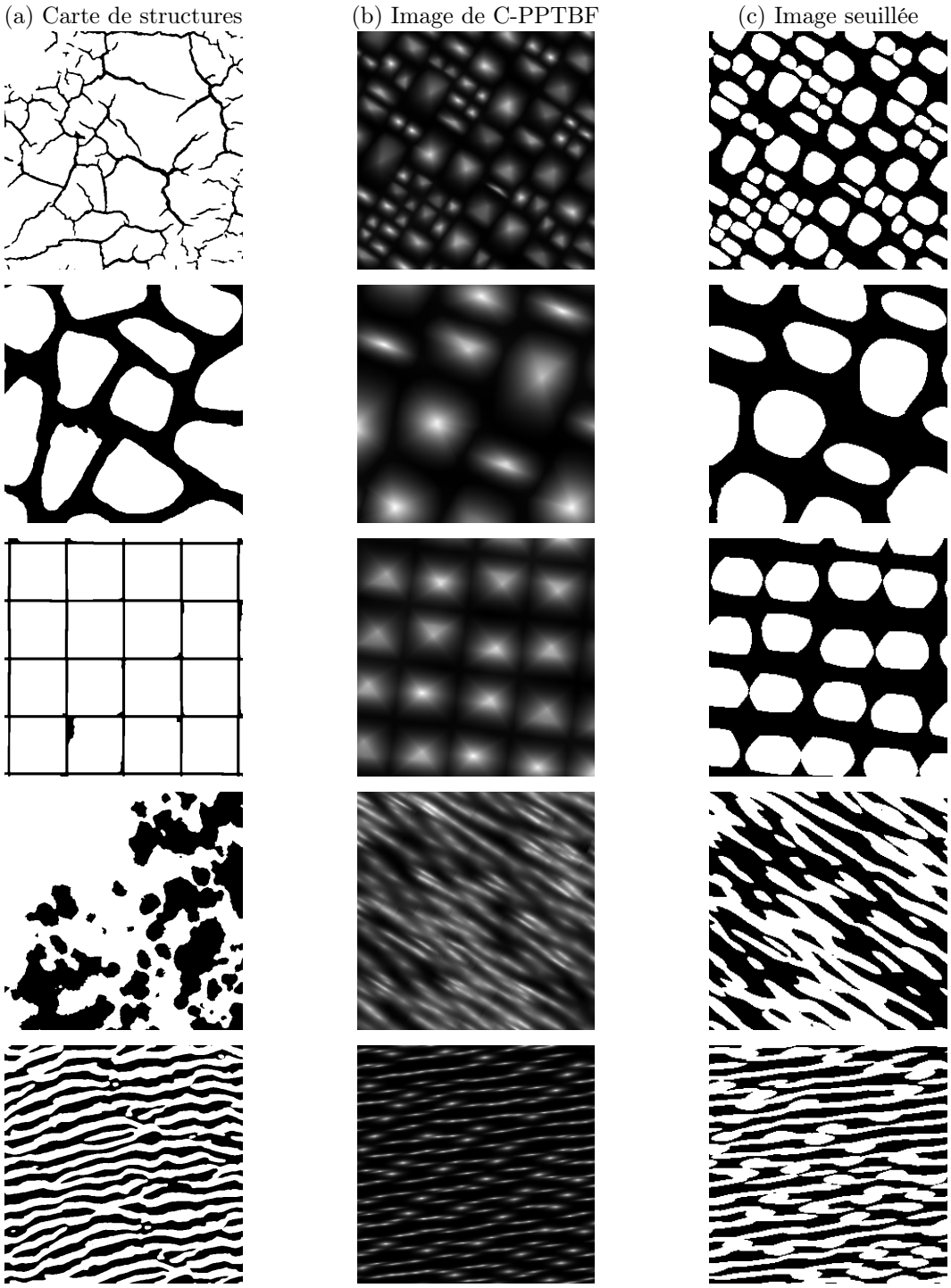

Figure 5.2 – Résultats de la Phase 1 d'estimation des paramètres. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés ; (c) : Version seuillée de (b).

Différences de résultats entre Pix2Pix et Pix2PixHD

Afin de voir la différence d'efficacité sur les estimations de la première phase en utilisant Pix2Pix ou Pix2PixHD pour la reconstruction de l'image de C-PPTBF plausible, nous générons un jeu de 10 000 images de C-PPTBF dont les paramètres sont échantillonnés de manière uniforme ainsi que leurs cartes de structures binaires. Nous utilisons des images synthétiques car nous connaissons les vrais paramètres, ce qui nous permet ainsi de juger de l'efficacité de l'estimation, ce qui n'est pas le cas avec des images naturelles. Nous reconstruisons 10 000 images de C-PPTBF plausibles à partir des cartes de structures binaires générées précédemment, avec Pix2Pix et avec Pix2PixHD, puis nous estimons le paramètre du type de pavage de l'intégralité des images des deux jeux et nous le comparons avec le vrai type de pavage utilisé pour générer l'image. Nous regardons si le type de pavage estimé est le bon, et si l'un des trois types de pavage avec les probabilités les plus élevées est le bon (*TOP-3*). Sur le jeu d'images de C-PPTBF reconstruites avec Pix2Pix la précision sur le type de pavage est de 29,52% pour le type exact, et de 51,74% pour le TOP-3. Sur le jeu d'images de C-PPTBF reconstruites avec Pix2PixHD en revanche la précision est de 50,03% pour le type exact, et de 75,96% pour le TOP-3.

Nous constatons alors que les résultats de la première phase pour le type de pavage sont bien meilleurs en reconstruisant les images avec Pix2PixHD. Nous avons cependant utilisé des images synthétiques et non des images naturelles, mais ce résultat devrait être le même sur des cartes de structures binaires extraites d'images de textures naturelles, justifiant alors le choix d'utiliser Pix2PixHD plutôt que Pix2Pix.

5.3 Deuxième phase de l'estimation

La valeur du type de pavage étant fixée et les valeurs initiales de tous les paramètres continus estimées, la deuxième phase de l'estimation peut être effectuée. Comme présenté dans la partie 4.7, nous utilisons une descente de gradient stochastique utilisant l'optimiseur Adam et l'algorithme de Basin-Hopping. Les valeurs des taux de décroissance exponentielles des estimations des premiers et seconds moments sont fixées à leurs valeurs par défaut, à 0,9 et 0,999, et le taux d'apprentissage est fixé à 0,05. Pour la Moyenne mobile exponentielle le taux de décroissance est fixé à 0,9 et la taille de fenêtre à 30 itérations de descente de gradient. Finalement le paramètre de température T du test d'acceptation du Basin-Hopping est fixé à 5. Nous utilisons 10 itérations de Basin-Hopping et pour chacune de ces itérations 200 itérations de descente de gradient stochastique sont effectuées. Une estimation satisfaisante des paramètres avec cette phase d'optimisation peut être obtenue en deux à

cinq minutes selon l'exemple en entrée.

Nous présentons les résultats obtenus en utilisant comme fonction objectif la SWD1D proposée dans l'équation 4.10. Les feature vectors utilisés sont extraits d'un réseau VGG19 pré-entraîné en utilisant les premières couches de convolution de chaque bloc du réseau. Les images données au réseau sont construites selon une combinaison utilisant l'image de C-PPTBF plausible reconstruite par le réseau Pix2PixHD $U(I)$ et la carte de structures en entrée I, les trois canaux sont alors définis selon la combinaison $I, U(I), 1 - U(I)$. Ces choix de couches à utiliser et de combinaison des trois canaux sont justifiés dans les études suivantes.

Dans les faits, nous n'effectuons pas la deuxième phase avec un seul type de pavage sur une image donnée. En effet, la première phase estime le type de pavage en donnant des pourcentages sur les 14 types de pavage possibles, nous allons alors sélectionner tous les types de pavage ayant une probabilité supérieure à 10%, car plusieurs types de pavage différents peuvent donner des bons résultats car certains sont visuellement proches, mais aussi pour pallier des potentielles d'erreurs d'estimation de notre CNN. Dans la grande majorité des cas, un seul type de pavage a une probabilité supérieure à 10%, le plus grand nombre que nous avons pu observer est de quatre types de pavage, mais cela est extrêmement rare, habituellement il y a plutôt deux types de pavage qui pourraient être sélectionnés. Nous pourrions imaginer par la suite une phase d'interaction avec l'utilisateur dans laquelle il choisirait un type de pavage en particulier si plusieurs types sont au-dessus des 10%.

À la fin de chaque itération de Basin-Hopping, une image est générée avec les paramètres permettant d'obtenir l'erreur sur la SWD1D la plus basse de cette itération. Nous obtenons donc dix images de C-PPTBF et leurs versions seuillées à la fin de la deuxième phase. Tous les résultats montrés dans ce chapitre sont ceux correspondant à l'erreur globale la plus basse, avec le type de pavage permettant d'obtenir l'erreur la plus basse quand plusieurs ont une probabilité supérieure à 10%.

Les résultats sont présentés dans la figure 5.3. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase (b) et sa version seuillée (c). Les exemples choisis ici sont les mêmes que ceux présentés pour les résultats de la première phase dans la figure 5.2.

Nous constatons tout d'abord une nette amélioration par rapport aux résultats obtenus avec uniquement les paramètres de la première phase, ce qui démontre que cette seconde phase est primordiale et donc que le fait de rendre le modèle différentiable par rapport à tous ses paramètres continus pour permettre une optimisation par le gradient est utile et efficace.

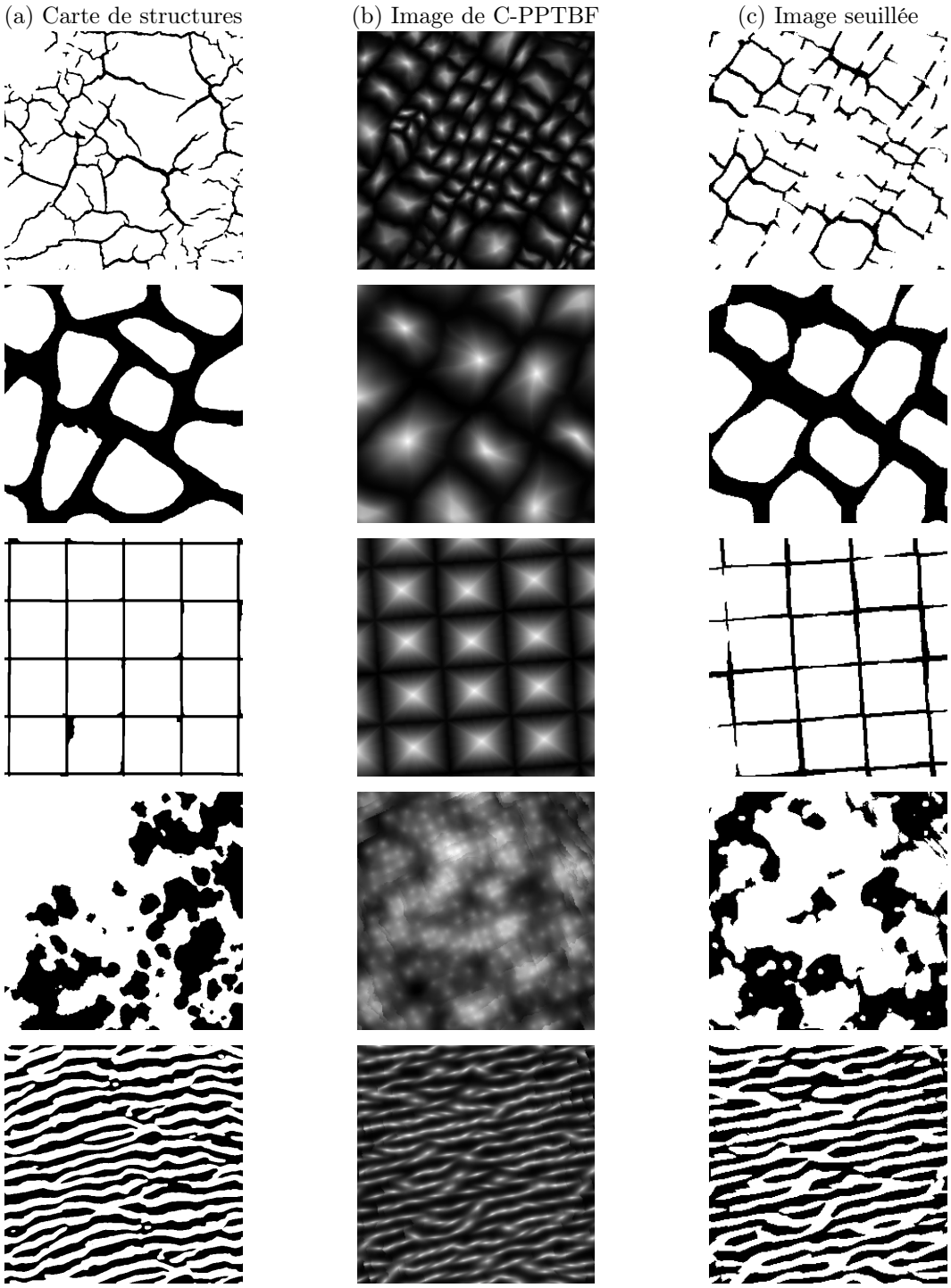

Figure 5.3 – Résultats de la Phase 2 d'estimation. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés ; (c) : Version seuillée de (b).

Nous présentons dans la figure 5.4 un exemple de carte de structures binaires (en haut) pour laquelle quatre types de pavage ont été estimés à plus de 10%, les types 4, 5, 6 et 7. Les quatre lignes présentent les résultats obtenus pour chaque type de pavage utilisé, de gauche à droite de l'image obtenue lors de l'itération de Basin-Hopping ayant abouti à l'erreur la plus basse de toute l'optimisation à l'image obtenue lors de l'itération de Basin-Hopping ayant abouti à la cinquième erreur la plus basse. L'image générée avec le jeu de paramètres permettant l'erreur de SWD1D la plus basse est la 1ère image de la quatrième ligne, donc avec le type de pavage 6, avec une erreur de 1,6403. Cependant nous pouvons observer visuellement qu'elle n'a pas l'air d'être l'image dont les structures ressemblent le plus à la carte de structures en entrée, les premières images de la deuxième (erreur de 1,7232) ou de la cinquième ligne (1,9966) par exemple ou encore la dernière image de la quatrième ligne (2,0147) pourraient être visuellement plus proches.

Nous présentons ensuite des résultats obtenus sur des images de C-PPTBF synthétiques dans la figure 5.5. Nous générons des images de C-PPTBF dont les paramètres ont été fixés de manière aléatoire et leurs versions seuillées, sur lesquelles nous appliquons la phase de reconstruction d'une image de C-PPTBF plausible, puis nous estimons les paramètres avec les deux phases. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase (b) et sa version seuillée (c). Nous pouvons constater visuellement que sur des exemples d'images synthétiques les résultats que nous obtenons sont très proches ce qui montre l'efficacité de notre méthode pour retrouver les paramètres de C-PPTBF.

Des résultats supplémentaires, obtenus en utilisant un réseau Pix2Pix pour la reconstruction de l'image de C-PPTBF plausible, sont disponibles en complément de notre article [BAD23] [1].

5.3.1 Étude sur les couches du réseau VGG19

Afin de déterminer quelle serait la configuration la plus efficace, nous décidons d'effectuer une étude sur différentes configurations possibles de couches de convolution du réseau VGG19 à utiliser. Pour ce faire, nous utilisons à nouveau notre jeu de données de 84 cartes de structures binaires extraites de textures naturelles auxquelles nous appliquons les étapes précédentes de reconstruction d'une image de C-PPTBF plausible à partir d'un réseau Pix2Pix et d'estimation du type de pavage et des valeurs initiales des paramètres continus. Puis nous effectuons cette deuxième phase d'estimation sur l'ensemble du jeu de données afin d'obtenir les paramètres

1. https://ars.els-cdn.com/content/image/1-s2.0-S0097849323000419-mmc1.zip

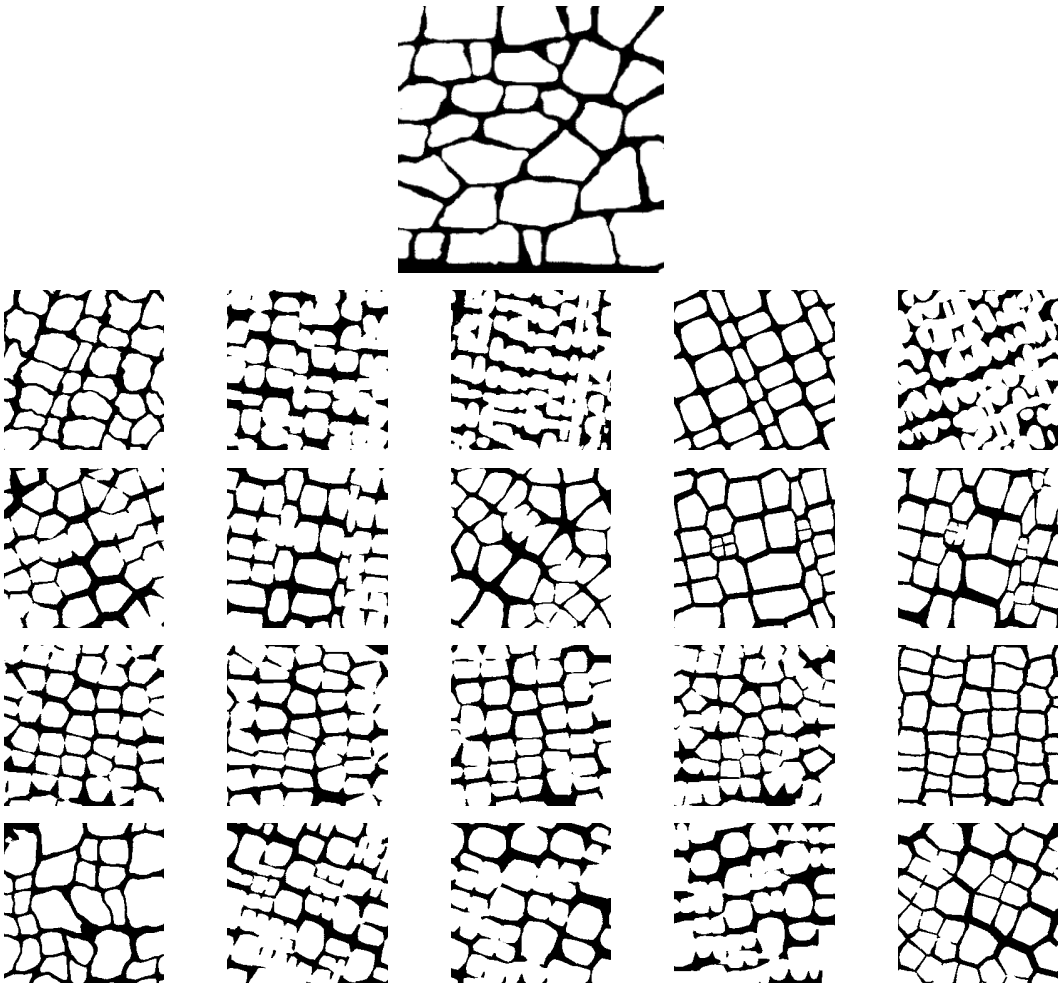

Figure 5.4 – Comparaison de résultats de la Phase 2 d'estimation avec plusieurs types de pavage sur les cinq meilleures itérations de Basin-Hopping, de gauche à droite de la meilleure itération à la cinquième meilleure itération. 1ère ligne : Carte de structures en entrée ; 2ème ligne : type de pavage 4 ; 3ème ligne : type de pavage 5 ; 4ème ligne : type de pavage 6 ; 5ème ligne : type de pavage 7.

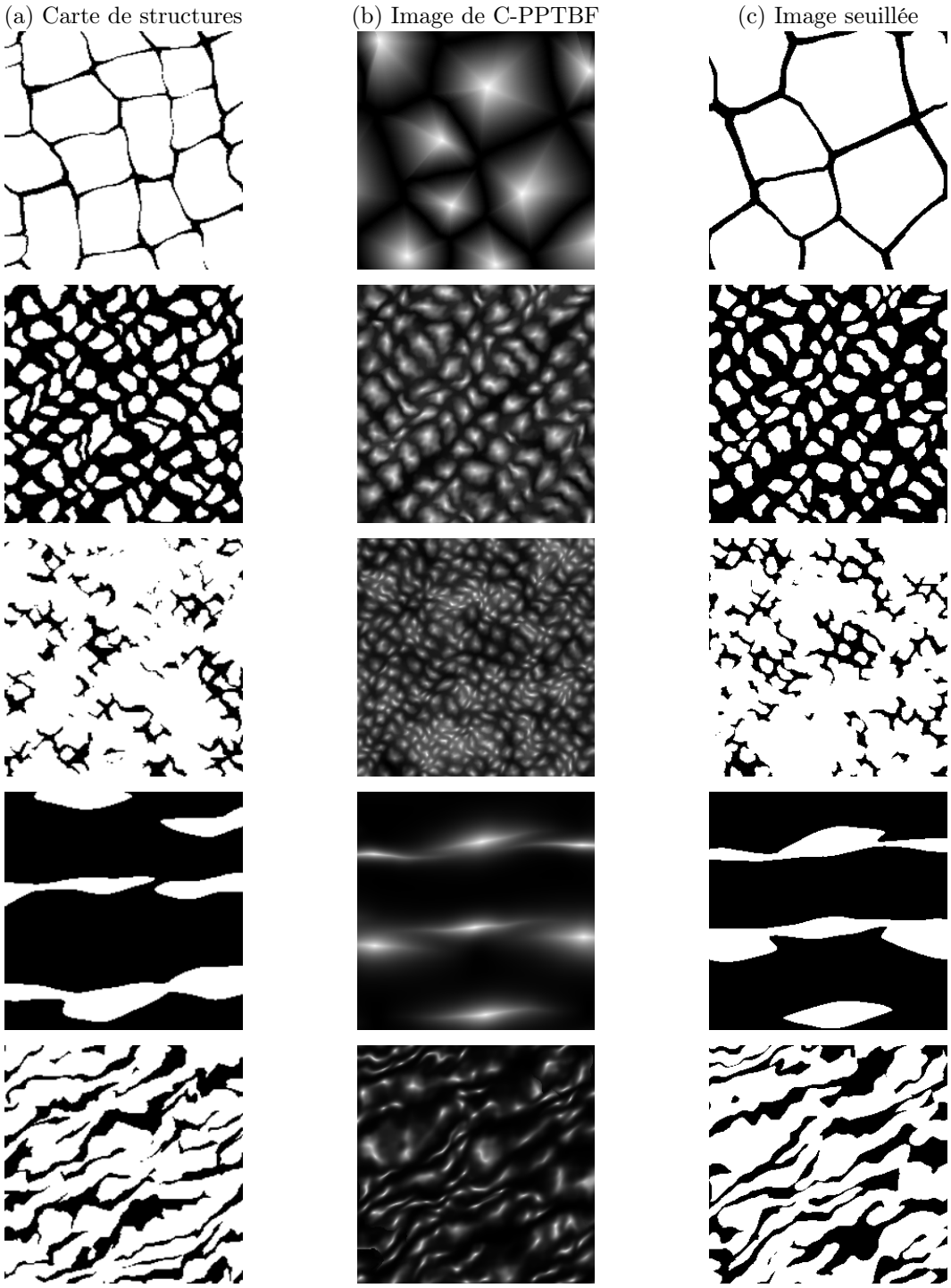

Figure 5.5 – Résultats de la Phase 2 d'estimation sur des images synthétiques. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés ; (c) : Version seuillée de (b).

les plus appropriés pour chacun des exemples en entrée et ainsi générer des nouvelles images de C-PPTBF. Nous seuillons ces images avec une valeur de seuil égale à la proportion de pixels noirs et blancs dans les exemples en entrée, et finalement nous calculons l'erreur reposant la SWD1D entre les exemples en entrée et ces nouvelles images binaires, en utilisant toutes les couches des quatre premiers blocs du réseau. Finalement une moyenne est calculée sur les 84 images du jeu de données, les moyennes de chaque configuration et les temps moyens d'itération de descente de gradient sont présentés dans le tableau 5.1.

Couches de convolution utilisées	**Erreur moyenne**	**Temps moyen d'itération**
Les 16 couches	1,20138	1,07s
Les 8 couches des 3 premiers blocs	1,18508	1,02s
Les dernières couches de chaque bloc	1,18019	0,73s
Les premières couches de chaque bloc	1,14443	0,72s
Les 12 couches des 4 premiers blocs	1,14698	1,05s
Les 2 premières couches du 1er bloc et les 2$^{\text{ème}}$s couches des 3$^{\text{ème}}$ et 4$^{\text{ème}}$ blocs	1,20501	0,71s
Les premières couches des 3 premiers blocs	1,18491	0,70s

Table 5.1 – Erreur reposant sur la SWD1D moyenne et temps d'itération sur le jeu de données d'images de textures naturelles selon les couches de convolutions utilisées pour l'extraction des feature vectors utilisés dans la SWD1D.

Nous constatons dans cette analyse purement reposant sur l'erreur que la meilleure configuration serait de prendre les premières couches de chaque bloc. On constate néanmoins que les différences ne sont pas forcément trop grandes entre les différentes configurations. De plus, les images utilisées sont celles obtenues après avoir effectué la deuxième phase de l'estimation des paramètres sur l'intégralité des images une seule fois, bien que les résultats soient assez similaires si on l'effectue une nouvelle fois il peut y avoir quelques variations. Finalement l'erreur utilisée pour calculer ces résultats repose sur toutes les couches des quatre premiers blocs, en sélectionnant d'autres couches les résultats peuvent varier, même si nous constatons que dans plusieurs configurations de couches utilisées pour calculer ces erreurs moyennes c'est toujours cette configuration fondée sur les premières couches de chaque bloc qui possède l'erreur moyenne la plus basse. Plus on utilise de couches de convolution, plus le temps moyen pour une itération de la descente de gradient est élevé, ce qui est un autre argument en faveur de l'utilisation des premières couches de chaque bloc. Des exemples de résultats selon plusieurs configurations sont présentés dans la figure 5.6.

Pour chaque exemple la carte de structures en entrée est donnée (a), puis la version seuillée de l'image de C-PPTBF générée lors de l'optimisation selon le nombre de couches utilisées, les 16 couches (b), les 8 couches des trois premiers blocs (c), les 12 couches des quatre premiers blocs (d) ou les premières couches de chaque bloc (e).

5.3.2 Étude sur les canaux

Nous proposons une étude sur différentes constructions possibles pour les trois canaux des images fournies au réseau VGG19, conduite de la même façon que l'étude précédente sur les couches convolutionnelles. Pour chaque construction envisagée, nous utilisons la SWD1D reposant sur toutes les couches des trois premiers blocs. Nous présentons dans le tableau 5.2 les résultats obtenus en termes d'erreur moyenne sur l'ensemble des 84 images du jeu de données avec les différentes constructions.

Construction proposée	**Erreur moyenne**
$U(I), U(I), U(I)$	1,62337
I, I, I	1,63351
$U(I), 1-U(I), I$	1,18508
$I, 1-U(I), U(I)$	1,26420
$U(I), U(I), I$	1,23525
$U(I), I, I$	1,24855
$I, U(I), 1-U(I)$	1,10557
$1-U(I), 1-U(I), 1-U(I)$	1,75298
$U(I), 1-U(I), 1-I$	1,57016
$I, U(I), U(I)$	1,19171

Table 5.2 – Erreur reposant sur la SWD1D moyenne sur le jeu de données d'images de textures naturelles selon la construction des trois canaux des images en entrée du réseau VGG19, avec I la carte de structures binaires en entrée et U(I) l'image de C-PPTBF plausible reconstruite.

Nous constatons alors qu'une construction composée de $I, U(I), 1-U(I)$ donne les meilleurs résultats lors de cette deuxième phase d'estimation selon cette étude purement fondée sur l'erreur. Comme pour l'étude précédente il s'agit des résultats obtenus sur des images obtenues après avoir effectué la deuxième phase de l'estimation une seule fois sur l'intégralité des images du jeu de données, et les erreurs sont calculées en prenant toutes les couches des quatre premiers blocs, mais à nouveau les résultats finaux seraient toujours plus ou moins similaires si l'on choisissait

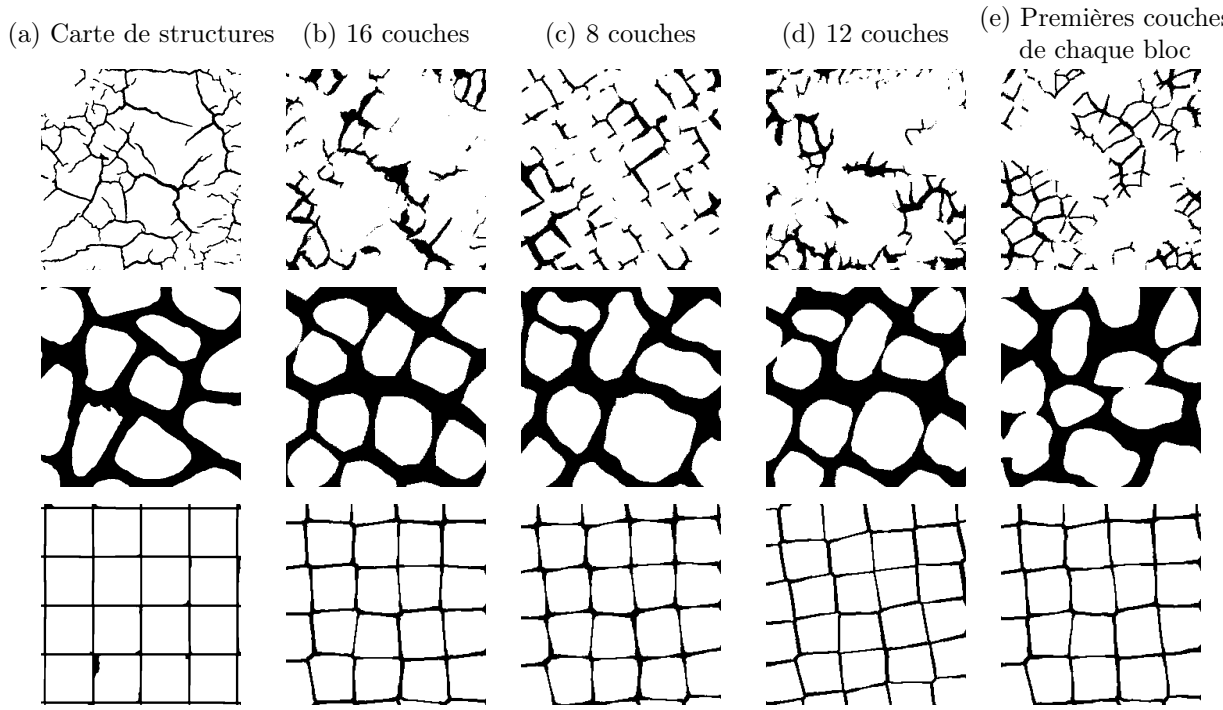

Figure 5.6 – Comparaison de nos résultats de la Phase 2 obtenus selon différentes configurations de couches de VGG19 utilisées. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF seuillée générée en utilisant les 16 couches de VGG19 ; (c) : Image de C-PPTBF seuillée générée en utilisant les 8 couches des trois premiers blocs de VGG19 ; (d) : Image de C-PPTBF seuillée générée en utilisant les 12 couches des quatre premiers blocs de VGG19 ; (e) : Image de C-PPTBF seuillée générée en utilisant les premières couches de chaque bloc de VGG19.

d'autres couches ou si l'on effectuait à nouveau cette étude. Les constructions proposées dans l'étude ont été choisies de manière empirique, il est possible que d'autres constructions proposent des résultats encore meilleurs mais nous n'avons pas vraiment d'intuitions sur les raisons pour lesquelles une construction serait plus performante qu'une autre, même s'il apparaît d'après les résultats qu'utiliser les images d'intensité et les images binaires ensemble est plus efficace que de n'utiliser qu'un seul type d'images. De plus, à notre connaissance, il n'existe pas d'étude portant sur cette problématique dans le domaine de la modélisation procédurale inverse. Des exemples de résultats selon plusieurs constructions des canaux sont présentés dans la figure 5.7. Pour chaque exemple la carte de structures en entrée est donnée (a), puis la version seuillée de l'image de C-PPTBF générée lors de l'optimisation selon les canaux utilisés, $U(I), U(I), U(I)$ (b), I, I, I (c), $U(I), 1-U(I), I$ (d) ou $I, U(I), 1-U(I)$ (e).

Il apparaît alors que les meilleurs résultats sur la deuxième phase de l'estimation en termes d'erreur moyenne s'obtiendraient selon nos études en utilisant les premières couches de chaque bloc pour les feature vectors à utiliser dans la SWD1D, et en fournissant au réseau VGG19 des images construites selon la configuration $I, U(I), 1-U(I)$. En combinant ces deux éléments, nous obtenons une erreur moyenne sur l'ensemble de notre jeu de données de 84 images de 1,0418. Ces deux études ont cependant été réalisées de manière indépendante mais elles pourraient être poussées plus loin en évaluant l'intégralité des combinaisons de couches et de canaux. Cependant vu le nombre important de possibilités, 80 si on ne prend uniquement celles décrites dans ces études, cela prendrait énormément de temps.

5.3.3 Études supplémentaires

Pour aller plus loin nous proposons différentes études supplémentaires sur plusieurs aspects de la deuxième phase.

Augmentation du nombre d'itérations de Basin-Hopping

Bien que les résultats soient déjà satisfaisants, il est possible d'augmenter le nombre d'itérations de Basin-Hopping pour permettre d'explorer davantage l'espace de recherche des paramètres. En augmentant ce nombre à 20 par exemple, nous obtenons une erreur moyenne de 0,9876 ce qui améliore un peu les résultats, mais peut multiplier alors par deux le temps de calcul. Des exemples de résultats sont présentés dans la figure 5.8. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase avec 10 itérations de Basin-Hopping (b) et sa version seuillée

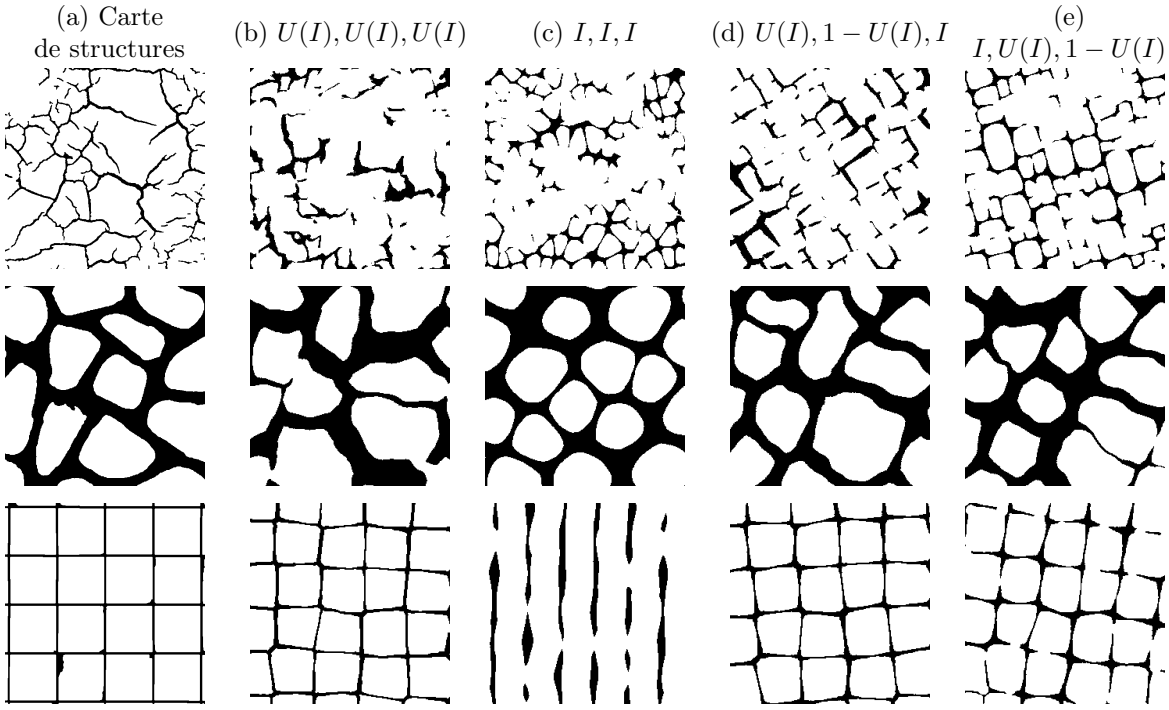

Figure 5.7 – Comparaison de nos résultats de la Phase 2 obtenus selon différentes constructions de canaux des images fournies à VGG19. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF seuillée générée en utilisant la construction $U(I), U(I), U(I)$(c) : Image de C-PPTBF seuillée générée en utilisant la construction I, I, I ; (d) : Image de C-PPTBF seuillée générée en utilisant la construction $U(I), 1-U(I), I$; (e) : Image de C-PPTBF seuillée générée en utilisant la construction $I, U(I), 1-U(I)$.

(c), et ensuite l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase avec 20 itérations de Basin-Hopping (d) et sa version seuillée (e).

Initialisation aléatoire des paramètres continus

Afin de démontrer l'utilité de la première phase de l'estimation, nous décidons d'exécuter la deuxième phase avec des paramètres continus aléatoirement initialisés à la place des valeurs estimées avec notre CNN. Pour le type de pavage néanmoins nous gardons la valeur estimée lors de la première phase car c'est un paramètre discret qui ne peut pas être optimisé lors de la descente de gradient, si nous décidons de l'initialiser à une valeur aléatoire les structures obtenues seraient bien trop éloignées

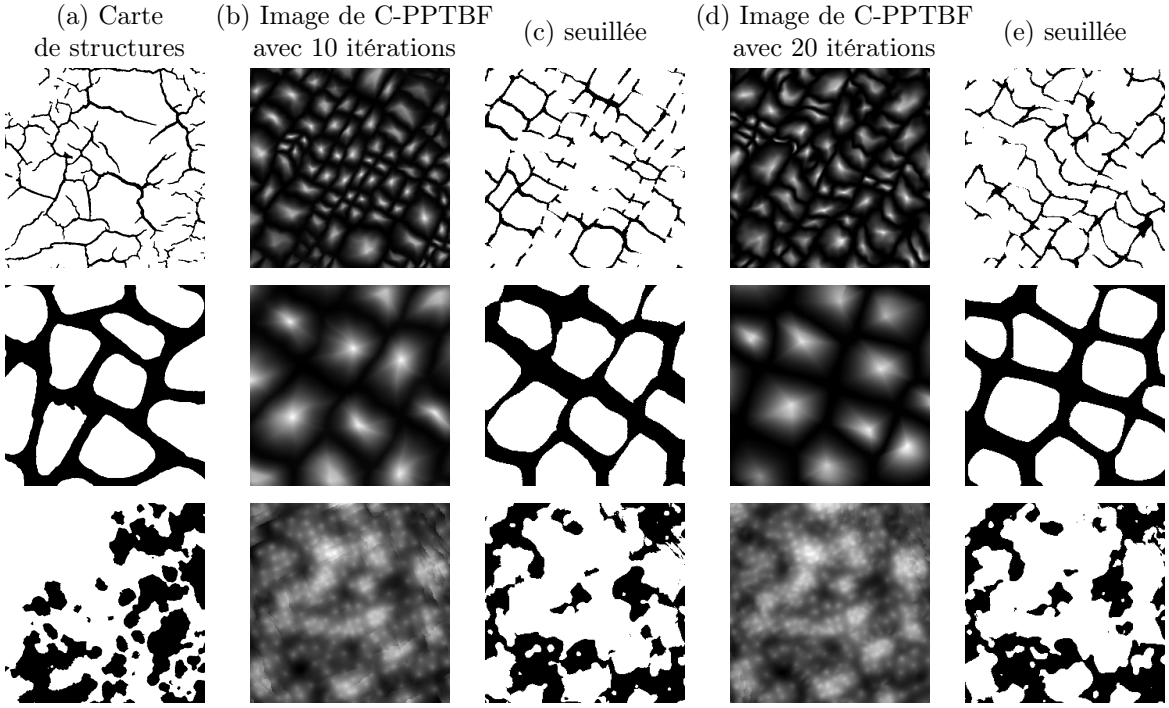

Figure 5.8 – Comparaison de nos résultats de la Phase 2 obtenus en 10 itérations et 20 itérations de Basin-Hopping. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés en 10 itérations ; (c) : Version seuillée de (b) ; (d) : Image de C-PPTBF générée avec les paramètres estimés en 20 itérations ; (e) : Version seuillée de (d).

de celles de l'image en entrée. Nous obtenons une erreur moyenne de 1,1351 ce qui est supérieur à l'erreur obtenue en utilisant normalement les deux phases, prouvant alors que les valeurs estimées par la première phase sur les paramètres continus sont pertinentes. La première phase reste de toute façon indispensable dans notre chaîne de traitement pour obtenir la valeur du type de pavage tout en gardant un modèle différentiable en fonction de tous ses paramètres. Des exemples de résultats sont présentés dans la figure 5.9. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase à partir des estimations de la première phase (b) et sa version seuillée (c), et ensuite l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase à partir de valeurs aléatoires pour les paramètres continus (d) et sa version seuillée (e).

Différences de résultats entre Pix2Pix et Pix2PixHD

Comme nous l'avons montré précédemment, les résultats de reconstruction d'une image de C-PPTBF plausible sont meilleurs en utilisant Pix2PixHD qu'avec Pix2Pix, et cela affecte alors aussi les résultats de l'estimation des paramètres. Nous le constatons dans la figure 5.10 dans laquelle des exemples de résultats de comparaison selon le GAN utilisé sont présentés. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase à partir de la reconstruction avec Pix2Pix (b) et sa version seuillée (c), et ensuite l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase à partir de la reconstruction avec Pix2PixHD (d) et sa version seuillée (e).

Comparaison entre la SWD1D et une erreur fondée sur les matrices de Gram

Afin de montrer l'efficacité de la SWD1D et de justifier le choix de cette fonction d'erreur, nous proposons une comparaison avec une erreur fondée sur les matrices de Gram, qui reste une fonction d'erreur très utilisée dans d'autres méthodes utilisant des optimisations par descentes de gradient. Nous remplaçons alors la SWD1D de la deuxième phase d'estimation par une erreur fondée sur les matrices de Gram. Les images étant générées par une optimisation suivant une erreur autre que la SWD1D, il ne convient pas de faire une comparaison numérique reposant sur l'erreur entre les résultats obtenus avec les deux fonctions d'erreur. Les résultats de cette comparaison sont présentés dans la figure 5.11. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec les paramètres estimés

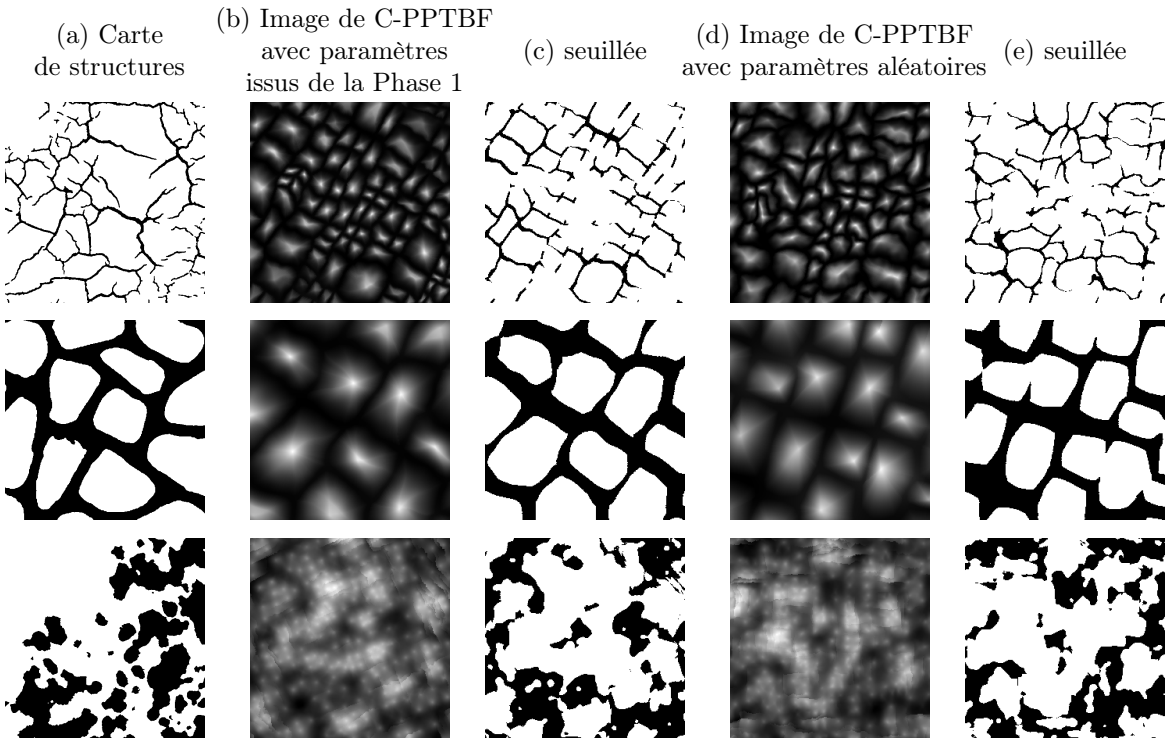

Figure 5.9 – Comparaison de nos résultats de la Phase 2 obtenus initialisant aléatoirement les paramètres ou avec les estimations de la première phase. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres initialisés par les résultats de la première phase ; (c) : Version seuillée de (b) ; (d) : Image de C-PPTBF générée avec les paramètres initialisés aléatoirement ; (e) : Version seuillée de (d).

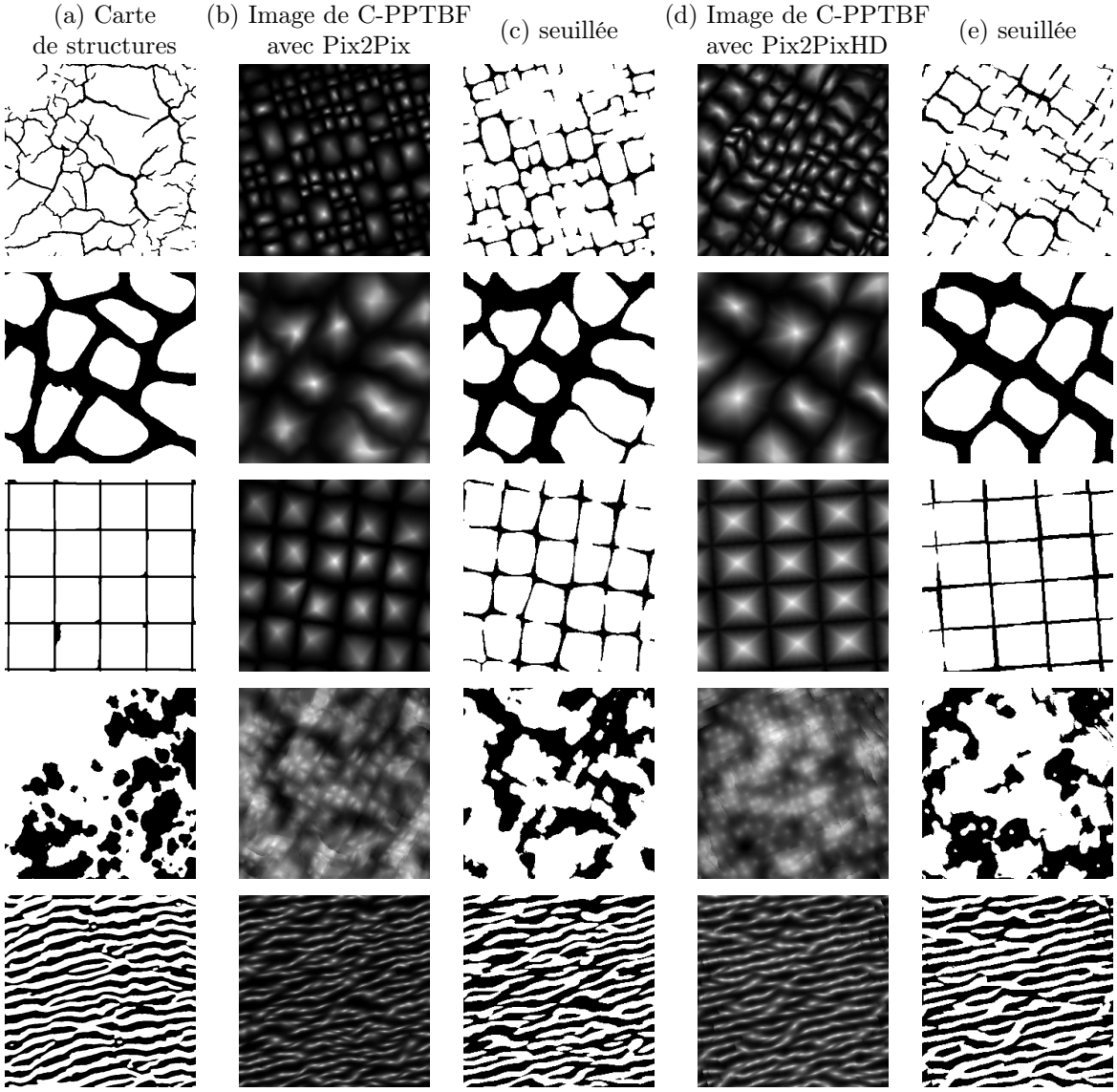

Figure 5.10 – Comparaison de nos résultats de la Phase 2 obtenus à partir d'images reconstruites par Pix2Pix et par Pix2PixHD. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés à partir de Pix2Pix ; (c) : Version seuillée de (b) ; (d) : Image de C-PPTBF générée avec les paramètres estimés à partir de Pix2PixHD ; (e) : Version seuillée de (d).

lors de cette seconde phase avec la SWD1D (b) et sa version seuillée (c), et ensuite l'image de C-PPTBF générée avec les paramètres estimés lors de cette seconde phase avec l'erreur fondée sur les matrices de Gram (d) et sa version seuillée (e).

On peut constater dans le troisième exemple que la rotation serait mieux estimée avec l'erreur fondée sur les matrices de Gram. À l'inverse les structures des deuxièmes et cinquièmes exemples semblent plus proches de celles de la carte en entrée avec la SWD1D au niveau des détails.

5.4 Comparaisons

5.4.1 Comparaison avec DiffProxy

Afin de s'assurer de l'efficacité de notre méthode, une comparaison est effectuée avec le modèle DiffProxy [Hu+22c] représentant l'état de l'art dans le domaine de la synthèse de matériaux et de textures par l'exemple. Nous entraînons le réseau utilisant une architecture StyleGAN2 sur un jeu de données composé de 300 000 images de C-PPTBF générées avec des paramètres échantillonnés de manière uniforme, selon les intervalles présentés dans le tableau 3.1, puis nous effectuons une optimisation par descente de gradient stochastique pour estimer les paramètres du modèle de C-PPTBF. La descente de gradient est initialisée avec les paramètres de C-PPTBF estimés par notre première phase de l'estimation, seuls les paramètres continus sont optimisés durant la descente de gradient, le type de pavage étant fixé. Nous remplaçons leur fonction d'erreur initiale fondée sur des matrices de Gram par une erreur fondée sur la SWD1D afin d'effectuer une comparaison équitable avec notre méthode.

Nous présentons des exemples de cette comparaison dans la figure 5.12. Nous pouvons constater sur ces exemples que nous obtenons des résultats (c) visuellement plus proches des cartes de structures binaires en entrée (a) avec notre méthode plutôt qu'avec DiffProxy (e). Dans le deuxième exemple, les contours des formes des structures obtenues avec DiffProxy sont trop arrondis par rapport à celles de la carte en entrée, nos structures étant plus proches des structures de cette dernière. Dans le troisième exemple, même si notre structure cellulaire n'est pas droite comme la carte en entrée, elle forme quand même un quadrillage dont les cellules sont alignées, alors que le résultat obtenu avec DiffProxy semble déformé. Dans le dernier exemple le résultat obtenu avec DiffProxy est complètement déformé et les bandes souhaitées ne sont pas du tout reconnaissables.

Nous proposons aussi une analyse numérique de cette comparaison entre les deux

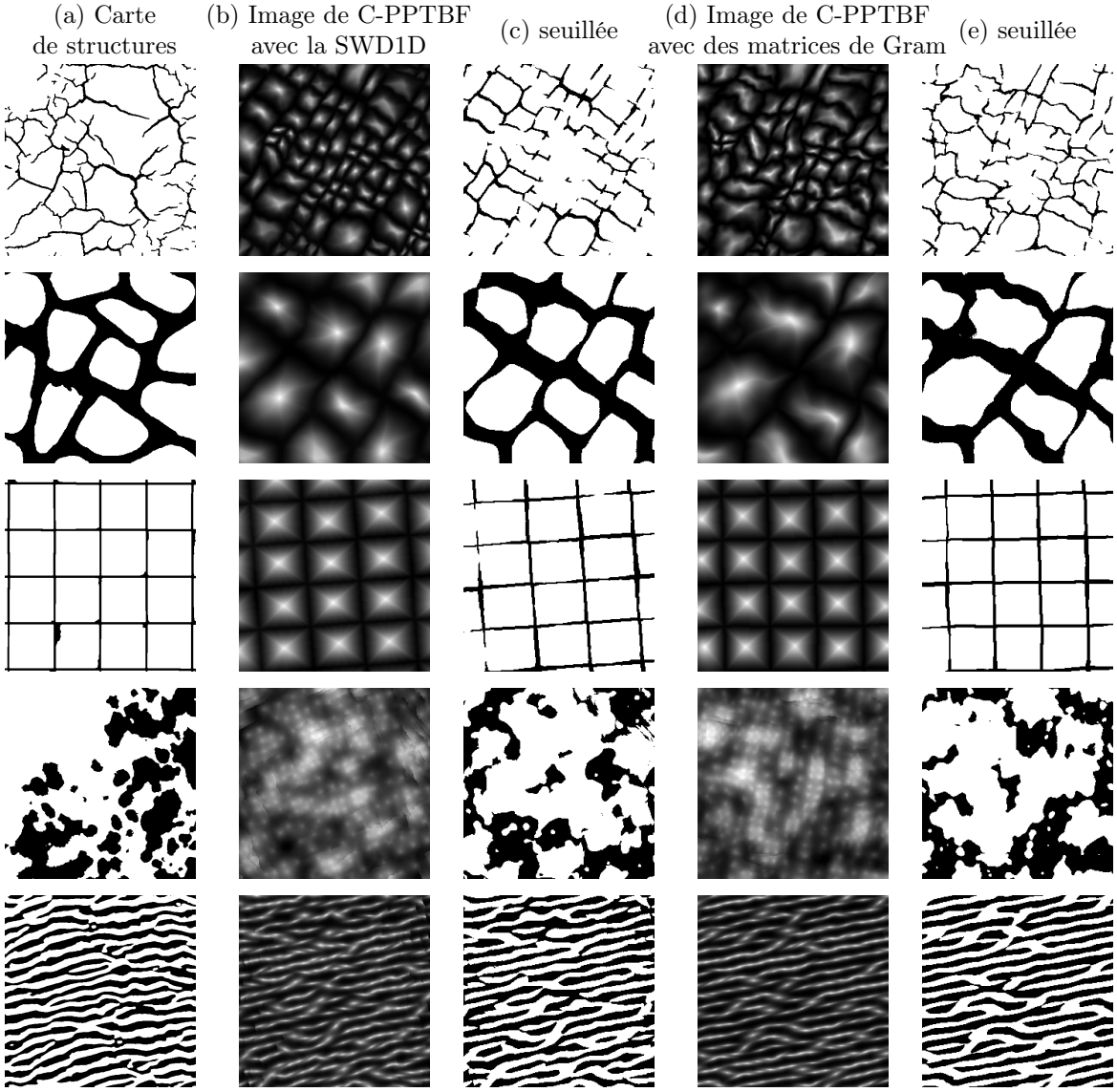

Figure 5.11 – Comparaison de nos résultats de la Phase 2 obtenus en utilisant la SWD1D et en utilisant des matrices de Gram. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés en utilisant la SWD1D ; (c) : Version seuillée de (b) ; (d) : Image de C-PPTBF générée avec les paramètres estimés en utilisant des matrices de Gram ; (e) : Version seuillée de (d).

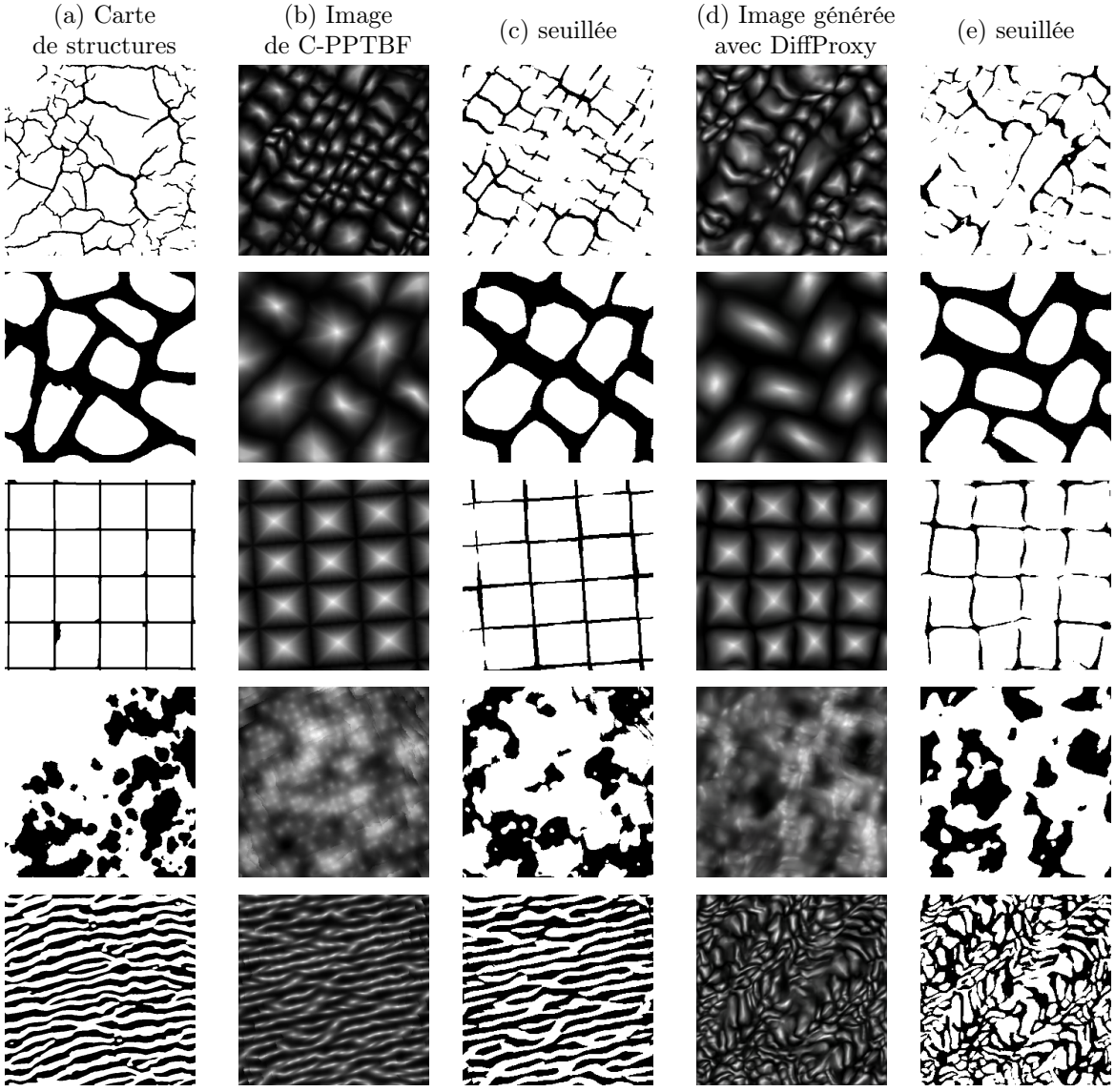

Figure 5.12 – Comparaison de nos résultats de la Phase 2 avec ceux obtenus avec DiffProxy. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés ; (c) : Version seuillée de (b) ; (d) : Image générée avec DiffProxy ; (e) : Version seuillée de (d).

méthodes, en calculant l'erreur reposant sur la SWD1D moyenne sur notre jeu de données de 84 images de cartes de structures binaires issues de textures naturelles. L'erreur moyenne calculée sur les images obtenues avec notre méthode est de 1,0418 contre 1,6169 sur les images obtenues avec DiffProxy. Sur les 84 images, nous obtenons une erreur inférieure avec notre méthode pour 52 images et donc une erreur inférieure avec DiffProxy pour 32 images. De plus, la différence moyenne de l'erreur entre les deux méthodes quand notre méthode est plus efficace est de 1,1577, alors que cette différence est de seulement 0,3722 quand c'est la méthode avec DiffProxy qui est plus efficace. Cela montre alors que même si pour une image donnée leur méthode offre de meilleurs résultats, notre méthode va donner des résultats très proches en terme d'erreur. À l'inverse, pour une image pour laquelle notre méthode va donner de meilleurs résultats les résultats obtenus avec DiffProxy peuvent être très éloignés en terme d'erreur.

Cela démontrerait alors que notre méthode est davantage robuste car elle est capable de quand même générer des résultats cohérents même si la carte de structures en entrée est trop éloignée du jeu de données d'entraînement. À l'inverse, DiffProxy aura des difficultés à généraliser sur des exemples trop éloignés, aboutissant alors à des erreurs très élevées.

Pour cette comparaison nous avons utilisé 300 000 images pour entraîner le réseau StyleGAN2, comme ce qui est indiqué par Hu et al. [Hu+22c] pour l'entraînement qu'ils réalisent pour le modèle de PPTBF, mais nous avons utilisé nos résultats de reconstruction et de la première phase pour initialiser les paramètres de la descente de gradient, ce qui fait qu'un total de 590 000 images a été nécessaire pour tous les entraînements. À titre de comparaison, nous n'avons besoin que de 290 000 images pour tous nos réseaux, ce qui ne représente alors qu'environ 49% des besoins en images de DiffProxy.

Intégration de DiffProxy dans notre chaîne de traitement

Pour aller plus loin sur l'utilisation de DiffProxy, nous proposons de l'intégrer à notre chaîne de traitement pour remplacer la première phase de l'estimation. Nous essayons deux configurations, dans la première tous les paramètres sont initialisés aléatoirement dans les intervalles définis dans le tableau 3.1 et ils sont optimisés durant la descente de gradient, le type de pavage devenant alors un paramètre continu et il est arrondi à l'entier le plus proche à la fin pour générer l'image. Dans la deuxième configuration, seuls les paramètres continus sont initialisés et optimisés, le type de pavage est récupéré de notre première phase d'estimation et il est fixé pendant la

descente de gradient. Les paramètres obtenus servent donc ensuite d'initialisation pour notre deuxième phase d'estimation que nous effectuons de la même façon que précédemment.

En effectuant une analyse purement numérique sur l'erreur reposant sur la SWD1D moyenne sur notre jeu de données de 84 cartes de structures binaires issues d'images de textures naturelles, nous obtenons une erreur de 1,7418 sur la première configuration et de 1,8613 sur la deuxième. Cela montre que notre première phase de l'estimation permettrait d'obtenir une meilleure initialisation des paramètres qu'avec une descente de gradient utilisant DiffProxy. Ces résultats sont alors moins bons que ceux que nous avons obtenus en initialisant aléatoirement les paramètres continus pour la deuxième phase, et aussi moins bons que les résultats que nous obtenons en utilisant uniquement DiffProxy. Des exemples de résultats sont présentés dans la figure 5.13. Pour chaque exemple la carte de structures en entrée est donnée (a), puis l'image de C-PPTBF générée avec notre chaîne de traitement (b) et sa version seuillée (c), et ensuite l'image de C-PPTBF générée en remplaçant notre première phase par DiffProxy (d) et sa version seuillée (e).

Cela signifierait que l'apprentissage des correspondances entre paramètres et apparences des images de C-PPTBF par le réseau StyleGAN2 de DiffProxy serait imparfait au moins pour les paramètres continus, car il apparaît que les types de pavages estimés sont toujours très similaires à ceux estimés par notre CNN. Il faut néanmoins tenir compte de la façon dont nous avons entraîné le réseau StyleGAN2 et des valeurs des hyperparamètres de la descente de gradient que nous avons fixées.

Précision de DiffProxy sur le type de pavage

Nous souhaitons ensuite tester la précision de DiffProxy sur l'estimation du type de pavage d'images synthétiques de C-PPTBF. L'optimisation par descente de gradient utilisant DiffProxy est effectuée sur un jeu de données de 1 000 images de C-PPTBF, dont les paramètres du modèle sont échantillonnés de manière uniforme, selon les intervalles présentés dans le tableau 3.1. Le problème de l'initialisation du type de pavage au début de la descente de gradient se pose, car si la valeur initiale est trop éloignée de la valeur que l'on cherche à obtenir, la descente de gradient peut ne jamais atteindre cette valeur car l'intervalle des valeurs, entre 0 et 13, est grand et qu'une autre valeur de type de pavage peut donner une résultat plutôt similaire ce qui n'encouragerait pas la descente de gradient à continuer sa recherche vers une autre valeur plus éloignée. Ainsi deux configurations sont testées, dans la première la valeur du type de pavage est initialisée aléatoirement, dans la seconde elle est initialisée à la valeur réelle afin de voir si cette valeur est conservée ou non. Comme la

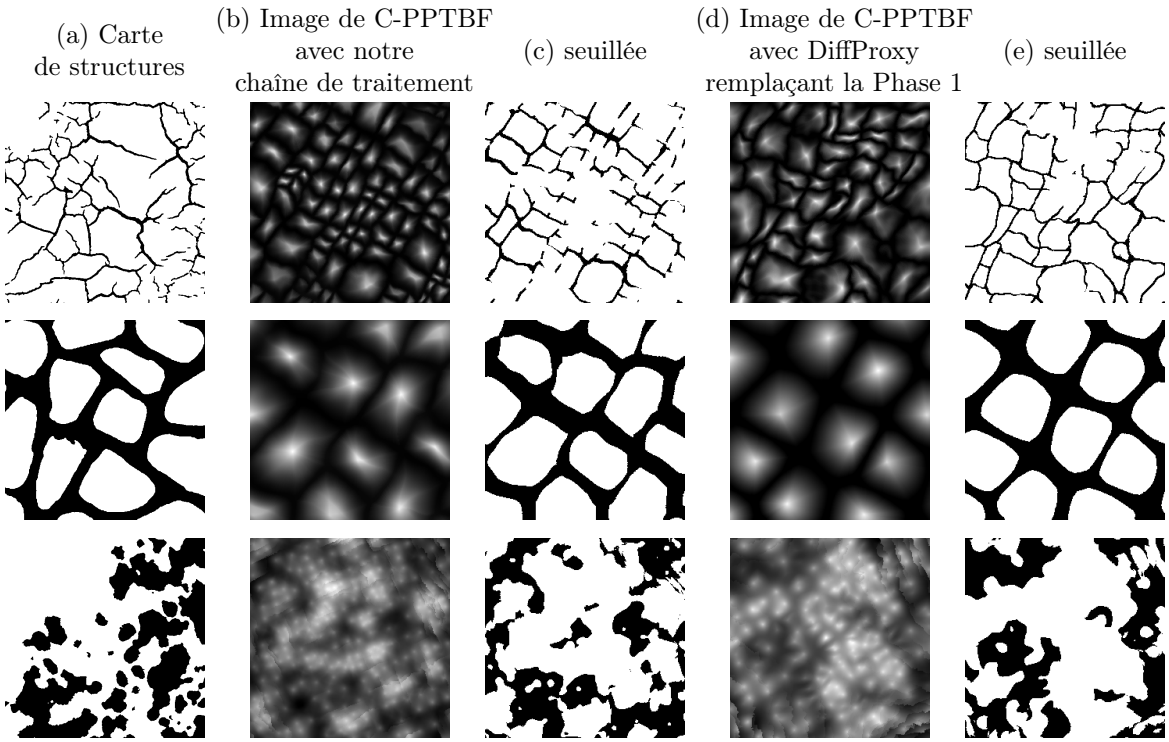

Figure 5.13 – Comparaison de nos résultats de la Phase 2 obtenus avec notre chaîne de traitement, et en remplaçant la Phase 1 par DiffProxy. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF générée avec les paramètres estimés en utilisant notre chaîne de traitement ; (c) : Version seuillée de (b) ; (d) : Image de C-PPTBF générée avec les paramètres estimés en remplaçant la Phase 1 par DiffProxy ; (e) : Version seuillée de (d).

descente de gradient donne des valeurs continues et non discrètes, nous arrondissons la valeur continue obtenue du type de pavage à l'entier le plus proche.

Pour la première configuration nous obtenons une précision de 8,6% et pour la seconde configuration une précision de 25,2%. Ainsi, si le type de pavage est initialisé aléatoirement il est très difficile de trouver le bon type de pavage lors de la descente de gradient car il est probable qu'une valeur permettant d'avoir une apparence proche soit trouvée au bout d'un certain nombre d'itérations, aboutissant alors à un minimum local dans lequel va continuer d'explorer la descente de gradient. Si le type de pavage est initialisé à la vraie valeur utilisée pour générer l'image de C-PPTBF la précision est quand même très basse, cela peut s'expliquer par le fait que l'optimisation est faite sur des valeurs continues et non discrètes. Le fait que des valeurs de types de pavage consécutives peuvent donner des apparences bien différentes peut alors fausser l'estimation. Il faut cependant prendre en compte le fait que les paramètres estimés lors de la descente de gradient ne correspondent pas vraiment aux paramètres de C-PPTBF, mais correspondent aux paramètres tels qu'appris par le réseau StyleGAN2 de DiffProxy lors de son entraînement. Ainsi même si les paramètres obtenus sont éloignés des paramètres utilisés pour générer l'image synthétique, ils peuvent quand même permettre d'obtenir une image à la structure proche de l'image cible car c'est ainsi que le réseau génératif aura appris la correspondance entre image et paramètres.

Ces deux résultats sont ainsi bien inférieurs avec ceux que nous obtenons lors de notre première phase, mais ceci peut être nuancé par le fait que la taille du jeu de données n'est pas la même. Dans notre étude sur la précision de l'estimation selon le type de pavage faite pour notre première phase d'estimation dans la partie 4.3 nous avions utilisé un jeu de données de 14 000 images, alors qu'ici nous utilisons uniquement un jeu de données de 1 000 images par soucis de temps de calcul. La comparaison n'est alors pas égale, même si elle donne une tendance notable.

5.4.2 Comparaison sur l'utilisation d'un noyau de Gabor

Nous décidons d'intégrer les modifications faites au modèle de C-PPTBF présentées dans la partie 3.4 pour observer les différences de résultats que nous obtenons avec l'utilisation d'un noyau de Gabor à la place du noyau gaussien utilisé dans le modèle de C-PPTBF. Cette étude est principalement réalisée afin d'observer l'influence de l'ajout de nouveaux paramètres dans le modèle de C-PPTBF, et ce qui en découle dans les différentes étapes de notre chaîne de traitement. En effet, cette étude ne rentre pas dans le cadre de notre travail sur des structures cellulaires car

l'utilisation d'un noyau de Gabor ne rend plus le modèle axé sur ce type de structures.

Ainsi nous entraînons le réseau Pix2Pix sur un jeu de données de 240 000 paires d'images de C-PPTBF et binaires, en intégrant les différents paramètres liés au noyau de Gabor pour générer les images de C-PPTBF, c'est-à-dire la fréquence des bandes du noyau, leur courbure et leur épaisseur. Nous entraînons ensuite notre CNN de la première phase d'estimation des paramètres en intégrant ces nouveaux paramètres, et nous estimons les paramètres des images de C-PPTBF plausibles reconstruites grâce au réseau Pix2Pix précédemment présenté. Finalement nous appliquons la deuxième phase d'estimation pour affiner les valeurs des paramètres continus, dont les nouveaux paramètres.

L'ajout des nouveaux paramètres va avoir une influence importante sur les résultats que nous obtenons. Ils apportent déjà une plus grande diversité de structures représentables ce qui rend d'abord l'apprentissage du réseau génératif plus complexe et donc la reconstruction de l'image de C-PPTBF plausible peut être moins réussie. De la même façon, pour la première phase d'estimation le CNN est entraîné désormais pour estimer 14 paramètres ce qui affecte forcément l'erreur durant l'entraînement, étant donné que davantage de paramètres y contribuent. De plus, comme il y a davantage de structures représentables, et donc davantage de possibilités de modifier les types de pavage initiaux, le réseau va être moins efficace pour estimer le type de pavage de l'image de C-PPTBF plausible. Avec notre modèle de C-PPTBF initial nous obtenions une précision sur la validation pour le type de pavage de 82,52%, contre une précision de 78,25% avec le modèle modifié utilisant le noyau de Gabor. Ainsi, tous ces éléments vont de facto influer sur la deuxième phase d'estimation, comme les types de pavage peuvent être mal fixés et qu'on ajoute davantage de paramètres à optimiser dans la descente de gradient.

Nous obtenons ici après la deuxième phase une erreur moyenne sur les 84 images du jeu de données de 1,3060, contre 1,1056 avec le modèle de C-PPTBF initial utilisant un noyau gaussien. Ainsi, le fait de remplacer le noyau gaussien par un noyau de Gabor dans le modèle de C-PPTBF ne semble pas améliorer les résultats globaux sur notre jeu de données de structures cellulaires.

Nous n'avons pas essayé d'utiliser plutôt une somme de plusieurs noyaux de Gabor car cela reviendrait à ajouter des paramètres discrets supplémentaires, en particulier le paramètre du nombre de noyaux de Gabor. Cela s'approcherait alors beaucoup du modèle de PPTBF initial.

5.4.3 Discussions

Résultats

Avec ces comparaisons nous remarquons que notre méthode est peu coûteuse en mémoire car elle nécessite uniquement de stocker les réseaux entraînés et non des bases de données d'images et de feature vectors qui peuvent être volumineuses. Le réseau Pix2Pix entraîné pèse 230 Mo et le réseau Pix2PixHD 750 Mo. Les deux réseaux ResNet152V2, celui pour estimer tous les paramètres sauf le paramètre de rotation et celui pour estimer uniquement le paramètre de rotation, pèsent en totalité 1,4 Go (700 Mo chacun) en raison du nombre important de couches présentes dans l'architecture de ResNet152V2. Dans nos travaux nous avons fait le choix de ResNet152V2 pour les bons résultats que nous obtenons avec dans un temps d'entraînement raisonnable, nous avons pu constater dans nos différentes études qu'avec d'autres modèles moins lourds des résultats tout aussi satisfaisants pourraient être obtenus, par exemple avec un réseau VGG19 pesant environ 240 Mo ou avec un réseau DenseNet169 pesant environ 150 Mo. Remplacer ResNet152V2 par une autre architecture moins lourde, mais un peu moins performante, altérerait les résultats de la première phase d'estimation des paramètres. Une étude approfondie sur les différences de résultats sur la suite de la chaîne de traitement qu'un tel changement apporterait pourrait être effectuée, cela pourrait montrer que le coût en mémoire de notre méthode pourrait alors être réduit sans affecter énormément les résultats. Cela reste ainsi moins coûteux en mémoire que d'autres méthodes utilisant des bases de données comme celle de Hu et al. [Hu+22a] qui pèse 8 Go.

Notre méthode est aussi robuste car elle permet d'obtenir des résultats satisfaisants même si les exemples en entrée sont trop éloignés du jeu de données d'entraînement. Nous l'avons remarqué par exemple dans notre comparaison avec DiffProxy, où nous obtenons de meilleurs résultats avec notre méthode qu'avec la leur sur ce type d'exemples.

Au niveau du temps d'exécution, notre méthode comporte l'entraînement des deux CNN, prenant environ quatre heures, et du GAN que nous entraînons pendant un mois. Le temps d'entraînement du GAN est alors très long, mais cette durée permet de maximiser la qualité des résultats. Un entraînement moins long pourrait permettre d'obtenir néanmoins de bons résultats de reconstruction, mais nous sommes confrontés au problème habituel de l'entraînement d'un GAN qui peut être aléatoire et dont l'erreur peut ne pas converger lors d'un entraînement mais peut converger lors d'un autre entraînement, et dans un nombre d'époques plus ou moins élevé. Une fois les réseaux entraînés et stockés, la reconstruction et la première phase de l'es-

timation des paramètres se font instantanément. La deuxième phase de l'estimation peut prendre entre deux à cinq minutes pour obtenir des résultats satisfaisants, nous avons cependant vu qu'on peut augmenter le nombre d'itérations de Basin-Hopping pour améliorer encore les résultats, mais cela augmente alors le temps d'exécution de l'optimisation. Hors entraînement des réseaux, notre méthode serait alors plus rapide que celles par exemple de Guehl et al. [Gue+20] ou de Hu et al. [Hu+22a], ou égale en termes de temps à DiffProxy.

Limites

Nos travaux comportent cependant plusieurs limites. Tout d'abord notre modèle de C-PPTBF se limite aux structures cellulaires, ainsi si nous avons en entrée une carte binaire comportant des structures complètement différentes nous ne pouvons pas avoir de résultats satisfaisants, même si la robustesse de notre approche permet d'obtenir des structures qui peuvent s'en approcher dans la limite de l'expressivité du modèle de C-PPTBF. Nous observons ensuite que nous pouvons avoir quelques erreurs sur des structures régulières, par exemple nous pouvons voir que l'estimation de la rotation et de la translation du troisième exemple de la figure 5.3 est incorrecte, car notre méthode se concentre sur des structures stochastiques. De plus, nous avons présenté dans ce manuscrit pour la deuxième phase de l'estimation les résultats obtenus en conservant les jeux de paramètres donnant l'erreur globale la plus basse pour une image donnée. Nous avons pu observer que les résultats obtenus sont proches pour des images de structures synthétiques en entrée, comme montré dans la figure 5.5. Pour des images de structures réelles en entrée, l'image produite n'est pas forcément la plus proche visuellement de l'exemple en entrée, d'autres images produites durant la deuxième phase ou avec un type de pavage différent pourraient être visuellement plus proches, comme observé dans la figure 5.4. Afin d'évaluer les performances de la SWD1D de façon approfondie, et afin de la comparer plus efficacement à d'autres fonctions d'erreur, une étude sur des jeux de données synthétiques serait nécessaire, qui fera l'objet de travaux futurs.

Comme toutes les différentes méthodes utilisant des bases de données créées à partir d'images issues de modèles procéduraux, notre approche reste dépendante de l'échantillonnage des paramètres utilisé pour construire la base de données, même si elle l'est moins que les autres méthodes en raison de sa robustesse. Nous pouvons noter que les jeux de données que nous utilisons pour l'entraînement de nos réseaux de neurones sont plus petits que ceux utilisés par les autres méthodes. Nous utilisons un jeu de 240 000 images pour l'entraînement de notre GAN et 50 000 pour l'entraî-

nement de nos CNN, pour un total de 290 000 images utilisées. Pour DiffProxy par exemple un jeu de 300 000 images est utilisé pour entraîner le réseau StyleGAN2, puis nous utilisons nos réseaux de reconstruction et de première phase pour initialiser les paramètres de la descente de gradient, ce qui donne un total de 590 000 images utilisées. Pour construire la base de données de Guehl et al. [Gue+20] 450 000 images sont utilisées, associées à des feature vectors alourdissant encore davantage le jeu de données. Augmenter la taille des jeux de données pourrait permettre d'échantillonner davantage les valeurs des paramètres et donc de réduire davantage la dépendance, mais au prix de l'augmentation du temps nécessaire pour l'entraînement des réseaux et de mémoire vive nécessaire pour gérer un jeu de données plus volumineux.

5.5 Applications

Nous pouvons effectuer une synthèse de textures semi-procédurale suivant l'algorithme PCTS à partir des images binaires des images de C-PPTBF que nous obtenons à la fin de la chaîne de traitement, qui sont utilisées pour guider une synthèse par l'algorithme PCTS, comme ce qui est effectué par Guehl et al. [Gue+20]. Nous présentons plusieurs résultats dans la figure 5.14, pour chaque exemple la ligne du haut montre la carte de structures binaires en entrée dans l'encadré et autour la carte de structures que nous obtenons à la fin de notre méthode, et la ligne du bas montre l'image de texture naturelle dont est issue la carte de structures binaires dans l'encadré et autour l'image obtenue par la synthèse.

Les structures générées par notre modèle de C-PPTBF, comme celles générées par le modèle de PPTBF, ne sont pas directement *filtrables*, ce qui est susceptible de produire des artefacts d'échantillonnage lors du rendu d'une surface plongée en 3D sur lequel serait plaquée une structure procédurale, ou bien une texture semi-procédurale. Filtrer nécessiterait de pouvoir calculer numériquement, et de façon efficace, une intégrale de la fonction procédurale sur des régions du plan, ce qui est laissé pour des travaux futurs.

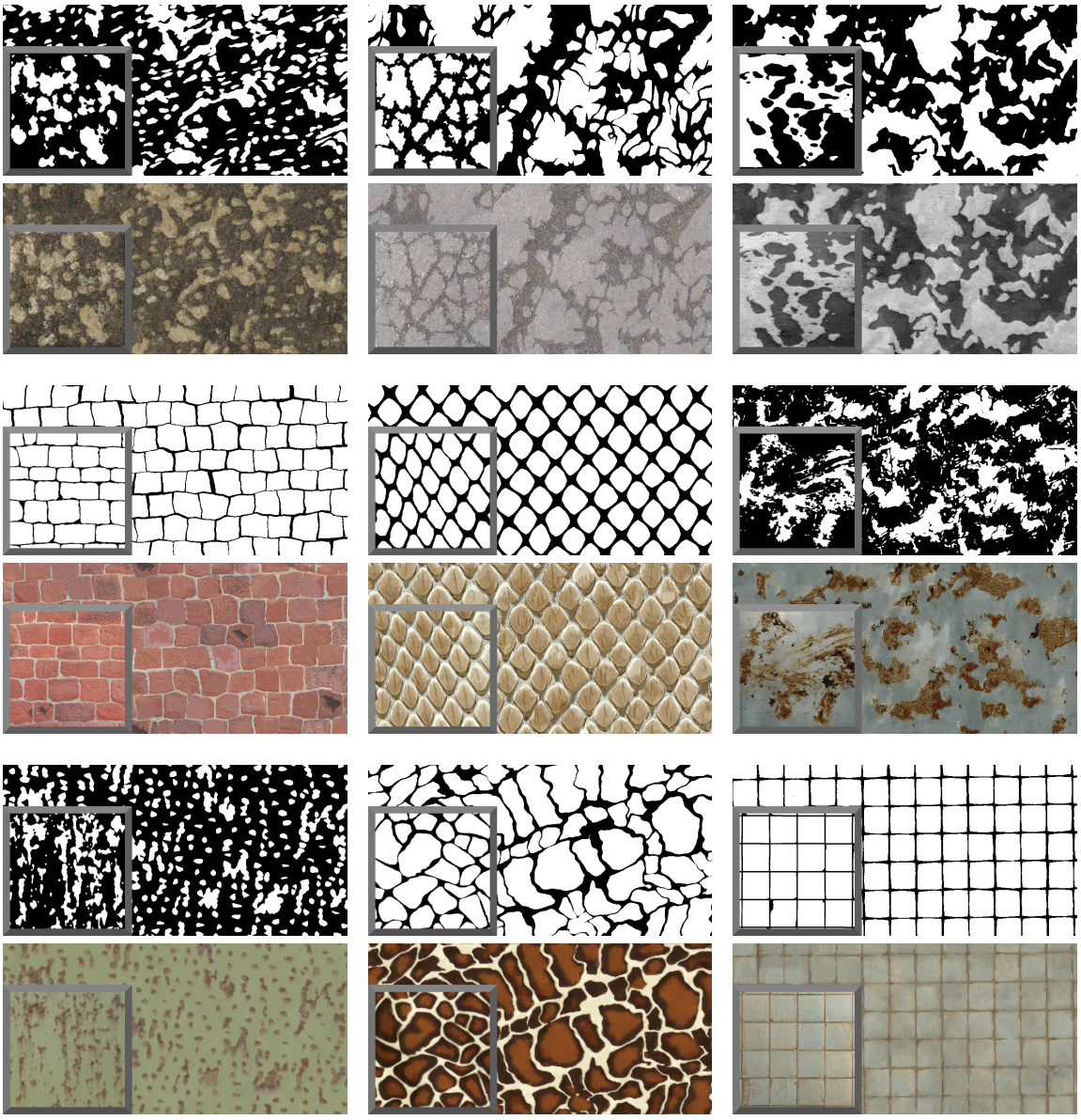

Figure 5.14 – Application d'une synthèse de textures semi-procédurale sur nos résultats. Pour chaque exemple, la ligne du haut montre la carte de structures en entrée (encadré) et l'image binaire résultante de notre méthode. La ligne du bas montre l'image de texture naturelle initiale (encadré) et le résultat de la synthèse de texture avec la méthode de Guehl et al. [Gue+20].

Chapitre 6

Conclusion

6.1 Bilan

Dans le cadre de cette thèse nous avons proposé des contributions portant sur la modélisation procédurale de structures stochastiques cellulaires et leur génération par l'exemple.

Tout d'abord nous avons mis en place un modèle procédural de génération de structures cellulaires stochastiques 2D selon la formulation du modèle de PPTBF des travaux de Guehl et al. [Gue+20] que nous nommons Cellular Point Process Texture Basis Function (C-PPTBF). Ce modèle génère dans une première étape de réalisation d'un processus ponctuel un ensemble de feature points suivant un type de pavage prédéfini, 14 types de pavage étant représentables pour permettre de construire un éventail de structures cellulaires différentes. Cet ensemble est ensuite convolué au produit d'une fonction Window construisant des cellules autour des points à partir d'une interpolation linéaire entre deux fenêtres définissant alors si le contenu des cellules est plus ou moins fusionné, et d'une fonction feature définissant les formes locales des cellules en utilisant un noyau gaussien. Notre modèle s'inscrit dans la formulation du modèle de PPTBF de Guehl et al. [Gue+20] en excluant quelques types de pavage, et remplaçant la somme de noyaux de Gabor anisotropes par un noyau gaussien, ce qui réduit ainsi le nombre de paramètres utilisé par rapport au modèle original. Les paramètres conservés ainsi que les fonctions utilisées assurent la possibilité de générer une grande diversité de structures cellulaires stochastiques, ce qui a justifié le choix d'alléger le modèle de PPTBF initial pour faciliter l'estimation des paramètres du modèle. Notre modèle de C-PPTBF n'est composé alors que d'un paramètre discret, le type de pavage, le reste des paramètres étant continus, avec des

fonctions Window et Feature différentiables par rapport à ces paramètres continus.

Notre seconde contribution porte ainsi sur l'estimation des paramètres du modèle de C-PPTBF, nécessaire pour effectuer une modélisation procédurale inverse à partir de cartes de structures binaires issues de textures naturelles. Nous avons conçu une chaîne de traitement composée de deux étapes. Tout d'abord à partir des cartes de structures binaires en entrée, nous reconstruisons des images de C-PPTBF plausibles en utilisant un GAN, cette étape est cruciale pour pouvoir estimer les paramètres sur des images de C-PPTBF, ce qui donne de meilleurs résultats que d'estimer les paramètres sur des images binaires contenant moins d'informations. La deuxième étape est celle de l'estimation des paramètres qui est divisée en deux phases : dans la première nous utilisons un CNN pour estimer le type de pavage et des valeurs initiales pour l'intégralité des paramètres continus ; dans la seconde nous effectuons une descente de gradient stochastique pour optimiser les valeurs des paramètres continus. Cette descente de gradient est permise par la différentiabilité du modèle en fonction de tous ses paramètres continus étant donné que le seul paramètre discret a sa valeur fixée de par son estimation lors de la première phase. Nous avons aussi proposé d'utiliser une fonction d'erreur appropriée pour cette optimisation, la distance de Wasserstein tranchée 1D inspirée de Heitz et al. [Hei+21], qui serait plus robuste que l'habituelle fonction d'erreur fondée sur des matrices de Gram. Cette fonction d'erreur est alors minimisée lors de la descente de gradient, sur laquelle nous appliquons l'algorithme du Basin-Hopping pour permettre d'éviter à la descente de gradient de rester bloquée dans des minima locaux. Ces différentes phases pourraient être complétées par la suite par une phase d'interaction avec l'utilisateur afin d'affiner encore davantage les valeurs des paramètres obtenus, ou tout simplement pour permettre à l'utilisateur de choisir manuellement certains paramètres, comme le type de pavage, et d'explorer davantage l'espace de recherche. Nous obtenons ainsi à la fin une image de C-PPTBF dont l'image seuillée binaire doit s'approcher de l'exemple en entrée. Cette carte de structures peut alors être utilisée par la suite dans une synthèse de textures semi-procédurale pour rajouter des détails de couleurs et obtenir ainsi une image de texture.

Nous avons réalisé plusieurs études sur les différentes étapes de l'estimation des paramètres pour justifier nos choix et pour permettre d'améliorer les résultats que l'on peut obtenir. Ce travail est néanmoins toujours perfectible, nous avons vu par exemple que l'étude sur les canaux des images à utiliser pour l'extraction des feature vectors à partir d'un réseau VGG19 pré-entraîné pourrait être complétée par la suite tant il existe de possibilités et que nous n'avons pas vraiment de justification sur les

potentielles raisons pour lesquelles certaines constructions pourrait être meilleures que d'autres.

Finalement nous avons proposé une comparaison avec la méthode de l'état de l'art DiffProxy développée par Hu et al. [Hu+22c] se reposant sur l'entraînement d'un GAN et d'une optimisation par descente de gradient, et la méthode de modélisation procédurale inverse proposée par Guehl et al. [Gue+20] utilisant une recherche du plus proche voisin à partir d'une base de données d'images de PPTBF et de descripteurs. Notre méthode présente l'avantage d'être automatique, elle ne nécessite pas de phases d'interaction avec l'utilisateur obligatoire pour obtenir des résultats satisfaisants au contraire de Guehl et al. [Gue+20] ou de Zhou et al. [Zho+23]. Nous avons de plus remarqué que notre méthode était moins coûteuse en mémoire que d'autres méthodes nécessitant de stocker des bases de données d'images car nous n'avons que les réseaux entraînés à stocker, et moins coûteuse en temps car elle ne nécessite qu'entre deux à cinq minutes hors entraînement des réseaux. Finalement notre méthode est robuste car nous observons que nous pouvons obtenir des résultats cohérents même si l'image en entrée est trop éloignée du jeu de données d'entraînement, ce qui n'est pas le cas pour DiffProxy par exemple comme nous avons pu le voir dans notre comparaison.

Notre méthode a cependant plusieurs limites. Tout d'abord sur la gamme de structures représentables étant donné que notre modèle de C-PPTBF est limité aux structures cellulaires, et le fait que nous nous concentrons sur des structures stochastiques peut poser problèmes pour des structures plus régulières. Malgré le fait que notre méthode soit plus robuste que d'autres méthodes, elle reste toujours dépendante à l'échantillonnage des paramètres utilisé pour construire les bases de données d'images d'entraînement. Finalement nous avons constaté que l'image générée lors de la deuxième phase d'estimation avec les paramètres permettant l'erreur la plus basse n'est pas toujours l'image la plus ressemblante visuellement à l'exemple en entrée, ce qui implique de devoir vérifier visuellement les différentes images générées à la fin de chaque itération de Basin-Hopping lors de la deuxième phase.

6.2 Perspectives

Il existe plusieurs pistes potentielles pour des futurs travaux sur notre thématique. Tout d'abord l'ajout de nouveaux paramètres au modèle de C-PPTBF pourrait permettre de pouvoir représenter d'autres types de structures comme des vagues, des branchements ou des motifs hiérarchiques. Cependant il faut pouvoir gérer ces nou-

veaux paramètres dans nos différentes étapes, il faudrait davantage d'images dans les jeux de données d'entraînement pour pouvoir échantillonner efficacement tous les paramètres, et rajouter ces paramètres dans les deux phases d'estimation pourrait rendre l'estimation un peu plus difficile et longue. Nous l'avons par exemple observé dans notre étude sur les résultats obtenus en remplaçant le noyau gaussien du modèle de C-PPTBF par un noyau de Gabor, les résultats d'estimation dans les différentes phases sont un peu moins bons car nous ajoutons des nouveaux paramètres pour pouvoir gérer des structures plus complexes. De plus, si on ajoute des paramètres discrets il faudra pouvoir les estimer efficacement dans la première phase pour pouvoir les fixer comme le type de pavage dans la deuxième phase afin de permettre l'optimisation par le gradient. Ainsi l'objectif serait de pouvoir représenter davantage de types de structures tout en gardant des fonctions Window et Feature différentiables en fonction de tous leurs paramètres.

Pour étendre la gamme de structures représentables, une alternative à l'ajout de paramètres aux fonctions Window et Feature peut être de construire la réalisation d'un processus stochastique P plus complexe, par exemple en remplaçant les feature points par des segments, qui seraient nommés *feature segments*, afin de pouvoir traiter des structures en formes de branchements, un type de structures qui n'était pas déjà traité par le modèle de PPTBF [Gue+20]. Par exemple Gaillard et al. [Gai+19] proposent une fonction procédurale générant des motifs en formes de branchements utilisés pour la génération de terrains, en créant un arbre composé de branches rectilignes, qui sont des segments, sur plusieurs niveaux de profondeur et de longueur en utilisant une fonction de contrôle reposant sur un bruit de Perlin. L'intensité en un point est alors sa distance au segment le plus proche. Les segments sont créés en reliant des points distribués dans l'image selon une fonction déterministe de perturbation à partir de centres de cellules d'une grille.

Dans des premières expérimentations d'utilisation de feature segments à la place de feature points, nous avons pu obtenir des structures en formes de branchements basiques comme montrées dans la figure 6.1. La construction de ces segments peut se faire de plusieurs façons différentes, selon le nombre de points traversés, selon si on cherche le point le plus proche ou un point aléatoire pour former un segment, ou encore les segments peuvent être transformés en courbes de Bézier pour permettre plus de diversité de structures. Cette piste pourrait ainsi faire l'objet de travaux futurs, en prenant en compte le fait qu'utiliser des feature segments augmente le temps de calcul car ce sont des structures de données plus lourdes que des feature points et qui nécessitent alors des algorithmes plus complexes.

(a) Image d'intensité (b) Image binaire

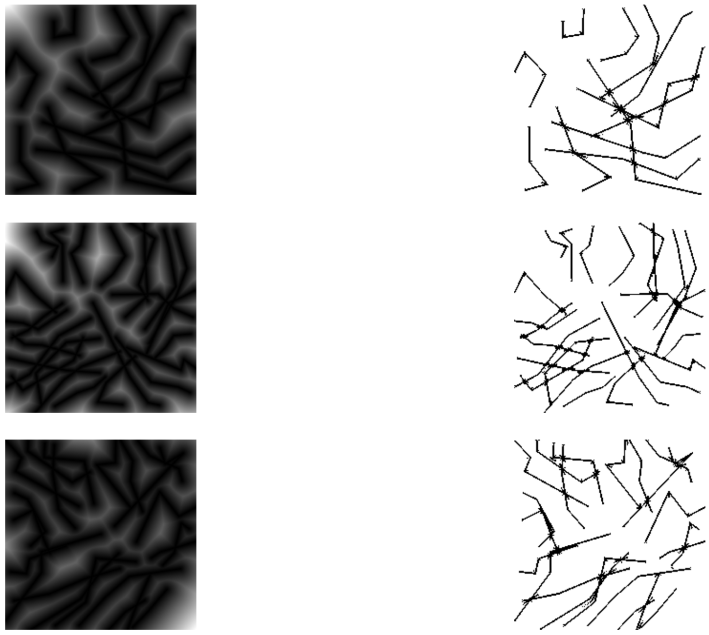

Figure 6.1 – Exemples d'images en remplaçant les feature points par des segments. (a) : Image d'intensité ; (b) : Image seuillée.

Nous avons précédemment souligné qu'une phase d'interaction avec l'utilisateur pourrait être utile pour améliorer encore les résultats de la deuxième phase en lui permettant de modifier certains paramètres ou de sélectionner un type de pavage en particulier parmi plusieurs types acceptables, ou encore de sélectionner une certaine image obtenue lors des itérations de Basin-Hopping de la deuxième phase. Nous pourrions effectuer une étude des choix des utilisateurs portant justement sur le type de pavage préféré sur des images qui leur seraient proposées, cela pourrait permettre d'ajuster les choix que nous faisons sur la sélection du type de pavage ([Liu+15]). Ces choix pourraient même être inclus dans une métrique de similarité perceptuelle ([Gao+20]) qui serait ajoutée dans la première phase pour influer sur l'estimation du type de pavage.

Nous avons observé que les déformations pouvaient influer sur l'efficacité de l'estimation des paramètres, par exemple sur le type de pavage estimé lors de la première phase. De travaux futurs pourraient se pencher sur l'inversion de déformation ([Li+19]), afin d'obtenir des images sans déformation permettant une meilleure esti-

mation, ce qui est d'autant plus utile étant donné que les structures issues de textures naturelles peuvent être très déformées.

Pour notre première phase de l'estimation nous avons utilisé un CNN utilisant une architecture ResNet152V2. On pourrait se demander si d'autres réseaux plus récents seraient encore plus efficaces. En particulier les réseaux utilisant des mécanismes d'attention ([Vas+17]) et les transformeurs ([Dos+20]), ou d'autres CNN plus évolués au niveau de l'architecture ([Liu+22]). De la même manière, pour la reconstruction d'une image de C-PPTBF plausible nous avons utilisé le réseau Pix2Pix puis Pix2PixHD, qui sont des réseaux performants mais d'autres réseaux génératifs ont été développés par la suite, qui pourraient peut-être améliorer encore les résultats de reconstruction, par exemple à nouveau les réseaux utilisant des mécanismes d'attention ([Tan+19]), ou encore les modèles de diffusion ([DN21]).

Par exemple nous avons essayé le réseau AttentionGAN [Tan+19], qui ajoute des mécanismes d'attention à une architecture classique de cGAN, que nous avons entraîné de la même façon que les réseaux Pix2Pix et Pix2PixHD, c'est-à-dire sur un jeu de 240 000 images de C-PPTBF, sans intégrer néanmoins notre terme d'erreur sur les images binaires, pendant 10 jours pour un total de 32 époques. Nous présentons quelques résultats que nous comparons aux résultats que nous obtenons avec le réseau Pix2PixHD dans la figure 6.2. Les résultats semblent plutôt prometteurs même si inférieurs aux nôtres, mais cette piste pourrait intéressante par la suite tant les réseaux génératifs se sont développés ces dernières années.

Cependant plusieurs éléments sont à prendre en compte. Tout d'abord tous ces réseaux sont coûteux en temps de calcul, car ils utilisent des architectures plus complexes que les CNN et GAN habituels que nous avons présentés dans ce manuscrit, et consomment davantage de ressources en mémoire. Ces deux éléments font qu'il peut être difficile de faire une vraie étude détaillée des potentielles améliorations amenées par l'utilisation de réseaux plus complexes.

Un autre axe de recherche pourrait consister à chercher à automatiser l'extraction de structures à partir de textures ou de matériaux. Il existe des méthodes de filtrage permettant d'extraire des éléments structurés saillants, dont les plus récentes utilisent des réseaux de neurones profonds [JLW23]. L'application de telles méthodes n'est pas suffisante, le problème de la binarisation des images de structures se pose également, rendue difficile par les variations d'apparence des motifs structurés. Une piste pourrait être de mettre au point un modèle génératif spécifique.

L'application de notre modèle de C-PPTBF à la synthèse de textures et de ma-

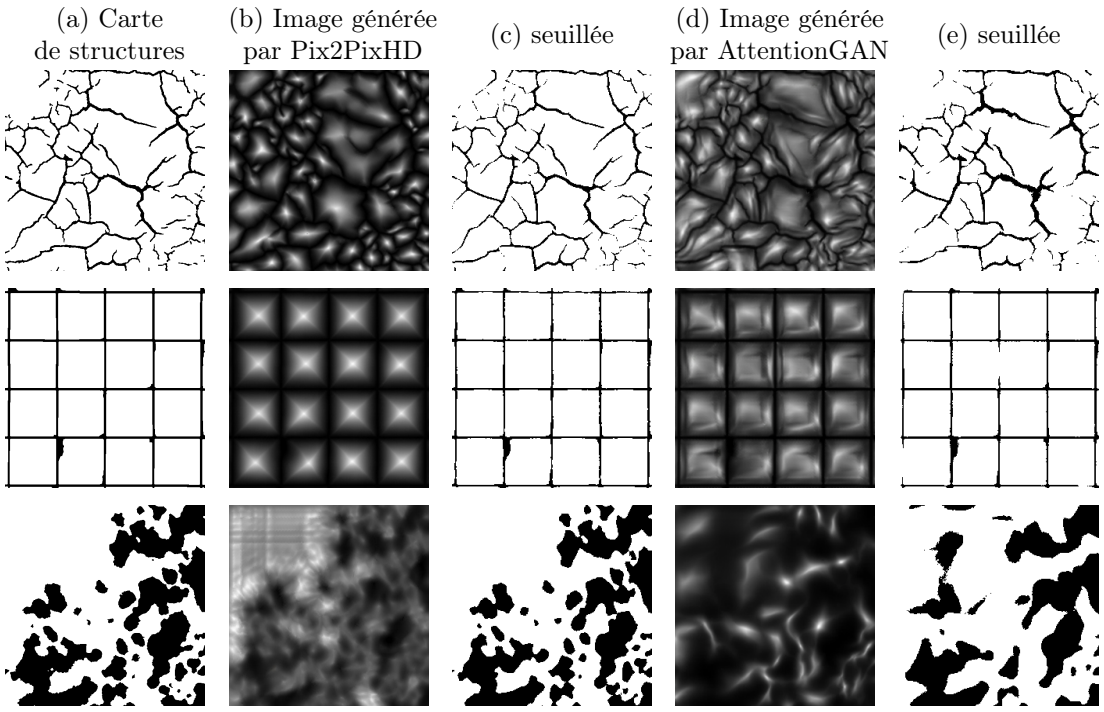

Figure 6.2 – Résultats de reconstructions d'images de C-PPTBF plausibles obtenues avec le réseau AttentionGAN comparées avec nos résultats obtenus avec le réseau Pix2PixHD. (a) : Carte de structures en entrée ; (b) : Image de C-PPTBF plausible reconstruite avec Pix2PixHD et (c) : sa version seuillée ; (d) : Image de C-PPTBF plausible reconstruite avec AttentionGAN et (e) : sa version seuillée.

tériaux présente des limites similaires à celles du modèle de PPTBF proposé par Guehl et al. [Gue+20 ; Gue22]. En particulier, nous n'avons pas étudié les problèmes de cohérence de la synthèse par pixels par l'algorithme PCTS lorsque les motifs des différentes cellules comportent des variations d'apparence. Ceci nécessiterait de passer de la génération de carte de structures procédurales binaires à des cartes à plus de deux labels, ce qui peut faire l'objet de travaux futurs. En outre, la question du filtrage pour le rendu sur des surfaces plongées en trois dimensions se pose.

On pourrait ensuite imaginer une extension de notre approche en trois dimensions. Par exemple, Paris et al. [Par+20] proposent une méthode pour générer des environnements de roches à partir de fractures construites procéduralement depuis un ensemble de points suivant une distribution de Poisson. Les fractures vont pou-

voir ensuite être utilisées pour générer des blocs contenant les détails volumétriques permettant de construire le terrain voulu, les formes des blocs et leur placement étant hautement contrôlables avec plusieurs paramètres. Ainsi nous pourrions nous aussi générer des environnements 3D avec un modèle de C-PPTBF adapté, tout en considérant que l'ajout d'une dimension complexifierait le modèle. Par exemple les types de pavage implémentés ne pourraient pas tous être adaptés facilement en 3D, il faudrait alors concevoir de nouveaux types de pavage. En 2D chaque cellule possède 8 cellules voisines, alors qu'en 3D elle en possède 26 ce qui augmente forcément le coût en temps de calcul.

Finalement, concevoir de nouveaux modèles procéduraux répondant aux besoins de l'industrie graphique, ou pour d'autres domaines comme le domaine médical, est un défi important. Plusieurs travaux récents ont montré qu'il était possible de directement générer des modèles procéduraux à l'aide de réseaux génératifs, comme par exemple ceux associés aux graphes de nœuds de Substance 3D Designer ([Gue+22] et [Hu+23]). Cette approche s'inscrit dans la lignée d'approches neuro-symboliques [Rit+23]. Les méthodes existantes restent limitées par une dépendance à des données d'entraînement massives et construites manuellement par des experts. Un axe de recherche pourrait consister à développer davantage ce type d'approches en cherchant à réduire la dépendance aux données d'entraînement, à contrôler la complexité et le coût en temps de calcul des modèles procéduraux générés, le caractère intuitif de leurs paramètres, et à améliorer la similarité aux exemples réels.

Bibliographie

[Bro66] Phil Brodatz. *Textures ; a photographic album for artists and designers*. Dover Publications, 1966. ISBN : 0486216691.

[Lew84] John-Peter Lewis. « Texture Synthesis for Digital Painting ». In : *SIGGRAPH Comput. Graph.* 18.3 (jan. 1984), p. 245-252. ISSN : 0097-8930. DOI : 10.1145/964965.808605. URL : https://doi.org/10.1145/964965.808605.

[Per85] Ken Perlin. « An Image Synthesizer ». In : *SIGGRAPH Comput. Graph.* 19.3 (juill. 1985), p. 287-296. ISSN : 0097-8930. DOI : 10.1145/325165.325247. URL : https://doi.org/10.1145/325165.325247.

[Wor96] Steven Worley. « A Cellular Texture Basis Function ». In : *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA : Association for Computing Machinery, 1996, p. 291-294. ISBN : 0897917464. DOI : 10.1145/237170.237267.

[DG97] J.-M. Dischler et D. Ghazanfarpour. « A Procedural Description of Geometric Textures by Spectral and Spatial Analysis of Profiles ». In : *Computer Graphics Forum* 16.s3 (1997), p. C129-C139. DOI : 10.1111/1467-8659.16.3conferenceissue.14.

[LP00] Laurent Lefebvre et Pierre Poulin. « Analysis and Synthesis of Structural Textures ». In : *Graphics Interface 2000*. Mai 2000, p. 77-86.

[Ebe+02] David S. Ebert et al. *Texturing and Modeling : A Procedural Approach*. 3rd. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2002. ISBN : 1558608486.

[BD04] Eric Bourque et Gregory Dudek. « Procedural Texture Matching and Transformation ». In : *Computer Graphics Forum* (2004). ISSN : 1467-8659. DOI : 10.1111/j.1467-8659.2004.00777.x.

[IO04] Masao Iwamatsu et Yutaka Okabe. « Basin hopping with occasional jumping ». In : *Chemical Physics Letters* 399 (2004), p. 396-400.

[Lin+04] Wen Chieh Lin et al. « A comparison study of four texture synthesis algorithms on near-regular textures ». English (US). In : *ACM SIGGRAPH 2004 Posters, SIGGRAPH 2004*. Sous la dir. de Ronen Barzel. International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2004 ; Conference date : 08-08-2004 Through 12-08-2004. Association for Computing Machinery, Inc, août 2004. doi : 10.1145/1186415.1186435.

[LH05] Sylvain Lefebvre et Hugues Hoppe. « Parallel Controllable Texture Synthesis ». In : *ACM Trans. Graph.* 24.3 (juill. 2005), p. 777-786. issn : 0730-0301. doi : 10.1145/1073204.1073261. url : https://doi.org/10.1145/1073204.1073261.

[DH06] Daniel Dunbar et Greg Humphreys. « A spatial data structure for fast Poisson-disk sample generation ». In : *ACM SIGGRAPH 2006 Papers*. SIGGRAPH '06. Boston, Massachusetts : Association for Computing Machinery, 2006, p. 503-508. isbn : 1595933646. doi : 10.1145/1179352.1141915. url : https://doi.org/10.1145/1179352.1141915.

[Lag+09] Ares Lagae et al. « Procedural Noise using Sparse Gabor Convolution ». In : *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2009)* 28.3 (juill. 2009), p. 54-64. doi : 10.1145/1531326.1531360.

[Wei+09] Li-Yi Wei et al. « State of the Art in Example-based Texture Synthesis ». In : *Eurographics 2009, State of the Art Report, EG-STAR*. Eurographics Association, 2009. url : http://www-sop.inria.fr/reves/Basilic/2009/WLKT09.

[Bus+10] P. P. Busto et al. « Instant Texture Synthesis by Numbers ». In : *International Symposium Vision, Modeling, and Visualization, The Eurographics Association*. 2010.

[GD10] G. Gilet et J-M. Dischler. « An Image-Based Approach for Stochastic Volumetric and Procedural Details ». In : *Computer Graphics Forum* 29.4 (2010), p. 1411-1419. doi : 10.1111/j.1467-8659.2010.01738.x.

[Lag+10] A. Lagae et al. « A Survey of Procedural Noise Functions ». In : *Computer Graphics Forum* (2010). issn : 1467-8659. doi : 10.1111/j.1467-8659.2010.01827.x.

[Gal+12] Bruno GALERNE et al. « Gabor Noise by Example ». In : *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)* 31.4 (juill. 2012), 73 :1-73 :9. DOI : 10.1145/2185520.2335424.

[Gil+14] Guillaume GILET et al. « Local Random-Phase Noise for Procedural Texturing ». In : *ACM Trans. Graph.* 33.6 (nov. 2014). ISSN : 0730-0301. DOI : 10.1145/2661229.2661249.

[Lef14] Sylvain LEFEBVRE. « Synthèse de textures par l'exemple pour les applications interactives ». Habilitation à diriger des recherches. Université de Lorraine (Nancy), juin 2014. URL : https://inria.hal.science/tel-01388378.

[GEB15a] Leon GATYS, Alexander S ECKER et Matthias BETHGE. « Texture Synthesis Using Convolutional Neural Networks ». In : *Advances in Neural Information Processing Systems*. Sous la dir. de C. CORTES et al. T. 28. Curran Associates, Inc., 2015.

[GEB15b] Leon A. GATYS, Alexander S. ECKER et Matthias BETHGE. *A Neural Algorithm of Artistic Style*. 2015. DOI : 10.48550/ARXIV.1508.06576.

[KB15] Diederik P. KINGMA et Jimmy BA. « Adam : A Method for Stochastic Optimization ». In : *International Conference on Learning Representations (ICLR'15)*. 2015.

[Liu+15] Jun LIU et al. « Visual Perception of Procedural Textures : Identifying Perceptual Dimensions and Predicting Generation Models ». In : *PLOS ONE* 10.6 (juin 2015), p. 1-22. DOI : 10.1371/journal.pone.0130335. URL : https://doi.org/10.1371/journal.pone.0130335.

[SZ15] Karen SIMONYAN et Andrew ZISSERMAN. « Very Deep Convolutional Networks for Large-Scale Image Recognition ». In : *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Sous la dir. d'Yoshua BENGIO et Yann LECUN. 2015. URL : http://arxiv.org/abs/1409.1556.

[GEB16] Leon A. GATYS, Alexander S. ECKER et Matthias BETHGE. « Image Style Transfer Using Convolutional Neural Networks ». In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 2414-2423. DOI : 10.1109/CVPR.2016.265.

[He+16a] Kaiming HE et al. « Deep Residual Learning for Image Recognition ». In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 770-778. DOI : 10.1109/CVPR.2016.90.

[He+16b] Kaiming He et al. « Identity Mappings in Deep Residual Networks ». In : *Computer Vision – ECCV 2016*. Sous la dir. de Bastian Leibe et al. Cham : Springer International Publishing, 2016, p. 630-645.

[JAF16] Justin Johnson, Alexandre Alahi et Li Fei-Fei. « Perceptual Losses for Real-Time Style Transfer and Super-Resolution ». In : *Computer Vision – ECCV 2016*. Sous la dir. de Bastian Leibe et al. Cham : Springer International Publishing, 2016, p. 694-711. isbn : 978-3-319-46475-6.

[Pav+16] Nicolas Pavie et al. « Procedural Texture Synthesis by Locally Controlled Spot Noise ». In : *Proceedings of the 24th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS Association*. 2016, p. 71-79.

[Sze+16] Christian Szegedy et al. « Rethinking the Inception Architecture for Computer Vision ». In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 2818-2826. doi : 10.1109/CVPR.2016.308.

[Cho17] F. Chollet. « Xception : Deep Learning with Depthwise Separable Convolutions ». In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA : IEEE Computer Society, juill. 2017, p. 1800-1807. doi : 10.1109/CVPR.2017.195. url : https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.195.

[Hua+17] Gao Huang et al. « Densely Connected Convolutional Networks ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[Iso+17] Phillip Isola et al. « Image-to-Image Translation with Conditional Adversarial Networks ». In : *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. 2017.

[Sze+17] Christian Szegedy et al. « Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning ». In : *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA : AAAI Press, 2017, p. 4278-4284.

[Vas+17] Ashish VASWANI et al. « Attention is All you Need ». In : *Advances in Neural Information Processing Systems*. Sous la dir. d'I. GUYON et al. T. 30. Curran Associates, Inc., 2017. URL : https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[ADM18] L. Raad ans A. DAVY, A. DESOLNEUX et J.-M. MOREL. « A survey of exemplar-based texture synthesis ». In : *Annals of Mathematical Sciences and Applications* 3.1 (2018), p. 89-148.

[Akl+18] Adib AKL et al. « A survey of exemplar-based texture synthesis methods ». In : *Computer Vision and Image Understanding* 172 (2018), p. 12-24. ISSN : 1077-3142. DOI : https://doi.org/10.1016/j.cviu.2018.04.001. URL : https://www.sciencedirect.com/science/article/pii/S1077314218300523.

[Wan+18] Ting-Chun WANG et al. « High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[Zho+18] Yang ZHOU et al. « Non-Stationary Texture Synthesis by Adversarial Expansion ». In : *ACM Trans. Graph.* 37.4 (juill. 2018). ISSN : 0730-0301. DOI : 10.1145/3197517.3201285. URL : https://doi.org/10.1145/3197517.3201285.

[Zop+18] Barret ZOPH et al. « Learning Transferable Architectures for Scalable Image Recognition ». In : juin 2018, p. 8697-8710. DOI : 10.1109/CVPR.2018.00907.

[ZWW18] Károly ZSOLNAI-FEHÉR, Peter WONKA et Michael WIMMER. « Gaussian material synthesis ». In : *ACM Trans. Graph.* 37.4 (2018), 76 :1-76 :14.

[FAW19] Anna FRÜHSTÜCK, Ibraheem ALHASHIM et Peter WONKA. « TileGAN : Synthesis of Large-Scale Non-Homogeneous Textures ». In : *ACM Trans. Graph.* 38.4 (juill. 2019). ISSN : 0730-0301. DOI : 10.1145/3306346.3322993. URL : https://doi.org/10.1145/3306346.3322993.

[Gai+19] Mathieu GAILLARD et al. « Dendry : A Procedural Model for Dendritic Patterns ». In : *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '19. Montreal, Quebec, Canada : ACM, 2019, 16 :1-16 :9. ISBN : 978-1-4503-6310-5. DOI : 10.1145/3306131.3317020.

[HDR19] Yiwei Hu, Julie Dorsey et Holly Rushmeier. « A Novel Framework for Inverse Procedural Texture Modeling ». In : *ACM Trans. Graph.* 38.6 (nov. 2019). ISSN : 0730-0301. DOI : 10.1145/3355089.3356516.

[Li+19] Xiaoyu Li et al. « Blind Geometric Distortion Correction on Images Through Deep Learning ». In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, p. 4855-4864.

[TL19] Mingxing Tan et Quoc V. Le. « EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks ». In : *CoRR* abs/1905.11946 (2019). eprint : 1905.11946.

[Tan+19] Hao Tang et al. « Attention-Guided Generative Adversarial Networks for Unsupervised Image-to-Image Translation ». In : *International Joint Conference on Neural Networks (IJCNN)*. 2019.

[Tri+19] Thibault Tricard et al. « Procedural Phasor Noise ». In : *ACM Trans. Graph.* 38.4 (juill. 2019). ISSN : 0730-0301. DOI : 10.1145/3306346.3322990.

[YLT19] Cem Yuksel, Sylvain Lefebvre et Marco Tarini. « Rethinking Texture Mapping ». In : *Computer Graphics Forum* (2019). ISSN : 1467-8659. DOI : 10.1111/cgf.13656.

[Dos+20] Alexey Dosovitskiy et al. « An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale ». In : *CoRR* abs/2010.11929 (2020). arXiv : 2010.11929. URL : https://arxiv.org/abs/2010.11929.

[Gao+20] Ying Gao et al. « A Perception-Inspired Deep Learning Framework for Predicting Perceptual Texture Similarity ». In : *IEEE Transactions on Circuits and Systems for Video Technology* 30.10 (2020), p. 3714-3726. DOI : 10.1109/TCSVT.2019.2944569.

[Gue+20] P. Guehl et al. « Semi-Procedural Textures Using Point Process Texture Basis Functions ». In : *Computer Graphics Forum* 39.4 (2020), p. 159-171. DOI : 10.1111/cgf.14061.

[Guo+20a] Y. Guo et al. « A Bayesian Inference Framework for Procedural Material Parameter Estimation ». In : *Computer Graphics Forum* 39.7 (2020), p. 255-266. DOI : 10.1111/cgf.14142.

[Guo+20b] Yu Guo et al. « MaterialGAN : Reflectance Capture Using a Generative SVBRDF Model ». In : *ACM Trans. Graph.* 39.6 (nov. 2020). ISSN : 0730-0301. DOI : 10.1145/3414685.3417779. URL : https://doi.org/10.1145/3414685.3417779.

[Par+20] A. Paris et al. « Modeling rocky scenery using implicit blocks ». In : *The Visual Computer* 36.10–12 (oct. 2020), p. 2251-2261. ISSN : 0178-2789. DOI : 10.1007/s00371-020-01905-6.

[Shi+20] Liang Shi et al. « MATch : Differentiable Material Graphs for Procedural Material Capture ». In : *ACM Trans. Graph.* 39.6 (nov. 2020). ISSN : 0730-0301. DOI : 10.1145/3414685.3417781.

[DN21] Prafulla Dhariwal et Alex Nichol. « Diffusion Models Beat GANs on Image Synthesis ». In : *CoRR* abs/2105.05233 (2021). arXiv : 2105.05233. URL : https://arxiv.org/abs/2105.05233.

[Hei+21] Eric Heitz et al. « A Sliced Wasserstein Loss for Neural Texture Synthesis ». In : *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2021.

[Kim+21] Hansoo Kim et al. « Edge-Based Procedural Textures ». In : *Vis. Comput.* 37.9–11 (juill. 2021), p. 2595-2606. ISSN : 0178-2789. DOI : 10.1007/s00371-021-02212-4.

[BAD22] Guillaume Baldi, Rémi Allègre et Jean-Michel Dischler. « Differentiable Point Process Texture Basis Functions for inverse procedural modeling of cellular stochastic structures ». In : *Journées Françaises de l'Informatique Graphique, (jFIG 2022), Bordeaux, France, 23 au 25 novembre 2022*. Bordeaux, France, nov. 2022. URL : https://hal.science/hal-03926460.

[Gai+22] Mathieu Gaillard et al. « Automatic Differentiable Procedural Modeling ». In : *Computer Graphics Forum* 41 (mai 2022), p. 289-307. DOI : 10.1111/cgf.14475.

[Gue22] Pascal Guehl. « Modèle d'apparence semi-procédural pour le contrôle et le passage à l'échelle de la synthèse de textures et de matériaux ». Theses. Université de Strasbourg, déc. 2022. URL : https://theses.hal.science/tel-04090150.

[Gue+22] Paul Guerrero et al. « MatFormer : A Generative Model for Procedural Materials ». In : *ACM Trans. Graph.* 41.4 (juill. 2022). ISSN : 0730-0301. DOI : 10.1145/3528223.3530173. URL : 10.1145/3528223.3530173.

[Hu+22a] Yiwei Hu et al. « An Inverse Procedural Modeling Pipeline for SVBRDF Maps ». In : *ACM Trans. Graph.* 41.2 (jan. 2022). ISSN : 0730-0301. DOI : 10.1145/3502431.

[Hu+22b] Yiwei Hu et al. « Controlling Material Appearance by Examples ». In : *Computer Graphics Forum* 41.4 (2022), p. 117-128. DOI : 10.1111/cgf.14591.

[Hu+22c] Yiwei Hu et al. « Node Graph Optimization Using Differentiable Proxies ». In : *ACM SIGGRAPH 2022 Conference Proceedings*. SIGGRAPH '22. Vancouver, BC, Canada : Association for Computing Machinery, 2022. ISBN : 9781450393379. DOI : 10.1145/3528233.3530733.

[Liu+22] Zhuang Liu et al. « A ConvNet for the 2020s ». In : *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, p. 11966-11976. DOI : 10.1109/CVPR52688.2022.01167.

[BAD23] Guillaume Baldi, Rémi Allègre et Jean-Michel Dischler. « Differentiable point process texture basis functions for inverse procedural modeling of cellular stochastic structures ». In : *Computers & Graphics* 112 (2023), p. 116-131. ISSN : 0097-8493. DOI : https://doi.org/10.1016/j.cag.2023.04.004. URL : https://www.sciencedirect.com/science/article/pii/S0097849323000419.

[Hu+23] Yiwei Hu et al. « Generating Procedural Materials from Text or Image Prompts ». In : *ACM SIGGRAPH 2023 Conference Proceedings*. 2023.

[JLW23] Lixi Jiang, Xujie Li et Yandan Wang. « Iterative unsupervised deep bilateral texture filtering ». In : *The Visual Computer* (juill. 2023). ISSN : 1432-2315. DOI : 10.1007/s00371-023-03010-w. URL : https://doi.org/10.1007/s00371-023-03010-w.

[LSM23] Beichen Li, Liang Shi et Wojciech Matusik. « End-to-End Procedural Material Capture with Proxy-Free Mixed-Integer Optimization ». In : *ACM Trans. Graph.* 42.4 (juill. 2023). ISSN : 0730-0301. DOI : 10.1145/3592132. URL : https://doi.org/10.1145/3592132.

[Rit+23] Daniel Ritchie et al. « Neurosymbolic Models for Computer Graphics ». In : *Computer Graphics Forum* 42.2 (2023), p. 545-568. DOI : https://doi.org/10.1111/cgf.14775. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14775. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14775.

[Zho+23] Xilong ZHOU et al. « A Semi-Procedural Convolutional Material Prior ». In : *Computer Graphics Forum* 42.6 (2023), e14781. DOI : https://doi.org/10.1111/cgf.14781. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14781.

[22] *Adobe Substance 3D*. https://www.substance3d.com. 2022.

Université de Strasbourg

Guillaume BALDI

Contributions à la modélisation procédurale de structures cellulaires stochastiques 2D et à leur génération par l'exemple

École doctorale Mathématiques, sciences de l'information et de l'ingénieur | ED 269 Université de Strasbourg

Résumé

La création de matériaux et de textures procéduraux demande une grande expertise et constitue un travail long, fastidieux et coûteux, c'est pourquoi on cherche à développer des outils permettant leur génération automatique à partir d'exemples en entrée fournis sous la forme d'images : on parle de *modélisation procédurale inverse*.
Dans cette thèse, nous proposons un modèle procédural appelé Cellular Point Process Texture Basis Function (C-PPTBF) permettant de représenter des structures cellulaires stochastiques 2D, impliquant des fonctions différentiables par rapport à la plupart de leurs paramètres, ce qui rend possible l'estimation de ces paramètres à partir d'exemples sans recourir entièrement à des réseaux de neurones profonds. Nous avons mis en place une chaîne de traitement permettant d'estimer les paramètres de notre modèle à partir d'exemples de structures fournis sous la forme d'images binaires, combinant une estimation réalisée à l'aide d'un réseau de neurones convolutif entraîné sur des images produites avec notre modèle de C-PPTBF et une phase d'estimation par descente de gradient directement sur les paramètres du modèle procédural.

Mots-clés : informatique graphique, modélisation procédurale inverse, synthèse de textures, modèle différentiable, réseaux de neurones, descente de gradient

Abstract

The creation of procedural materials and textures requires considerable expertise, and is time-consuming, tedious and costly. We are therefore looking to develop tools for the automatic generation of procedural textures and materials from input exemplars provided in the form of images: This is known as *inverse procedural modeling*.
In this thesis, we propose a procedural model called Cellular Point Process Texture Basis Function (C-PPTBF) for representing 2D stochastic cellular structures, involving functions that are differentiable with respect to most of their parameters, making it possible to estimate these parameters from examples without resorting entirely to deep neural networks. We have set up a processing pipeline to estimate the parameters of our model from structural examples provided in the form of binary images, combining an estimation performed using a convolutional neural network trained on images produced with our C-PPTBF model and an estimation phase using gradient descent directly on the parameters of the procedural model.

Keywords: computer graphics, inverse procedural modeling, texture synthesis, differentiable model, neural networks, gradient descent