Processus stochastiques et algorithmes pour la synthèse et le rendu de textures

Nicolas Lutz

Université de Strasbourg

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

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

THÈSE

Présentée par : **Nicolas Lutz**

Soutenue le : **1er juillet 2021**

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

Discipline/Spécialité : **Informatique**

Processus stochastiques et algorithmes pour la synthèse et le rendu de textures

Thèse dirigée par :

M. DISCHLER Jean-Michel	Professeur des universités, Université de Strasbourg
M. SAUVAGE Basile	Maître de conférences, Université de Strasbourg

Rapporteurs :

M. GALERNE Bruno	Professeur des universités, Université d'Orléans
M. LEFEBVRE Sylvain	Directeur de recherches, Inria Nancy

Autres membres du jury :

M. HEITZ Eric	Director of research, Unity Grenoble
M. PARIS Sylvain	Fellow and lab director, Adobe Systems Inc.

Remerciements

J'aimerais remercier en premier lieu mes collègues. Tout d'abord, un grand merci à Basile et Jean-Michel, qui ont été des encadrants, enseignants, pédagogues, conseillers ou co-auteurs exceptionnels, et ce depuis ma première année à l'université. J'aimerais aussi remercier les doctorants et post-doctorants ; David, Florian, Geoffrey, les Guillaumes, Katia, Mélinda, Pascal, Paul, Quentin, Sabah, Simon et Xavier, que ce soit pour les contributions scientifiques, mais aussi pour les pauses, petits-déjeuners, déjeuners et goûters dans la salle de pause ou celle de contrepause, pour avoir respecté la sieste de 12h30, m'encourager à aller à la piscine ou au lac, me donner un petit sourire avec des messages privés espacés irrégulièrement, et bien d'autres choses. Merci à nos ingénieurs de recherche, qui m'ont apporté leur aide à différents niveaux, Frédéric, Joris, Jérôme, Sylvain. Merci également à ceux que j'ai eu le plaisir de voir en dehors du travail, dont Claire et Johnathan.

Merci à Bruno Galerne, Sylvain Lefebvre, Eric Heitz et Sylvain Paris pour avoir accepté de participer au jury, et pour le temps qu'ils ont consacré à mon manuscrit et à ma soutenance. Les contributions de ce manuscrit s'appuient entre autres sur certaines de leurs contributions majeures.

J'aimerais remercier les membres de ma famille, quelque soit l'échelle ou la portée de leurs contributions. Merci à Chloé, que j'ai rencontré tôt pendant ma thèse et avec qui j'entretiens maintenant une relation romantique. Merci à mon traducteur en chef secret, Matthew Cazaley, qui repasse sur tous mes textes anglais et dont le serveur de discussion et ses 10 membres figurent en tête de ma liste depuis le début de mon master. Merci également à toutes mes connaissances qui ne rentrent pas dans les différentes catégories de remerciements, vous êtes nombreux, et vous savez qui vous êtes.

Direction de la **recherche**
et de la **valorisation**
Collège **doctoral**
Université de Strasbourg

J'affirme être informé que le plagiat est une faute grave susceptible de mener à des sanctions administratives et disciplinaires pouvant aller jusqu'au renvoi de l'Université de Strasbourg et passible de poursuites devant les tribunaux de la République Française.

Je suis conscient(e) que l'absence de citation claire et transparente d'une source empruntée à un tiers (texte, idée, raisonnement ou autre création) est constitutive de plagiat.

Au vu de ce qui précède, **j'atteste sur l'honneur que le travail décrit dans mon manuscrit de thèse est un travail original et que je n'ai pas eu recours au plagiat ou à toute autre forme de fraude.**

I affirm that I am aware that plagiarism is a serious misconduct that may lead to administrative and disciplinary sanctions up to dismissal from the University of Strasbourg and liable to prosecution in the courts of the French Republic.

I am aware that the absence of a clear and transparent citation of a source borrowed from a third party (text, idea, reasoning or other creation) is constitutive of plagiarism.

In view of the foregoing, **I hereby certify that the work described in my thesis manuscript is original work and that I have not resorted to plagiarism or any other form of fraud.**

Nom : **Lutz**
Prénom : **Nicolas**
École doctorale : **MSII**
Laboratoire : **ICube**
Date : **28/07/2021**
Signature :

Nicolas Lutz

Processus stochastiques pour la synthèse et le rendu de textures

Résumé

En informatique graphique, les mondes virtuels suivant des considérations d'esthétisme demandent de représenter des objets avec de nombreux détails, classiquement apportés par les texture. La synthèse de textures permet d'introduire de l'aléa contrôlé dans le processus de génération et de rendu des textures, qui peuvent alors être plus larges, moins coûteuses en occupation mémoire et limiter les artefacts de répétition et d'alignement.

Dans cette thèse, nous formalisons la synthèse de textures en tant qu'implémentation de processus stochastiques. Cette formalisation nous aide à comprendre les algorithmes de bruit procédural, et nous montrons qu'une classe de processus n'ayant pas été utilisé dans la synthèse par le passé, les processus cyclostationnaires, peuvent être exploités pour synthétiser des textures à organisation globale régulière (comme des murs de briques).

Une de nos contributions porte également sur la création du filtrage d'une synthèse par pavage apériodique nommée « échange de contenus ». Le filtrage permet d'effectuer le rendu d'une surface texturée avec cette synthèse sans artefacts de scintillements ou effets de Moiré.

Mots-clés : informatique, synthèse de textures, bruit procédural, rendu, filtrage

Résumé en anglais

In computer graphics, virtual worlds following aestheticism considerations require to represent objects with much detail, typically provided by textures. Texture synthesis allows to introduce controlled randomness in the process of texture generation and rendering, enabling them to be larger, less expensive in memory consumption, and to limit repetition and alignment artifacts.

In this thesis, we formalize texture synthesis as implementations of stochastic processes. This formalization helps us understand procedural noise algorithms, and we show that a class of processes whcih had not yet been used in texture synthesis, cyclostationary processes, can be exploited to synthesize textures with a regular global organization (such as brick walls).

One of our contributions also concerns the creation of a texture filtering technique, for an aperiodic tiling synthesis technique called "content exchange". This filtering enables to render a surface textured with the content exchange, without flickering artifacts or Moiré effects.

Keywords : computer science, texture synthesis, procedural noise, rendering, filtering

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Mondes virtuels

FIGURE 1.1 – Exemples de mondes virtuels. À gauche : le jeu *The Legend of Zelda : Breath of the Wild* (2017). À droite, en haut : rendu schématique d'une simulation d'intervention médicale sur le foie [HPCE16]. À droite, en bas : Visualisation d'un plan à travers une application de réalité augmentée [BCL15].

En informatique graphique, les mondes virtuels sont utilisés pour représenter des scènes sur un support numérique permettant leur visualisation. Ces scènes peuvent être des reproductions de la réalité ou de pures créations. Les mondes virtuels peuvent être utilisés dans de nombreux domaines, dont certains sont illustrés en Figure 1.1. Ils sont notamment utilisés dans :

— le domaine du divertissement, par exemple dans le jeu vidéo, l'animation et la post-production cinématographique ;
— le domaine médical, avec l'imagerie médicale, par exemple par la reconstruction tri-dimensionnelle de données tomographiques ou la planification chirurgicale virtuelle ;
— le domaine de l'agro-alimentaire, par exemple avec la reconstruction et la visualisations de plants ;
— le domaine de l'industrie, par exemple avec les maquettes virtuelles de bâtiments et de machines ;
— le domaine de la formation, avec la réalité virtuelle.

Les mondes virtuels ont parfois pour vocation de sembler réels, parfois d'aider à la visualisation schématique et illustrative, et parfois d'être plus irréalistes. Dans tous les cas, ils sont conçus de façon à être immersifs. Le sentiment d'immersion est caractérisé par le degré avec lequel un monde virtuel parvient à induire la participation mentale de l'utilisateur, et dépend donc de la qualité de l'expérience proposée par le monde virtuel à l'utilisateur. L'expérience utilisateur est au cœur des problématiques intrinsèques à l'amélioration des outils de création des mondes virtuels. En particulier, dans les cas où il est attendu que la construction du monde virtuel suive des considérations de réalisme ou d'esthétisme, cette expérience dépend fortement de l'apparence des objets virtuels présents dans le monde. Additionnellement, un monde virtuel peut avoir vocation à être visualisé en temps réel, auquel cas les images de la scène doivent être calculées rapidement pour satisfaire l'utilisateur. Dans ce genre d'applications, il est attendu que les mondes virtuels soient très larges, très détaillés et que leur visualisation soit calculée en temps réel afin de proposer l'expérience la plus immersive possible.

Concrètement, un monde virtuel est un ensemble d'objets de nature différente, comme une caméra, des lumières, et des objets tri-dimensionnels comme des surfaces, positionnés dans une scène. Ces objets peuvent être dynamiques : dans certaines applications, ils peuvent par exemple changer d'apparence au cours du temps ou interagir entre eux. Le but est de prendre en compte les propriétés attribuées aux objets, comme leur position, leur orientation, leurs propriétés colorimétriques, leur réflectance et leurs propriétés dynamiques pour générer une image de la scène appelée un « rendu » selon la position et l'orientation d'une caméra.

Nous nous intéressons particulièrement aux objets surfaciques, qui sont munis d'une topologie décrite par des équations ou par des agencements de primitives, comme des triangles. À cette topologie sont attachés des « plongements », c'est-à-dire des attributs comme des positions ou des normales, qui servent à décrire la position et l'orientation de la surface dans le monde virtuel de façon plus ou moins précise. Il est également possible de la plonger par des attributs tels que de l'albedo, du relief, de la brillance ou de la rugosité, qui permettent de calculer une couleur en tout point de la surface en fonction des paramètres d'éclairage donnés. Pour augmenter la quantité de détail, il faut alors des sur-

faces de haute résolution très détaillées, ce qui pose plusieurs problèmes : la création de la surface devient plus difficile, et son occupation mémoire ainsi que son temps de calcul augmente avec la résolution. Souvent, de nos jours, la géométrie est grossière, et de nombreux détails, comme la couleur, les normales ou les variations de relief sont apportés par les textures.

1.2 Textures

En informatique graphique, une texture est une fonction d'un ensemble d'indices appelés les coordonnées de texture vers un ensemble de valeurs. Les textures ont le rôle d'ajouter des détails à des objets numériques sans augmenter leur résolution.

Nous nous intéressons aux textures bi-dimensionnelles, définies par des coordonnées de texture à deux dimensions. Elles ont pour vocation d'habiller la surface par la représentation graphique d'un matériau (bois, métal...) ou d'une substance (eau, peinture...). Elle est représentée sur la surface par sa paramétrisation, qui permet d'associer des coordonnées de texture en tout point de la surface. La texture est souvent représentée sous la forme d'une image numérique, c'est-à-dire un ensemble de valeurs indexées sur des positions entières dont les éléments sont appelées les texels. L'ensemble des valeurs d'une texture est typiquement unidimensionnel (niveau de gris) ou tridimensionnel (couleurs), mais d'autres formes sont possibles, par exemple à quatre dimensions (couleur et transparence) ou plus (avec une dimension représentant les normales, les hauteurs, les tangentes...). On appelle « canal » chaque dimension des valeurs d'une texture, terme emprunté aux images numériques colorées et pour distinguer la dimension des indices de la dimension des valeurs.

1.3 Création des textures et rendu

Dans un pipeline graphique classique (Figure 1.2), la création des contenus et le rendu sont consécutifs. La création des textures, en particulier, est effectuée de différentes façons en fonction de la nature de la texture. Par exemple, la couleur peut être créée via un simple dessin numérique ou acquise par la photographie ; les données de matériau ou la réflectance peuvent être obtenus par acquisition numérique, par exemple à l'aide d'un spectrophotomètre ; le relief peut être obtenu grâce à des données géographiques acquises via la télémétrie à grande échelle, ou à petite échelle avec l'acquisition par franges lumineuses. Les textures peuvent passer par une phase de traitement, où leur contenu est altéré manuellement ou automatiquement en fonction des besoins de l'utilisateur. Elles peuvent par exemple être pré-filtrées pour accélérer le rendu dans les

applications temps réel, organisées dans un atlas pour texturer des maillages ou rendues périodiques pour paver une large surface sans discontinuités visibles.

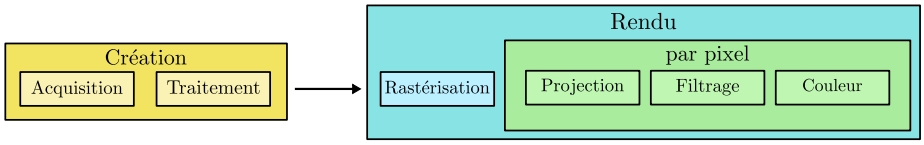


FIGURE 1.2 – Pipeline graphique classique pour une surface texturée. La texture est crée et traitée pour le rendu. Pendant le rendu, les objets sont rastérisés pour être transformés en un ensemble de pixels. L'empreinte du pixel sur la surface est projetée sur la texture, puis filtrée ; enfin, la couleur du pixel, qui dépend des texels dans l'empreinte et des interactions lumineuses de la scène, est calculée.

Lors du rendu, les objets de la scène sont rastérisés, c'est-à-dire qu'ils sont transformés selon la position et l'orientation de la caméra de la scène en un ensemble discret de pixels qui constituent l'image finale de scène. Pour calculer la couleur d'un pixel, son empreinte sur les objets de la scène doit être calculée. L'empreinte correspond à la surface du pixel projetée depuis la position de la caméra sur les objets de la scène. Dans le cas des surfaces texturées, elle est calculée et projetée dans l'ensemble d'indices de la texture. Elle est ensuite filtrée pour éviter les artefacts d'aliassage ; cette étape est décrite plus précisément dans la section 1.5.

Les textures sont directement concernées par les besoins accrus en matière de mondes virtuels. Au fur et à mesure que les besoins se complexifient, par l'augmentation significative des détails, de la taille du monde, et/ou de contraintes de temps de calcul du rendu, les besoins en terme de textures augmentent proportionnellement : il faut des textures plus larges et plus détaillées en ralentissant le moins possible l'étape de rendu. Se posent alors des questions pratiques : comment créer efficacement des textures qui répondent aux nouveaux besoins en terme de mondes virtuels ? Comment les utiliser pour créer un rendu immersif et efficace ?

1.4 Synthèse de textures

La synthèse de textures est un ensemble d'algorithmes automatiques ou semi-automatiques qui ont pour vocation d'augmenter le pipeline graphique avec des algorithmes de génération de textures basées sur une certaine quantité d'aléa contrôlé. Elle a pour vocation de fournir des outils qui permettent de créer plus facilement des textures correspondant mieux aux besoins, c'est-à-dire plus larges, de plus haute résolution, et sans répétitions. La nature aléatoire des modèles de synthèse a pour but de permettre la génération de textures sans devoir recourir à des algorithmes rudimentaires, par exemple, à la répétition d'un

même échantillon rectangulaire de données de texture, qui cause des artefacts de répétition (le même motif est répété à différents endroits) et des artefacts d'alignement (une grille régulière est visible) qui cassent l'immersion montré dans la Figure 1.3. On notera que les artefacts d'alignement ne sont pas problématiques pour les textures aux motifs distribués régulièrement, comme les murs de briques, mais elles sont tout de même concernées par les artefacts de répétition.

On désigne par « une synthèse » un algorithme de synthèse de textures. Une texture générée par une synthèse est désignée par « texture de sortie ».

FIGURE 1.3 – Rendus du jeu *MONSTER HUNTER RISE* (2021). Des artefacts de répétitions et d'alignement peuvent se voir sur chaque capture (gauche/droite). Sur les vues du bas, des artefacts d'alignement (gauche), et des artefacts de répétition (droite) sont illustrés.

Nous classons les modèles de synthèse dans trois catégories différentes, en fonction de la façon dont leur part d'aléatoire est définie :

— Des modèles algorithmiques ; la texture de sortie est définie comme le résultat d'une suite d'instructions algorithmiques contenant une part de pseudo-aléatoire.
— Des modèles statistiques ; la texture de sortie est définie comme la réalisation d'une fonction aléatoire dont les statistiques sont déterminées par des paramètres.
— Des modèles par optimisation ; la texture de sortie est définie comme l'optimum d'un système dont l'état initial est aléatoire.

1.4.1 Emploi de la synthèse

Les synthèses de textures sont munies de paramètres qui permettent de contrôler l'apparence des textures générées et la part d'aléa qui participe à la génération. Il existe différentes façons de paramétrer une synthèse, qui varient fortement en fonction du modèle algorithmique utilisé. Parmi elles, les modèles de synthèse dits « par l'exemple » ont la particularité de se servir d'une texture (appelée exemple) en entrée pour en déduire les paramètres de la synthèse. Elles ont pour but de contrôler plus facilement l'apparence de la texture de sortie : au lieu de gérer les paramètres de la synthèse manuellement, l'algorithme va lui-même déduire les paramètres de la synthèse. La synthèse par l'exemple peut, en outre, servir à :

1. augmenter la taille de la texture sans agrandir les motifs pour texturer une surface plus large ;
2. augmenter la résolution de la texture, pour pouvoir mieux détailler une surface ;
3. créer des textures ressemblantes mais non identiques pour texturer différentes surfaces avec le même type d'apparence.

La synthèse peut intervenir dans les deux grandes étapes de la génération d'un monde virtuel, comme montré dans la Figure 1.4.

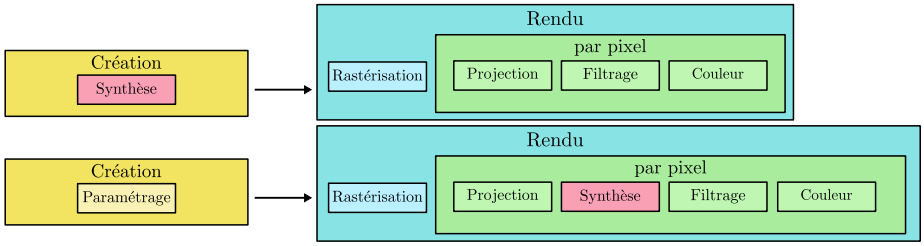


FIGURE 1.4 – Pipeline graphique altéré pour y rajouter des étapes de synthèse. La synthèse peut intervenir dans les deux grandes étapes du pipeline.

Création. La synthèse peut remplacer la création : par exemple, le logiciel vendu par Adobe, Substance Designer (Figure 1.5) est un exemple d'outil qui permet à un utilisateur de créer interactivement des textures par des procédés de synthèse, par exemple en combinant des motifs et une fonction de l'espace générant des valeurs aléatoires, appelé bruit procédural.

Lorsque l'utilisateur dispose déjà d'une texture, elle peut être traitée grâce à des algorithmes de synthèse : la périodisation (rendre une texture continue aux bords) et la stationnarisation (éliminer les motifs saillants) des textures sont des synthèses par l'exemple.

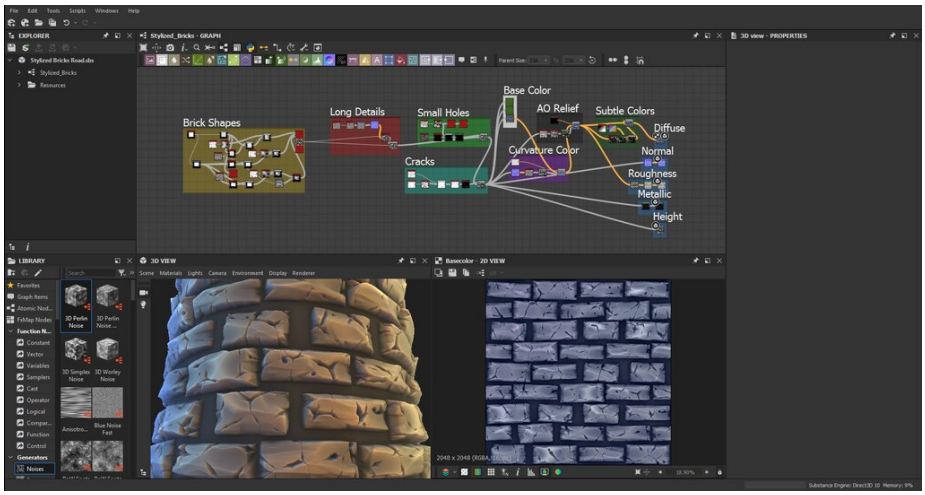


FIGURE 1.5 – Substance Designer, par Adobe. Le logiciel propose de créer des textures interactivement, en combinant le dessin, des images numériques et des motifs procéduraux.

La création des textures au travers de la synthèse permet donc de faciliter la production des textures et d'adapter des textures existantes à une application voulue.

Rendu. La synthèse peut aussi s'insérer dans le rendu lui-même, après la projection qui détermine l'empreinte d'un pixel sur la surface texturée : la texture, non stockée en mémoire jusque-là, est alors calculée et filtrée à ce moment-là.

On dit de cette étape de synthèse qu'elle est « à la volée » lorsque tous les calculs de texels peuvent être effectués indépendemment des texels voisins (ou dépendant d'un petit voisinage), ce qui permet de calculer n'importe quel point de la surface sans devoir calculer ou pré-calculer ses voisins. De plus, si la synthèse est suffisamment rapide pour les applications temps réel, elle est aussi qualifiée de « temps réel ».

Cette étape de synthèse est nécessaire lorsque la texture utilisée dans le monde virtuel n'est pas connue à l'avance, ce qui arrive lorsque sa taille est non bornée ou que la texture est indexée sur un domaine continu (il est possible de zoomer dessus à l'infini) ; l'impossibilité de stocker la texture sous la forme d'une image numérique justifie alors l'emploi de la synthèse de textures.

1.4.2 Enjeux de la synthèse

Comme la plupart des algorithmes, la synthèse de textures a des enjeux de performance, qui se traduisent par des enjeux d'occupation mémoire et de temps de calcul.

Occupation mémoire. L'occupation mémoire doit être adaptée à la capacité du matériel informatique, qui est limitée. C'est pour cette raison qu'il n'est pas possible de stocker une texture très grande ou très détaillée, notamment sur une carte graphique, où la quantité de mémoire est particulièrement limitée. Souvent, les larges données de textures rendues en temps réel sont gérées par des méthodes de « texture streaming », qui consistent à réserver une plage mémoire pour charger et décharger progressivement les textures du monde virtuel à différentes résolutions, mais ces méthodes peuvent causer des ralentissements et l'apparition et la disparition subite de détails. Les algorithmes de synthèse de textures doivent éviter d'occuper trop de mémoire et donc limiter au maximum l'occupation mémoire des données utilisées pour la synthèse, notamment par rapport au stockage explicite de la texture synthétisée.

Temps de calcul. Le temps de calcul doit être minimisé pour que l'application graphique ne soit pas ralentie. Lors de l'utilisation d'un algorithme de synthèse de textures dans l'étape de rendu, les algorithmes doivent être parallélisables et particulièrement rapides en évitant au maximum les accès mémoire et les longs calculs ; ainsi, les synthèses qui demandent plusieurs étapes d'itérations, comme les synthèses par optimisation, sont difficiles à adapter au temps réel. Paradoxalement, pré-calculer et stocker des données supplémentaires peut permettre de réduire la quantité de calculs, mais de larges données prennent plus de temps à être traitées par le matériel informatique, à cause des défauts de cache. Lors de la création d'une texture à l'aide d'un outil interactif, une génération en temps réel ou en temps interactif d'une texture permet une création plus efficace, en permettant par exemple à l'utilisateur de visualiser sans latence un aperçu de son travail.

1.4.3 Contribution à la synthèse de textures régulières

La synthèse de textures profite d'une panoplie de méthodes procédurales, qui sont des synthèses se basant entièrement sur des processus stochastiques et dont les propriétés sont connues. Seulement, ces synthèses tendent à toujours être stationnaires, c'est-à-dire que les statistiques sont identiques sur tout le domaine de la texture ; cette propriété limite la classe de textures qu'il est possible de traiter. Pour étendre cette classe, nous introduisons les processus cyclostationnaires de façon théorique sous des termes connus dans la section 3.3, et des modèles de

synthèse cyclostationnaires, plus particulièrement, de synthèse cyclostationnaire Gaussienne dans le chapitre 4. Nous proposons notamment d'adapter plusieurs algorithmes de synthèse stationnaire, qui s'insèrent dans l'étape de rendu ou prennent la place de l'étape de création. Les algorithmes que nous proposons permettent le rendu temps réel de textures dont les motifs sont des agencements réguliers. Nous proposons également un transfert d'histogramme cyclostationnaire basé sur le modèle du transport optimal de statistiques pour la génération de textures dont les statistiques ne sont pas nécessairement Gaussiennes.

1.5 Filtrage

L'étape de filtrage survient lors de l'étape de rendu, après la projection de l'empreinte d'un pixel dans la texture. L'étape de filtrage consiste à calculer l'intégrale de la lumière réfléchie de chaque point de la surface texturée vers le pixel. Cette étape est cruciale pour éviter les artefacts d'aliassage, qui sont le résultat du sous-échantillonage des hautes fréquences de la texture et se traduisent par des effets de Moiré ou des scintillements qui cassent l'immersion.

Dans les applications graphiques où les images sont calculées en temps réel, le filtrage est muni d'une étape de traitement appelée le « pré-filtrage » qui intervient dans la phase de création de la texture. Le principe du pré-filtrage consiste à effectuer les calculs d'intégrale coûteux en amont du rendu pour éviter d'avoir à les effectuer en temps réel. La technique la plus courante de pré-filtrage, utilisée pour filtrer des textures explicites, est le *MIP-mapping*, qui consiste à calculer une suite ordonné de textures appelée *MIP-map* représentant la texture à filtrer à différentes résolutions. Chaque élément de la *MIP-map* est une texture dont la résolution est diminuée de moitié par rapport à la précédente, en moyennant les texels 4 à 4. Pendant le rendu, la texture utilisée est alors celle dont la taille d'un texel est la plus proche de la taille de l'empreinte du pixel sur la surface texturée et filtrée. La pile contenant les textures de chaque résolution est parfois nommée « pyramide » à cause de la résolution décroissante des textures en fonction de la position dans la suite. L'idée derrière le pré-filtrage est de pré-calculer des intégrales des valeurs de la texture pour ne pas avoir à le faire pendant le rendu, où le temps de calcul doit être minimisé (Figure 1.6).

La synthèse à la volée interfère parfois avec l'étape de filtrage lorsqu'elle est utilisée conjointement avec les méthodes de pré-filtrage, parce que la texture n'est pas connue à l'avance et son pré-filtrage ne peut donc pas être calculé. Parfois, le pré-filtrage de l'exemple lui-même peut suffire [GLM17, HN18]. Cependant, cette solution n'est pas toujours valide, notamment dans le cas d'une synthèse par pavage apériodique à tuiles irrégulières nommée « synthèse par échange de contenus » (voir chapitre 2), qui implémente un processus discret dont des régions sont échangées avec d'autres régions pré-calculées depuis un exemple d'entrée. Nous proposons une méthode de pré-filtrage basé sur une

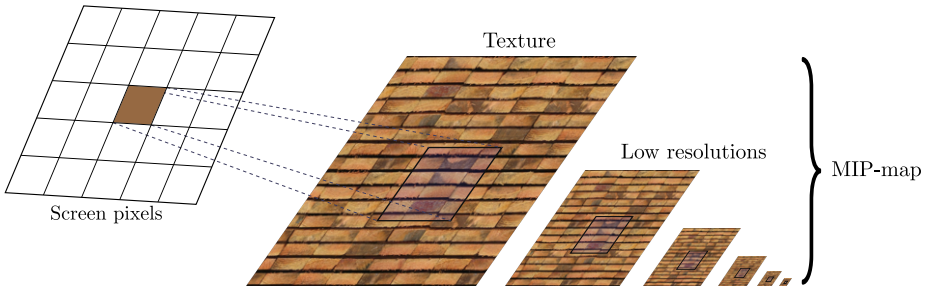


FIGURE 1.6 – Filtrage et pré-filtrage. Le principe du filtrage est d'intégrer la lumière partant de la surface vers la caméra. Le MIP-mapping est une technique de pré-filtrage consistant à calculer une MIP-map, formée de la texture et de résolutions plus basses de la texture, pour que l'empreinte corresponde à un minimum de texels, et ainsi réduire fortement le nombre de calculs.

forme de pré-filtrage de ces régions, ainsi que des propositions pour améliorer l'apparence des résultats de la synthèse elle-même.

1.6 Plan

Le plan de ce manuscrit se décline de la manière suivante :
- D'abord, nous effectuons un **état de l'art** de la synthèse de textures ;
- ensuite, nous présentons un pan de la théorie des **processus stochastiques** sous une forme utile pour la synthèse en informatique graphique ;
- puis nous présentons nos contributions sur la **synthèse cyclostationnaire Gaussienne**, qui consiste à s'appuyer sur la théorie des processus cyclostationnaires et des processus Gaussiens pour synthétiser des textures ayant des statistiques périodiques ;
- puis, nos contributions sur la **synthèse par échange de contenus**, et plus particulièrement du filtrage de cette synthèse ;
- enfin, nous donnons la **conclusion** de ce manuscrit.

Chapitre 2

État de l'art

Dans ce chapitre, nous dressons un état de l'art en lien avec nos travaux.

Nous nous intéressons à la synthèse par bruit procédural, plus particulièrement à la synthèse par convolution parcimonieuse et à phase aléatoire, qui est exploitée par la contribution principale du chapitre 4. Ce chapitre se positionne également comme une amélioration de la synthèse de textures régulières, c'est-à-dire dont le motif représenté a une forme d'organisation globale régulière (murs de briques, mosaïques, etc). Nous nous intéressons aussi aux synthèses de textures qui s'appuient sur la réorganisation de contenu en temps réel et aux problèmes de filtrage qui en découlent, et qui concernent la contribution exposée dans le chapitre 5. En outre, comme précisé dans l'introduction, les applications pour lesquelles nos contributions sont faites sont des applications temps réel qui demandent des synthèses adaptées au temps réel.

2.1 Synthèse par réorganisation

Nous appelons « synthèse par réorganisation » la famille des synthèses de textures par l'exemple qui génèrent des textures en réorganisant les données de l'exemple, de façon à limiter l'apparition d'artefacts de discontinuité (ou « coutures »), généralement en cherchant à obtenir des voisinages dans la texture de sortie qui correspondent à des voisinages dans l'exemple. L'intuition de ces synthèses est que l'exemple d'entrée contient une certaine forme de répétition statistique de motifs similaires, et que leur reproduction dans la texture de sortie permet d'obtenir une texture cohérente par rapport au contenu de l'exemple. Ces synthèses se basent sur l'hypothèse que l'exemple est la réalisation d'un champ de Markov, c'est-à-dire un processus aléatoire dont chaque texel dépend

uniquement d'un voisinage de taille variable autour de ce texel.

Les premiers exemples de telles synthèses sont les synthèses par texel, qui consistent à copier séquentiellement des texels tirés d'un exemple dans la sortie. Les texels choisis à une étape donnée sont ceux dont le voisinage correspond au mieux dans la sortie déjà générée aux étapes précédentes. L'algorithme tente de minimiser une fonction d'erreur d qui calcule une différence entre une région de l'exemple, et une région de la sortie (pour les texels déjà générés). C'est une forme de synthèse par optimisation où le contrôle est apporté par l'exemple choisi et la taille du voisinage pris en compte, et où l'aléa de la synthèse dépend de l'initialisation et du choix de voisinages, qui peut se faire parmi une liste des n meilleurs candidats déterminés par d et appelés les n plus proches voisins. Dans cette famille, on retrouve la synthèse proposée par Efros et Leung [EL99] (Figure 2.1).

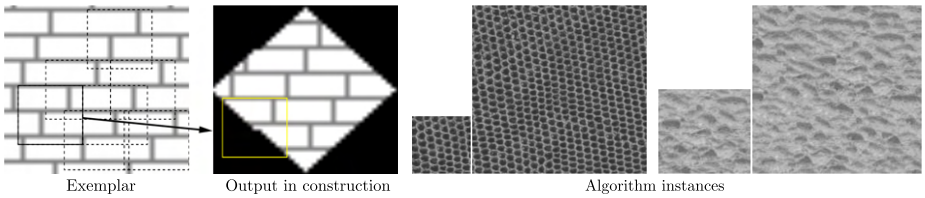


FIGURE 2.1 – Technique de synthèse par pixel présentée par Efros et Leung [EL99]. Le principe (à gauche) est de remplir progressivement la texture de sortie avec des texels choisis lorsque le voisinage dans l'entrée correspond bien au voisinage présent dans la sortie. Le voisinage des textures générées (à droite) correspond rigoureusement à celui de l'exemple.

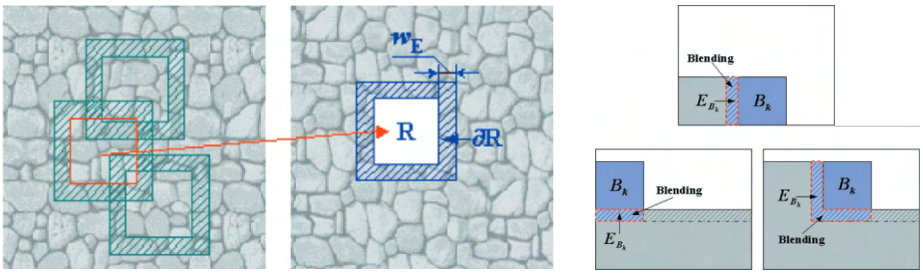
Des améliorations ont été proposées par Wei et Levoy [WL00] pour augmenter la vitesse de la synthèse et rendre les textures de sorties périodiques pour paver une surface sans discontinuités visibles, notamment en effectuant une recherche des plus proches voisins dans des basses résolutions de l'exemple en premier si la taille du voisinage utilisé est grande.

L'avantage de ces synthèses est qu'elles reproduisent bien les voisinages, même lorsque ceux-ci sont sémantiquement complexes, comme la géométrie des contenus de la Figure 2.1 à droite.

Les inconvénients de ces synthèses sont, d'une part, qu'elles génèrent des textures sans créer du contenu inédit, car tous les texels ou régions de la texture de sortie se retrouvent dans l'exemple d'entrée, ce qui limite la variété des contenus en sortie. Le processus itératif de choix des plus proches voisins peut conduire à la divergence des voisinages choisis par rapport à l'exemple et générer du contenu qui s'éloigne de l'exemple (nommé « *garbage* » en anglais). D'autre part, que ce sont des processus discrets, et qu'ils doivent donc compter sur l'interpolation bilinéaire pour texturer une surface de manière continue. De plus,

la taille du voisinage est un paramètre significatif pour déterminer l'apparence de la sortie : un voisinage plus large signifie que l'algorithme trouve des plus proches voisins qui correspondent mieux à l'entrée, mais qui sont moins variés. Pourtant, ces algorithmes ont du mal à reproduire l'organisation globale des textures.

2.1.1 Synthèse par patchs



Exemplar Output with patch Patch blending

FIGURE 2.2 – *Patch-based sampling* présenté par Lin et al. [LLX+01]. Des patchs réguliers sont placés séquentiellement et régulièrement dans la sortie. Les patchs contiennent une marge sur laquelle un mélange est effectué avec les patchs déjà placés. La vue de droite donne des exemples de mélanges possibles.

La synthèse par patchs consiste à construire une texture en collant de larges régions de l'exemple appelées « patchs » tirés d'un exemple, plutôt que des texels individuels. Les premiers exemples de cette famille sont le *patch-based sampling* [LLX+01] et le *quilting* [EF01].

Le *patch-based sampling* consiste à coller un ensemble de patchs carrés de façon régulière (les patchs sont alignés sur une grille régulière dans la sortie) en mélangeant les bords des patchs dans la sortie (Figure 2.2). Le calcul des plus proches voisins est effectué en cherchant le minimum d'une distance sur ces mêmes bords, entre l'exemple et la partie déjà définie de la sortie.

Le *quilting* [EF01], illustré dans la Figure 2.3 reprend le *patch-based sampling* [LLX+01], et l'algorithme « tisse » intelligemment les patchs réguliers au long de leurs bords, de façon à minimiser les discontinuités entre différents patchs. Le contrôle de la synthèse est apporté par l'exemple et par la taille des patchs réguliers tirés, et l'aléa est apporté par la position à laquelle ces patchs sont pris dans l'exemple.

Un exemple similaire est le *graphcut* [KSE+03], illustré dans la Figure 2.4, qui ramène la synthèse de textures à une coupe de graphe, où les noeuds du graphe sont les texels dans la marge entre des patchs irréguliers. La texture de

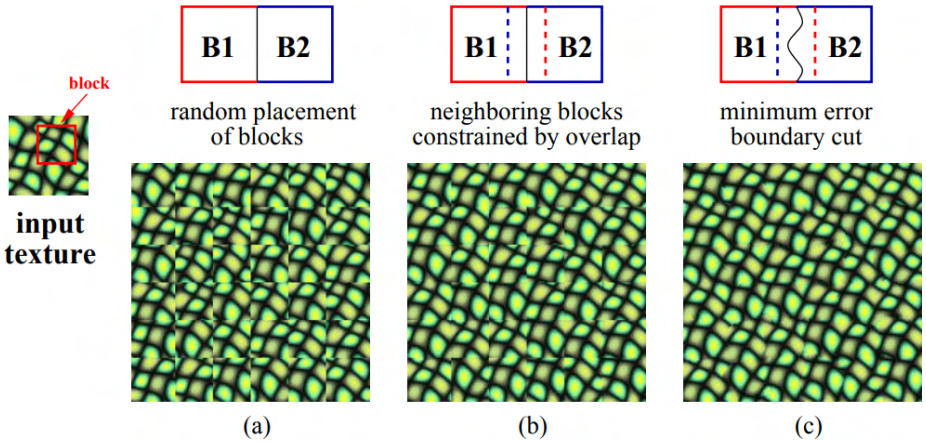


FIGURE 2.3 – *Quilting* présenté par Efros et Freeman [EF01]. Des patchs réguliers sont tirés aléatoirement dans l'exemple en (a) ; de meilleurs patchs sont cherchés pour minimiser l'erreur sur leur chevauchements en (b), et les patchs sont découpés sur leurs marges de façon à minimiser l'écart entre les patchs, et ainsi éviter au mieux les coutures visibles en (c).

sortie est initialisée par des patchs de formes différentes contenant des régions de l'exemple à des positions différentes, qui apportent l'aléa de la synthèse. Puis, avec un algorithme de *graph cut*, les coupes et les valeurs d'erreur causées par les coupes sont retenues et prises en compte pour optimiser la forme des patchs, supprimer les patchs irréguliers causant des erreurs trop grandes, et en rajouter de nouveaux. La technique est donc plus complexe que le *quilting*, mais plus générale, dans le sens où les patchs peuvent prendre des formes plus variées. Cette technique permet d'éviter l'apparition d'une grille quasi-régulière de coutures et de potentiellement trouver de meilleures coupes.

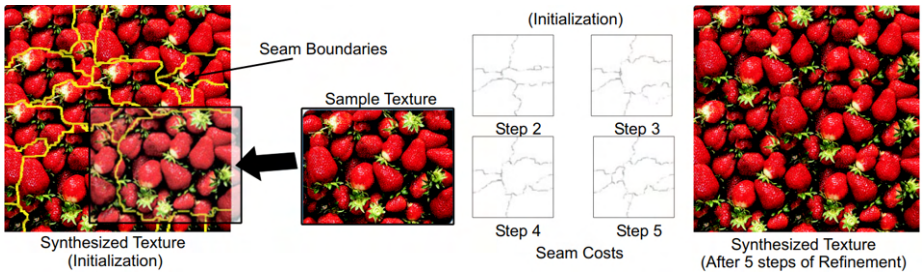


FIGURE 2.4 – *Graphcut* présenté par Kwatra et al. [KSE+03]. Des patchs irréguliers contenant différentes régions sont initialisés aléatoirement, puis optimisés par *graph cut*.

Toutes ces synthèses sont conçues pour être plus rapides que les algorithmes

de synthèse par texel, mais elles restent lentes et inadaptées au temps réel, soit parce qu'elles ne sont pas parallélisables, soit parce qu'elles demandent de nombreux accès mémoire. De plus, la taille des motifs préservés est déterminée par la taille des patchs ; mais comme pour la synthèse par texels, la régularité des motifs n'est pas nécessairement préservée (voir Figure 2.5).

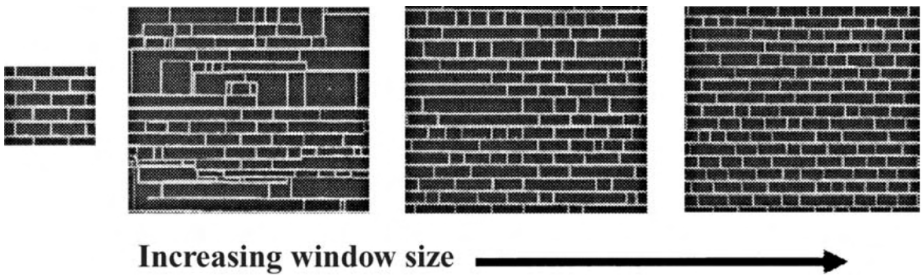


FIGURE 2.5 – Figure de Liu et al. [LTL05] montrant que l'augmentation de la taille du voisinage des synthèses par texels ou l'augmentation de la taille des patchs des synthèses par patchs ne suffit pas à préserver l'organisation globale des textures.

2.1.2 Synthèse par pavage apériodique

Le pavage périodique consiste à agencer des copies d'une seule tuile périodique les unes à côté des autres pour paver une surface. Cette technique est encore largement utilisée aujourd'hui, car peu coûteuse et simple à mettre en œuvre, mais elle a l'inconvénient de générer des artefacts de répétitions et des artefacts d'alignements qui peuvent casser l'immersion de l'utilisateur.

La synthèse par pavage apériodique a pour but de remédier à ce problème, et forme un ensemble de synthèses qui ont pour but d'agencer des régions de texture ensemble de façon à paver une surface sans périodicité, grâce à différentes « tuiles » de texture qui se raccordent naturellement bien lorsqu'elles sont mises les unes à côté des autres. Elles s'appuient souvent sur des techniques issues de la synthèse par patchs, soit pour extraire les régions, soit pour générer les tuiles. Nous distinguons la synthèse par pavage apériodique à tuiles régulières et la synthèse par pavage apériodique à tuiles irrégulières. La qualité de ces synthèses est conditionnée par la façon dont les contenus sont calculés, et il peut être possible de préserver l'organisation globale des textures régulières. Tout comme les synthèses par réorganisation, les synthèses par pavage sont des processus discrets, mais contrairement à ces derniers, ils s'adaptent bien aux applications temps réel : un enjeu important de ce genre de synthèses est alors la capacité de les filtrer en temps réel, qui n'est pas triviale.

Pavage apériodique avec tuiles régulières

Cette synthèse consiste à paver la surface avec un ensemble de tuiles de formes régulières, et nous nous intéressons particulièrement aux tuiles de forme carrées. Ce sont des textures pouvant être mises les unes à côté des autres sans discontinuités visibles.

En utilisant plusieurs tuiles, une part d'aléa est ajoutée par rapport au pavage périodique par la synthèse de Stam [Sta97], fondée sur les tuiles (ou carreaux) de Wang. Un identifiant de couleur est donné à chaque bord de chaque tuile ; deux tuiles peuvent être agencées l'une à côté de l'autre lorsque l'identifiant de leur bord commun est le même. Les tuiles sont générées par bruit procédural, et de sorte à garantir une continuité entre les tuiles lorsque leur bord correspond.

Une synthèse par l'exemple a été proposée par Cohen et al. [CSHD03] pour pré-calculer un ensemble de tuiles de Wang [JR15]. La synthèse utilisée par Cohen et al. est le *quilting* [EF01] présenté dans la section précédente. Nous montrons un exemple de cette synthèse en Figure 2.6.

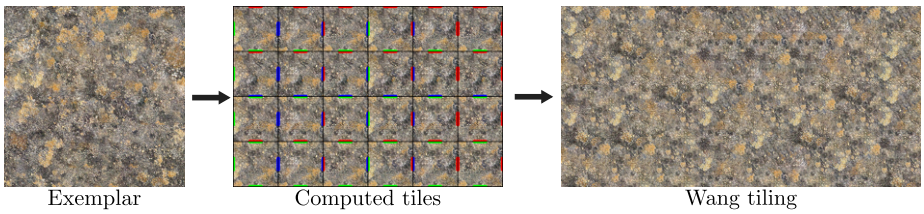


FIGURE 2.6 – Pavage de Wang tel que conçu par Cohen et al. [CSHD03]. Les tuiles sont synthétisées par *quilting* [EF01].

Plus tard, Wei [Wei04] montre comment adapter cet algorithme à des conditions de rendu temps réel, en montrant notamment d'une part qu'il est possible de calculer la synthèse rapidement et à la volée, et d'autre part qu'il est possible de la filtrer. Wei montre notamment une formule analytique et simple à calculer permettant de déterminer la couleur d'un texel de la texture de sortie sans connaître ses voisins. Comme les tuiles sont calculées à partir d'un exemple via le *quilting*, une synthèse relativement lente et inadaptée au rendu à la volée, il est impossible de filtrer la sortie du pavage de Wang en pré-filtrant l'exemple. Wei pré-filtre l'ensemble des tuiles calculées en les disposant ensemble dans un atlas (une agrégation de sous-textures dans une seule texture), de façon à ce que les bords de chaque tuile dans l'atlas correspondent vis-à-vis de leurs identifiants de couleur (Figure 2.7).

D'autres algorithmes de pavage basées sur les tuiles de Wang ont plus tard vu le jour. Par exemple, Ng et al. [NWT+05] calculent les tuiles à partir du graphcut plutôt que du quilting ; Lagae et al. [LD06] calculent les contraintes sur les coins

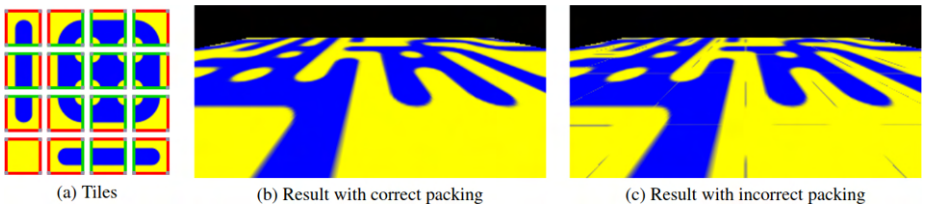


FIGURE 2.7 – Filtrage du pavage de Wang. Les différentes tuiles carrées sont disposées dans un atlas de façon à ce que les identifiants de couleurs des bords communs soient les mêmes pour éviter les artefacts à basse résolution. Figure de Wei [Wei04].

des tuiles pour préserver les diagonales ; Xue et al. [XZJ+07] calculent des tuiles avec uniquement un identifiant de couleur par axe, et à partir d'une synthèse par texels.

L'avantage de ces synthèses est qu'elles sont compatibles avec des applications temps réel et permettent de créer facilement de la variété ; cependant, cette variété n'est pas nécessairement suffisante pour éviter de nombreuses répétitions, qui dépend du nombre de tuiles ; or, augmenter le nombre de tuiles implique une augmentation du coût mémoire de la synthèse. De plus, les répétitions sont fréquentes et des alignements restent visibles, ce qui casse l'immersion.

Pavage apériodique avec tuiles irrégulières

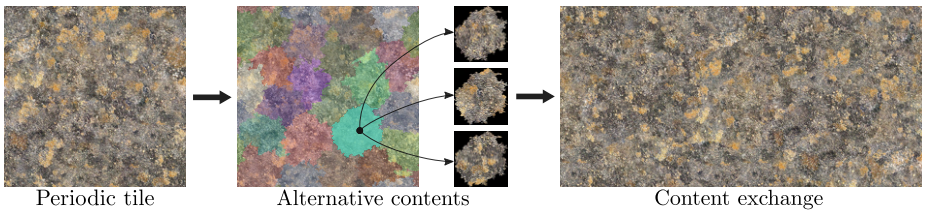


FIGURE 2.8 – Pavage apériodique avec tuiles irrégulières effectué par échange de contenu [VSLD13]. Une texture périodique est répétée, mais un contenu aléatoire est attribué à chaque patch parmi les contenus pré-calculés. Les contenus alternatifs sont obtenus par translation dans la texture périodique.

Ce pavage, illustré en Figure 2.8, consiste à paver une surface grâce des tuiles de forme et de taille irrégulières (là où le pavage précédent utilise des tuiles de forme carrée, c'est-à-dire de forme constante et de tailles identiques). La première synthèse par pavage à tuiles irrégulières connue est l'échange de contenus [VSLD13], qui consiste à répéter une texture périodique sur une surface, en modifiant à la volée certaines régions de la texture. Les tuiles sont appelées

les « patchs », et leurs valeurs sont appelées les « contenus ». Ces patchs et ces contenus sont déterminés aléatoirement en fonction d'une mesure ; Vanhoey et al. [VSLD13] proposent de découper les patchs aux endroits où les bords sont francs, et de choisir les contenus avec une mesure perceptuelle. Le contrôle est alors apporté uniquement par le choix de l'exemple ; l'aléa est apporté, en temps réel, par le contenu choisi pour chaque patch. Nous montrons un exemple de cette technique en Figure 2.8.

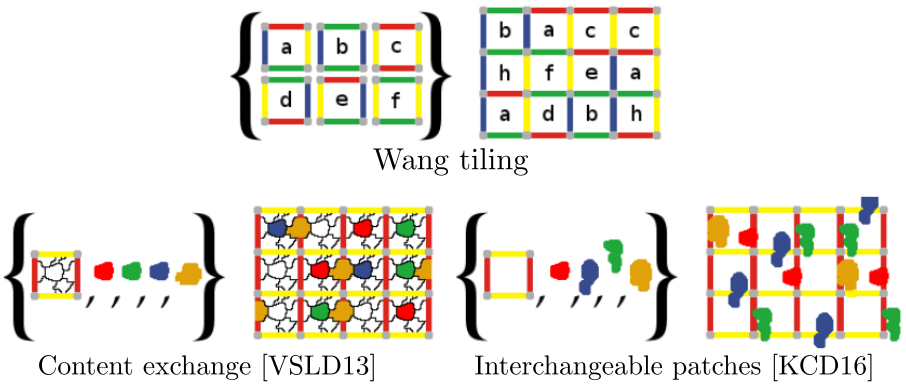


FIGURE 2.9 – Principe du pavage Wang comparé au pavage par échange de contenu [VSLD13] et patchs interchangeables [KCD16]. Les patchs sont visualisés par des régions colorées.

Plus tard, Kolář et al. [KCD16] développent une synthèse semblable où les patchs ne recouvrent pas entièrement la texture périodique, mais peuvent se chevaucher. Pendant le rendu, le contenu d'un ensemble aléatoire de patchs disjoints sont remplacés par des contenus alternatifs. Le contrôle et l'aléa sont apportés de la même façon que pour l'échange de contenus. Ces deux synthèses sont illustrées en Figure 2.9 et leurs résultats sont comparés en Figure 2.10.

L'avantage de ces pavages est que la variété qu'ils génèrent est accrue par rapport au pavage apériodique à tuiles régulières : ils reviennent à un pavage apériodique à tuiles régulières dont le nombre de tuiles différentes augmente exponentiellement par rapport à la complexité de la découpe en patchs ; par exemple, dans le cas de l'échange de contenus [VSLD13], étant donné un nombre p de patchs et un nombre c de contenus par patch, le nombre de tuiles possibles correspond à c^p. L'inconvénient de ces synthèses est que des contenus alternatifs qui peuvent bien remplacer les contenus par défaut sont parfois difficiles à trouver et peuvent causer des coutures visibles dans la sortie. Les répétitions sont moins souvent alignées, mais restent fréquentes : pour l'échange de contenus, entre deux répétitions de l'exemple dans la sortie, la probabilité de retrouver exactement le même contenu sur la même tuile irrégulière est de $\frac{1}{c}$. Enfin, les tuiles n'étant pas connues avant le rendu, le filtrage est particulièrement difficile ; nous présentons notre filtrage de l'échange de contenus en tant qu'une de

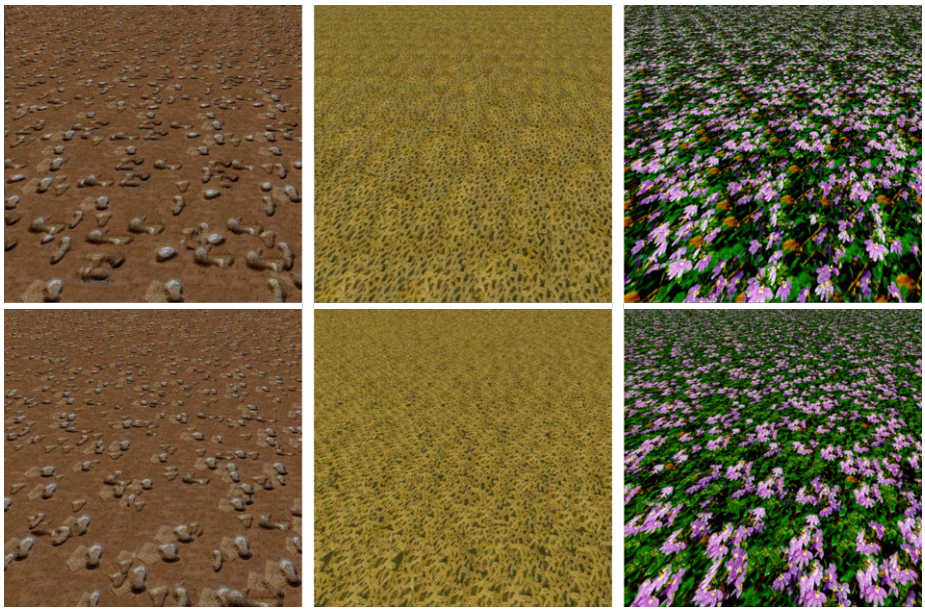


FIGURE 2.10 – Pavage apériodique de Wang [CSHD03] avec 16 tuiles (haut) comparé à l'échange de contenus [VSLD13] avec 3 contenus par patch (bas). L'échange de contenus limite fortement les répétitions alignées. Figure de Vanhoey et al. [VSLD13].

nos contributions principales dans le chapitre 5, mais le filtrage de la synthèse par patchs interchangeables de Kolář et al. [KCD16] reste un problème ouvert.

2.2 Bruit procédural

Le bruit procédural est utilisé par une famille de synthèses de textures définissant des textures à l'aide d'une fonction aléatoire. Parmi les termes de cette fonction, certains sont des paramètres servant à la contrôler, et d'autres créent l'aléa du résultat.

La première occurrence d'un bruit procédural est le bruit de Perlin [Per85], qui propose une synthèse sans exemple, calculée en deux parties. La première partie consiste à calculer une fonction noise($\mathbf{x}$) en tirant des vecteurs de direction aléatoires sur une grille régulière (voir Figure 2.11). Étant donné un point $\mathbf{x}$, noise($\mathbf{x}$) est égal à l'interpolation du produit scalaire entre $\mathbf{x}$ et les quatre coins de la cellule à laquelle $\mathbf{x}$ appartient, en ajustant le résultat pour qu'il soit continu au moins en dérivée première à travers différentes cellules. Pour augmenter la

gamme de fréquences, une deuxième partie consiste à calculer

$$\sum_{l=0}^{L} \frac{\text{noise}(2^l \mathbf{x})}{2^l} \tag{2.1}$$

et permet d'apporter un certain contrôle fréquentiel en spécifiant le nombre d'octaves L (puissances de 2). L'aléa est entièrement géré par les vecteurs de la grille régulière.

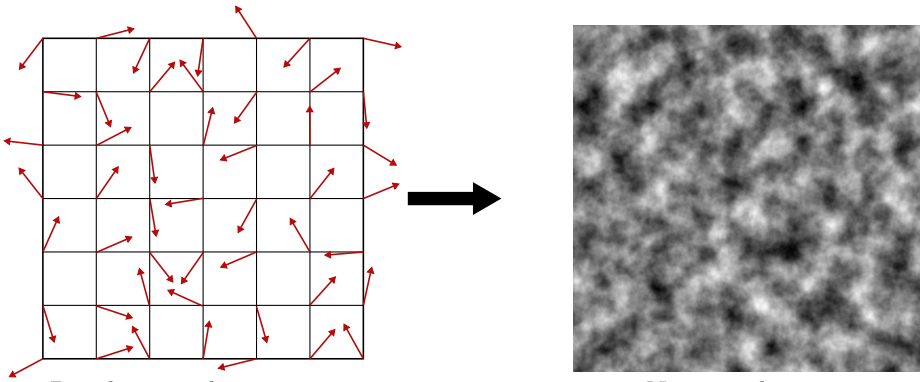


FIGURE 2.11 – Bruit de Perlin (partie droite de la figure de Perlin 1985 [Per85]). Une carte de gradients aléatoires et l'équation (2.1) est utilisée pour synthétiser une texture.

Le bruit de Perlin peut générer diverses textures, mais ses statistiques sont toujours invariantes par translation (stationnaires) et par rotation (isotropes), et ne permet pas beaucoup de contrôle, limitant fortement la classe de textures pouvant être représentées avec le bruit de Perlin. Le bruit de Perlin a été amélioré en 2002 [Per02] pour qu'il soit plus rapide et mieux adapté aux espaces de plus grandes dimensions (la version de base nécessitant de calculer une interpolation avec 2^d coins, où d est la dimension de l'ensemble des indices de la texture), mais le principe reste le même.

2.2.1 Convolution parcimonieuse

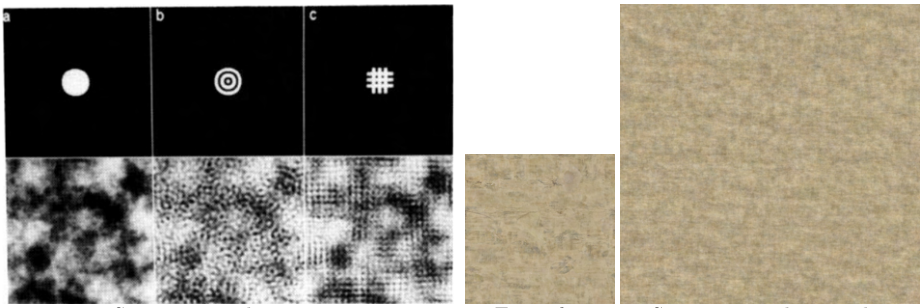
Dans ce manuscrit, nous nous intéressons surtout à la convolution parcimonieuse, qui est exploitée pour la synthèse cyclostationnaire dans le chapitre 4. La convolution parcimonieuse a été introduite par Lewis [Lew84], qui reprend notamment la création de textures à l'aide d'un spectre et de la transformée de Fourier, utilisés pour la génération de motifs aléatoires dans des travaux présentés par Saupe [Sau88]. De nombreux bruits sont basés sur ce modèle, qui consiste à effectuer la convolution d'une fonction à des positions aléatoires et de façon

à limiter le nombre d'additions nécessaires pour que le résultat converge vers la forme voulue, d'où le terme « parcimonieux ». Lagae et al. [LLC+10] proposent la formulation explicite de la convolution parcimonieuse avec un noyau k et un processus de Poisson γ :

$$N(\mathbf{x}) = \int \gamma(\mathbf{u})\, k(\mathbf{x} - \mathbf{u}). \tag{2.2}$$

Le processus de Poisson est notamment utilisé à la place d'un bruit blanc, car il permet d'obtenir une meilleure couverture du domaine avec un tirage de positions aléatoires discrètes, tout en étant asymptotiquement équivalent à celui-ci.

Spot noise. Dans cette famille de bruits, le *spot noise* (Figure 2.12) a été créé par Van Wijk [vW91] et propose de synthétiser des textures avec un bruit stationnaire et anisotrope qui simule directement la convolution parcimonieuse.



Spot noise with patterns | Exemplar | Spot noise with exemplar

FIGURE 2.12 – *Spot noise* présenté par Van Wijk [vW91] à gauche, où des motifs sont convolués par un bruit de Poisson pour créer des textures. *Spot noise* par l'exemple à droite, généré par la synthèse de Galerne et al. [GGM11]

Le principe est de calculer N à partir d'un noyau explicite ou implicite k à valeurs centrées en 0, en additionnant le noyau à des positions déterminées (ce qu'on appelle un « tir » – *shot* en anglais) par un processus stochastique de Poisson. Cette technique de synthèse permet de synthétiser la texture d'une surface non bornée à la volée. Le contrôle de la synthèse s'effectue entièrement via le noyau utilisé dans la convolution parcimonieuse, et l'aléa est apporté par le processus de Poisson. Le degré de contrôle est accru par rapport au bruit de Perlin, et l'algorithme permet notamment de générer des textures anisotropes (comme le troisième motif de la Figure 2.12), contrairement au bruit de Perlin.

Discrete spot noise. Galerne et al. [GGM11] appellent le *spot noise* discret (DSN) l'implémentation du *spot noise* avec un noyau discret et sur des positions

discrètes. Le *spot noise* discret dit « d'ordre n » sur l'ensemble d'indices X de la texture est [GGM11]

$$N(\mathbf{x}) = \sum_{p=1}^{n} k(x - Y_p), \quad x \in X \tag{2.3}$$

où Y_p sont des variables aléatoires identiquement distribuées et uniformes ayant comme support X. Galerne et al. [GLM17] montrent qu'il est possible d'utiliser $\mathbb{R}^2$ comme support de Y_p et de x, en interpolant bilinéairement k.

Galerne et al. [GGM11] montrent des propriétés de sa version asymptotique (avec un nombre de tirs qui tend vers l'infini) appelée *asymptotic discrete spot noise* (ADSN), qui est un modèle de bruit procédural correspondant à la convolution d'un noyau k par un bruit blanc. Ils montrent notamment qu'il est la réalisation d'un processus Gaussien et stationnaire, dont l'amplitude de Fourier de l'ADSN correspond à l'amplitude de Fourier du noyau multipliée point par point par un bruit de Rayleigh : le *spot noise* altère donc le spectre de puissance du noyau dans la sortie. De plus, ils montrent la possibilité d'utiliser un exemple à la place d'un motif simple pour synthétiser des textures de taille non bornée, en adoucissant les bords de l'exemple pour éviter l'apparition d'artefacts linéaires. Enfin, ils montrent que la moyenne et la fonction d'autocovariance de l'exemple est préservée par ce processus. Nous adaptons l'algorithme du *spot noise* discret et le modèle de l'ADSN au contexte cyclostationnaire dans le chapitre 4.

Texton noise. Dans la même famille, le *texton noise* [GLM17] consiste à calculer le *spot noise* avec une version « résumée » de l'exemple appelé « texton », et qui contient les informations fréquentielles importantes, afin de réduire le nombre de tirs nécessaires pour que le résultat converge. Cette synthèse requiert en moyenne 30 accès mémoire au texton pour calculer la valeur d'un texel de la sortie avec un degré de convergence suffisant, ce qui fait que son utilisation peut être considérée pour des applications temps réel, mais reste relativement coûteuse par rapport à d'autres synthèses (notamment les synthèses par pavage apériodique). L'inconvénient et que la sortie perd quelques détails de l'exemple à cause de ce « résumer ».

2.2.2 Bruit de Gabor

Gabor noise. Le bruit de Gabor [LLDD09, LLD11] est un autre type de convolution parcimonieuse alliant le domaine spatial et le domaine spectral (de Fourier). Un spectre analytique est défini grossièrement comme un ensemble de lobes, qui sont des intervalles de fréquences et d'orientations dans le domaine de Fourier bi-dimensionel. Le principe du bruit de Gabor est de mélanger des noyaux de Gabor, de façon similaire au *spot noise*, où chaque noyau de Gabor

est la multiplication d'une enveloppe Gaussienne et d'une harmonique, dont la fréquence et l'orientation est tirée aléatoirement dans l'ensemble des lobes du spectre analytique. L'utilisation du spectre apporte un contrôle spectral à l'utilisateur, et n'utilise pas de données échantillonnées discrètement en entrée, contrairement au *spot noise* discret. Le bruit de Gabor est illustré sur la première grande ligne de la Figure 2.13. Nous adaptons le bruit de Gabor au contexte cyclostationnaire dans le chapitre 4.

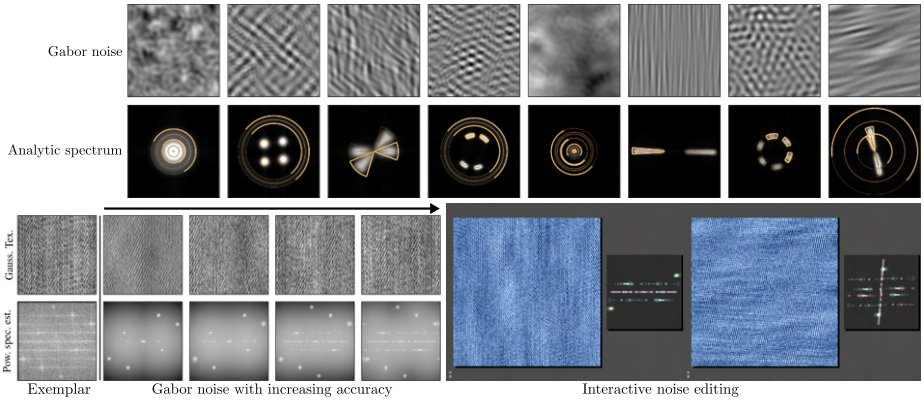


FIGURE 2.13 – Bruit de Gabor présenté par Lagae et al. [LLDD09] et bruit de Gabor par l'exemple présenté par Galerne et al. [GLLD12]. Des bruits de Gabor sont obtenus par des spectres analytiques (haut). Le spectre d'un exemple d'entrée peut être grossi en un spectre analytique à précision variable et utilisé pour la synthèse (gauche). Le spectre analytique peut être modifié interactivement pour créer intuitivement des motifs (droite).

Gabor noise by example. Le bruit de Gabor par l'exemple [GLLD12] consiste à utiliser le spectre de l'exemple, à le subdiviser en un ensemble de lobes, et à utiliser le spectre analytique résultant pour la synthèse. La précision du bruit de Gabor par l'exemple ainsi que sa complexité algorithmique varient alors selon le nombre de lobes utilisés. La synthèse par l'exemple du bruit de Gabor est illustrée sur la seconde grande ligne de la Figure 2.13.

2.2.3 Bruit à phase aléatoire

Random phase noise. Holten et al. [HWM06] présentent un algorithme de synthèse spectrale basé sur la randomisation de la phase d'une texture dans le domaine de Fourier. Le modèle sous-jacent est explicité dans Galerne et al. [GGM11], qui l'appellent *random phase noise (RPN)*, ou bruit à phase aléatoire. Le modèle du *RPN* consiste à fournir un motif discret dans le domaine de Fourier appelé spectre de puissance (*power spectral density* ou *PSD*) qui est

le carré de l'amplitude dans le domaine de Fourier, avant de calculer la transformée de Fourier inverse de l'amplitude avec une phase aléatoire. L'aléa de la synthèse est donc entièrement apporté par la phase.

Cet algorithme de synthèse est compatible avec la synthèse par l'exemple, en calculant le spectre d'un exemple via sa transformée de Fourier et en attribuant à ce spectre une phase aléatoire. Le spectre est également la transformée de Fourier de la fonction d'autocovariance ; ainsi, la fonction d'autocovariance de l'exemple est préservée dans la sortie par définition. De façon similaire au *spot noise*, la synthèse est stationnaire ; de plus, elle peut prendre une forme Gaussienne, mais cela dépend du spectre utilisé. Le contrôle de la synthèse est entièrement apporté par le spectre (fourni ou calculé depuis l'exemple) ; l'aléa est apporté par la phase aléatoire. Pour les textures à plusieurs canaux, il faut en plus préserver l'écart des phases entre chaque canal, ce qui permet de préserver les fonctions d'autocovariances croisées, dont la préservation est indispensable pour conserver au mieux l'apparence d'un exemple.

Cette synthèse propose une méthode alternative de contrôle au *spot noise* avec l'instanciation via un spectre donnant les informations fréquentielles du résultat de la synthèse. Contrairement au *spot noise*, elle a comme propriété, par définition, de conserver entièrement le spectre de puissance de l'exemple dans la sortie. En revanche, toujours contrairement au *spot noise*, cet algorithme ne permet pas d'étendre la taille de la sortie au-delà de la taille du spectre sans que la sortie ne soit périodique au-delà de la taille de l'exemple. Le bruit à phase aléatoire est illustré dans la Figure 2.14. Nous adaptons le bruit à phase aléatoire au contexte cyclostationnaire dans le chapitre 4.

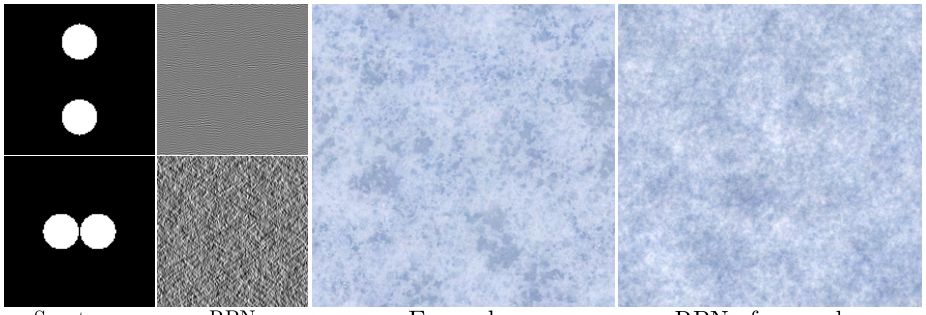
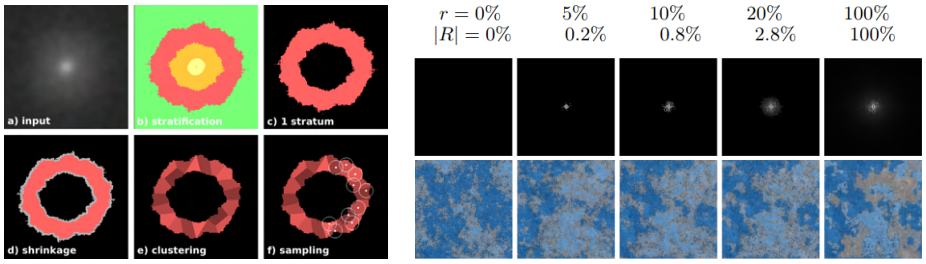


FIGURE 2.14 – Random phase noise de Galerne et al. [GGM11]. Sans l'exemple, une phase aléatoire est attribuée à un spectre. Avec un exemple, c'est le spectre de l'exemple qui est utilisé.

Local random-phase noise. Gilet et al. [GSV+14] proposent des améliorations directes au *RPN*. Le bruit est défini comme un mélange de bruits locaux à phase aléatoire disposés sur une grille régulière. La synthèse par l'exemple est

optimisée en subdivisant le spectre de l'exemple en strates et lobes de fréquences dans chaque strate le spectre (Figure 2.15, gauche), et la synthèse non bornée est rendue possible grâce à la grille régulière.



Spectrum simplification Phase conservation

FIGURE 2.15 – Local random-phase noise présenté par Gilet et al. [GSV+14]. Un spectre est simplifié en strates de fréquences, puis en lobes de fréquences (gauche). La phase d'une certaine quantité r d'énergie du spectre (sur une aire de $|R|$) peut être conservée pour préserver l'apparence de l'exemple (droite).

2.2.4 Limite du bruit procédural stationnaire et Gaussien

Les synthèses par convolution parcimonieuse ont l'avantage de pouvoir générer des textures non bornées, d'être compatibles avec les applications temps réel, et de bénéficier d'un certain degré de contrôle, soit par un contrôle spectral, soit par un contrôle spatial avec des motifs simples ou des exemples. Le résultat d'un bruit procédural basé sur des convolutions (*spot noise*, *texton noise*, bruit de *Gabor*) et du bruit à phase aléatoire tend vers une forme stationnaire et Gaussienne, car elle implémente un processus stationnaire Gaussien (ou quasi-Gaussien dans le cas du bruit à phase aléatoire) dont les statistiques sont contrôlées dans leur intégralité par les deux premiers moments, qui sont la moyenne et la fonction d'autocovariance (ou, de façon équivalente, le spectre de puissance) [LLC+10]. De ce fait, dans la synthèse par l'exemple, les textures générées sont similaires à l'exemple si celui-ci correspond à la réalisation d'un processus Gaussien et stationnaire, auquel cas ses statistiques d'ordre 1 (probabilités pour chaque position) et d'ordre 2 (probabilités jointes entre deux positions définies par un écart fixe) sont préservées dans la sortie, et certaines propriétés liées à son apparence (histogramme, corrélations) sont également préservées. S'il ne correspond pas à la réalisation d'un processus Gaussien et stationnaire, l'apparence de la sortie peut être fortement altérée, car ces synthèses ne sont pas conçues pour générer des motifs dont les statistiques sont non Gaussiennes ou non stationnaires, ce qui limite fortement la gamme de textures pouvant être traitées.

Nous voyons dans la section 3 qu'il existe des processus non stationnaires, dont les statistiques varient spatialement, et des processus non Gaussiens, dont

les deux premiers moments (moyenne et fonction d'autocovariance ou spectre de puissance) ne suffisent pas à définir entièrement le processus. Pour générer des motifs inédits tout en s'appuyant sur les propriétés des bruits procéduraux, des méthodes s'appliquent à éliminer la contrainte des statistiques stationnaires, et d'autres, à éliminer celle des statistiques Gaussiennes.

2.2.5 Bruit procédural non stationnaire

La stationnarité contraint les statistiques de la réalisation d'un bruit procédural à être invariantes en tout point de la surface. Il est alors impossible de synthétiser naturellement et correctement des textures dont les statistiques varient spatialement, comme des textures régulières (voir la section 2.3) ou des textures sémantiquement complexes (voir *bi-layer noise* [GSDC17]).

Contrôle du *spot noise*. Des alternatives au *spot noise* ont été proposées pour générer des motifs non stationnaires. Pavie et al. [PGDG16] proposent d'apporter un degré de contrôle au processus de Poisson utilisé pour la synthèse. Ils contrôlent les positions tirées par un champ de densité (périodique sur la Figure 2.16), ce qui permet de concentrer les mélanges à certains endroits de la texture de sortie. Les motifs sont des compositions d'ellipsoïdes Gaussiennes, dont la formulation analytique garantit la continuité du motif. Cavalier et al. [CGG19] améliorent cette synthèse en proposant notamment un filtrage analytique, rendant la synthèse viable pour le rendu temps réel, et une formulation des ellipsoïdes qui soit plus rapide à calculer.

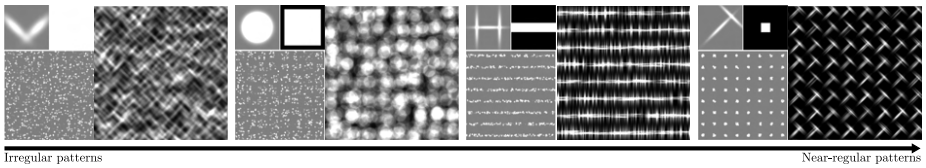


FIGURE 2.16 – Algorithme de synthèse illustrée par Pavie et al. [PGDG16]. Sur chaque vue, à gauche : motif composé d'ellipsoïdes Gaussiennes et son champ de densité périodique (haut), et tirage des positions (bas) ; à droite : motif synthétisé. Un degré de contrôle est ajouté au processus de Poisson utilisé par le *spot noise* grâce à un champ de densité périodique qui permet d'ajouter des variations spatiales aux motifs.

Le problème de cette synthèse est qu'elle ne fonctionne bien que pour des motifs qui sont des agencements d'ellipsoïdes Gaussiennes. Pour cette raison, elle n'est pas facilement adaptable à la synthèse de textures à plusieurs canaux, et la synthèse par l'exemple est difficile car il faut transformer un exemple en un agencement d'ellipsoïdes Gaussiennes. Néanmoins, l'idée d'obtenir des

statistiques variant spatialement en contrôlant le processus de distribution des positions, qui contrôle l'aléa de la synthèse, est une idée intéressante que nous reprenons dans nos travaux.

Contrôle du bruit à phase aléatoire. Pour synthétiser une plus large gamme de textures, Gilet et al. [GSV+14] agissent sur la phase aléatoire du *RPN*, en figeant la partie des phases pour lesquelles l'amplitude est la plus forte, afin de préserver les corrélations entre les fréquences les plus significatives dans le calcul de la texture de sortie (Figure 2.15, droite). Cependant, cette synthèse revient à faire tendre la texture de sortie vers une texture périodique en fonction de la quantité de phases fixes, et limite ainsi la variété des textures générées.

Bi-layer noise Guingo et al. [GSDC17] proposent le *bi-layer noise*, qui consiste à séparer une texture hétérogène marquée par de larges régions à bord forts en deux couches. Une couche décrit l'organisation spatiale globale de la texture, et une couche décrit l'ensemble des bruits homogènes qui composent la texture. La synthèse consiste alors à synthétiser des masques qui représentent les différentes régions homogènes, et à synthétiser ces régions homogènes grâce à un bruit procédural. De cette façon, les textures synthétisées sont hétérogènes, et ont notamment des statistiques qui varient spatialement. Cette synthèse peut être instanciée avec un exemple en identifiant les masques et les différentes régions homogènes qui composent la texture, et en synthétisant la première couche grâce à une synthèse par patchs ou par pavage (pavage apériodique à tuiles irrégulières par échange de contenus [VSLD13] sur la Figure 2.17).

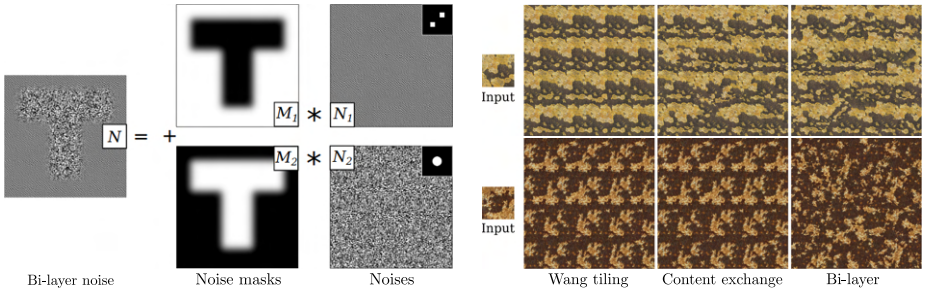


FIGURE 2.17 – Algorithme de synthèse présentée par Guingo et al. [GSDC17]. Des masques sont utilisés pour représenter des régions de différents bruits. Un exemple hétérogène est synthétisé en synthétisant ses masques avec une synthèse par pavage et ses régions homogènes avec un bruit procédural. Par rapport au pavage de Wang et à l'échange de contenus, la synthèse (*bi-layer*) réduit fortement les alignements et les répétitions.

Le problème de cette synthèse est que l'identification de ces régions peut être difficile, car un exemple est potentiellement la réalisation d'un très grand

ensemble de bruits procéduraux ; de plus, la transition entre différentes régions n'est pas nécessairement la réalisation d'un mélange linéaire de bruits comme elle l'est considérée dans l'article.

2.2.6 Bruit procédural non Gaussien

Les statistiques Gaussiennes contraignent les valeurs générées par un bruit procédural à suivre une loi normale multivariée, et correspondent à un ensemble de valeurs centrées en une (ou plusieurs, dans le cas non stationnaire – voir section 3.3) moyenne, formant une apparence particulière.

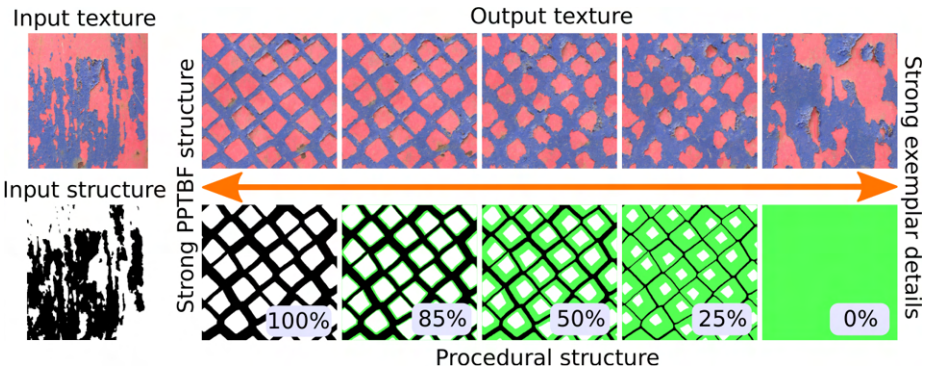


FIGURE 2.18 – Algorithme de synthèse semi-procédurale présentée par Guehl et al. [GAD+20]. Un masque est synthétisé au travers d'un processus de génération de points très général. Le contenu du masque est rempli avec une synthèse par l'exemple.

Le bruit procédural non Gaussien consiste à utiliser les algorithmes issus du bruit procédural pour synthétiser des motifs dont les statistiques ne sont pas Gaussiennes. Par exemple, le bruit de Worley [Wor96] s'appuie sur les diagrammes de Voronoï pour générer des motifs cellulaires, et Guehl et al. [GAD+20] proposent de synthétiser un masque décrivant des régions via un bruit procédural avant de remplir ces régions via la synthèse par l'exemple (Figure 2.18).

Phasor noise. La synthèse non Gaussienne est plus difficile avec le bruit procédural par convolution parcimonieuse, et les synthèses par l'exemple qui en découlent, car la convolution a tendance à générer naturellement des textures ayant des statistiques Gaussiennes. Le *phasor noise* [TEZ+19] peut générer, via les algorithmes du bruit de Gabor et du bruit à phase aléatoire, des motifs non Gaussiens, via l'intermédiaire d'une manipulation de la phase et d'une fonction de seuillage. Le principe est de faire varier la phase spatialement pour que le spectre de variance (spectre de puissance du signal au carré) ne contienne pas

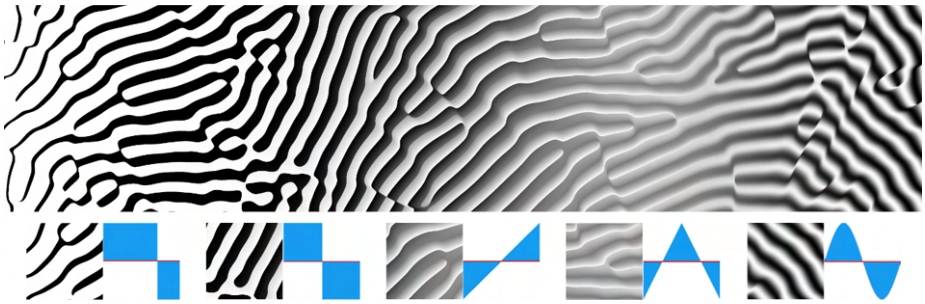


FIGURE 2.19 – *Phasor noise* présenté par Tricard et al. [TEZ+19]. Une fonction de transfert appelée le « profil » est définie pour modifier l'apparence de la réalisation d'un bruit procédural.

de basses fréquences, ce qui se traduit dans le domaine spatial par l'absence de régions « floues ». Une fonction de seuillage appelée le « profil » peut ensuite être appliquée pour modifier l'apparence de la réalisation du bruit procédural. Cette synthèse, illustrée dans la Figure 2.19, permet de générer certains motifs non Gaussiens (notamment à contours francs) à partir de bruit procéduraux Gaussiens. Cependant, elle n'a pas été adaptée à la synthèse par l'exemple, et ne dispose pas de méthode de filtrage de texture rapide.

Transport optimal. Le transport optimal est un modèle mathématique qui permet de transporter une densité de probabilité vers une autre, et a notamment été utilisée pour transporter l'histogramme d'une image vers une autre, par exemple avec les travaux de Dischler et al. [DGF98] et de Papadakis et al. [PPC10].

Cette approche a été adaptée aux synthèses Gaussiennes par Galerne et al. [GLR17] en calculant analytiquement la carte de transfert vers l'histogramme de la réalisation de la synthèse à partir de la densité de probabilité d'une Gaussienne. Elle est reprise et adaptée au temps réel par Heitz et Neyret [HN18]. Une carte de transfert de l'histogramme de l'exemple E vers la subdivision d'une densité de probabilité Gaussienne analytique est calculée, ainsi que son inverse, qui est stocké dans la carte graphique sous la forme d'une carte de l'ensemble des intervalles Gaussiennes vers les intervalles de l'histogramme de l'exemple. Si l'ensemble des valeurs de la texture est multidimensionnel, alors les ensembles de départ et d'arrivée de la carte peuvent être multi-dimensionnels eux aussi, mais Deliot et Heitz [DH18] optent pour une carte par dimension en décorrélant les valeurs à l'aide d'une PCA. Cette carte de transfert est appliquée à l'exemple texel par texel pour générer un exemple E'. L'exemple E' (dont l'histogramme est Gaussien) est synthétisé pour générer la sortie I', qui a des statistiques Gaussiennes. I' est ensuite transformée en I texel à texel via la carte de transfert inverse ; I conserve donc l'histogramme de l'exemple. Cette

technique est illustrée en Figure 2.20 avec la méthode de synthèse de Heitz et Neyret [HN18].

Cette technique permet de renforcer le contrôle sur la sortie d'une synthèse Gaussienne grâce à un exemple, en préservant son histogramme dans la sortie, qui est un aspect important de l'apparence de la texture. Nous l'adaptons à la synthèse cyclostationnaire dans le chapitre 4.

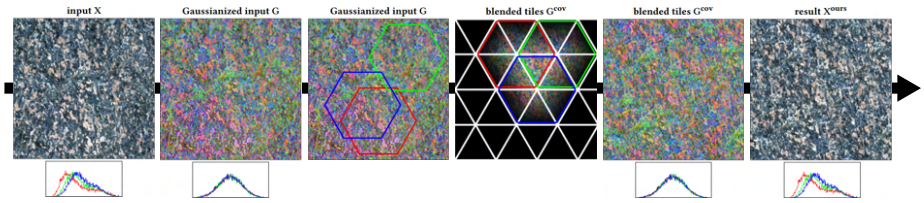


FIGURE 2.20 – Synthèse présentée par Heitz et Neyret. [HN18]. Une version « Gaussienne » de l'exemple est calculée par transfert d'histogramme. Une synthèse de cette version est exécutée en copiant des hexagones sur une grille régulière et en les mélangeant avec les hexagones adjacents. Un transfert inverse d'histogramme est ensuite effectué pour obtenir la texture de sortie.

High performance noise Heitz et Neyret [HN18] proposent un algorithme de synthèse, inspiré de la convolution parcimonieuse mais aussi de la synthèse par pavage apériodique (section 2.1.2), nommé *high performance noise*, qui consiste à mélanger de larges hexagones pris aléatoirement dans l'exemple sur une grille régulière (Figure 2.20). La valeur de chaque texel de la sortie est alors le résultat d'un mélange barycentrique entre trois texels de l'exemple.

Le contrôle est apporté par l'exemple et par la taille des hexagones ; l'aléa est apporté par les positions sur lesquelles les hexagones sont choisis.

La synthèse permet de couvrir le domaine de sortie de façon optimale avec les hexagones, là où la convolution parcimonieuse ne le permet pas avec les noyaux. De plus, elle est particulièrement adaptée au rendu temps réel, car chaque texel ne requiert que trois accès mémoire à l'exemple pour être calculé, chaque texel peut être calculé indépendemment, et le filtrage se calcule grâce à un simple pré-filtrage de l'exemple. Enfin, grâce au transfert d'histogramme présenté précédemment, l'histogramme de l'exemple est conservé dans la sortie.

Limiter le nombre de mélanges à trois peut toutefois causer l'apparition d'artefacts de répétitions, similaires à ceux des synthèses par pavage apériodique de la section 2.1.2. Moins important, elle requiert aussi de stocker une carte de transfert, qui demande des accès texture supplémentaires si elle est stockée sous la forme d'une texture.

Nous adaptons cet algorithme de synthèse au contexte cyclostationnaire dans

le chapitre 4.

2.3 Synthèse de textures quasi-régulières

Liu et al. [LLH04] ont défini les textures quasi-régulières comme des textures régulières (un agencement de motifs identiques dont l'organisation globale est régulière), dont la régularité est altérée en géométrie (les motifs peuvent être déformés) ou/et en valeur (les motifs peuvent être légèrement différents). La synthèse de ces textures est habituellement réservée à des modèles de synthèse par réorganisation ; cependant, comme montré par Liu et al. [LTL05], la préservation naturelle du voisinage proposé par les modèles par réorganisation ne suffit généralement pas à reproduire la régularité des exemples quasi-réguliers (voir Figure 2.5). La synthèse proposée par Liu et al. [LLH04] gère à la fois les altérations géométriques et colorimétriques (en séparant l'intensité lumineuse et l'albedo).

La régularité géométrique d'un exemple peut être représentée par deux vecteurs qui forment une base de $\mathbb{R}^2$ qu'on appelle les « vecteurs de période ». Ces vecteurs sont fournis par l'utilisateur, qui les trace à partir d'un sommet s choisi manuellement dans l'exemple. L'utilisateur est également chargé de donner les autres sommets congruents au sommet s en prenant en compte l'altération de la géométrie de l'exemple. Un champ de distortion donnant l'écart entre la version géométriquement périodique et altérée de l'exemple est calculé et permet de créer une version géométriquement périodique de l'exemple. Pendant la synthèse de l'exemple, le champ de distortion inverse est également synthétisé pour altérer la sortie de la synthèse.

Un exemple régulier peut être divisé en un nombre de tuiles se chevauchant grâce aux vecteurs de période, comme montré par Liu et al. [YCT04]. L'irrégularité colorimétrique d'un exemple régulier peut être représentée comme la tuile moyenne M, plus une variation aléatoire. Liu et al. [LLH04] font l'extraction de ces variations grâce à une PCA de très haute dimension sur les texels de chaque tuile composant l'exemple. Chaque vecteur propre $V_1, V_2, \ldots, V_n$ est une tuile à laquelle est associé une valeur propre $\lambda_1, \lambda_2, \ldots, \lambda_n$. De nouvelles tuiles T peuvent alors être synthétisées comme étant

$$T = M + \sum_{i=0}^{n} w_i V_i, \tag{2.4}$$

où les $w_1, w_2, \ldots, w_n$ sont des nombres aléatoires suivant une loi normale $\mathcal{N}(0, \lambda_i)$. Pendant la synthèse, ces tuiles peuvent être agencées ensemble en mélangeant leur bord ou en utilisant le *quilting* [EF01] pour limiter les discontinuités. La synthèse de Liu et al. [LLH04] est illustrée dans la Figure 2.21.

Bien que son application dans le rendu temps réel n'ait pas été démontrée,

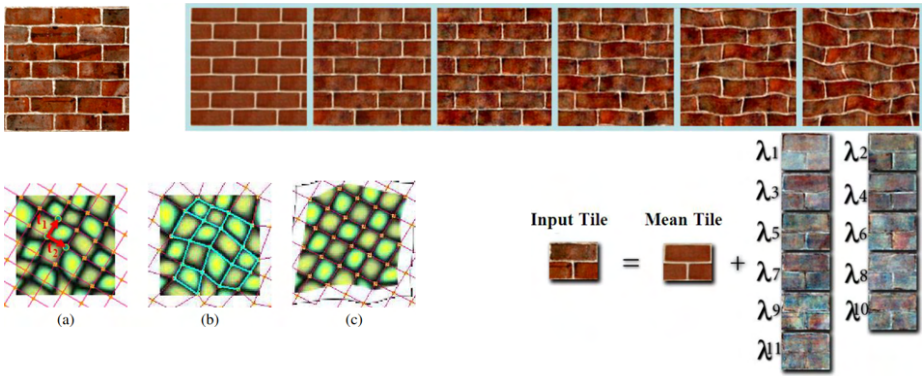


FIGURE 2.21 – Synthèse de textures quasi-régulières présentée par Liu et al. [LLH04]. Leur synthèse traite simultanément les irrégularités géométriques (gauche) et colorimétriques (droite). L'utilisateur fournit un sommet et les vecteurs de période (a), ainsi que les sommets congruents en prenant en compte l'irrégularité géométrique (b) ; un exemple géométriquement régulier peut alors être calculé via le champ de distortion déduit (c). L'irrégularité colorimétrique est exprimée par un agencement de tuiles moyennes auxquelles sont additionnées des variations de valeur. Ces valeurs sont synthétisées en choisissant aléatoirement un point (qui correspond à une tuile) dans l'espace PCA engendré par toutes les variations.

les exemples dont l'irrégularité est uniquement colorimétrique peuvent être synthétisés en temps réel en mélangeant les tuiles calculées. De plus, il n'a pas été montré comment synthétiser un champ de distortion en temps réel.

Ces travaux ont été repris plus tard, par des études comparatives [WHC+06], des méthodes d'analyse permettant de détecter la régularité des textures [PBCL09, DED05, CB11, CB13, LPVV17, LNS+15], et des modifications diverses de la synthèse [NMMK05, HH09, RHE11]. Par exemple, Haindl et al. [HH09] ont noté que de nombreuses textures quasi-régulières sont organisées comme un agencement de tuiles ayant un bord et un intérieur (une vision faisant penser à Guingo et al. [GSDC17]). Ces deux régions différentes sont identifiées ; le bord est synthétisé par patchs et l'intérieur est synthétisé avec un bruit procédural. Cette synthèse est illustrée dans la Figure 2.22.

La difficulté soulevée par ces synthèses réside dans la gestion des variations de couleur. Lorsque les variations de couleur sont fortes, la PCA utilisée par Liu et al. [LLH04] et le bruit stationnaire utilisé et Haindl et al. [HH09] ne parviennent pas à reproduire les détails fins des textures ; de plus, les copies de régions utilisées dans Haindl et al. [HH09], mais aussi dans Nicoll et al. [NMMK05] et Recas et al. [RHE11] génèrent des répétitions visibles et font perdre en variété.

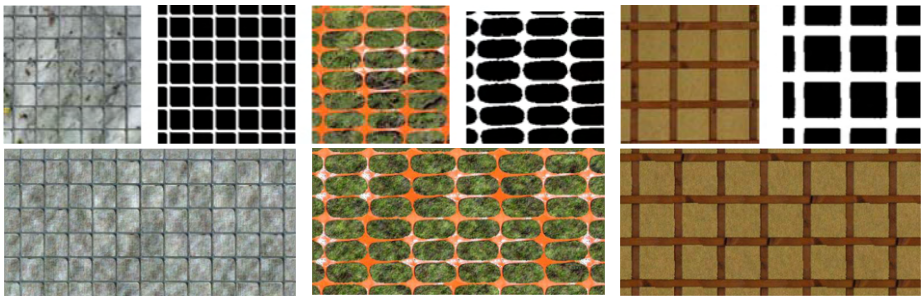


FIGURE 2.22 – Synthèse de textures quasi-régulières présentée par Haindl et Hatka. [HH09]. Sur chaque vue, un exemple (en haut à gauche) est séparé en deux régions, un bord et un intérieur (identifiés en haut à droite). Le bord est synthétisé par patchs et le contenu est synthétisé avec un bruit procédural par l'exemple.

La synthèse par réseaux de neurones [GEB15] peut aussi permettre de traiter des textures quasi-régulières (par exemple avec *Texture networks* [ULVL16] ou *deep correlations* [SCO17]), mais ces synthèses souffrent de la lenteur des algorithmes sous-jacents et de la lourdeur de leur évalution. Les réseaux de neurones forment une nouvelle catégorie de synthèses proches de l'optimisation mais exploitant une formulation convolutive qui n'est pas encore arrivée à maturité. De nouveaux réseaux commencent à faire de la synthèse plus rapide et ont des résultats prometteurs [GRGH18], sans arriver à la fiabilité et compacité des bruits procéduraux. Les contributions que nous présentons dans le chapitre 4 visent à permettre la synthèse de textures très similaires aux textures quasi-régulières, et exploitent notamment les vecteurs de périodes de Liu et al. [YCT04].

Chapitre 3

Processus stochastiques pour la synthèse

3.1 Introduction

Dans ce chapitre, nous présentons des processus stochastiques en lien avec la synthèse de textures. Un processus stochastique est la généralisation d'une variable aléatoire sur plusieurs composantes indexées (typiquement par la position temporelle ou la position spatiale). Ces processus permettent de modéliser des générateurs de signaux aléatoires et sont donc fondamentaux dans le domaine du traitement du signal.

Un processus stochastique Y est un objet mathématique caractérisé par un espace probabilisé $(\Omega, \mathcal{A}, P)$, un ensemble d'indices X et un ensemble d'états S mesurable avec sa σ-algèbre Σ [FF02]. Ω représente l'ensemble des observables (que nous notons $\omega_1, \omega_2, \ldots, \omega_n$), $\mathcal{A}$ représente les événements que nous souhaitons mesurer (par exemple, l'évènement $\omega = \omega_1$; l'évènement $\omega \neq \omega_1$ et $\omega \neq \omega_2$...), et P est une application $P : \mathcal{A} \to [0, 1]$ qui représente la mesure des probabilités des évènements de $\mathcal{A}$. On dit d'un processus stochastique Y qu'il est caractérisé par l'espace probabilisé $(\Omega, \mathcal{A}, P)$, indexé sur X et à valeurs dans S. Il est également possible de représenter l'espace probabilisé par l'ensemble des fonctions de densité du processus, qui sont des fonctions $p_n(y_1 \ldots y_n; \mathbf{x_1} \ldots \mathbf{x_n})$ d'ordre croissant n [AM80], et nous appelons p_n les *statistiques d'ordre* n de Y.

Dans le contexte de ce manuscrit, nous considérons que les synthèses de textures sont des implémentations directes ou indirectes de modèles de processus stochastiques, et que les textures sont des réalisations de ces processus. Une façon d'exprimer un modèle de synthèse de textures est alors de l'exprimer sous

la forme d'un processus stochastique. Cette expression ne doit pas surprendre : les textures générées sont des ensembles de valeurs dans S, indexées par des coordonnées de texture dans X. Introduire la synthèse de textures revient à introduire un espace probabilisé $(\Omega, \mathcal{A}, P)$ dans la définition de la texture de sortie :toutes les synthèses de textures ont un ensemble des observables, qui renvoie à l'ensemble des issues d'un générateur pseudo-aléatoire, et des évènements ayant diverses probabilités peuvent être observés. En somme, l'espace probabilisé $(\Omega, \mathcal{A}, P)$ désigne les *statistiques* d'un modèle de synthèse de texture, où Ω représente toutes les issues, $\mathcal{A}$ est une tribu de Ω, et P représente les probabilités sur $\mathcal{A}$. X représente des *positions* dans une texture (par exemple, des texels entiers de $\mathbb{Z}^2$ en synthèse discrète ou des positions dans $\mathbb{R}^2$ en synthèse continue), et S représente l'ensemble des *valeurs* que peuvent prendre une texture en une position $\mathbf{x} \in X$ (par exemple, $\mathbb{R}$ en niveau de gris, ou $\mathbb{R}^3$ en couleur). Une synthèse Y est évaluable sur X et Ω, et on écrit $Y : X \times \Omega \to S$.

Lorsque $\omega \in \Omega$ est fixe, $Y(\omega)$ est une *réalisation* d'une synthèse ; cette réalisation est une fonction de X appelée la *trajectoire* de Y en ω, et correspond à une texture unique pouvant être générée par la synthèse. Lorsque $\mathbf{x} \in X$ est fixe, $Y(\mathbf{x})$ est une variable aléatoire.

Nous désignons par $\widetilde{\Omega}$ l'ensemble des observés, qui permet de représenter les textures $Y(\widetilde{\omega})$ dont nous disposons déjà pour la synthèse de textures ; cet ensemble est couramment utilisé dans la synthèse par l'exemple, où il y a typiquement une seule texture E (appelée l'« exemple »), et est utilisé pour instancier l'espace probabilisé $(\Omega, \mathcal{A}, P)$, en inférant les statistiques à partir de l'exemple. Inférer un espace probabilisé à partir d'un seul exemple ou à partir d'un nombre très restreint de paramètres requiert l'exploitation de certaines hypothèses sur le processus. Dans cette section, nous nous intéressons à deux couples d'hypothèses : la stationnarité et ergodicité, et celles que nous avons exploité dans une de nos contributions principale, la cyclostationnarité et cycloergodicité. Nous nous intéressons également au cas particulier où ces synthèses sont Gaussiennes, qui permet de définir un processus avec peu de données.

Pour plus de clarté, nous omettons souvent l'aléa dans la notation et écrivons simplement une synthèse de texture Y comme une fonction $Y : X \to S$, en explicitant si Y est une synthèse ou une texture issue de la synthèse.

3.2 Synthèse stationnaire

Soit Y un processus stochastique défini tel qu'au début du chapitre 3. Nous définissons ici les notions d'invariance par translation et de stationnarité pour les statistiques p_n d'ordre n.

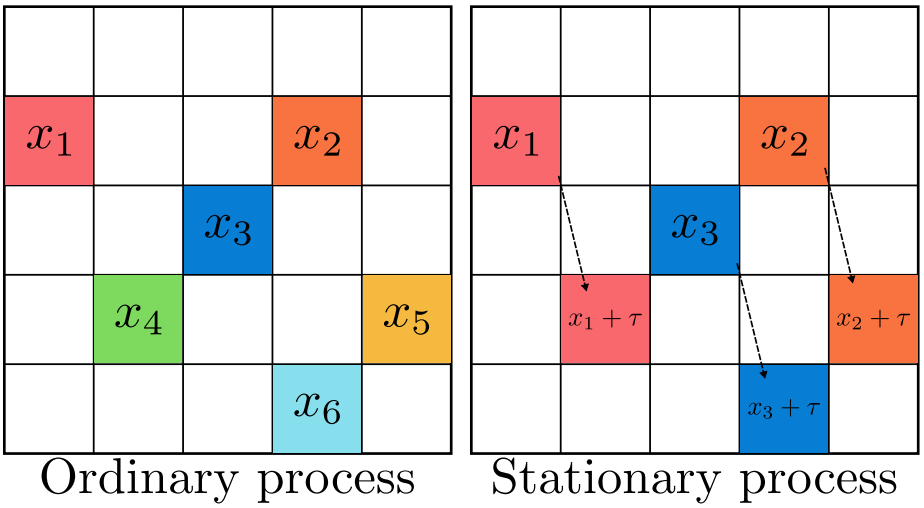


FIGURE 3.1 – Deux processus stochastiques discrets, dont un stationnaire. Les statistiques d'ordre n des processus stationnaires au sens strict sont invariantes *pour toute* translation τ.

Definition 3.1 *p_n est invariant par translation $\tau \in X$ si et seulement si :*

$$p_n(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = p_n(\mathbf{x_1} + \tau, \mathbf{x_2} + \tau, \ldots, \mathbf{x_n} + \tau), \quad \forall \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n} \in X$$

Definition 3.2 *Y est stationnaire au sens strict (SSS) si et seulement si p_n est invariant par toute translation $\tau \in X$, pour tout $n \geq 1$.*

Example 3.3 *Par exemple, les statistiques d'ordre 1 p_1 d'un processus stationnaire Y à m valeurs possibles de S ont la forme d'un histogramme à m classes, et représentent la probabilité qu'un texel prenne une valeur dans S ; de plus, cette probabilité est la même pour n'importe quel texel. Ainsi, dans la Figure 3.2, les statistiques d'ordre 1 en $\mathbf{x_1}$ et en $\mathbf{x_2}$ sont les mêmes. De plus, si Y est ergodique (voir section 3.2.3), l'histogramme d'une seule réalisation approxime p_1.*

La propriété de stationnarité simplifie le problème de la synthèse de texture, pour celles qui sont concernées : au lieu de devoir décrire les statistiques d'une synthèse pour toutes les positions, il suffit de les décrire à partir d'une seule position et pour un ensemble de translations. Ainsi, sur la Figure 3.1, on a par exemple $p_1(\mathbf{x_1}) = p_1(\mathbf{x_1}+\tau)$, ou encore $p_3(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}) = p_3(\mathbf{x_1}+\tau, \mathbf{x_2}+\tau, \mathbf{x_3}+\tau)$.

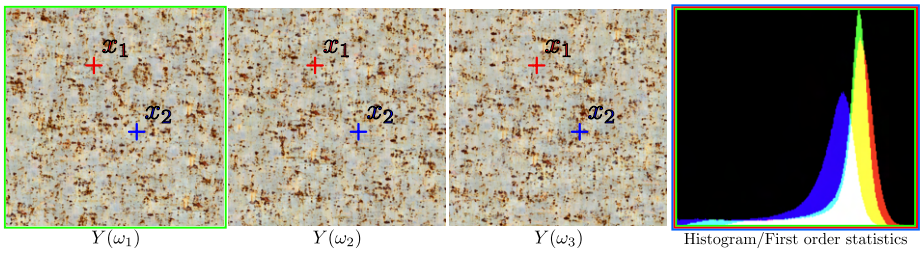


FIGURE 3.2 – Les statistiques d'ordre 1 d'un processus stationnaire Y sont identiques en tout indice de Y (ici, par exemple, en $\mathbf{x_1}$ et $\mathbf{x_2}$). Sous l'hypothèse souvent admise d'ergodicité (section 3.2.3), l'histogramme d'une seule réalisation suffisamment large correspond également aux statistiques d'ordre 1 du processus Y (ici, par exemple, l'histogramme de la réalisation $Y(\omega_1)$).

3.2.1 Espérance et Moments

Les moments sont des indicateurs de dispersion d'une variable aléatoire, et sont généralisés pour les processus. Dans ce manuscrit, nous utilisons les appellations de Adomian et al. [AM80], qui généralise les moments aux processus stochastiques, sans perdre en complexité ; souvent, seules la moyenne et la fonction d'autocovariance sont définies pour les processus.

Classiquement, les moments sont généralisés aux processus depuis les variables aléatoires, où l'on parle de moments d'ordre n pour décrire l'ordre de l'exposant utilisé. La notation de Adomian et al. [AM80] inclut cet exposant, mais aussi le nombre de variables aléatoires du processus Y ; on parle du k-ème moment d'ordre n pour décrire le moment étudié sur n variables avec un ordre d'exposant de k. Nous nous concentrons sur quelques-uns de ces moments, utiles dans notre contexte, et les raccordons avec des termes connus dans la synthèse de textures.

Moments usuels

Definition 3.4 *La* moyenne *ou l'*espérance d'ordre 1 *d'une synthèse stationnaire Y est son premier moment d'ordre 1. C'est une constante $\mathbf{m}^1$ dans S définie par*

$$\mathbf{m}^1 \triangleq \mathbf{m}_1^1 = \int_S y\, p_1(y;\mathbf{x})\, dy \tag{3.1}$$

Pour n'importe quelle position $\mathbf{x} \in X$.

La moyenne peut par exemple servir à centrer, autour de 0, une texture dont la synthèse est considérée comme stationnaire, ce qui est parfois nécessaire lors

d'un mélange (par exemple pour le *spot noise* [GGM11]). Elle peut aussi servir à définir des espaces probabilisés, notamment celui des processus Gaussiens (voir section 3.2.2).

Definition 3.5 *La* dispersion d'ordre 2 *d'une synthèse stationnaire Y est son second moment d'ordre 2 [AM80]. C'est une fonction $\mathbf{m}_2^2 : X \to S$ définie par*

$$\mathbf{m}_2^2(\tau) \triangleq \int_S \int_S y_1 y_2 \, p_2(y_1, y_2; \mathbf{x}, \mathbf{x}+\tau) \, dy_1 \, dy_2 \tag{3.2}$$

Pour n'importe quelle position $\mathbf{x} \in X$.

Le fait qu'un processus puisse être représenté comme un ensemble de variables aléatoires ne signifie pas qu'il n'existe pas de lien de corrélation entre elles, et les moments fournissent des outils pour détecter ces liens. La fonction de dispersion représente la moyenne du produit des valeurs entre deux positions, l'une en $\mathbf{x}$, l'autre en $\mathbf{x}+\tau$. Elle peut servir à définir le spectre d'une synthèse stationnaire (par exemple pour le bruit à phase aléatoire [GGM11]), ou à définir des espaces probabilisés, notamment celui des processus Gaussiens lorsque S est unidimensionnel (voir section 3.2.2).

Definition 3.6 *On dit d'un processus stochastique qu'il est* stationnaire au sens large *(SSL) si et seulement si son premier moment d'ordre 1 et son second moment d'ordre 2 sont invariants pour toute translation τ.*

Un processus SSS est aussi SSL, mais un processus SSL n'est pas nécessairement SSS. Cette distinction est utile pour les processus Gaussiens stationnaires (voir section 3.2.2).

Il est possible de généraliser le deuxième moment à l'ordre n, bien que nous n'utilisons pas cette définition dans ce manuscrit ; on parle alors de *dispersion d'ordre n* de la synthèse [AM80].

Moments centrés et/ou réduits Le moment peut être donné sous plusieurs formes ; par exemple, les k-èmes moments centrés d'ordre n $\mathbf{mc}_n^k$ sont les k-èmes moments d'ordre n auquel on a retiré la moyenne $\mathbf{m}^1$ à chaque terme y_i. $\mathbf{mc}_1^1$ est donc nul.

Definition 3.7 *La* variance *d'un processus stationnaire Y est son second moment centré d'ordre 1. C'est une constante $\mathbf{mc}^2$ dans S définie comme*

$$\mathbf{mc}^2 \triangleq \mathbf{mc}_1^2 = \int_S \left(y - \mathbf{m}^1\right)^2 p_1(y; \mathbf{x}) \, dy \tag{3.3}$$

La variance sert à donner un indicateur de la dispersion de chaque variable aléatoire du processus Y ; en synthèse de textures, la conservation de la moyenne et de la variance permet notamment de garantir que les statistiques d'ordre 1 ne s'éloignent pas trop des valeurs voulues, car des valeurs trop éloignées créent de larges contrastes qui se voient facilement dans les textures.

Definition 3.8 *La* fonction d'autocovariance *d'un processus stationnaire Y est son second moment centré d'ordre 2. C'est une fonction $\mathbf{mc}_2^2 : X \to S$ définie par*

$$\mathbf{mc}_2^2(\tau) \triangleq \int_S \int_S \left(y_1 - \mathbf{m}^1\right)\left(y_2 - \mathbf{m}^1\right)\, p_2(y_1, y_2; \mathbf{x}, \mathbf{x}+\tau)\, dy_1\, dy_2 \tag{3.4}$$

et on a $\mathbf{mc}_2^2(\mathbf{0}) = \mathbf{mc}_1^2$.

La fonction d'autocovariance est un indicateur des covariances entre des variables aléatoires situées à deux positions différentes d'un processus. Elle est utilisée pour de nombreux calculs et son étude est essentielle pour les processus Gaussiens (voir section 3.2.2).

Le k-ème moment d'ordre n, pour tout $k \geq 2$, peut aussi être réduit, en divisant chaque terme y_i (pour i allant de 1 à n) par la variance $\mathbf{mc}^2$ de Y ; on note le k-ème moment d'ordre n réduit $\mathbf{mr}_n^k$. Son existence n'est possible que si $\mathbf{mc}^2 \neq 0$. Enfin, ces deux notions peuvent être combinées : on note le k-ème moment centré réduit d'ordre n $\mathbf{mcr}_n^k$. $\mathbf{mcr}^2$ est donc égal à 1 sur toutes les composantes de S s'il existe.

Definition 3.9 *La* fonction d'autocorrélation *d'un processus stationnaire Y est son second moment centré réduit d'ordre 2. C'est une fonction $\mathbf{mcr}_2^2 : X \to S$ définie par*

$$\mathbf{mcr}_2^2(\tau) \triangleq \int_S \int_S \frac{y_1 - \mathbf{m}^1}{\mathbf{mc}^2} \frac{y_2 - \mathbf{m}^1}{\mathbf{mc}^2}\, p_2(y_1, y_2; \mathbf{x}, \mathbf{x}+\tau)\, dy_1\, dy_2 \tag{3.5}$$

et on a $\mathbf{mcr}_2^2(\mathbf{0}) = \mathbf{mcr}_1^2 = 1$.

La fonction d'autocorrélation est un indicateur des corrélations entre des variables aléatoire situées à deux positions différentes d'un processus : une valeur de 1 sur les dimensions de S indique une forte corrélation positive, une valeur de -1 indique une forte corrélation négative, et une valeur de 0 indique l'absence de corrélation.

Definition 3.10 *Le* coefficient d'asymétrie *d'un processus stationnaire Y est le troisième moment d'ordre 1 centré réduit. C'est une constante $\mathbf{mcr}^3$ définie*

sur S par

$$\mathbf{m}^3 \triangleq \mathbf{m}_1^3 = \int_S \left(\frac{y - \mathbf{m}^1}{\mathbf{mc}^2} \right)^3 p_1(y; \mathbf{x})\, dy \tag{3.6}$$

Le coefficient d'asymétrie donne une indication sur la forme de la distribution des statistiques d'ordre 1 de chaque dimension de S : si la valeur est positive, alors la distribution est décalée à gauche de la médiane ; si la valeur est négative, alors elle est est décalée à droite de la médiane. Une valeur nulle peut ne pas indiquer la symétrie des statistiques d'ordre 1, et il existe des contre-exemples de distributions non symétriques ayant un coefficient d'asymétrie de 0.

Definition 3.11 *Le* kurtosis *d'un processus stationnaire Y est le quatrième moment d'ordre 1 centré réduit. C'est une constante* $\mathbf{mcr}^4$ *définie sur S par*

$$\mathbf{m}^4 \triangleq \mathbf{m}_1^4 = \int_S \left(\frac{y - \mathbf{m}^1}{\mathbf{mc}^2} \right)^4 p_1(y; \mathbf{x})\, dy \tag{3.7}$$

Le kurtosis est décrit comme le coefficient d'aplatissement des statistiques d'ordre 1 de chaque dimension de S. Ainsi, si le kurtosis est égal à 3, alors elles correspondent à celles d'une loi normale.

Moments croisés Lorsque l'espace d'état S est multidimensionnel, il est parfois utile de s'intéresser aux moments croisés. Le processus générant chaque dimension de S peut être vu comme un vecteur de processus indexé sur D, où D est le nombre de dimensions de S, aussi appelé nombre de canaux des textures synthétisée. Chaque canal est alors un processus indexé sur X et à valeurs dans S^-. Par exemple, lorsque $S = \mathbb{R}^3$, $S^- = \mathbb{R}$ et le processus se décompose en trois sous-processus. On appelle ce canal Y_d, où d est une des dimensions de S, compris entre 1 et D. Dans cette section en particulier, on note certains moments comme des fonctions $\mathbf{m} : Y_d \to S^-$ pour signifier qu'ils sont calculés à partir de Y_d plutôt que Y.

Definition 3.12 *La* matrice de covariance *d'un processus stationnaire Y à valeurs dans S est la matrice symétrique regroupant l'ensemble des covariances entre ses canaux à valeurs dans S^-. Chaque ligne i et colonne j représente un des canaux Y_i ou Y_j. Chaque cellule est le moment croisé* $\mathbf{m}^{1+1}$ *entre le canal i et le canal j, qui est une constante dans S^- :*

$$\mathbf{mc}_1^{1+1} = \int_{S^-} \int_{S^-} \left(y_i - \mathbf{m}^1(Y_i) \right) \left(y_j - \mathbf{m}^1(Y_j) \right) P(y_i, y_j; \mathbf{x}, \mathbf{x})\, dy_i\, dy_j, \tag{3.8}$$

où y_i est sur Y_i, et y_j est sur Y_j. Sur la diagonale $i = j$, le moment est égal au moment d'ordre 2 de Y_i ; autrement dit, $\mathbf{mc}_1^{1+1} = \mathbf{mc}_1^2$.

La matrice de covariance sert à détecter les covariances/corrélations entre les canaux. Elle est utilisée par l'algorithme de la PCA pour décorréler des variables, et permet de décorréler les statistiques d'ordre 1 d'un processus. Il est important de noter que la PCA ne décorrèle pas les statistiques d'ordre 2 et plus. Le second moment d'ordre 1 de Y se retrouve sur la diagonale de la matrice.

Definition 3.13 *La* fonction d'autocovariance croisée *d'un processus stationnaire Y à valeurs dans S est l'ensemble des matrices symétriques regroupant les covariances entre ses canaux à valeurs dans S^-, avec un décalage de τ. Chaque ligne i et colonne j représente un des canaux Y_i ou Y_j. Chaque cellule est le second moment centré d'ordre 2 entre le canal i à la position $\mathbf{x}$ et le canal j à la position $\mathbf{x}+\tau$:*

$$\mathbf{mc}_2^{1+1}(\tau) = \int_{S^-}\int_{S^-} \left(y_i - \mathbf{m}^1(Y_i)\right)\left(y_j - \mathbf{m}^1(Y_j)\right)\, P(y_i, y_j; \mathbf{x}, \mathbf{x}+\tau)\, dy_i\, dy_j, \tag{3.9}$$

où y_i est sur Y_i, et y_j est sur Y_j. De plus, on a $\mathbf{mc}_2^{1+1}(\mathbf{0}) = \mathbf{mc}_1^{1+1}$.

La fonction d'autocovariance croisée est une généralisation de la fonction d'autocovariance de Y, qui est représentée par l'ensemble des diagonales des matrices de $\mathbf{m}_2^{1+1}$. Elle est notamment utilisée pour définir les processus Gaussiens multivariés (voir section 3.2.2).

Definition 3.14 *La* fonction d'autocorrélation croisée *est l'ensemble des moments croisés centrés réduits $\mathbf{mcr}_2^{1+1}$ entre le canal i à la position $\mathbf{x}$ et le canal j à la position $\mathbf{x}+\tau$:*

$$\mathbf{mcr}_2^{1+1}(\tau) = \int_{(S^-)^2} \frac{y_i - \mathbf{m}^1(Y_i)}{\mathbf{mc}^2(Y_i)} \frac{y_j - \mathbf{m}^1(Y_j)}{\mathbf{mc}^2(Y_j)}\, P_{ij}\, dy_i\, dy_j, \tag{3.10}$$

où y_i est sur Y_i, et y_j est sur Y_j et $P_{ij} = P(y_i, y_j; \mathbf{x}, \mathbf{x}+\tau)$. On a $\mathbf{mcr}_2^{1+1}(\mathbf{0}) = \mathbf{mcr}_1^{1+1}$.

La fonction d'autocorrélation croisée fonctionne similairement à la fonction d'autocovariance, sauf que les valeurs sont comprises entre -1 et 1.

Exemples La Figure 3.3 montre des exemples de synthèses stationnaires. L'étude de leur moment donne des indications sur leurs statistiques ; le tableau 3.2.1 regroupe certains des moments significatifs.

1. Y_1 est un bruit blanc, c'est-à-dire que chaque variable de chaque canal est indépendante et suit une loi normale, en l'occurrence une loi $\mathcal{N}(\frac{1}{2}, \frac{1}{2})$. De

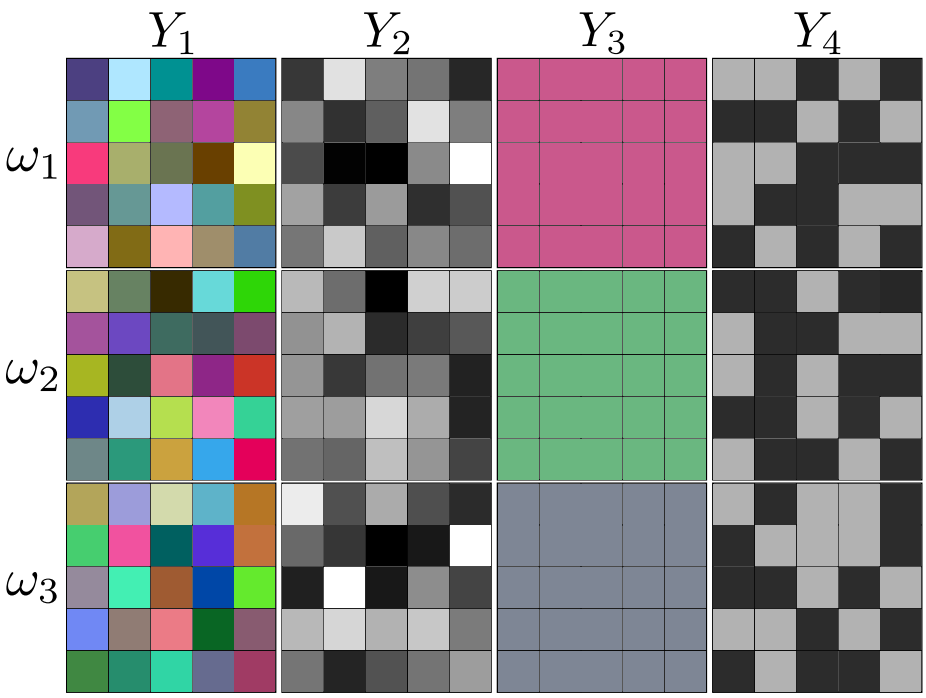


FIGURE 3.3 – Exemple de 4 processus stationnaires discrets $Y_1 \dots Y_4$ particuliers. Y_1 est un bruit blanc aux couleurs indépendantes sur chaque canal. Y_2 est un bruit blanc où les couleurs sont dépendantes entre les canaux. Y_3 est un processus où une couleur unie aléatoire de loi normale est choisie pour chaque réalisation. Y_4 est un processus où chaque variable est soit claire, soit foncée. Leurs moments nous donnent des informations utiles.

plus, les valeurs de chaque canal sur S sont indépendantes. Sa moyenne est donc $\mathbf{m}^1 = \left\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right\}$; sa variance est $\mathbf{mc}^2 = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\}$; sa fonction d'autocorrélation $\mathbf{mcr}_2^2$ est de $\mathbf{0}$ partout sauf en $\tau = \mathbf{0}$ où elle est égale à $\mathbf{1}$; sa fonction d'autocorrélation croisée $\mathbf{mcr}_2^{1+1}$ est de 0 partout, sauf en $\tau = \mathbf{0}$ lors de l'étude de canaux identiques où elle est égale à 1 ; son coefficient d'asymétrie est égal à $\mathbf{0}$ (indiquant la symétrie des statistiques d'ordre 1 autour de la moyenne) et son kurtosis est égal à $\{3, 3, 3\}$ (indiquant le fait que les statistiques d'ordre 1 soient Gaussiennes).

2. Y_2 est similaire à Y_1, sauf que les valeurs de chaque canal sur S sont identiques. La différence avec Y_1 se détecte notamment sur la fonction d'autocorrélation croisée $\mathbf{mcr}_2^{1+1}$, qui est toujours égale à 1 sur $\tau = \mathbf{0}$ pour n'importe quelle paire de canaux.

3. Y_3 est fait d'une couleur unie tirée avec une loi $\mathcal{N}(\frac{1}{2}, \frac{1}{2})$ sur chaque canal et pour chaque réalisation. La différence avec Y_1 se détecte notamment sur la fonction d'autocorrélation, où elle est toujours égale à $\mathbf{1}$ sur chaque

Processus	Y_1	Y_2	Y_3	Y_4
$\mathbf{m}^1$	$\begin{pmatrix}\frac{1}{2} & \frac{1}{2} & \frac{1}{2}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{2} & \frac{1}{2} & \frac{1}{2}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{2} & \frac{1}{2} & \frac{1}{2}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{2} & \frac{1}{2} & \frac{1}{2}\end{pmatrix}$
$\mathbf{mc}^2$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$
$\mathbf{mcr}^3$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$
$\mathbf{mcr}^4$	$\begin{pmatrix}3 & 3 & 3\end{pmatrix}$	$\begin{pmatrix}3 & 3 & 3\end{pmatrix}$	$\begin{pmatrix}3 & 3 & 3\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\end{pmatrix}$
$\mathbf{mcr}_2^2(\tau \neq \mathbf{0})$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$
$\mathbf{mcr}_1^{1+1}$	$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$
$\mathbf{mcr}_2^{1+1}(\tau \neq \mathbf{0})$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$

TABLE 3.1 – Ensemble de moments significatifs des exemples de la Figure 3.8.

canal peu importe la valeur de τ.

4. Y_4 est un ensemble de variables suivant une loi de Bernoulli (soit $\frac{1}{4}$ sur chaque canal, soit $\frac{3}{4}$ sur chaque canal). Les moments d'ordre 1 sont très similaires à ceux de Y_2 ; la différence se détecte cependant sur le kurtosis $\mathbf{mcr}^4$, qui est égal à $\mathbf{1}$, indiquant un aplatissement maximal des statistiques d'ordre 1.

3.2.2 Synthèse Gaussienne

Les processus Gaussiens sont des processus pour lesquels n'importe quelle combinaison linéaire de variable aléatoire du processus suit une loi normale (ou normale multivariée si S est multidimensionnel). Ainsi, la distribution des variables $Y(\mathbf{x})$ et $Y(\mathbf{y})$ du processus ont une forme normale, et leur jointure ont une forme normale multivariée (voir la Figure 3.4).

Une synthèse Gaussienne Y est entièrement caractérisée par sa moyenne $\mathbf{m}^1$ et par son second moment d'ordre 2 (centré ou pas), habituellement par sa fonction d'autocovariance $\mathbf{mc}_2^2$ (ou l'ensemble de ses fonctions d'autocovariance croisée $\mathbf{mc}_2^{1+1}$ si S est multidimensionnel). Toute synthèse SSL Gaussienne est par conséquent SSS Gaussienne, puisque les statistiques sont entièrement caractérisées par ces moments.

Dans la synthèse de textures, ces processus permettent d'implémenter des

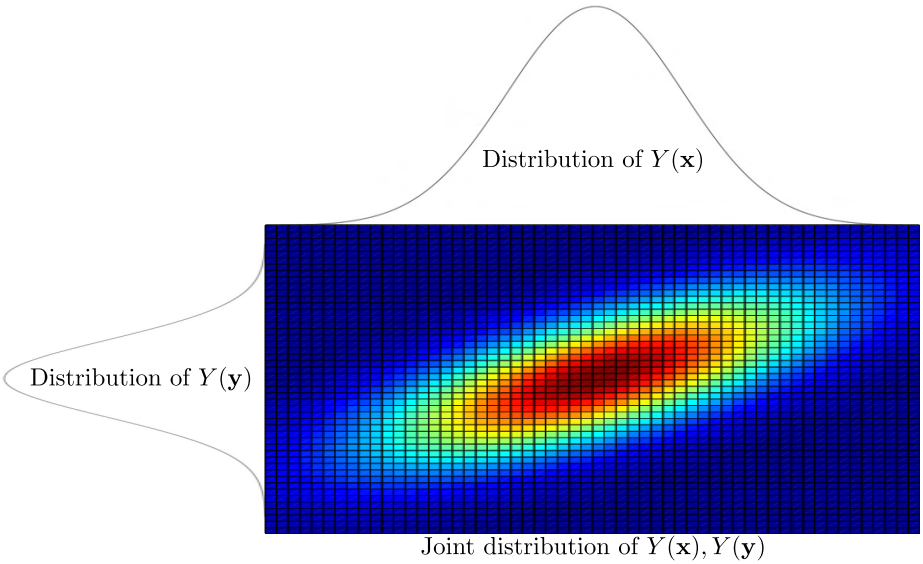


FIGURE 3.4 – Forme des statistiques d'ordre 2 d'un processus Gaussien, qui sont représentées par l'ensemble des probabilités p_2 des couples $(Y(\mathbf{x}), Y(\mathbf{y}))$ pour tout $\mathbf{x}, \mathbf{y} \in X$. Les distributions marginales p_1 de chaque variable du processus sont Gaussiennes (ici, celles de $Y(x)$ et de $Y(y)$), et leur jointure l'est aussi.

bruits procéduraux Gaussiens (voir section 2.2).

Dans la Figure 3.3, par exemple, Y_1, Y_2 et Y_3 sont des processus Gaussiens par définition. Y_4, en revanche, n'est pas un processus Gaussien, ce qui se remarque notamment à ses statistiques d'ordre 1 (ou son kurtosis).

Le théorème central limite établit que la somme de variables aléatoires converge vers une loi normale, et se généralise aux processus. Ainsi, la synthèse Gaussienne s'appuie souvent sur des mélanges ou des convolutions de noyaux aléatoires.

En revanche, les statistiques Gaussiennes et stationnaires limitent fortement la gamme de textures pouvant être produites par ces synthèses : des organisations spatiales complexes sont impossibles à obtenir avec ce genre de synthèse.

3.2.3 Ergodicité

Nous avons vu que les moments et les statistiques d'ordre n des processus stationnaires sont calculées par des intégrales sur l'ensemble des valeurs dans S pouvant être prises par un processus. Sous la propriété d'ergodicité,

les propriétés statistiques d'une synthèse Y peuvent être approximées à partir d'une seule réalisation $\{Y(\omega) \,|\, \omega \in \Omega\}$ en faisant la moyenne sur les texels. Lors de la synthèse stationnaire par l'exemple, la propriété d'ergodicité est essentielle : elle garantit qu'une seule texture (un exemple) ait besoin d'être utilisée comme exemple pour la synthèse (à condition qu'elle soit suffisamment large), et les statistiques nécessaires à la synthèse peuvent être estimées implicitement ou explicitement sans information additionnelle. Ainsi, on peut considérer que toutes les synthèses par l'exemple stationnaires font l'hypothèse de processus ergodiques.

Definition 3.15 *Un processus stationnaire Y est* ergodique au sens de la moyenne *si et seulement si sa moyenne $\mathbf{m}^1$ est égale à sa moyenne spatiale lorsque le domaine X' (domaine de définition de la texture sur X) croît, c'est-à-dire que*

$$\mathbf{m}^1 = \lim_{|X'|\to\infty} \frac{1}{|X'|} \int_{X'} Y(\mathbf{x},\omega)\, d\mathbf{x} \tag{3.11}$$

avec une probabilité égale à 1.

Cette propriété permet l'estimation de $\mathbf{m}_1^1$ sur une seule réalisation $Y(\omega)$ (sur un seul exemple).

Definition 3.16 *Un processus stationnaire Y est* ergodique au sens de la fonction d'autocovariance *si sa fonction d'autocovariance $\mathbf{mc}_2^2$ est égale à sa fonction d'autocovariance spatiale lorsque le domaine X' croît, c'est-à-dire que*

$$\mathbf{mc}_2^2(\tau) = \lim_{|X'|\to\infty} \frac{1}{|X'|} \int_{X'} \left(Y(\mathbf{x},\omega) - \mathbf{m}_1^1\right) \left(Y(\mathbf{x}+\tau,\omega) - \mathbf{m}_1^1\right)\, d\mathbf{x} \tag{3.12}$$

avec une probabilité égale à 1.

Cette propriété permet l'estimation de $\mathbf{m}_2^2$ sur une seule réalisation $Y(\omega)$ (sur un seul exemple).

Un processus stationnaire au sens large qui est ergodique au sens de la moyenne et au sens de la fonction d'autocovariance est parfois appelé *ergodique au sens large* (ESL). Un processus ESL Gaussien ne nécessite donc qu'une seule réalisation (un seul exemple) pour être estimé et reproduit avec plus ou moins de précision.

On peut généraliser l'ergodicité aux statistiques d'ordre n. Notamment, sous l'hypothèse d'ergodicité des statistiques d'ordre 1, ces dernières peuvent être

représentées par l'histogramme d'une réalisation suffisamment large, et donc estimées par l'étude d'une seule réalisation.

Par exemple, sur la Figure 3.3, Y_1, Y_3, et Y_4 sont ergodiques par définition : il est possible d'étendre leur domaine et de calculer de façon précise les moments sur X'. En revanche, Y_2 n'est pas ergodique, notamment au sens de la moyenne $\mathbf{m}^1$, qui varie à chaque réalisation ; en outre, l'histogramme d'une seule réalisation de Y_2 ne correspondra jamais aux statistiques d'ordre 1 du processus, même en étendant X'.

3.2.4 Exemples

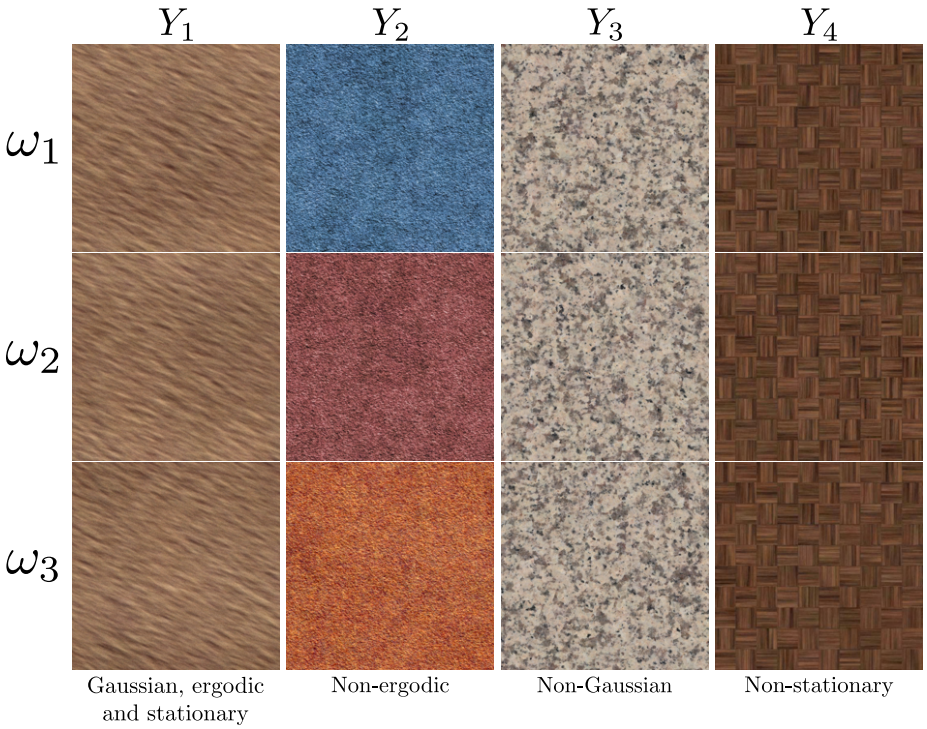


FIGURE 3.5 – Plusieurs réalisations (lignes) de différentes synthèses (colonnes) sur des exemples réels. Y_1 est une synthèse stationnaire, Gaussienne et ergodique. Y_2 est une synthèse stationnaire et Gaussienne, mais pas ergodique. Y_3 est une synthèse stationnaire, ergodique mais pas Gaussienne. Enfin, Y_4 n'est pas une synthèse stationnaire.

Nous avons donné des propriétés de la synthèse de textures stationnaire. La Figure 3.5 donne des exemples de résultats concrets de synthèse.

— Y_1 est une synthèse stationnaire, Gaussienne et ergodique ; ainsi, non seulement elle est définie entièrement par les moments $\mathbf{m}^1$ et $\mathbf{m}_2^2$, mais en plus, $\mathbf{m}^1$ est égale à la moyenne spatiale et $\mathbf{m}_2^2$ à l'autocovariance spatiale ; ils peuvent donc être estimés à partir d'une réalisation si on suppose qu'elle est suffisamment grande et leur estimation est nécessaire et suffisante à la définition entière du processus.
— Y_2 est stationnaire mais pas ergodique en la moyenne, car $\mathbf{m}^1$ est fait pour changer d'une réalisation à l'autre. L'estimation des statistiques sur un seul exemple est donc impossible.
— Y_3 est stationnaire et ergodique mais pas Gaussienne, notamment aux statistiques d'ordre 1 (ou, similairement, dans les histogrammes de chaque réalisation). La reproduction de ce genre de textures dans la synthèse par l'exemple peut-être approximée, notamment avec des synthèses par patchs ou par pavage.
— Y_4 n'est pas stationnaire. Cependant, les statistiques sont Gaussiennes et suivent un schéma périodique particulier ; ce genre de processus est le sujet de la section suivante.

3.3 Synthèse cyclostationnaire

Definition 3.17 *Les processus cyclostationnaires au sens strict (CSSS) sont définis par la périodicité de leurs statistiques sur X. Sous la propriété d'invariance des statistiques données dans la section précédente, l'invariance est alors vraie pour au moins une valeur de τ égale à $k\mathbf{t_0}$, avec $k \in \mathbb{Z}$ et $\mathbf{t_0} \in X$.*

La propriété de cyclostationnarité est plus méconnue en synthèse de textures. Elle permet pourtant de représenter n'importe quel processus dont les statistiques sont périodiques, et permettent notamment de générer des textures dont le motif se répète régulièrement avec une perturbation aléatoire. Dans la Figure 3.6, on considère que le processus est cyclostationnaire car il a au moins une translation $\mathbf{t_0} = \tau$ telle que les statistiques soient invariantes par application de cette translation. On dit que $\mathbf{t_0}$ est une *période* du processus cyclostationnaire s'il n'existe pas de vecteur $\mathbf{t_0}'$ colinéaire à $\mathbf{t_0}$ et plus petit que $\mathbf{t_0}$ tel que les statistiques de Y sont invariantes par translation de $k\mathbf{t_0}'$. Les processus cyclostationnaires généralisent les processus stationnaires : on peut voir un processus stationnaire comme un processus cyclostationnaire ayant un ensemble de périodes orthogonales infiniment petites (ou de la taille d'un texel dans le cas ou X est discret). Dans le cas des textures, la cyclostationnarité a souvent deux périodes $\mathbf{t_0}$ et $\mathbf{t_1}$. Ces périodes décrivent un parallélogramme dont la forme se répète périodiquement à travers X, comme montré sur la réalisation de la Figure 3.6. On appelle le domaine du premier parallélogramme X^- et il est possible de se ramener de X à X^- par congruence en utilisant l'opérateur

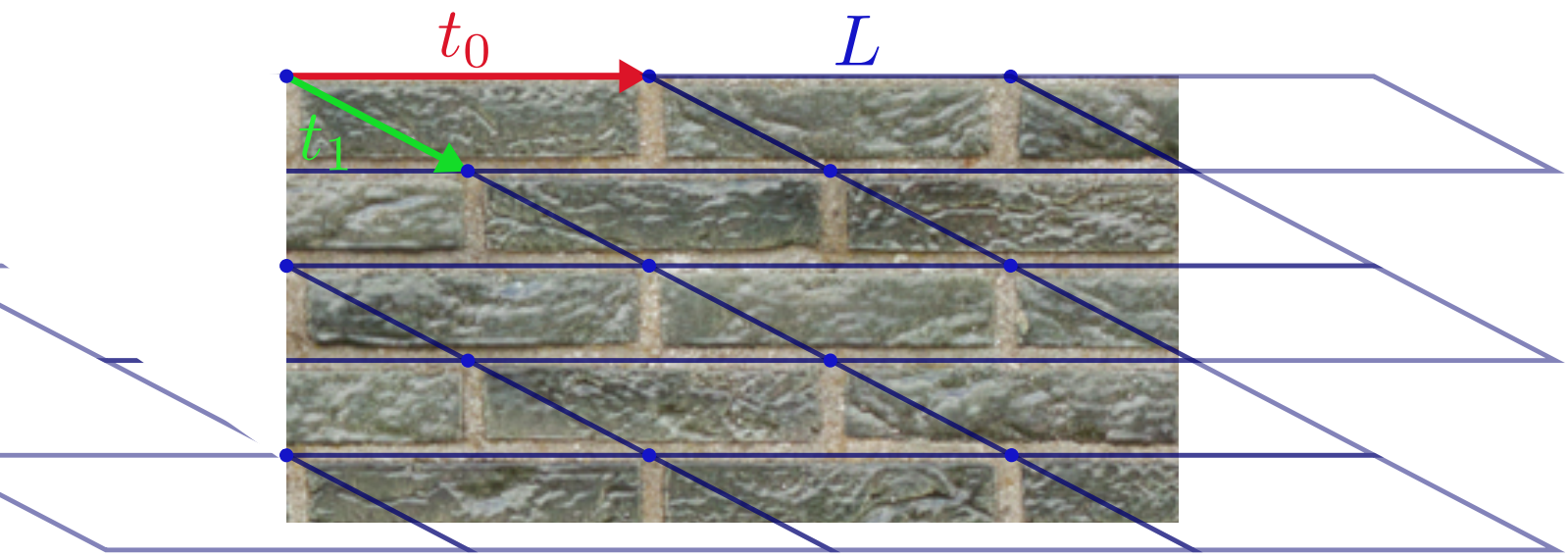


FIGURE 3.6 – Exemple d'une texture cyclostationnaire où les vecteurs $\mathbf{t_0}$ et $\mathbf{t_1}$ sont montrés. Ces deux vecteurs décrivent L, une structure en grille régulière dont chaque cellule est un parallélogramme. Le premier parallélogramme, en particulier, représente X^-. Le contenu de chaque cellule varie, mais reste visuellement proche : c'est le résultat de la périodicité des statistiques.

$\mathcal{P} : X \to X^-$ qui projette tout $\mathbf{x}$ modulo $(\mathbf{t_0}, \mathbf{t_1})$:

$$\mathbf{x} = \mathcal{P}(\mathbf{x}) + k\mathbf{t_0} + l\mathbf{t_1}, \quad \text{avec } \mathcal{P}(\mathbf{x}) \in X^- \text{ et } (k,l) \in \mathbb{Z}^2. \tag{3.13}$$

Un processus cyclostationnaire avec seulement un vecteur de période $\mathbf{t_0}$ peut être vu comme un processus cyclostationnaire avec un vecteur $\mathbf{t_1}$ orthogonal à $\mathbf{t_0}$ et infiniment petit s'il est stationnaire dans cette direction, ou infiniment grand s'il a des statistiques arbitraires.

Un processus cyclostationnaire peut se décomposer en un ensemble de processus stationnaires de diverses façons [Nap19]. Dans ce manuscrit, nous nous concentrons sur la décomposition en composantes polyphases.

Definition 3.18 *La* $\mathbf{x}$*-ème composante polyphase d'un processus cyclostationnaire* Y *est un sous-processus de* Y *noté* $Y^{\mathbf{x}}$ *pour tout* $\mathbf{x} \in X^-$. $Y^{\mathbf{x}}$ *est un processus stationnaire discret défini sur* $\{\mathbf{x} + k\mathbf{t_0} + l\mathbf{t_1} | \mathbf{x} + k\mathbf{t_0} + l\mathbf{t_1} \in X\}$ *et à valeurs dans* S.

Example 3.19 *Les statistiques d'ordre 1 d'un processus cyclostationnaire* Y *peuvent être représentées par l'ensemble des statistiques d'ordre 1 de ses composantes polyphases. Ainsi, dans la Figure 3.7, la composante* $Y^{\mathbf{x_1}}$ *n'a pas néces-*

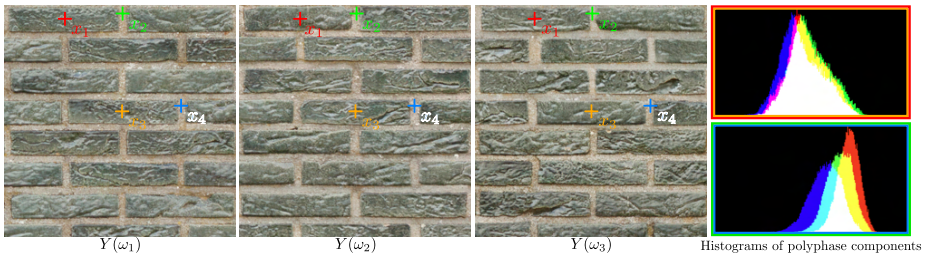


FIGURE 3.7 – Les statistiques d'ordre 1 d'un processus cyclostationnaire Y sont les mêmes sur plusieurs réalisations $(\omega_1, \ldots, \omega_n)$, et sont invariantes sur l'ensemble d'indices d'une même composante polyphase. Ici, la composante $Y^{\mathbf{x_1}}$ inclut la position $\mathbf{x_3}$ et la composante $Y^{\mathbf{x_2}}$ inclut la position $\mathbf{x_4}$. Les statistiques d'ordre 1 (à droite) de Y en $\mathbf{x_1}$ et $\mathbf{x_3}$ sont identiques, mais celles en $\mathbf{x_1}$ et $\mathbf{x_4}$ sont différentes.

sairement les même statistiques que la composante $Y^{\mathbf{x_2}}$. Sous l'hypothèse de cycloergodicité, l'ensemble des histogrammes des composantes polyphases calculés sur une seule réalisation se rapproche des statistiques d'ordre 1 de Y ; cependant, comparé à des processus stationnaires ergodiques, la convergence est plus lente.

3.3.1 Espérance et Moments

Contrairement à la synthèse stationnaire, les statistiques varient (périodiquement) sur X ; les moments sont aussi concernés par cette propriété d'invariance périodique.

Moments usuels

Definition 3.20 *La moyenne d'une synthèse CSSS Y est une fonction $\mathbf{m}^1 : X \to S$ définie par*

$$\mathbf{m}^1(\mathbf{x}) \triangleq \int_S y\, p_1(y; \mathbf{x})\, dy \tag{3.14}$$

et respecte la périodicité

$$\mathbf{m}^1(\mathbf{x}) = \mathbf{m}^1(\mathbf{x} + k\mathbf{t_0}) \quad \forall k \in \mathbb{Z}.$$

Contrairement aux processus stationnaires, la moyenne n'est pas une constante et varie sur X^- avant de se répéter périodiquement au-delà de X^-. La moyenne

en $\mathbf{x}$, $\mathbf{m}^1(\mathbf{x})$ représente aussi la moyenne stationnaire de la $\mathbf{x}$-ème composante polyphase $Y^{\mathbf{x}}$, et cette propriété reste vraie pour tous les premiers moments (comme la variance cyclostationnaire) car une seule variable est étudiée.

Definition 3.21 *La dispersion d'ordre 2 [AM80] d'une synthèse CSSL Y est une fonction $\mathbf{m}_2^2 : X \times X \to S$ définie par*

$$\mathbf{m}_2^2(\mathbf{x}, \tau) \triangleq \int_S \int_S y_1 y_2 \, p_2(y_1, y_2; \mathbf{x}, \mathbf{x} + \tau) \, dy_1 \, dy_2 \tag{3.15}$$

et respectant la périodicité

$$\mathbf{m}_2^2(\mathbf{x}, \tau) = \mathbf{m}_2^2(\mathbf{x} + k\mathbf{t_0}, \tau) \quad \forall k \in \mathbb{Z}.$$

La dispersion d'ordre 2 d'un processus cyclostationnaire, comme pour son homologue stationnaire, dépend de τ, qui représente n'importe quel écart $\mathbf{x_2} - \mathbf{x_1}$ égal à τ. Comme pour la moyenne cyclostationnaire, elle varie sur X^- avant de se répéter périodiquement au-delà de X^-. La dispersion d'ordre 2 en $\mathbf{x}$, $\mathbf{m}_2^2(\mathbf{x})$, contrairement à $\mathbf{m}^1(\mathbf{x})$, *ne représente pas la dispersion de la $\mathbf{x}$-ème composante polyphase $Y^{\mathbf{x}}$*. De façon générale, les seconds moments ne sont pas l'ensemble des seconds moments des composantes polyphases, car ils dépendent de différentes composantes polyphases.

Moments centrés et/ou réduits Dans cette section, on note certains moments comme des fonctions $\mathbf{m} : Y_d \to S^-$ pour signifier qu'ils sont calculés à partir de Y_d plutôt que Y.

Les moments stationnaires centrés, réduits ou croisés se généralisent tous bien au cas cyclostationnaire. Nous donnons uniquement ceux qui nous sont utiles dans ce manuscrit. Tous ces moments sont périodiques en $\mathbf{x}$.

Definition 3.22 *La variance d'une synthèse CSSL Y est une fonction $\mathbf{mc}^2 : X \to S$ définie par*

$$\mathbf{mc}^2(\mathbf{x}) \triangleq \int_S \left(y - \mathbf{m}^1(\mathbf{x})\right)^2 \, p_1(y; \mathbf{x}) \, dy \tag{3.16}$$

et la variance en $\mathbf{x}$ représente aussi la variance de la $\mathbf{x}$-ème composante polyphase $Y^{\mathbf{x}}$.

Definition 3.23 *La fonction d'autocovariance d'une synthèse CSSL Y est une*

fonction $\mathbf{mc}_2^2$ *définie par*

$$\mathbf{mc}_2^2(\mathbf{x},\tau) \triangleq \int_S \int_S \big(y_1 - \mathbf{m}^1(\mathbf{x})\big)\,\big(y_2 - \mathbf{m}^1(\mathbf{x}+\tau)\big)\; p_2(y_1,y_2;\mathbf{x},\mathbf{x}+\tau)\,dy_1\,dy_2 \tag{3.17}$$

et on a $\mathbf{mc}_2^2(\mathbf{x},0) = \mathbf{mc}_1^2(\mathbf{x})$.

Definition 3.24 *La fonction d'autocorrélation d'une synthèse CSSL* Y *est une fonction* $\mathbf{mcr}_2^2$ *définie par*

$$\mathbf{mcr}_2^2(\mathbf{x},\tau) \triangleq \int_S \int_S \frac{y_1 - \mathbf{m}^1(\mathbf{x})}{\mathbf{mc}^2(\mathbf{x})}\,\frac{y_2 - \mathbf{m}^1(\mathbf{x}+\tau)}{\mathbf{mc}^2(\mathbf{x}+\tau)}\,p_2(y_1,y_2;\mathbf{x},\mathbf{x}+\tau)\,dy_1\,dy_2 \tag{3.18}$$

et on a $\mathbf{mcr}_2^2(\mathbf{x},0) = \mathbf{mcr}_1^2(\mathbf{x}) = 1$.

Moments réduits Cette section généralise les moments croisés de la section 3.2.1 à la cyclostationnarité. Dans cette section, on note certains moments comme des fonctions $\mathbf{m} : Y_d \times X^n \to S^-$ pour signifier qu'ils sont calculés à partir de Y_d plutôt que Y.

Definition 3.25 *La* fonction d'autocovariance croisée *d'un processus cyclostationnaire* Y *à valeurs dans* S *est l'ensemble des matrices symétriques regroupant les covariances entre ses canaux à valeurs dans* S^-*, avec un décalage de* τ *et pour toute valeur de* $\mathbf{x}$ *(avec périodicité au-delà de* X^-*). Chaque ligne* i *et colonne* j *représente un des canaux* Y_i *ou* Y_j*. Chaque cellule est le moment croisé* $\mathbf{mc}_2^{1+1}$ *entre le canal* i *à la position* $\mathbf{x}$ *et le canal* j *à la position* $\mathbf{x}+\tau$ *:*

$$\mathbf{mc}_2^{1+1}(\mathbf{x},\tau) = \int_{S^-} \int_{S^-} \big(y_i - \mathbf{m}_1^1(Y_i;\mathbf{x})\big)\,\big(y_j - \mathbf{m}_1^1(Y_j;\mathbf{x}+\tau)\big)\; P_{ij}\,dy_i\,dy_j, \tag{3.19}$$

où y_i *est sur* Y_i*, et* y_j *est sur* Y_j *; et* $P_{ij} = P(y_i,y_j;\mathbf{x},\mathbf{x}+\tau)$.

Definition 3.26 *De façon similaire, la* fonction d'autocorrélation croisée *est*

$$\mathbf{mcr}_2^{1+1}(\mathbf{x},\tau) = \int_{S^-} \int_{S^-} \frac{y_i - \mathbf{m}_1^1(Y_i;\mathbf{x})}{\mathbf{m}_1^2(Y_i;\mathbf{x})}\,\frac{y_j - \mathbf{m}_1^1(Y_j;\mathbf{x}+\tau)}{\mathbf{m}_1^2(Y_j;\mathbf{x}+\tau)}\,P_{ij}\,dy_i\,dy_j, \tag{3.20}$$

où y_i *est sur* Y_i*, et* y_j *est sur* Y_j *; et* $P_{ij} = P(y_i,y_j;\mathbf{x},\mathbf{x}+\tau)$*. Par rapport à la fonction d'autocovariance croisée, ce moment permet d'obtenir des valeurs entre* -1 *et* 1.

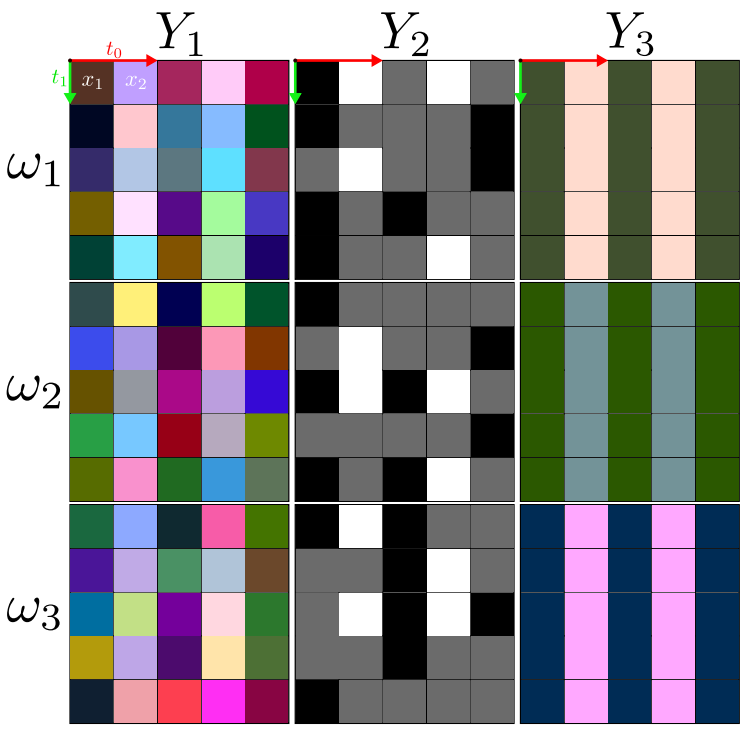


FIGURE 3.8 – Exemple de 3 processus cyclostationnaires discrets $Y_1 \dots Y_3$ particuliers ayant les mêmes périodes $\mathbf{t_0}$ et $\mathbf{t_1}$. Y_1 est composé de deux bruits blancs différents, $Y_1^{\mathbf{x_1}}$ et $Y_1^{\mathbf{x_2}}$. Y_2 est composé de deux processus de Bernoulli différents. Y_3 est un processus où une couleur unie aléatoire de loi normale est choisie pour chaque texel de chaque composante polyphase, avec les mêmes lois normales que pour Y_1.

Exemples La Figure 3.8 montre des exemples de synthèses cyclostationnaires. L'étude de leur moment donne des indications sur leurs statistiques.

1. Y_1 est un processus cyclostationnaire défini par ses composantes polyphases. La composante polyphase $Y_1^{\mathbf{x_1}}$ est un bruit blanc dont chaque composante de chaque canal suit une loi normale $\mathcal{N}(\frac{1}{4}, \frac{1}{4})$, et la composante polyphase $Y_1^{\mathbf{x_2}}$ est un bruit blanc dont chaque composante de chaque canal suit une loi normale $\mathcal{N}(\frac{3}{4}, \frac{1}{4})$. La moyenne $\mathbf{m}^1$ de Y_1 est donc $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$ pour toutes les positions congruentes à $\mathbf{x_1}$, et de $\{\frac{3}{4}, \frac{3}{4}, \frac{3}{4}\}$ sur les positions congruentes à $\mathbf{x_2}$. Sa variance $\mathbf{mc}^2$ est $\{\frac{1}{8}, \frac{1}{8}, \frac{1}{8}\}$ partout ; sa fonction d'autocorrélation $\mathbf{mcr}_2^2$ est de $\mathbf{0}$ partout sauf en $\tau = \mathbf{0}$ où elle est égale à $\mathbf{1}$; sa fonction d'autocorrélation croisée $\mathbf{mcr}_2^{1+1}$ est de $\mathbf{0}$ partout sauf en $\tau = \mathbf{0}$ lors de l'étude de canaux identiques où elle est égale à $\mathbf{1}$.
2. Y_2 est un processus cyclostationnaire défini par ses composantes poly-

Processus	Y_1	Y_2	Y_3
$\mathbf{m}^1(\mathbf{x_1})$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{4} & \frac{1}{4} & \frac{1}{4}\end{pmatrix}$
$\mathbf{m}^1(\mathbf{x_2})$	$\begin{pmatrix}\frac{3}{4} & \frac{3}{4} & \frac{3}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{3}{4} & \frac{3}{4} & \frac{3}{4}\end{pmatrix}$	$\begin{pmatrix}\frac{3}{4} & \frac{3}{4} & \frac{3}{4}\end{pmatrix}$
$\mathbf{mc}^2(\mathbf{x_1})$	$\begin{pmatrix}\frac{1}{8} & \frac{1}{8} & \frac{1}{8}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{8} & \frac{1}{8} & \frac{1}{8}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{8} & \frac{1}{8} & \frac{1}{8}\end{pmatrix}$
$\mathbf{mc}^2(\mathbf{x_2})$	$\begin{pmatrix}\frac{1}{8} & \frac{1}{8} & \frac{1}{8}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{8} & \frac{1}{8} & \frac{1}{8}\end{pmatrix}$	$\begin{pmatrix}\frac{1}{8} & \frac{1}{8} & \frac{1}{8}\end{pmatrix}$
$\mathbf{mcr}_2^2(\mathbf{x_1},\{1,0\})$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$
$\mathbf{mcr}_2^2(\mathbf{x_1},\{2,0\})$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\end{pmatrix}$
$\mathbf{mcr}_2^{1+1}(\mathbf{x_1},\{0,0\})$	$\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$
$\mathbf{mcr}_2^{1+1}(\mathbf{x_1},\{1,0\})$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$
$\mathbf{mcr}_2^{1+1}(\mathbf{x_1},\{2,0\})$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$

phases. La composante polyphase $Y_2^{\mathbf{x_1}}$ suit une loi de Bernoulli (soit 0, soit $\frac{1}{2}$ sur chaque canal), et $Y_2^{\mathbf{x_2}}$ aussi (soit $\frac{1}{2}$, soit 1 sur chaque canal). La différence avec Y_1 se remarque notamment sur $\mathbf{mcr}_1^{1+1}$ ou $\mathbf{mcr}_2^{1+1}$ lorsque $\tau = \mathbf{0}$.

3. Y_3 est un processus cyclostationnaire défini par ses composantes polyphases. La composante polyphase $Y_3^{\mathbf{x_1}}$ est égale à une constante fixée par une loi normale $\mathcal{N}(\frac{1}{4}, \frac{1}{4})$ sur chaque canal, et $Y_3^{\mathbf{x_2}}$ est égale à une constante fixée par une loi normale $\mathcal{N}(\frac{3}{4}, \frac{1}{4})$ sur chaque canal. La différence avec Y_1 et Y_2 se remarque notamment sur $\mathbf{m}_2^2$ ou $\mathbf{mcr}_2^{1+1}$ lorsque $\tau = (2, 0)$.

3.3.2 Synthèse Gaussienne

La synthèse Gaussienne est aussi possible dans le cadre cyclostationnaire. De façon similaire au cas stationnaire, un processus Gaussien peut être entièrement caractérisé par sa moyenne $\mathbf{m}^1$ et par son second moment d'ordre 2 (centré ou pas), habituellement par sa fonction d'autocovariance $\mathbf{mc}_2^2$ (ou l'ensemble

de ses fonctions d'autocovariance croisée $\mathbf{mc}_2^{1+1}$ si S est multidimensionnel). Ainsi, toute synthèse CSSL est CSSS, puisque les statistiques d'ordre n sont entièrement caractérisées par ces moments.

La question de la synthèse Gaussienne cyclostationnaire est une de nos contributions principales, et est évoquée plus en détail dans le chapitre 4.

3.3.3 Cycloergodicité

L'ergodicité de la section 3.2.3 se généralise au cas cyclostationnaire ; c'est ce qu'on appelle la cycloergodicité.

Definition 3.27 *Un processus cyclostationnaire Y est* cycloergodique au sens de la moyenne *si et seulement si sa moyenne $\mathbf{m}^1(\mathbf{x})$ est égale à la moyenne périodique lorsque le domaine X' (domaine de définition de la texture sur X) croît, c'est-à-dire que*

$$\mathbf{m}^1(\mathbf{x}) = \lim_{|X'|\to\infty} \frac{1}{|X'^{\mathbf{x}}|} \sum_{\mathbf{x_0}\in X'^{\mathbf{x}}} Y(\mathbf{x_0},\omega) \tag{3.21}$$

avec une probabilité égale à 1, où $X'^{\mathbf{x}}$ représente l'ensemble des indices de la $\mathbf{x}$-ème composante polyphase sur lesquels la texture est définie.

Cette propriété permet l'estimation de $\mathbf{m}^1$ sur une seule réalisation $Y(\omega)$.

Definition 3.28 *Un processus cyclostationnaire Y est* cycloergodique au sens de la fonction d'autocovariance *si et seulement si sa fonction d'autocovariance $\mathbf{mc}_2^2(\mathbf{x},\tau)$ est égale à la fonction d'autocovariance périodique lorsque le domaine X' croît, c'est-à-dire que*

$$\mathbf{mc}_2^2(\mathbf{x},\tau) = \lim_{|X'|\to\infty} \frac{1}{|X'^{\mathbf{x}}|} \sum_{\mathbf{x_0}\in X'^{\mathbf{x}}} \big(Y(\mathbf{x_0},\omega) - \mathbf{m}^1(\mathbf{x})\big)\big(Y(\mathbf{x_0}+\tau,\omega) - \mathbf{m}^1(\mathbf{x})\big) \tag{3.22}$$

avec une probabilité égale à 1.

Cette propriété permet l'estimation de $\mathbf{m}_2^2$ sur une seule réalisation $Y(\omega)$.

On appelle un processus cyclostationnaire au sens large qui est ergodique au sens de la moyenne et au sens de la fonction d'autocovariance un processus *cycloergodique au sens large* (CESL). Un processus CESL Gaussien ne nécessite

donc qu'une seule réalisation (un seul exemple) pour être estimé et reproduit avec plus ou moins de précision.

Comme pour les processus stationnaires, on peut généraliser la cycloergodicité aux statistiques d'ordre n. Notamment, sous l'hypothèse d'ergodicité des statistiques d'ordre 1, ces dernières peuvent être représentées par *l'ensemble des histogrammes des estimations des composantes polyphases* d'une réalisation suffisamment large, et donc estimées par l'étude d'une seule réalisation.

Le fait que l'estimation des statistiques d'ordre 1 ne puisse se faire que sur $X'^{\mathbf{x}}$ peut fortement limiter la précision de l'estimation en fonction de la taille de X'^{-} ; plus X'^{-} est grand, plus l'estimation se fait sur des échantillons éloignés dans X', et plus X' doit être grand pour compenser cette perte d'information par rapport à l'ergodicité dans le cas stationnaire. De ce fait, l'estimation précise d'un processus cyclostationnaire ayant généré une texture est plus difficile que l'estimation précise d'un processus stationnaire, rendant la synthèse cyclostationnaire particulièrement ardue.

3.3.4 Exemples

La Figure 3.9 donne des exemples de synthèses de textures.

— Y_1 est cyclostationnaire, Gaussienne et cycloergodique ; elle est donc définie uniquement par $\mathbf{m}^1$ et $\mathbf{m}_2^2$, et ces moments peuvent être estimés avec une seule réalisation suffisamment large.

— Y_2 est cyclostationnaire et Gaussienne, mais pas cycloergodique ; l'estimation des statistiques sur une seule réalisation est donc impossible.

— Y_3 est cyclostationnaire et cycloergodique, mais pas Gaussienne, notamment aux statistiques d'ordre 1 (ou, similairement, dans l'ensemble des histogrammes de chaque composante polyphase).

— Y_4 n'est pas cyclostationnaire, et est une synthèse purement théorique rendant l'image naturelle d'un arbre à un point dans le temps aléatoire.

La reproduction de Y_4 sort du cadre de la synthèse de textures : la synthèse a plutôt été pensée indirectement pour des processus stationnaires ou cyclostationnaires, et certains auteurs (dont Julesz [Jul81]) ne considèrent pas les images naturelles dans leur définition de ce qu'est une « texture ». Nous élaborons cependant la question de synthèse de textures exploitant des processus stochastiques sans stationnarité ni cyclostationnarité dans la conclusion (chapitre 6).

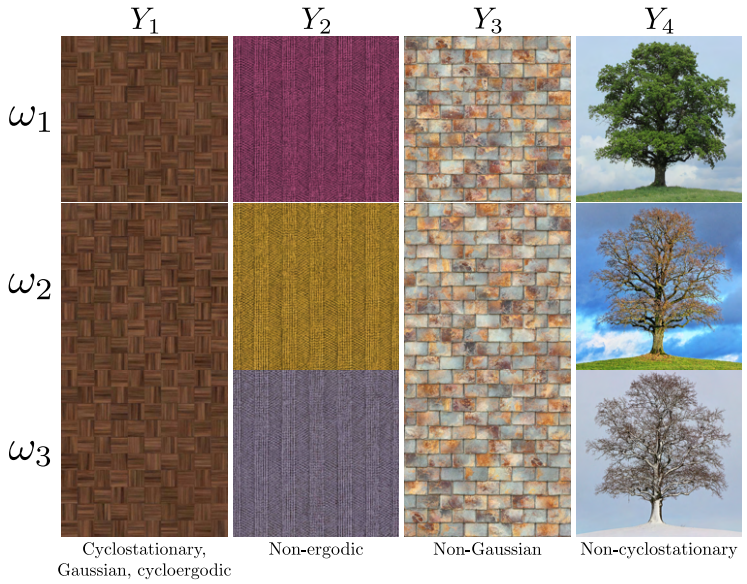


FIGURE 3.9 – Plusieurs réalisations (lignes) de différentes synthèses (colonnes) sur des exemples réels. Y_1 est une synthèse cyclostationnaire, Gaussienne et cycloergodique. Y_2 est une synthèse cyclostationnaire et Gaussienne, mais pas cycloergodique. Y_3 est une synthèse cyclostationnaire et cycloergodique, mais pas Gaussienne. Enfin, Y_4 n'est pas une synthèse cyclostationnaire ; c'est une synthèse purement théorique qui s'éloigne du cadre de la synthèse de textures.

Chapitre 4

Synthèse cyclostationnaire Gaussienne

Nous avons vu dans le chapitre précédent que les processus cyclostationnaires sont un cas particulier de processus stochastiques définis par la périodicité de leurs statistiques. Ces processus, qui généralisent les processus stationnaires, sont capables de générer des textures dont les motifs sont répartis périodiquement sur le domaine de la texture, comme ceux de la Figure 4.1.

FIGURE 4.1 – Exemples de textures cyclostationnaires. Cette classe de textures est caractérisée par des statistiques périodiques, c'est-à-dire, un motif périodique avec de fortes variations de couleur visibles en zoomant.

Nos travaux, publiés dans Lutz et al. [LSD21], nous ont amené à nous intéresser à la synthèse de textures désignées « cyclostationnaires », et qui correspondent à des textures dont les motifs sont répartis régulièrement, sont similaires, mais pas nécessairement identiques. Le point d'entrée utilisé pour ces

travaux est de généraliser des bruits procéduraux stationnaires et Gaussiens présentés dans la section 2.2 au cas cyclostationnaire.

Nous adaptons notamment le *spot noise* [vW91, GGM11], le bruit à phase aléatoire [GGM11, GSV+14], le bruit de Gabor [LLDD09, LLD11, TNVT19], le *high performance noise* [HN18], et le *phasor noise* [TEZ+19]. Le bruit à phase aléatoire en particulier n'est pas un modèle de bruit procédural Gaussien, mais s'y apparente fortement et génère des textures similaires [GGM11]. Le *phasor noise* [TEZ+19] n'est pas non plus un algorithme de bruit procédural Gaussien, mais s'appuie sur des modèles issus du bruit procédural Gaussien.

4.1 Modèle de synthèse cyclostationnaire

Dans cette section, nous complétons le modèle cyclostationnaire présenté en section 3.3. Les mêmes notations sont utilisées ; ainsi, un processus cyclostationnaire de synthèse de texture est noté Y, son espace probabilisé est $(\Omega, \mathcal{A}, P)$, il est indexé sur l'ensemble d'indices X et à valeurs dans S (notamment $S = \mathbb{R}$ en niveau de gris et $S = \mathbb{R}^3$ pour les textures à trois canaux de couleur). Nous considérons ici que X est continu ; une texture à ensemble d'indices discrets peut être rudimentairement ramenée à un ensemble d'indices continus par interpolation bilinéaire, comme le fait une carte graphique. La particularité de ces processus est que les statistiques sont périodiques ; donc P est périodique, et nous notons ses vecteurs de période $\mathbf{t_0}$ et $\mathbf{t_1}$.

Textures cyclostationnaires et leurs statistiques. Dans ce chapitre, nous définissons les « textures périodiquement corrélées » ou « textures cyclostationnaires » les réalisations de cette classe de processus dans le contexte de la synthèse de textures, pour signifier leur appartenance à l'ensemble des observables d'un processus cyclostationnaire quelconque.

Tout comme un processus cyclostationnaire et cycloergodique, connu ou pas, les ayant générées, leurs statistiques sont périodiques, notamment la moyenne $\mathbf{m}_1^1$ et la fonction d'autocovariance $\mathbf{mc}_2^2$ (voir section 3.3.1), tous deux estimables à partir de la texture.

En particulier, la cycloergodicité permet d'estimer les moments du processus sous-jacent à un exemple. Nous nous intéressons particulièrement au premier moment d'ordre 1 et au second moment centré d'ordre 2, qui, comme montré dans la section 3.3.2, permettent de définir entièrement les statistiques d'un processus Gaussien.

L'estimateur $\widetilde{\mathbf{m}_1^1}$ du premier moment d'ordre 1 (moyenne) à partir d'un

exemple E est :

$$\widetilde{\mathbf{m}_1^1}(\mathbf{x}) = \frac{1}{\#ech} \sum_{k,l} E(\mathbf{x} + k\,\mathbf{t_0} + l\,\mathbf{t_1}) \tag{4.1}$$

étant donné les vecteurs de période connus $\mathbf{t_0}$ et $\mathbf{t_1}$. $\#ech$ correspond aux nombres d'échantillons pour lesquels $\mathbf{x} + k\,\mathbf{t_0} + l\,\mathbf{t_1}$ tombent dans l'ensemble d'indices de l'exemple E. Cette équation est utilisée plus tard pour le modèle de l'ADSN dans la section 4.2.3 et y est visualisée dans la ligne du bas de la Figure 4.7.

L'estimateur $\widetilde{\mathbf{mc}_2^2}$ du second moment d'ordre 2 centré (fonction d'autocovariance) à partir d'un exemple E est :

$$\widetilde{\mathbf{mc}_2^2}(\mathbf{x}, \tau) = \frac{1}{\#ech} \sum_{k,l} \left(E(\mathbf{y}) - \widetilde{\mathbf{m}_1^1}(\mathbf{x})\right) \left(E(\mathbf{y} + \tau) - \widetilde{\mathbf{m}_1^1}(\mathbf{x} + \tau)\right), \tag{4.2}$$

où $\mathbf{y} = \mathbf{x} + k\,\mathbf{t_0} + l\,\mathbf{t_1}$ et $\#ech$ correspond aux nombres d'échantillons pour lesquels à la fois $\mathbf{y}$ et $\mathbf{y} + \tau$ tombent dans l'ensemble d'indices de l'exemple E.

Dans le cas stationnaire, la fonction d'autocovariance peut être contrôlée dans le domaine spectral par le spectre de puissance (PSD), qui est la transformée de Fourier de $\mathbf{mc}_2^2$. Dans le cas cyclostationnaire, il est possible de passer par le spectre de puissance instantané (IPSD), qui correspond à la transformée de Fourier de $\mathbf{mc}_2^2$ en $\mathbf{x}$ pour tout X. Cependant, ce spectre est difficile à définir ; c'est pourquoi nous passons par les spectres des composantes polyphases (voir section 3.3), qui correspondent chacun à un spectre de puissance classique sur une composante polyphase donnée, contrôlent $\mathbf{mc}_2^2$ [KGE18], et sont exploités dans la section 4.2.2.

Les processus sont définis entièrement par $\mathbf{m}_1^1$ et $\mathbf{mc}_2^2$, car nous traitons essentiellement des processus cyclostationnaires Gaussiens (voir section 3.3.2) ou s'y apparentant.

Textures à phase aléatoire Les textures à phase aléatoire sont le résultat de bruit à phase aléatoire. Elles sont aussi appelées « textures Gaussiennes », au même titre que des résultats d'autres bruits procéduraux, car elles ont habituellement des statistiques similaires à celles des réalisations de processus Gaussiens. Elles sont définies par Galerne et al. [GGM11] comme des textures dont la phase dans le domaine de Fourier est la réalisation d'un bruit blanc :

$$I(\mathbf{x}) = \sum_{\xi} A(\xi)\, \cos\left(2\pi\xi\mathbf{x} + \Phi(\xi)\right) \tag{4.3}$$

pour toutes les fréquences $\xi \in \hat{X}$, avec une amplitude A (dont le carré est le spectre de puissance) et avec des phases Φ uniformément distribuées entre $-\pi$ et π.

Textures cyclostationnaires à phase aléatoire Nous définissons les textures cyclostationnaires à phase aléatoire (CSRP pour *cyclostationary random phase*) pour modéliser des textures cyclostationnaires à travers une phase aléatoire :

$$I(\mathbf{x}) = \sum_{\xi} A(\mathbf{x}, \xi) \cos\left(2\pi\xi\mathbf{x} + \Phi(\xi)\right). \tag{4.4}$$

La nouveauté de ce modèle est que l'amplitude $A : X \times \hat{X} \to \mathbb{R}$ varie spatialement et périodiquement avec les périodes $\mathbf{t_0}$ et $\mathbf{t_1}$. Φ est aléatoire, et A définit entièrement les statistiques des textures CSRP.

Bien que toutes les textures CS ne sont pas CSRP, ce modèle les approxime de la même manière que le modèle des textures à phases aléatoires approxime les textures étant des réalisations de processus stationnaires (appelées micro-textures par Galerne et al. [GGM11]), et établit la possibilité de synthétiser des textures en définissant une amplitude variant spatialement et en randomisant les phases. Il est important de noter que les textures à phase aléatoire et les textures dites « Gaussiennes » sont très similaires, mais pas équivalentes [GGM11].

4.2 Bruits cyclostationnaires Gaussiens

Dans cette section, nous introduisons les bruits cyclostationnaires Gaussiens. Ils consistent en diverses formulations de processus Gaussiens cyclostationnaires permettant la synthèse de textures cyclostationnaires.

4.2.1 Modulation d'un signal stationnaire

Les processus cyclostationnaires peuvent être exprimés comme une agrégation de composants stationnaires définie périodiquement sur X [Nap19]. Une façon simple d'obtenir un signal cyclostationnaire est de moduler un ensemble de signaux jointement stationnaires $\{Y_1, ..., Y_K\}$ comme

$$I(\mathbf{x}) = \sum_{s} w_s(\mathbf{x})\, Y_s(\mathbf{x}), \quad \forall \mathbf{x} \in X, \tag{4.5}$$

où $w_s : X \to \mathbb{R}$ sont des fonctions de poids $(\mathbf{t_0}, \mathbf{t_1})$-périodiques.

Cette approche a été utilisée par Guingo et al. [GSDC17] dans un cas non-cyclostationnaire pour générer des textures hétérogènes à partir d'un exemple depuis lequel des régions stationnaires et leurs fonctions de poids sont extraites. Dans une moindre mesure, on retrouve aussi cette stratégie dans le travail de

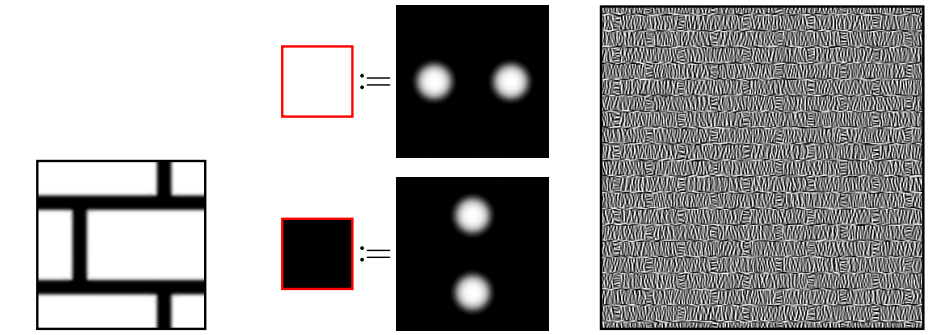


FIGURE 4.2 – Modulation de deux signaux stationnaires créés à partir de deux bruits à phase aléatoire. Les poids sont définis à gauche par une texture scalaire périodique. Les signaux sont interpolés linéairement sur les zones de transition (entre noir et blanc).

Haindl et al. [HH09], où des textures dites « quasi-régulières » sont considérées comme un ensemble de cellules bordées et adjacentes dans lesquelles un bruit est extrait et synthétisé. Le problème de cette approche est la définition des poids w_s et des signaux stationnaires Y_s : leur création manuelle est fastidieuse et leur extraction automatique à partir d'un exemple est un problème difficile. L'approche est illustrée dans la Figure 4.2.

4.2.2 Bruit à phase aléatoire cyclostationnaire

Nous avons établi dans l'équation (4.4) que les textures cyclostationnaires à phase aléatoire peuvent être exprimées sous la forme d'une somme de cosinus aléatoirement déphasés, avec des statistiques étant contrôlées par l'amplitude A, qui varie spatiallement. À partir de là, les bruits procéduraux stationnaires basés sur l'utilisation d'une PSD (power spectral density) dans le domaine fréquentiel peuvent être redéfinis à partir d'une PSD variant spatialement, ce qui inclut le bruit à phase aléatoire [GGM11, GSV+14], mais aussi le bruit de Gabor [LLDD09] et le *phasor noise* [TEZ+19]. La difficulté est de contrôler la PSD variant spatialement $A^2(\mathbf{x}, \bullet) = S^{\mathbf{x}}$.

Nous validons cette idée dans la Figure 4.3 par synthèse dans le domaine de Fourier en implémentant directement l'équation (4.4), où A^2 est interpolé linéairement à partir d'un petit ensemble de spectres de puissance définissant une variation périodique. Dans la Figure 4.4, nous avons adapté le bruit de Gabor, le bruit à phase aléatoire locale, et le *phasor noise*, qui correspondent à différentes façons d'approximer l'équation (4.4) en temps réel. Le *phasor noise* [TEZ+19] modifie en plus la phase aléatoire et applique une fonction de transfert sur le

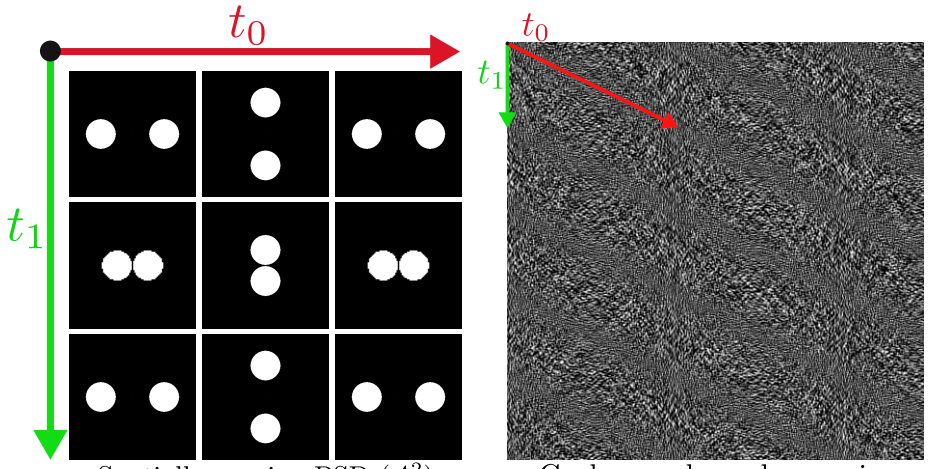


FIGURE 4.3 – Bruit cyclostationnaire à phase aléatoire généré par transformée de Fourier inverse d'une amplitude A variant spatialement et périodique, elle-même calculée à partir d'un ensemble de PSD $S^{\mathbf{x}}$ et d'une phase aléatoire unique. L'ensemble des PSD a été construit par interpolation bilinéaire de 9 spectres et défini périodiquement sur le parallélogramme décrit par $\mathbf{t_0}$ et $\mathbf{t_1}$.

signal. Dans cet exemple, A^2 est une fonction périodique générant un bi-lobe dont l'orientation varie selon une seule période $\mathbf{t_0}$ (vecteur rouge).

4.2.3 ADSN cyclostationnaire

L'*asymptotic discrete spot noise* (ADSN) [GGM11] est un processus Gaussien stationnaire et ergodique qui calcule une sortie I de taille arbitraire comme la convolution parcimonieuse d'une texture à ensemble d'indices discrets J (appelée *spot* discret) :

$$I = \frac{1}{\sqrt{|J|}} (J - \mathbf{1}\,\mathbf{m}_1^1) * W, \tag{4.6}$$

où W est la réalisation d'un bruit blanc dont les composants sont des variables non corrélées de distribution $\mathcal{N}(0,1)$, $\mathbf{m}_1^1$ est la moyenne stationnaire de I (estimable sur J), $\mathbf{1}$ est une fonction constante égale à 1 partageant le support de J, et $|J|$ est le nombre de texels de J (nombre de texels). Notons que W peut avoir un ensemble d'indices continus, même si J a un ensemble d'indices discrets [GLM17].

Notre modèle est un processus Gaussien cyclostationnaire et cycloergodique

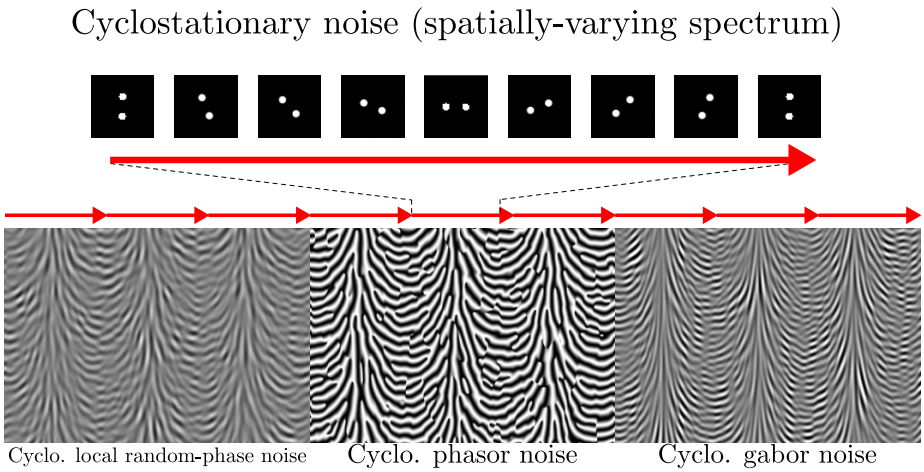


FIGURE 4.4 – Bruit cyclostationnaire spectral permettant la synthèse de textures cyclostationnaires, qui représentent des motifs exhibant une périodicité dans leurs statistiques. Nous portons des bruits stationnaires spectraux existants dans un contexte cyclostationnaire, permettant la synthèse de textures avec un contrôle dans le domaine de Fourier.

qui calcule une sortie I à partir d'une texture

$$I = \frac{1}{\sqrt{|J|}} (J - \mathbf{m}_1^1) * W_L, \tag{4.7}$$

où $\mathbf{m}_1^1$ est cette fois la moyenne cyclostationnaire du processus I, et où W_L est un processus discret défini sur la grille dont les sommets sont congruents à $\mathbf{0}$ par les vecteurs de période $\mathbf{t_0}$ et $\mathbf{t_1}$. W_L est une somme de Diracs (un par sommet) avec des valeurs décorrélées et aléatoire d'après la distribution $\mathcal{N}(0, |X^-|)$. I est un processus dont le premier moment est égal à $\mathbf{0}$ partout, et requiert l'addition de $\mathbf{m}_1^1$ après l'application du processus.

Fonction d'autocovariance. La fonction d'autocovariance de J est la fonction d'autocovariance de I.

Preuve. Nous le prouvons de façon similaire à la façon dont Galerne et al. [GGM11] l'ont prouvé pour le cas stationnaire. Pour faire le parallèle entre nos preuves et permettre l'estimation robuste du second moment de la texture J, nous considérons que celle-ci est périodique. Soit L la grille décrite par les vecteurs $\mathbf{t_0}$ et $\mathbf{t_1}$ comme sur la Figure 4.1. Nous appelons $|L|$ le nombre de sommets sur la grille L qui tombent dans l'espace d'indices de J.

Le second moment centré d'ordre 2 de J est estimé par

$$\widetilde{\mathbf{m}_2^2}(J;\mathbf{x},\tau) = \frac{1}{|L|}\sum_{\mathbf{u}\in L}\left(J(\mathbf{x}-\mathbf{u})-\mathbf{m}_1^1(\mathbf{x})\right)\left(J(\mathbf{x}-\mathbf{u}+\tau)-\mathbf{m}_1^1(\mathbf{x}+\tau)\right) \quad (4.8)$$

car $\mathbf{m}_1^1(\mathbf{x}) = \mathbf{m}_1^1(\mathbf{x}-\mathbf{u})$ pour tout $\mathbf{x}\in X$, $\mathbf{u}\in L$. Définissons $\dot{J} = \frac{1}{\sqrt{|J|}}(J-\mathbf{m}_1^1)$. Le second moment centré d'ordre 2 de I est

$$\begin{aligned}\mathbf{m}_2^2(I;\mathbf{x},\tau) &= \mathbb{E}[\sum_{\mathbf{u}\in L}\dot{J}(\mathbf{x}-\mathbf{u})\,W_L(\mathbf{u})\sum_{\mathbf{v}\in L}\dot{J}(\mathbf{x}+\tau-\mathbf{v})\,W_L(\mathbf{v})]\\ &= |X^-|\sum_{\mathbf{u}\in L}\dot{J}(\mathbf{x}-\mathbf{u})\dot{J}(\mathbf{x}+\tau-\mathbf{u}) \qquad (4.9)\\ &= \widetilde{\mathbf{m}_2^2}(J;\mathbf{x},\tau)\end{aligned}$$

car $\frac{|X^-|}{|J|} = \frac{1}{|L|}$, $\mathbb{E}[W_L(\mathbf{u})\,W_L(\mathbf{v})]$ est égal à 0 quand $\mathbf{u}\neq\mathbf{v}$, $\mathbf{u}\notin L$ ou $\mathbf{v}\notin L$, et $|X^-|$ sinon. I est donc un processus Gaussien cyclostationnaire de second moment d'ordre 2 $J(\widetilde{\mathbf{m}_2^2})(\mathbf{x},\tau)$.

Cette preuve s'adapte également bien pour l'ensemble des seconds moments croisés d'ordre 2, en remplaçant $\mathbf{mc}_2^2$ par $\mathbf{mc}_2^{1+1}$ et $\mathbf{m}_1^1$ par le premier moment centré d'ordre 1 du canal concerné, justifiant que I est également un processus Gaussien multivarié cyclostationnaire.

Le modèle de l'ADSN cyclostationnaire est proche d'une catégorie spécifique de *spot noise* contrôlé localement élaboré par Pavie et al. [PGDG16] and Cavalier et al. [CGG19], où les résultats sont rendus réguliers en contrôlant la position où les copies de J sont tirées : notre conjecture est que notre CS-ADSN est le modèle cyclostationnaire sous-jacent pour la version parfaitement régulière, qu'ils présentent sans considération pour les moments de la sortie I, ce qui nous permet d'aller une étape plus loin et d'arriver à la synthèse par l'exemple pour les textures cyclostationnaires dans la section suivante.

4.3 Synthèse par l'exemple

Les paramètres des modèles des bruits peuvent être difficiles à définir manuellement, surtout dans le cas cyclostationnaire ; le bruit par l'exemple consiste à estimer les paramètres à partir d'un exemple d'entrée E. Dans notre contexte, l'idée est de trouver un processus cyclostationnaire Y qui aurait pu générer l'exemple.

Le modèle cyclostationnaire par phase aléatoire de l'équation (4.4) n'est malheureusement pas un modèle viable pour la synthèse par l'exemple, parce qu'il requiert une estimation précise de la PSD des composantes polyphases,

et nous ne disposons que d'une poignée parcimonieuse d'échantillons dans E pour chaque composante polyphase. Pour cette raison, nous élaborons sur le modèle CS-ADSN présenté dans la section précédente. Nous l'appliquons sur deux bruits procéduraux en utilisant l'exemple comme un *spot* : le *spot noise* lui-même (section 4.3.1) et le *high performance noise* (section 4.3.2). Des résultats sont disponibles Figure 4.6 ainsi que dans la Figure 4.5.

Dans la section suivante, nous supposons que $\mathbf{t_0}$ et $\mathbf{t_1}$ sont déjà estimés à partir de l'exemple traité ; on montre comment effectuer cette estimation dans la section 4.4.

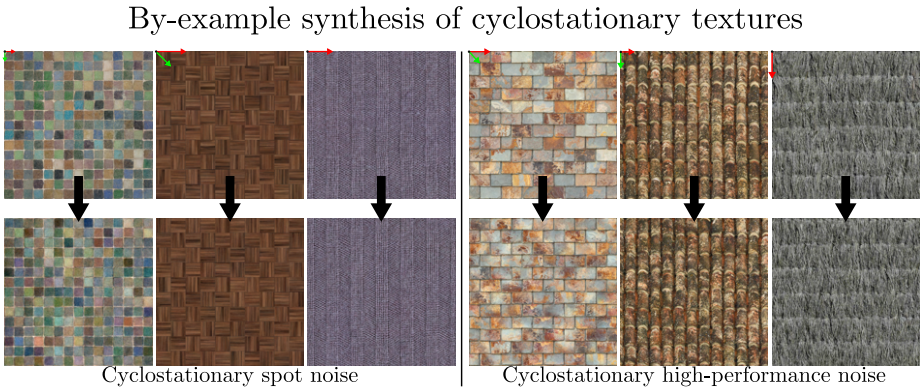


FIGURE 4.5 – Exemples synthétisés avec un algorithme cyclostationnaire adapté à leur reproduction : *CS-spot noise* à gauche et *CS-HPN* à droite.

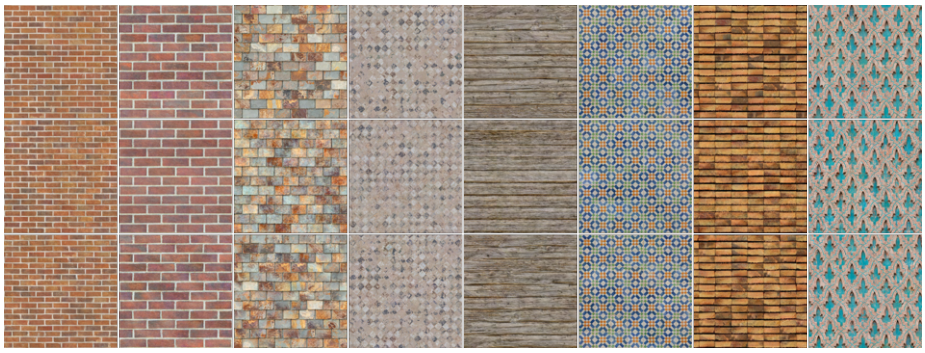


FIGURE 4.6 – Série d'exemples (haut) synthétisés avec notre *spot noise* cyclostationnaire (milieu) et notre *high performance noise* cyclostationnaire (en bas). Notre transfert des statistiques du premier ordre (section 4.3.3) a été appliquée à tous ces résultats. Les textures sont à taille non bornée, continues, et peuvent être échantillonnées en temps réel à n'importe quel position de $\mathbb{R}$. Nous montrons un échantillon de même résolution de l'entrée.

4.3.1 *Spot noise* cyclostationnaire

L'algorithme du *spot noise* par l'exemple [GGM11] est une implémentation directe du modèle d'ADSN de l'équation (4.6) qui consiste à tirer E (le *spot* $J = E$) sur des positions aléatoires. Ces positions sont déterminées par un processus de Poisson [vW91] plutôt qu'un bruit blanc W, qui permet d'augmenter la couverture spatiale.

Nous définissons le *spot noise* cyclostationnaire à partir de l'équation (4.7), en tirant E uniquement sur les sommets de la grille régulière L, comme montré dans la Figure 4.7. Comme les poids W_L sont souvent peu nombreux, il peut être plus rapide de calculer le *spot noise* cyclostationnaire directement plutôt qu'en tirant E plusieurs fois comme dans la section 2.2.1. Pour ce processus, le premier moment $\mathbf{m}_1^1$ doit être estimé à partir de l'exemple E utilisé grâce à l'équation (4.1). Le résultat asymptotique de ce bruit par l'exemple prend la forme d'un processus Gaussien cyclostationnaire ayant les statistiques du CS-ADSN de E.

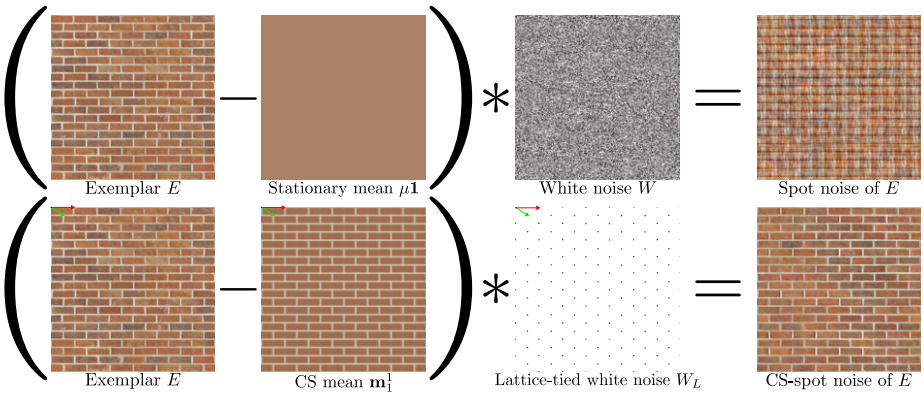


FIGURE 4.7 – Comparaison entre la version stationnaire (haut) et cyclostationnaire (bas) du *spot noise*, en utilisant un exemple d'entrée. L'estimation de la grille L et du premier moment $\mathbf{m}_1^1$ sont des étapes critiques pour la conservation des statistiques de l'exemple. L'addition de $\mathbf{m}_1^1$ stationnaire (respectivement cyclostationnaire) requise après l'équation (4.6) (respectivement l'équation (4.7)) sont omises.

4.3.2 *High performance noise* cyclostationnaire

Le *high performance noise* (stationnaire) [HN18, DH18, Bur19] est une synthèse temps réel par l'exemple désignée pour bénéficier à la fois d'avantages des algorithmes de pavage irréguliers et des algorithmes de bruit procéduraux. Il consiste à mélanger des tuiles hexagonales centrées sur les sommets d'une grille

triangulaire. Pour chaque sommet $\mathbf{x}$ de la grille, une tuile hexagonale est copiée à partir de l'exemple E à une position aléatoire donnée par une fonction de hachage $\mathbf{h}(\mathbf{x})$. Les tuiles superposées sont mélangées linéairement dans chaque triangle. Comme montré dans la Figure 4.8 à gauche, ce processus n'est pas capable de reproduire des textures CS. Le problème est que $\mathbf{x}$ et $\mathbf{h}(\mathbf{x})$ ne sont pas congruents.

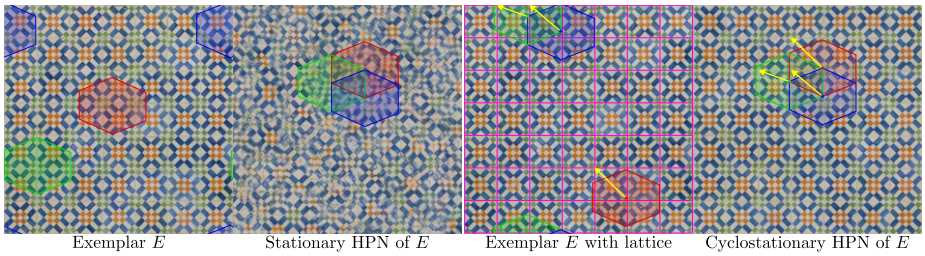


FIGURE 4.8 – *High performance noise* : version stationnaire (gauche) et cyclostationnaire (droite). Pour la seconde, les positions des tuiles hexagonales dans l'exemple ($\mathbf{h}(\mathbf{x})$) et dans le résultat ($\mathbf{x}$) sont congruentes.

Nous définissons une variante cyclostationnaire en contrôlant les positions sur lesquelles les tuiles hexagonales sont prises. La fonction de hachage $\mathbf{h}(\mathbf{x})$ retourne une position aléatoire congruente à $\mathbf{x}$. Autrement dit, $\mathcal{P}(\mathbf{h}(\mathbf{x})) = \mathcal{P}(\mathbf{x})$ (voir les vecteurs jaunes de la Figure 4.8). Cet algorithme est compatible avec la variante de Burley [Bur19] qui modifie les poids de mélange.

4.3.3 Transfert des statistiques du premier ordre

Les algorithmes précédents sont faits pour synthétiser des textures CSRP, c'est-à-dire, les textures CS étant les réalisations d'un processus Gaussien cyclostationnaire. Cependant, nombre de textures naturelles stochastiques n'ont pas des statistiques Gaussiennes. De ce fait, de nombreux travaux ont essayé de pousser les algorithmes au-delà du pur Gaussien. Parmi eux, le transport optimal a été utilisé avec succès pour contrôler les statistiques du premier ordre [GLR17, HN18, DH18], afin de traiter les textures sans histogrammes Gaussiens.

Nous rappelons ici le modèle du transfert d'histogrammes, montrons leurs limitations, proposons un transfert d'histogramme cyclostationnaire, discutons les problèmes de filtrage et de robustesse.

Transfert d'histogramme stationnaire

Soit E_H un exemple avec un histogramme non Gaussien H. Le principe du transfert d'histogramme est le suivant :

- Pré-calculer une fonction de transfert T qui associe chaque élément de H aux éléments d'un histogramme Gaussien quelconque, en respectant leur ordre dans H ;
- Pré-calculer T^{-1} dans une table de recherche.
- Pré-calculer $E_G = T(E_H)$, une version de l'exemple ayant des statistiques Gaussiennes.
- En temps réel, calculer le résultat $T^{-1}(Synthesis(E_G))$ en appliquant l'algorithme de synthèse à E_G, puis T^{-1} au résultat.

Dans sa version théorique idéale, les résultats ont un histogramme non Gaussien H, comme l'exemple E_H. Cependant, lorsqu'on applique un transfert d'histogramme à une texture CS, des artefacts apparaissent, comme montré sur la Figure 4.9. Ces artefacts sont liés à la nature globale de l'histogramme, qui ne capture les statistiques d'ordre 1 que lorsque l'exemple est une texture stationnaire (voir section 3.2), et ne prend donc pas en compte les variations de statistiques des textures cyclostationnaires.

Transfert d'histogramme cyclostationnaire

Comme expliqué dans la section 4.1, un processus CS peut être décomposé en un ensemble de composantes polyphases (PC pour polyphase components) qui sont des processus jointements stationnaires. L'intuition de notre transfert est de préserver l'histogramme de chacune d'entre elles indépendemment.

Soit E une texture CS que nous souhaitons synthétiser, et qu'on suppose être le résultat d'un processus cyclostationnaire. Nous appelons les $E^{\mathbf{x}}$ les *estimations des composantes polyphases* : pour chaque $\mathbf{x} \in X^-$, c'est une estimation, depuis l'exemple E, des $\mathbf{x}$-èmes composantes polyphases du processus CS sous-jacent. En d'autres termes, $E^{\mathbf{x}}$ est un sous-échantillonage de E fait de $|E|/|X^-|$ échantillons. Soit $H^{\mathbf{x}}$ l'histogramme de $E^{\mathbf{x}}$, qui est une estimation des statistiques du premier ordre de la $\mathbf{x}$-ème PC. Alors, la collection $\{H^{\mathbf{x}}\}_{\mathbf{x}\in X^-}$ est une estimation discrète des statistiques du premier ordre du processus cyclostationnaire sous-jacent.

Notre transfert d'histogramme CS opère sur les composantes polyphases similairement à la section 4.3.3 :

- Pré-calculer, pour chaque $H^{\mathbf{x}}$, une fonction de transfert $T_{\mathbf{x}}$ et son inverse $T_{\mathbf{x}}^{-1}$.
- Pré-calculer $E_G(\mathbf{x}) = T_{\mathcal{P}(\mathbf{x})}(E(\mathbf{x}))$, car chaque $\mathbf{x}$ appartient à la $\mathcal{P}(\mathbf{x})$-ème PC.
- En temps réel, calculer le résultat $T^{-1}_{\mathcal{P}(\mathbf{x})}(Synthesis(E_G))$.

Comme montré dans la Figure 4.9, ce transfert cyclostationnaire aide à résoudre les artefacts de fausses couleurs. Il aide notamment à enlever les artefacts de *region bleeding* (quand les statistiques de la texture de sortie sont incorrectement interprétées comme celles d'une région différente de l'exemple), et réduit les artefacts de *ghosting* (des éléments fantômes apparaissant dans la sortie) lorsqu'on tente de synthétiser des textures ayant de faibles irrégularités géométriques.

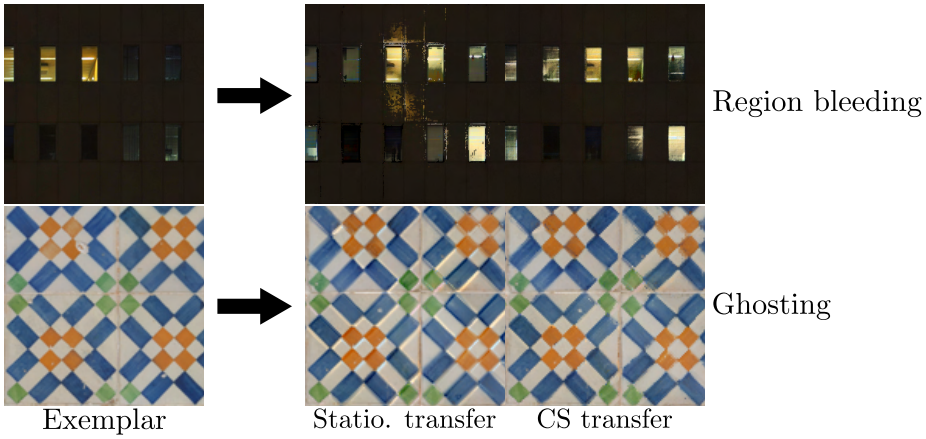
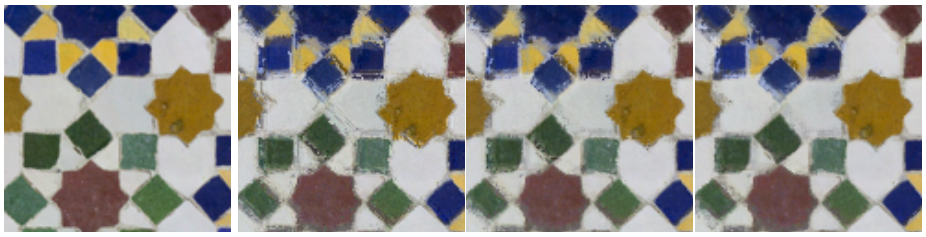


FIGURE 4.9 – Transfert des statistiques d'ordre 1 : comparaison entre le transfert stationnaire (gauche) et cyclostationnaire (droite) appliqués après l'algorithme de synthèse hautes performances [HN18]. Notre transfert CS aide à réduire les artefacts de couleur.

Estimation robuste de l'histogramme CS

Le transfert d'histogramme CS (section précédente) s'appuie sur les estimations $H^{\mathbf{x}}$. Cependant, pour des petits exemples avec de larges périodes, $|E^{\mathbf{x}}|$ devient petit, ce qui peut poser problème pour les estimations d'histogrammes et causer des artefacts liés à de fortes différences de transfert entre des texels voisins (voir la Figure 4.10). Pour rendre cette estimation plus robuste, nous supposons une forme de stationnarité locale : dans une petite fenêtre de voisinage de taille δ autour de $\mathbf{x}$, nous supposons que les PC sont similaires. En pratique, nous ajoutons dans $E^{\mathbf{x}}$ les échantillons des estimations des PC voisines : $\delta = 1$ augmente l'échantillonnage par un facteur 9, $\delta = 2$ par un facteur 25, etc. Comme montré dans la Figure 4.10, dans les cas où c'est nécessaire, étendre le voisinage fournit un compromis entre le transfert stationnaire (mauvaise couleurs, flou visible) et cyclostationnaire (meilleures couleurs, bruit visible). Dans sa limite (lorsque le voisinage couvre l'exemple entier), toutes les estimations des PC sont égales et on retombe sur un transfert stationnaire. Dans nos expériences, nous n'utilisons cette amélioration que si la périodicité des motifs est perturbée.



Exemplar $d = 0$; sampling $\times 1$ $d = 1$; sampling $\times 9$ $d = 2$; sampling $\times 25$

FIGURE 4.10 – Estimation renforcée d'histogramme qui absorbe les estimations des composantes polyphases dans un petit voisinage de taille δ. L'extension de δ échange les artefacts de bruits pour des artefacts de flou et est utile pour les exemples à géométrie régulière perturbée.

4.3.4 Rendu temps réel et filtrage

Les algorithmes de synthèse présentés dans ce chapitre sont conçus pour la synthèse temps réel et le rendu, tout en gardant un faible coût mémoire. Le *high performance noise*, en particulier, ne requiert que trois accès mémoire pour le calcul de chaque texel. Le rendu nécessite la capacité de filtrer les réalisations en temps réel pour éviter les artefacts d'alliassage (voir section 1.5). Le *spot noise* [vW91] et le *high performance noise* [HN18] peuvent être filtrés en utilisant une MIP-map pré-calculée de l'exemple E ; cette technique est toujours valide avec notre modèle cyclostationnaire.

Nous appliquons directement notre transfert cyclostationnaire sur le résultat, comme Heitz et Neyret [HN18]. Deliot et Heitz [DH18] montrent cependant que l'application du transfert sur le résultat filtré n'est pas équivalent à filtrer après avoir appliqué le transfert sur le résultat, qui est l'approche souhaitée. Pour obtenir cette équivalence, ils calculent une table de recherche pré-filtrée par niveau de MIP-map : cette technique peut être adaptée en utilisant une table de recherche par niveau et par composante polyphase, mais l'implémentation pratique de cette technique reste à étudier.

4.4 Estimation des vecteurs de période

Dans cette section, nous étudions l'estimation des périodes $\mathbf{t_0}$ et $\mathbf{t_1}$ à partir d'un exemple E.

Des outils de détection de régularité ont été développées [PBCL09, DED05, CB11, CB13, LPVV17, LNS+15] permettent d'estimer manuellement, semi-automatiquement ou automatiquement des motifs répétitifs dans une texture. Ces approches sont utilisables aussi dans le cadre des processus cyclostation-

naires, mais manquent parfois de précision pour nos algorithmes par l'exemple, causant des artefacts de ghosting similaires à ceux observables dans la Figure 4.9. Quand bien même l'une ou l'autre technique pourrait être adaptée ou adaptable, elles sont créées pour des problèmes bien plus compliqués de vision et utilisent des programmes complexes comme des réseaux de neurones, qui sont lourds pour la tâche demandée.

Nous proposons d'utiliser les techniques les plus simples (par exemple, une estimation manuelle) pour faire une estimation initiale, et de raffiner cette estimation en exploitant les caractéristiques des textures cyclostationnaires.

Notre algorithme de raffinement est basé sur l'observation que $(\mathbf{t_0}, \mathbf{t_1})$ représente les translations vers l'élément suivant du motif répétitif, qui correspond à un autre échantillonage des mêmes composantes polyphases. Nous supposons que les composantes polyphases sont fortement cohérentes, c'est-à-dire que deux points $\mathbf{y}$ et $\mathbf{z}$ dans une composante polyphase donnée $E^{\mathbf{x}}$ sont similaires. Nous définissons donc

$$d_{PC}(\mathbf{t_0}, \mathbf{t_1}, E^{\mathbf{x}}) = \frac{1}{|E^{\mathbf{x}}|^2} \sum_{\mathbf{y} \in E^{\mathbf{x}}} \sum_{\mathbf{z} \in E^{\mathbf{x}}} \sqrt{(E(\mathbf{y}) - E(\mathbf{z}))^2}, \tag{4.10}$$

qui est une mesure de cohérence interne de $E^{\mathbf{x}}$. La mesure globale de cohérence des vecteurs $(\mathbf{t_0}, \mathbf{t_1})$ est alors donnée par

$$d(\mathbf{t_0}, \mathbf{t_1}) = \int_{\mathbf{x} \in X^-} d_{PC}(\mathbf{t_0}, \mathbf{t_1}, E^{\mathbf{x}}) \, \mathrm{d}\mathbf{x}. \tag{4.11}$$

Pour raffiner l'estimation initiale $(\mathbf{t_0}, \mathbf{t_1})$, nous effectuons la minimisation locale de d en utilisant une descente de gradient stochastique. Une visualisation de d est montré dans la Figure 4.11, où un point optimal est trouvé, même dans le troisième exemple qui exhibe des irrégularités géométriques (on peut d'ailleurs observer l'irrégularité par la large bande bleue). Il est important de noter que d n'est pas convexe ; cependant, la procédure converge de façon robuste vers un minimum local pour peu que l'estimation initiale soit suffisamment précise.

4.5 Résultats et discussion

Implémentation Une implémentation de la synthèse par bruit procédural instanciée par un spectre est disponible sur *shadertoy*[1].

Lorsqu'un exemple discret doit être échantillonné sur des positions continues (entre les texels), il est bilinéairement interpolé entre les 4 texels autour de cette position. Le *CS-spot noise* est implémenté avec l'amélioration de Galerne et

1. https://www.shadertoy.com/view/wdtcRs

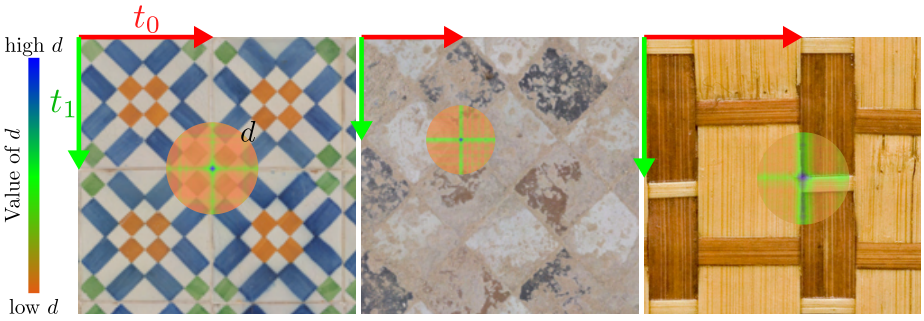


FIGURE 4.11 – Fonction de distance d appliquée autour d'un minimum local de d (qui correspond à la vérité terrain des vecteurs $\mathbf{t_0}$ et $\mathbf{t_1}$) sur diverses textures. d a seulement été évalué avec des vecteurs orthogonaux pour produire cette visualisation.

al. [GGM11] qui consiste à adoucir les bords de l'exemple pour supprimer les artefacts linéaires qu'ils causent dans la sortie. Comme pour le *high performance noise* original [HN18], sa version cyclostationnaire est plus facile à implémenter avec un exemple périodique ; sinon, les hexagones qui dépassent du bord doivent être évités, ce qui peut potentiellement réduire significativement la variété si les hexagones sont grands par rapport à l'exemple.

Les performances des algorithmes stationnaires et cyclostationnaires sont similaires. Le transfert cyclostationnaire des statistiques d'ordre 1 requiert cependant une utilisation mémoire plus importante (maximum 100% de la taille de l'exemple). Nos algorithmes héritent des avantages et des inconvénients des algorithmes originaux.

Synthèse de textures CSRP Nous avons étendu au cas cyclostationnaire plusieurs bruits issus de l'état de l'art : La Figure 4.3 est une implémentation directe du bruit à phase aléatoire ; le bruit de Gabor, le bruit à phase aléatoire locale, et le *phasor noise* sont montrés dans la Figure 4.4.

Nous avons créé une PSD variant spatialement et périodiquement en utilisant des fonctions ou une interpolation linéaire entre des PSD discrètes. D'autres opérateurs d'interpolation pourraient être utilisés, comme l'interpolation b-spline ou l'interpolation de Wasserstein [BPC16]. Une estimation de ce spectre à partir d'un exemple pourrait être un outil puissant. C'est cependant une tâche difficile car les estimations des composantes polyphases sont parcimonieuses. De plus, Guingo et al. [GSDC17] montrent que l'extraction des variations spatiales d'un spectre est un problème particulièrement difficile.

Synthèse par l'exemple Nous avons adapté le *spot noise* et le *high performance noise* De nombreux résultats sur des motifs différents sont montrés dans la Figure 4.6. Les textures sont non-bornées et continues et peuvent être échantillonnées à n'importe quelle position sur le plan infini. Un transfert d'histogramme étend l'application à des exemples ayant un histogramme non Gaussien, et aide à réduire les artefacts (Figure 4.9).

La Figure 4.12 montre des comparaisons entre ces deux algorithmes : la ligne 1 montre les exemples, la ligne 2 notre *spot noise* cyclostationnaire sans transfert d'histogramme (afin de montrer à quoi des réalisations CS Gaussiennes non altérées ressemblent), et la ligne 3 montre notre *high performance noise* avec transfert d'histogramme CS. La force majeure de notre *CS-spot noise* est de créer beaucoup de variété. Par exemple, les colonnes A et B exhibent des tuiles colorées pour lesquelles chaque couleur est proche du résultat d'une variable aléatoire Gaussiennes : le *spot noise* estime implicitement un processus pouvant avoir généré ce pavage et crée de nouvelles couleurs qui correspondent à ce processus. Ce résultat n'est possible que si la distribution des couleurs de chaque composante polyphase correspond au résultat d'un processus Gaussien ; sinon, il créé de fausses couleurs (colonnes C et D). Les statistiques d'ordre n prenant une forme Gaussienne, les réalisations ont de faibles variations d'intensité centrées autour de la moyenne, alors que de fortes variations sont envisageables dans l'exemple. Cette moyenne étant périodique, l'aspect régulier des exemples est tout de même conservé ; on observe ainsi des pertes de détails fins dans les colonnes E et F sans impact significatif sur l'apparence global des textures.

Les textures CS correspondant à la réalisation d'un processus Gaussien sont plutôt rares. Notre *CS-HPN* est plus à même de reproduire les détails fins et les structures complexes de ces textures, notamment ceux des colonnes E et F. Cependant, le *high performance noise* est avant tout un pavage : il crée donc des répétitions malgré les mélanges. De plus, ce pavage ne respecte pas non plus nécessairement les statistiques d'ordre supérieur à 1 : sur certains exemples comme les tuiles colorées uniformément (colonnes A, B, C et D), des motifs non voulus sont créés par le mélange linéaire.

Pour résumer, notre *spot noise* cyclostationnaire devrait être utilisé lorsque la variété est particulièrement importante, les détails fins ne le sont pas, ou l'exemple a des statistiques Gaussiennes cyclostationnaires ; notre *high performance noise* cyclostationnaire doit être utilisé à la place si les détails fins sont importants, que les tuiles aberrantes ne sont pas visuellement évidentes, et que la présence de répétitions n'est pas critique.

Notre transfert d'histogramme cyclostationnaire peut être utilisé pour améliorer les couleurs, comme il est capable de corriger les statistiques du premier ordre. Cependant, il n'agit pas sur les statistiques d'ordre supérieur, et échoue donc pour des cas difficiles comme la colonne D de la Figure 4.12

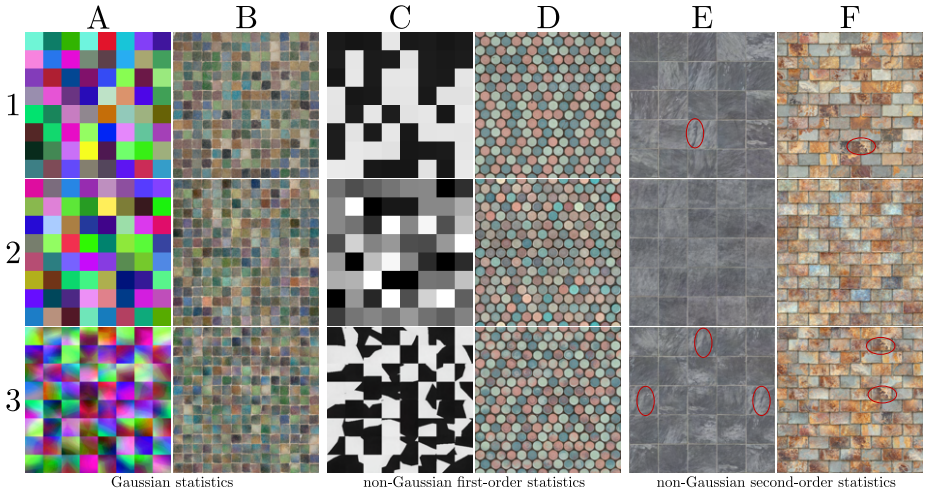


FIGURE 4.12 – Des exemples (ligne 1) synthétisés avec un *CS-spot noise* sans transfert d'histogramme (ligne 2), et avec un *high performance noise* CS + transfert d'histogramme CS (ligne 3). La nature Gaussienne de notre *CS-spot noise* le rend particulièrement efficace avec des exemples ayant des statistiques Gaussiennes cyclostationnaires ou proches (A, B). Notre *high performance noise* préserve à la place des structures non-Gaussiennes complexes (E, F), mais peut produire des répétitions visibles (entourées en rouge) et de nouvelles tuiles aberrantes (A, B, C, D).

Parmi les bruit procéduraux par l'exemple, il serait intéressant d'étudier le bruit texton [GLM17]. Cependant, une estimation robuste du texton cyclostationnaire n'est pas triviale.

Comparaison avec la synthèse de textures quasi-régulières Les textures CS représentent un sous-ensemble de textures quasi-régulières pour lesquelles il n'y a pas, ou très peu, d'altération de géométrie, mais d'importantes variations de couleurs. Liu et al. [LLH04] synthétisent les transformations de couleurs en utilisant une analyse des composantes principales (PCA pour principal component analysis). Dans nos expériences, nous avons observé que la PCA génère d'importantes pertes de contraste et une plus grande perte de détails comparée à notre *CS-spot noise*, comme montré sur la Figure 4.13. Comme Liu et al. [LLH04] séparent les tuiles individuelles, les tuiles de sorties doivent être cousues ensembles, et le temps réel est donc difficile à atteindre, à moins qu'un mélange de basse qualité soit utilisé. Haindl et al. [HH09] proposent une synthèse hors ligne pour une sous-classe de textures CS, qui sont caractérisées par un bruit stationnaire avec une structure périodique. Nous montrons dans la Figure 4.14 que même le *CS-spot noise* préserve mieux les détails fins.

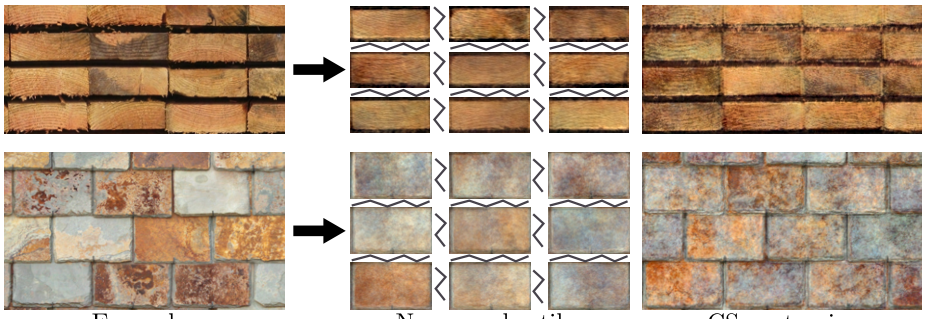


FIGURE 4.13 – Comparaison avec le synthétiseur de tuiles de Liu et al. [LLH04]. Les tuiles produites par leur technique s'appuyant sur la PCA (milieu) perdent des détails et du contraste, et nécessitent un tissage entre les tuiles pour produire des textures de sortie. Notre *CS-spot noise* évite mieux ces inconvénients.

Comparaison avec les techniques par optimisation et réseaux de neurones À l'heure actuelle, les techniques de synthèse par optimisation et réseaux de neurones ne sont pas faites pour le rendu temps réel de textures non bornées, contrairement aux modèles de synthèses par bruit. Ces techniques se concentrent plutôt sur la génération hors-ligne de textures basse résolution. *Deep correlations* [SCO17] tente de déterminer automatiquement des structures répétitives dans des exemples, mais la synthèse échoue pour certaines structures ; la raison de ces échecs est difficile à identifier. À l'inverse, notre bruit cyclostationnaire garantit le succès de la reproduction des textures cyclostationnaires, à condition qu'elles tombent dans la classe des textures cyclostationnaires bien reproduites par des processus Gaussiens cyclostationnaires. Nous montrons une comparaison côte à côte dans la Figure 4.15, exhibant différents résultats obtenus avec le code proposé par [SCO17] pour des textures ayant des structures similairement régulières.

Nous pensons que les techniques par réseaux de neurones pourraient être utiles dans la détection optimale de vecteurs de période selon des critères plus complexes, comme des critères visuels, et remplacer la méthode d'estimation des vecteurs de périodes décrite dans la section 4.4, ou fournir une une façon plus robuste d'estimer les statistiques d'ordre 1 cyclostationnaires, et remplacer la méthode de transfert de la section 4.3.3.

Variations géométriques Les variations géométriques ne sont pas gérées par notre approche. Liu et al. [LLH04] appliquent à la fois des variations géométriques et des variations de couleur. Les variations géométriques sont a priori compatibles avec nos synthèses par l'exemple, car elles sont appliquées indépendemment des variations de couleur, que nos algorithmes gèrent bien.

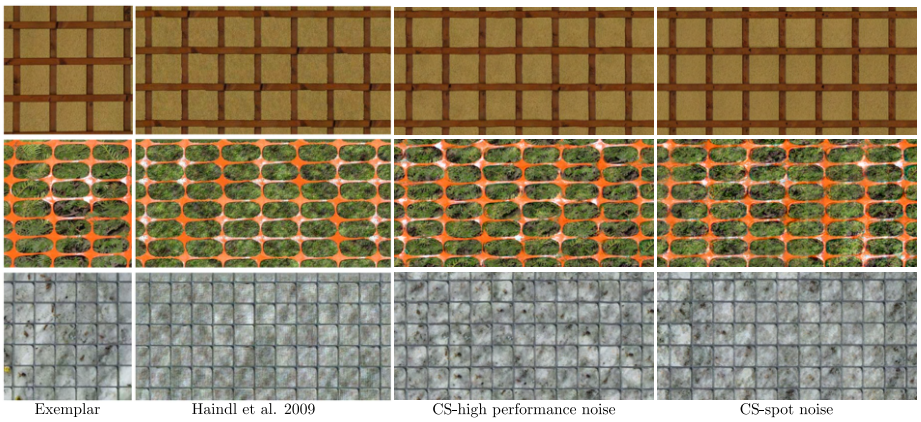


FIGURE 4.14 – Comparaison avec la synthèse hors ligne de Haindl et al. [HH09] qui reproduit une sous-classe de textures CS. Notre *CS-spot noise* et notre *CS-HPN* sont temps réel et préservent mieux les motifs des textures structurées et contrastées.

Les variations de géométrie sont crées par des déformations spatiales de la grille régulière dans le domaine des indices de la texture. Cependant, nous avons expérimenté que cela ne fonctionne bien que si les couleurs ont de faibles contrastes et de basses fréquences. Quand les couleurs sont caractérisées par de hautes fréquences et de forts contrastes, les déformations spatiales génèrent des artefacts visuels évidents comme des détails trop étirés, des distorsions artificielles et des artefacts d'échantillonage.

Il est possible que ces exemples puissent être théoriquement générés par des processus cyclostationnaires complexes mettant en scène des probabilités et des corrélations difficiles à modéliser ; l'incapacité de synthétiser ces exemples avec notre modèle tiendrait alors plus du fait que nos modèles soient Gaussiens que cyclostationnaires. Une piste de recherche est donc l'étude de la synthèse implémentant des processus qui soient cyclostationnaires, mais pas Gaussiens.

4.6 Conclusion

Dans ce chapitre, nous avons introduit les bruits cyclostationnaires Gaussiens à partir des modèles de processus pour la synthèse de textures donnés dans le chapitre 3. Les synthèses cyclostationnaires sont caractérisés par la périodicité de leurs statistiques, et généralisent les bruits stationnaires. Ils permettent en outre de synthétiser un sous-ensemble d'une classe de textures appelée « textures quasi-régulières » rapidement et avec des résultats de bonne qualité. Nous avons montré comment générer de telles textures de différentes façons : en modulant

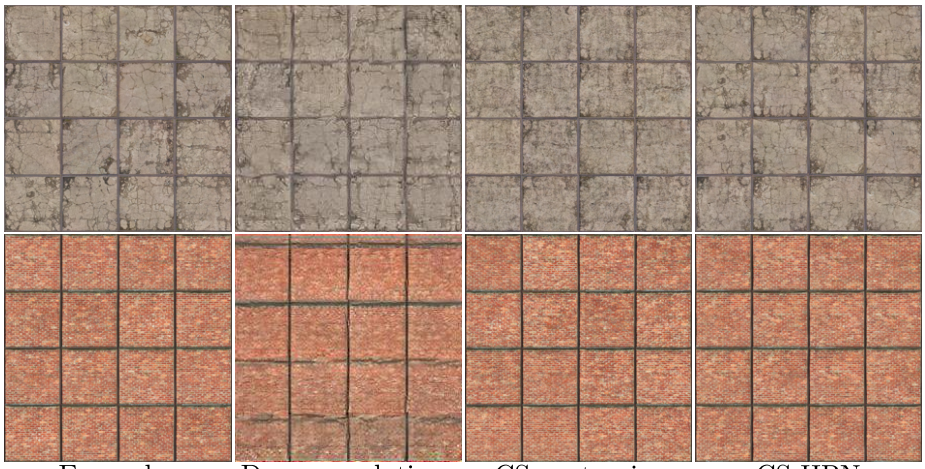


FIGURE 4.15 – Comparaison avec le synthétiseur par réseaux de neurones de Sendik et al. [SCO17]. Cette technique de synthèse ne s'adresse pas spécifiquement aux textures cyclostationnaires et la capture de la périodicité des statistiques échoue parfois. Additionnellement, il ne fonctionne que hors ligne et pour synthétiser de petites textures bornées, soit l'opposé des résultats de notre approche.

des signaux stationnaires, en créant un spectre de puissance variant spatialement et périodiquement, ou en passant par un exemple. Nous avons donc pu généraliser cinq bruits procéduraux : le bruit local à phase aléatoire, le bruit de Gabor, le *phasor noise*, le *spot noise* et le *high performance noise*. Nous avons également montré comment exécuter une correction des statistiques du premier ordre sur les sorties des synthèses par l'exemple pour étendre la synthèses à des textures qui ne sont pas nécessairement Gaussiennes au premier ordre. Nous avons montré comment nous estimons les vecteurs de période d'un exemple, nécessaires pour la synthèse par l'exemple. Enfin, nous avons comparé notre synthèse avec des méthodes issues de l'état de l'art et discuté de quelques limites et futurs travaux, dont certains sont élaborés dans le chapitre 6.

Chapitre 5

Filtrage de l'échange de contenus

5.1 Introduction

Dans le chapitre 2, nous avons vu que les synthèses par pavages permettent de synthétiser une large gamme de textures, y compris des textures quasi-régulières, peuvent conserver une partie du voisinage, et servir à texturer une surface en temps réel. Les synthèses par pavage apériodique avec tuiles régulières, notamment celles qui s'appuient sur le pavage de Wang [Sta97, CSHD03, NWT+05, LD06, XZJ+07] sont des processus discrets qui permettent de limiter une partie des répétitions qui apparaissent lorsqu'un exemple périodique est répété sur une surface. Cependant, une partie des répétitions reste, et ces répétitions sont alignées, créant à la fois des artefacts de répétition et d'alignement. Cet effet peut être évité en augmentant le nombre de tuiles, mais augmenter le nombre de tuiles requiert d'augmenter l'occupation mémoire de la synthèse, car les tuiles sont stockées explicitement dans la mémoire.

Les synthèses par pavage apériodique à tuiles irrégulières comme la synthèse par échange de contenus [VSLD13] et la synthèse par patchs interchangeables [KCD16] consistent à modifier à la volée le contenu de l'exemple répété périodiquement. La synthèse par échange de contenus [VSLD13] consiste à générer à la volée des tuiles à partir d'une seule tuile périodique en remplaçant certaines régions appelées « patchs » par les contenus de régions de la même forme appelées « contenus », provenant de l'exemple (voir Figure 5.1). Cette synthèse revient à un pavage apériodique à tuiles régulières avec un très grand nombre de tuiles, mais sans augmenter significativement l'occupation mémoire, et limite les artefacts d'alignement en fonction du nombre de contenus par patch.

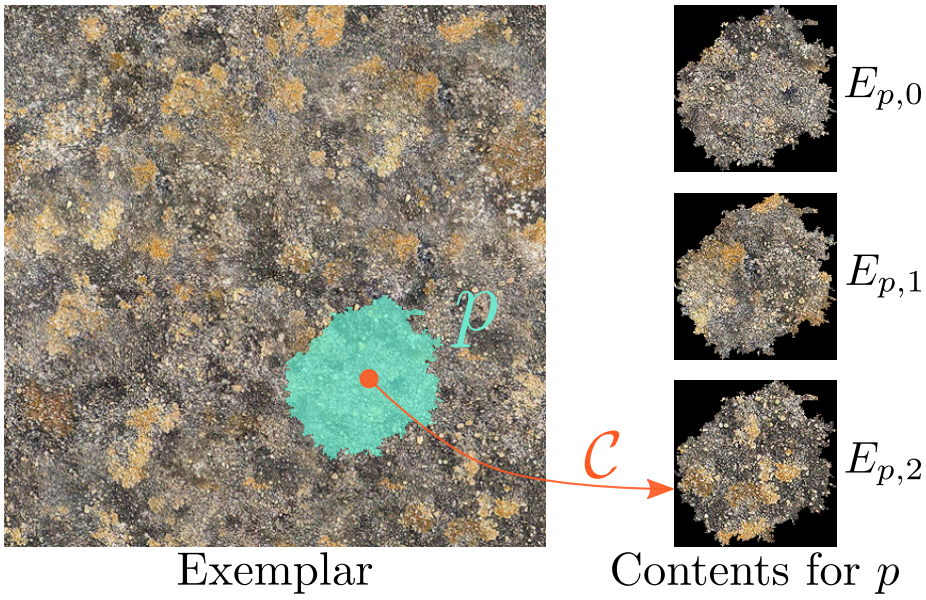


FIGURE 5.1 – Sélection de contenus. Pendant le rendu, en temps réel, chaque occurence de patch p est remplie avec un contenu $E_{p,c}$ choisi aléatoirement par $\mathcal{C}$ parmi un ensemble de candidats pré-définis.

Un des enjeux intrinsèques à la synthèse de textures temps réel est la possibilité de filtrer les textures générées en temps réel afin d'éviter les artefacts de filtrage comme les effets de Moiré et les scintillements. Bien que le filtrage en temps réel de ces synthèses n'est pas trivial, les synthèses par pavage apériodique à tuiles régulières peuvent être pré-filtrées de façon à permettre le filtrage en temps réel [Wei04, Lef08]. Cependant, les synthèses par pavage apériodique à tuiles irrégulières ne disposent pas de méthode de pré-filtrage. Le problème est difficile parce que des contenus adjacents choisis à la volée dans la sortie ne sont pas nécessairement adjacents dans l'exemple, rendant un simple pré-filtrage de l'exemple incorrect. Il est possible de les filtrer à la volée en intégrant l'empreinte de chaque pixel qui arrive sur la surface texturée, mais comme expliqué dans le chapitre d'introduction, cette méthode n'est pas adaptée aux applications temps réel car elle est trop coûteuse.

Nos travaux, publiés dans Lutz et al. [LSLD19], nous ont amené à montrer comment filtrer efficacement les textures générées par la synthèse par échange de contenus. Notre solution consiste à stocker explicitement les basses résolutions des contenus, chacun d'entre eux étant pré-filtré indépendemment. La *patch-map*, une structure qui indique le patch auquel chaque texel appartient, est aussi MIP-mappée en utilisant une opération booléenne OU bit à bit spéciale. Nous montrons que notre méthode permet un filtrage pour toute résolution tout en restant relativement peu coûteuse en termes d'occupation mémoire et

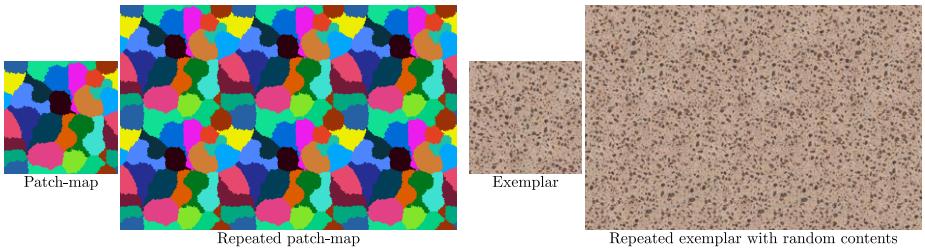


FIGURE 5.2 – L'échange de contenus est calculé en répétant une patch-map périodiquement. Comme les contenus associés à chaque patch sont choisis aléatoirement à la volée, la sortie n'est pas un pavage périodique.

de nombre de calculs. Notre méthode est détaillée dans la section 5.2 et 5.3 et évaluée dans la section 5.4. Nous donnons quelques perspectives dans la section de conclusion 5.5.

5.2 Énoncé du problème

5.2.1 Échange de contenus

Nous donnons ici une description détaillée de l'algorithme de Vanhoey et al. [VSLD13]. Nous considérons une seule tuile périodique qui est l'exemple utilisé pour la synthèse. La tuile est partitionnée en P patchs numérotés $1 \leqslant p \leqslant P$. Ils sont encodés par une patch-map $\mathcal{P}$ tel que $\mathcal{P}(\mathbf{x}) = p$ si le texel $\mathbf{x}$ appartient au patch p. Comme la tuile est supposée périodique, la patch-map est également répétée périodiquement comme sur la Figure 5.2. Soit $E_{p,c}$ le c-ème contenu du patch p : $E_{p,c}(\mathbf{x})$ donne une valeur de texel (Figure 5.1). Sans perte de généralité, nous supposons que C contenus sont attribués à chaque patch, numérotés $1 \leqslant c \leqslant C$. À chaque répétition de la tuile, chaque patch p est répété, mais son contenu change : une fonction $\mathcal{C}(p)$, qui contrôle la part d'aléa de la synthèse, donne un numéro de contenu c aléatoire. La texture synthétisée est alors définie, pour chaque position $\mathbf{x}$, par

$$I(\mathbf{x}) = E_{\mathcal{P}(\mathbf{x}),\mathcal{C}(\mathcal{P}(\mathbf{x}))}(\mathbf{x}). \tag{5.1}$$

5.2.2 Filtrage de texture

Lors du rendu, chaque pixel de l'écran a une empreinte qui est projetée dans l'ensemble d'indices de la texture. Le filtrage nécessite d'approximer l'intégrale de I sur cette empreinte. Dans un pipeline standard, où I est pré-calculé, il est

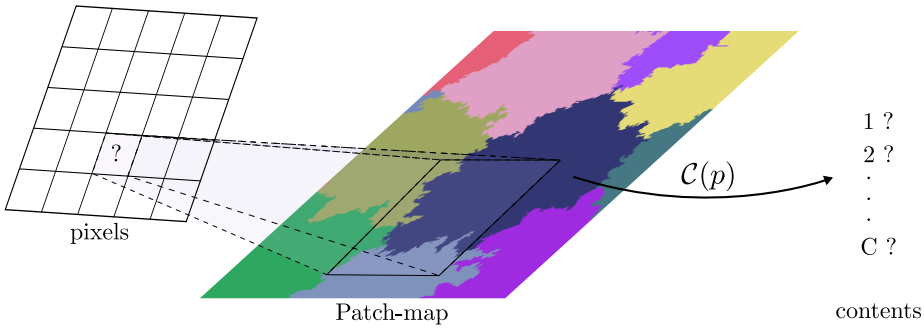


FIGURE 5.3 – Lors du rendu, l'empreinte de chaque pixel est projetée sur la surface texturée. L'empreinte peut inclure différents patchs, et le contenu attribué à chaque patch p dans l'empreinte par $\mathcal{C}$ n'est pas connu à l'avance, rendant difficile le calcul d'une MIP-map adaptée.

possible de s'appuyer sur la MIP-map $I^{(\ell)}$ de la texture : au fur et à mesure que le niveau ℓ augmente, I est sous-échantillonné en moyennant sur des carrés de taille $2^\ell \times 2^\ell$ de résolution décroissante (voir Figure 1.6). Dans notre contexte, la difficulté consiste à calculer $I^{(\ell)}$ en temps réel. Les synthèses par pavage résolvent ce problème en MIP-mappant les tuiles avant le rendu et en assemblant les résolutions appropriées ensemble en temps réel [Wei04, Lef08]. Cette approche ne fonctionne pas dans notre cas, parce que les patchs ont des formes irrégulières. Dans notre cas, à basse résolution, les régions carrées sur lesquelles les texels sont moyennés peuvent chevaucher plusieurs patchs différents, alors que les contenus ne sont connus qu'en temps réel (voir Figure 5.3). Il n'est pas non plus possible de calculer toutes les combinaisons possibles de contenus et de les pré-filtrer individuellement : cette méthode produirait C^P tuiles, qui ne peuvent pas être stockées et pré-filtrées par manque de mémoire.

5.3 Filtrage de la synthèse

Notre méthode suit la même intuition que le filtrage du pavage apériodique à tuiles régulières [Wei04, Lef08] :

Comme on ne peut pas se permettre d'assembler I avant le rendu et de le pré-filtrer en tant que $I^{(\ell)}$, nous pré-filtrons les données de façon à calculer $I^{(\ell)}$ à la demande, en temps réel. Nous obtenons ce résultat en :

— Pré-filtrant la patch-map comme une MIP-map $\mathcal{P}^{(\ell)}$ de façon à ce que les chevauchements entre les patchs puissent être détectés à chaque niveau ℓ (Figure 5.4).

— Pré-filtrant tous les contenus $E^{(\ell)}_{p,c}$ et en les empaquetant dans un atlas de texture (Figure 5.6).

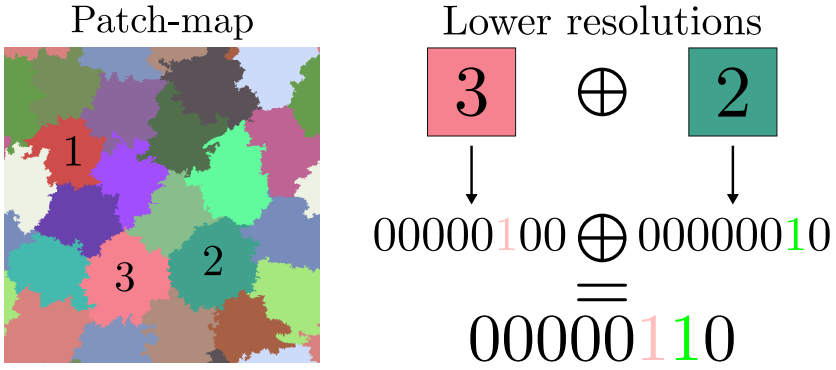


FIGURE 5.4 – Adaptation de la patch-map. Au lieu d'encoder chaque patch p par un nombre naturel, ils sont représentés avec un bit à 1, c'est-à-dire par 2^{p-1} en binaire. Cette structure permet de représenter un mélange entre deux patchs ou plus par un « OU » binaire.

— Assemblant $I^{(\ell)}$ en temps réel depuis $\mathcal{P}^{(\ell)}$ et l'atlas de contenus (équation (5.2)).

5.3.1 Pré-filtrage de la patch-map

La patch-map est encodée par un *bitmask* de P bits : à la plus haute résolution, le p-ème bit de $\mathcal{P}^{(0)}(\mathbf{x})$ est mis à 1 si $\mathbf{x}$ appartient au patch p, comme montré dans la Figure 5.4. Ensuite, la MIP-map est calculée en utilisant une opération OU bit à bit, à la place d'une moyenne. Concrètement, la valeur $\mathcal{P}^{(\ell)}(\mathbf{x})$ au texel $\mathbf{x}$ est le OU bit à bit de ses quatre parents dans le niveau précédent $\mathcal{P}^{(\ell-1)}$.

La conséquence est que le p-ème bit de $\mathcal{P}^{(\ell)}(\mathbf{x})$ est égal à 1 si et seulement si le patch p chevauche le texel $\mathbf{x}$ qui correspond à un carré de taille $2^\ell \times 2^\ell$ à haute résolution. Ces patchs sont ceux pour lesquels les contenus contribuent à la valeur finale $I^{(\ell)}(\mathbf{x})$.

5.3.2 Pré-filtrage des contenus

Pour calculer le pré-filtrage des contenus $E^{(\ell)}_{p,c}$, nous exécutons l'algorithme suivant. Pour chaque contenu c de chaque patch p :

— La tuile est remplie avec des valeurs nulles partout sauf sur le patch p, qui est remplie avec le contenu $E_{p,c}$ (Figure 5.5, à gauche).

— La tuile est MIP-mappée avec le moyennage classique de 4 texels pour 1. Les texels au bord sont alors mélangés avec les valeurs nulles autour du

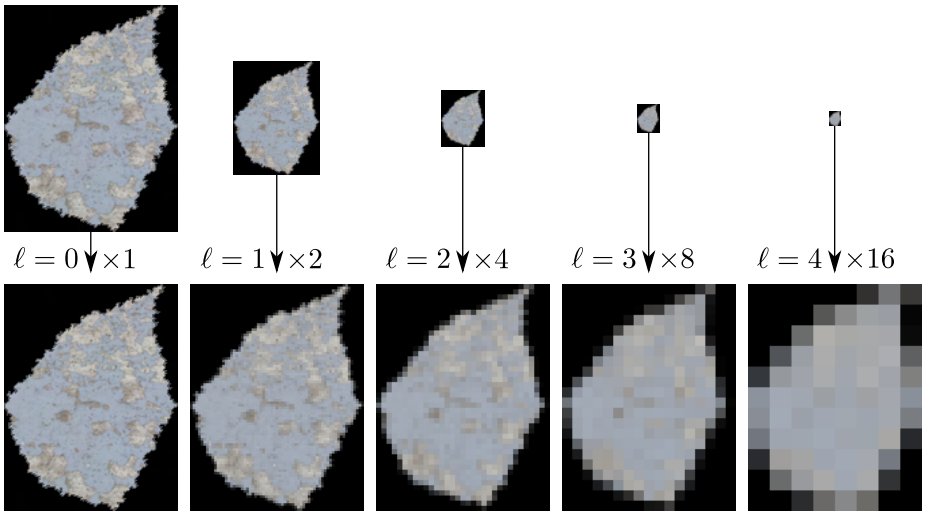


FIGURE 5.5 – Chaque contenu $E_{p,c}$ est pré-filtré à différents niveaux de résolution. Haut : résolution fixe. Bas : taille fixe.

contenu (Figure 5.5, droite).

— À chaque niveau $\ell \geq 1$, le contenu $E^{(\ell)}_{p,c}$ est coupé et paqué dans un atlas (voir section 5.3.4 et Figure 5.6).

Il est important que la MIP-map soit calculée avec le contenu à son emplacement dans la tuile, faute de quoi la forme du contenu risque de ne pas être correcte à basse résolution.

5.3.3 Assembler la sortie filtrée

Quand le rendu de la texture est effectué, il faut calculer I à la demande pour chaque niveau ℓ et à chaque texel $\mathbf{x}$. Nous le calculons par

$$I^{(\ell)}(\boldsymbol{x}) = \sum_{p \in \mathcal{P}^{(\ell)}(\boldsymbol{x})} E^{(\ell)}_{p,\mathcal{C}(p)}(\boldsymbol{x}), \tag{5.2}$$

où la somme mélange tous les contenus $E^{(\ell)}_{p,\mathcal{C}(p)}$ qui chevauchent $\mathbf{x}$ au niveau ℓ. Les patchs p concernés sont identifiés grâce à la patch-map pré-filtrée $\mathcal{P}^{(\ell)}$. En conséquence, l'équation (5.2) est une évaluation exacte de la MIP-map de I, sans besoin de la stocker explicitement. Les opérations de filtrage standard, comme le filtrage trilinéaire (mélange entre deux niveaux adjacents de MIP-map variants selon la distance à la surface) ou anisotrope (mélange avec les échantillons proches dans l'empreinte du pixel sur la surface de façon à prendre en compte l'angle de la caméra), peuvent alors être appliqués comme avec n'importe quelle MIP-map.

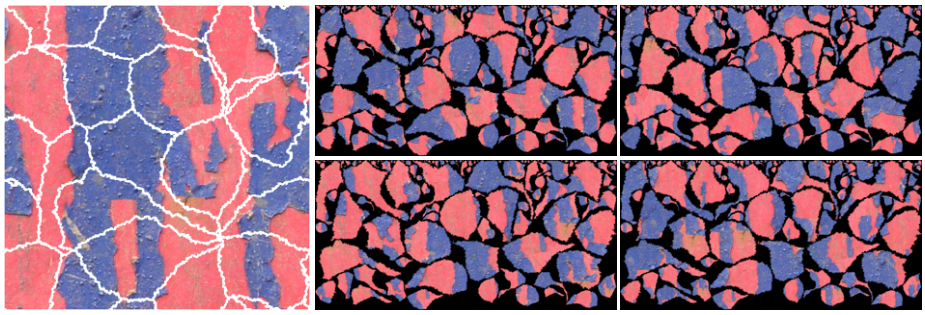


FIGURE 5.6 – Stockage des contenus. Gauche : la haute résolution ($\ell = 0$) n'a pas besoin de stockage spécial. Droite : pour chaque contenu identifié c, tous les contenus de basse résolution $E^{(\ell)}_{p,c}$ sont empaquetés dans un atlas.

5.3.4 Stockage et performances

Stockage. Notre méthode demande de stocker les patch-maps à toutes les résolutions comme un tableau de *bitmasks* (de P bits pour P patchs), ainsi que les contenus pour toutes les résolutions. Pour limiter l'impact des formes irrégulières de patchs dans la mémoire, nous construisons un atlas (un pour chaque numéro de contenu c) qui inclut tous les contenus de basse résolution $E^{(\ell)}_{p,c}$ de chaque patch ($\ell \geq 1$, $p \in \mathcal{P}$), comme montré dans la Figure 5.6. Il est bon de noter que nous ne stockons que les niveaux de MIP-map isotropes, depuis lesquels le filtrage anisotrope peut être calculé. Comme les contenus de plus haute résolution n'ont pas les mêmes problèmes de chevauchement que ceux à plus basse résolution, ils n'ont pas besoin d'être stockés dans un atlas, mais peuvent être accédés directement depuis la tuile d'entrée en utilisant leurs *offsets* de translation comme dans Vanhoey et al. [VSLD13]. L'occupation mémoire est illustrée en Figure 5.7 et détaillée dans le tableau 5.1. L'utilisation mémoire grandit linéairement avec le nombre N de pixels et le nombre C de contenus. Pour un usage typique, avec une tuile de de 1024^2 texels et 6 contenus par patch, le coût additionnel est d'à peu près $5\times$, tout en rendant possible la génération de textures de haute résolutions et de tailles non bornées pour moins de $20Mo$.

Performances. Nous avons effectué le rendu de nos résultats sur une surface raide couverte de $2.6G$ texels (51200×51200) en utilisant une fenêtre de 800×700 pixels. Nous avons activé le filtrage trilinéaire, le filtrage anisotrope et la magnification linéaire (les texels sont interpolés de façon à pouvoir calculer l'empreinte sur des positions non entières de texels). Le rendu produit 90 images par seconde sur une carte NVidia GTX 1060.

Lorsque la résolution ℓ augmente, le nombre d'accès texture requis pour la somme de l'équation (5.2) augmente jusqu'à un maximum de P accès texture,

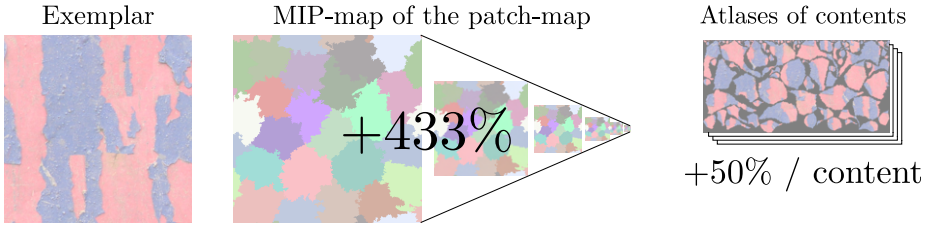


FIGURE 5.7 – Résumé de l'occupation mémoire par rapport à la taille de l'exemple. La patch-map prend une occupation mémoire fixe de +433% (+300% pour passer de 8 bits à 32 bits et +133% pour les résolutions supplémentaires – toujours en 32 bits). Chaque atlas de contenu prend à peu près une occupation mémoire de +50% par contenu.

Données (N/C)	Sans filtrage (Mo)		Notre filtrage (Mo)	
	contenus	patch-map	contenus	patch-map
512^2 / 3	**0.79**	**0.26**	**2.0** (+150%)	**1.4** (+433%)
512^2 / 6	**0.79**	**0.26**	**3.2** (+300%)	**1.4** (+433%)
1024^2 / 3	**3.1**	**1.05**	**8.1** (+150%)	**5.6** (+433%)
1024^2 / 6	**3.1**	**1.05**	**13** (+300%)	**5.6** (+433%)
Asymptotique	$\Theta(N)$		$\Theta(NC)$	

TABLE 5.1 – Analyse de l'occupation mémoire pour différentes tailles d'exemples N et différents nombres de contenus C. Le coût additionnel pour la patch-map est fixe (+433%). Le coût pour les contenus est +50% par contenu, qui doit être comparé à +33% si l'empaquetage dans l'atlas était parfait.

comme montré sur la Figure 5.8.

5.3.5 Mélange des contenus

Les synthèses par patchs, qui sont notamment utilisées dans le pré-calcul des synthèses par pavage apériodique à tuiles irrégulières, ont tendance à introduire des discontinuités visibles sur les bords des patchs (Figure 5.9, gauche). Ceux-ci peuvent être cachés en mélangeant les contenus adjacents (Figure 5.9, droite). Notre technique peut filtrer les contenus mélangés en effectuant les changements suivants :

— Les patchs sont agrandis par dilatation et changent plusieurs bits à 1 dans $\mathcal{P}^{(0)}(\boldsymbol{x})$ quand $\mathbf{x}$ est près d'un bord.
— Des poids de mélange sont définis sur les zones de chevauchement créées par la dilatation. Dans nos tests, nous avons utilisé une simple distance décroissante aux bords des patchs.
— Les $E_{p,c}$ sont pondérés d'après les contenus de haute résolution $E^{(0)}_{p,c}$.

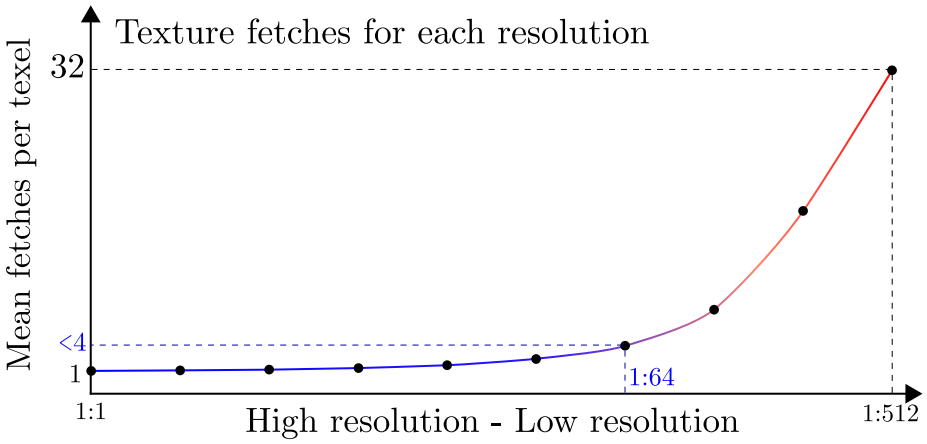


FIGURE 5.8 – Nombre d'accès texture de notre technique en fonction de la résolution de l'exemple. Jusqu'à une résolution de 1 : 64, notre filtrage demande en moyenne moins de 4 accès texture pour le calcul de chaque texel, mais ce nombre augmente ensuite rapidement à chaque niveau jusqu'à P (ici, $P = 32$).

Ensuite, notre pipeline (sections 5.3.1, 5.3.2, et 5.3.3) reste inchangé, et génère les basses résolutions qui intègrent automatiquement les pondérations dans les contenus filtrés. Le mélange est donc quasiment gratuit : un léger coût mémoire additionnel est causé par le stockage des poids de mélange et plus de termes sont impliqués dans la somme de l'équation (5.2).

La Figure 5.9 montre l'amélioration d'un simple mélange basé sur la distance des texels aux bords des patchs. Dans les régions où les coutures entre les différents contenus reste visible, le mélange réduit l'impact visuel.

5.4 Résultats

La Figure 5.10 présente les résultats de notre technique de filtrage : les vues sont indistinguables de la vérité terrain (milieu), tandis que les résultats non filtrés sont bruités à basse résolution. Notre technique est également compatible avec d'autres améliorations d'anti-aliassage comme l'anti-aliassage temporel. Le mélange de la section 5.3.5 a été activé pour tous ces résultats.

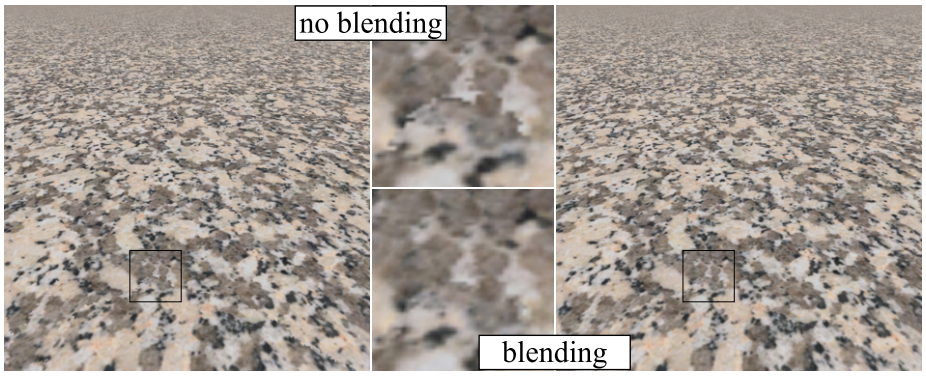


FIGURE 5.9 – Rendu d'une texture avec (bas/droite) et sans (haut/gauche) mélange aux frontières entre les patchs. Les coutures entre les différents contenus sont moins visibles lorsque le mélange est activé.

5.5 Conclusion

Nous avons présenté dans ce chapitre une technique de filtrage pour la synthèse par échange de contenus qui traite le problème difficile de gérer différents contenus provenant de différentes régions de l'exemple et choisis aléatoirement à la volée pendant le rendu. Le pré-filtrage de ces contenus et de la patch-map permet une évaluation à la volée de la MIP-map de la texture générée. Pour obtenir ce résultat, notre méthode a besoin de stocker explicitement les basses résolutions pré-filtrées des contenus ainsi que la patch-map à chaque résolution ; pourtant, le coût mémoire total reste acceptable, typiquement quelques dizaines de Mo pour des textures non bornées. De plus, nous avons montré que notre algorithme est capable d'effectuer un mélange aux bords des patchs par une simple modification de notre pré-calcul. Ce mélange permet de réduire l'impact visuel des coutures apparaissant potentiellement pendant le rendu d'une surface texturée. Cette technique fonctionne en temps réel sur une carte graphique moderne et peut être implémentée avec un langage de *shader* standard.

Limites Notre filtrage n'est pas parfait : lorsque l'empreinte d'un pixel sur la surface texturée a une taille supérieure à une tuile périodique, alors chaque tuile est représentée par une couleur différente, générant du bruit. Pour filtrer correctement la surface à ce moment-là, il faudrait mélanger la couleur de plusieurs tuiles adjacentes, ce qui est trop coûteux à calculer, car le calcul de la couleur d'une seule tuile de résolution minimale est déjà coûteux et s'additionne pour toutes les tuiles. Habituellement, ce problème n'est pas visible, car les textures de résolution minimale se ressemblent ; s'il l'est, nous pouvons afficher la couleur moyenne de toutes les tuiles possibles à la place, mais ce n'est qu'une approximation du résultat correct.

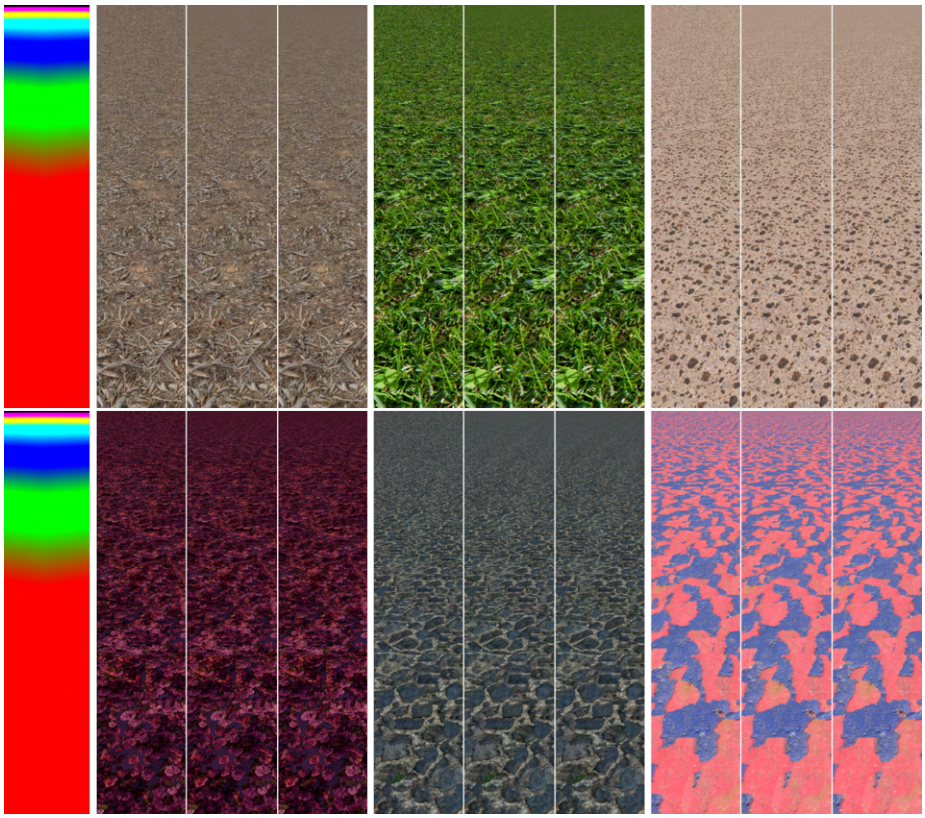


FIGURE 5.10 – Exemples de résultats pour divers exemples. Notre méthode de filtrage (droite) est comparée à la vérité terrain (milieu) et sans filtrage (gauche). La vérité terrain est calculée par un filtrage exact de la haute résolution. La vue la plus à gauche indique les niveaux de MIP-map utilisés.

De plus, même si la technique est compatible avec les applications temps réel, le calcul des plus basses résolutions est particulièrement long (jusqu'à autant d'accès texture que de nombre de patchs différents, soit entre 16 et 32).

Perspectives Nous nous intéressons à l'étude de l'extension de notre technique de filtrage. Notamment, elle a été spécifiquement conçue pour fonctionner avec la synthèse par échange de contenus [VSLD13], mais ne peut pas être appliquée directement à la synthèse par patchs interchangeables [KCD16] à cause de différences dans les données. Il pourrait être possible d'étendre notre algorithme pour pouvoir filtrer cette synthèse. De plus, nous pensons que notre algorithme peut fonctionner avec d'autres synthèses par pavage apériodique (par exemple le pavage de Wang [Sta97], qui dispose d'une méthode de pré-filtrage décente mais pas parfaite [Wei04]) en considérant l'ensemble des tuiles comme diffé-

rents contenus d'un unique patch carré. Des expériences pourraient être menées pour valider cette hypothèse. Il pourrait également être intéressant d'adapter notre méthode à des données plus complexes qui ne peuvent pas être filtrées linéairement, comme les cartes de normales ou les cartes d'ombre.

Chapitre 6

Conclusion et perspectives

Nous avons présenté deux contributions principales visant à améliorer la panoplie d'outils de synthèse de textures, en proposant de nouvelles façons de synthétiser des textures régulières d'une part, et une façon de filtrer la sortie d'une synthèse existante d'autre part. Nous donnons d'abord un bilan de ces travaux, en nous penchant sur l'intérêt que ces contributions apportent pour l'état de l'art, avant de donner des axes de direction pour de futurs travaux.

6.1 Extension à la cyclostationnarité

Dans le chapitre 4, nous avons montré qu'il était possible de synthétiser des textures à motifs organisés régulièrement à l'aide de processus cyclostationnaires, que nous avons présentés en détail dans le chapitre 3. Une fois les processus stationnaires connus, l'intuition des processus cyclostationnaires est simple. Contrairement aux processus stationnaires, leurs statistiques varient spatialement, mais pas n'importe comment : elles varient périodiquement selon une période $\mathbf{t_0}$. En l'occurrence, pour les textures, qui sont des signaux en deux dimensions, elles varient également selon une autre période $\mathbf{t_1}$, et les périodes sont des vecteurs de $\mathbb{R}^2$. Ces deux vecteurs coïncident parfaitement avec les vecteurs proposés par Liu et al. [LH05], qui décrivent le motif périodique de la texture, et peuvent donc être trouvés avec les mêmes méthodes.

Nous avons proposé un modèle de bruit cyclostationnaire permettant de créer des motifs dont les statistiques sont périodiques, sans que le motif lui-même ne le soit. Cette contribution permet de synthétiser une forme de textures quasi-régulières que nous avons appelées « textures cyclostationnaires » et dont des exemples sont donnés en Figure 6.1. Nos travaux se décomposent en deux

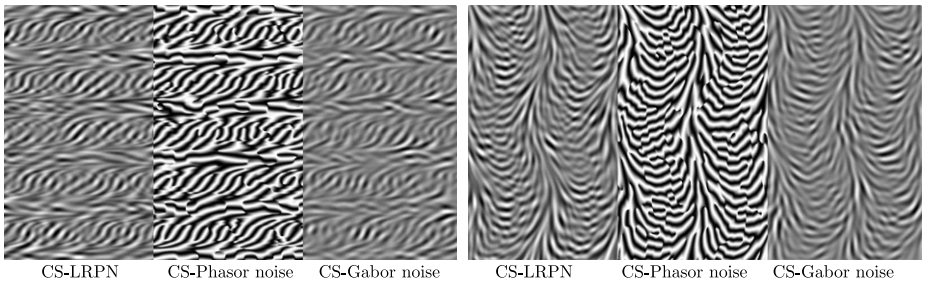


FIGURE 6.1 – Motifs cyclostationnaires générés à l'aide du *local random-phase noise* [GSV+14], *phasor noise* [TEZ+19], et bruit de Gabor [GLLD12], adaptés à une forme cyclostationnaire.

familles d'algorithmes : les algorithmes instanciés à partir de spectres, et ceux instanciés à partir d'un exemple.

Les algorithmes utilisés pour l'instanciation à partir de spectres exploitent le modèle algorithmiques du bruit à phase aléatoire stationnaire de Galerne et al. [GGM11], qui consiste à randomiser les phases dans le domaine spectral. Dans le cas cyclostationnaire que nous proposons, le spectre est défini de façon à varier spatialement et périodiquement. Les résultats sont générés en temps réel et permettent de représenter une nouvelle gamme de motifs.

Les algorithmes utilisés pour la synthèse par l'exemple exploitent le modèle algorithmique de la convolution parcimonieuse stationnaire pour adapter les algorithmes du *spot noise* [vW91] et du *high performance noise* [HN18] au cas cyclostationnaire. Nous avons également généralisé le transfert d'histogramme proposé par Heitz et Neyret [HN18], pour améliorer l'aspect des textures générées en sortie en permettant de reproduire des textures dont les statistiques d'ordre 1 (les histogramme estimés des composantes polyphases) ne sont pas Gaussiennes ; les résultats peuvent être générés en temps réel, sont de haute qualité, et peuvent être évalués en tout point d'une surface texturée.

La cyclostationnarité n'a pas directement été exploitée dans la littérature concernant la synthèse de textures avant nous. Cependant, certaines idées s'y retrouvent. Par exemple, l'idée de mélanger les contenus de l'exemple en contrôlant la position des mélanges à partir des vecteurs de périodes $\mathbf{t_0}$ et $\mathbf{t_1}$ se retrouve dans la synthèse de textures quasi-régulières de Liu et al. [LH05]. De plus, dans la synthèse instanciée par spectres, l'idée de modifier l'amplitude du spectre en fonction de la position se retrouve dans certaines implémentations du bruit de Gabor.

Nos travaux offrent de nombreuses perspectives que nous listons dans les sous-sections suivantes.

6.1.1 Création et stockage des spectres des composantes polyphases

Les algorithmes du bruit à phase aléatoire, bruit local à phase aléatoire, bruit de Gabor, et *phasor noise* cyclostationnaires utilisent un spectre variant spatialement (voir équation (4.4), Figure 4.4), qui correspond à l'ensemble des spectres des composantes polyphases. L'utilisation de ce spectre pose plusieurs problèmes. L'un d'entre eux est que le spectre soit défini pour toute position $\mathbf{x}$ dans X^-, comme si un spectre « stationnaire » était attribué à chaque position $\mathbf{x} \in X^-$. Comme nous définissons en plus des processus continus, il est impossible de stocker explicitement le spectre entier.

Spectre analytique La solution que nous avons employé pour nos bruits procéduraux cyclostationnaires est similaire à celle de Lagae et al. [LLDD09], qui consiste à définir le spectre de façon analytique. Les spectres stationnaires sont définis analytiquement par un ensemble de lobes, définis par un intervalle d'orientations et de fréquences dans le domaine spectral. Le spectre cyclostationnaire est plus délicat à définir, car il varie spatialement : on doit donc définir une fonction de variation pour chaque lobe. Dans nos exemples, notamment ceux de la Figure 6.1, nous utilisons uniquement des bi-lobes (un lobe répété par symétrie centrale) dont l'orientation et/ou l'intervalle d'orientations varie.

Interpolation de spectres Nous avons exploré brièvement l'interpolation entre des spectres, qui permettrait de définir un spectre pour n'importe quelle position à partir d'un petit ensemble de spectres. Les spectres stockés sont alors des « points de contrôle » de l'interpolation (que nous désignons comme les « spectres de contrôle »). La difficulté de l'interpolation est de choisir une fonction d'interpolation qui convienne à l'application que l'utilisateur veut faire du bruit cyclostationnaire. On peut par exemple choisir d'interpoler linéairement les spectres, à l'aide de l'interpolation de Wasserstein [BPC16] (basée sur le transport optimal de mesures), ou d'interpoler les paramètres du spectre analytique (l'orientation des lobes en radians, leur taille...), que nous désignerons comme l'*interpolation analytique*.

Le résultat de bi-lobes soumises à différentes interpolations est illustré dans la Figure 6.2. L'interpolation linéaire est facile à calculer et efficace, mais revient à effectuer un mélange dans le domaine spatial, ce qui peut provoquer des effets de *ghosting*. L'interpolation de Wasserstein évite ces effets, mais est difficile à définir lorsque les énergies des lobes ne sont pas identiques et est difficile à calculer en temps réel ; l'interpolation analytique est une alternative efficace à l'interpolation de Wasserstein que nous avons exploitée dans nos exemples avec des bi-lobes uniquement, mais elle n'est définie que lorsque le nombre de lobes est identique dans tous les spectres de contrôle.

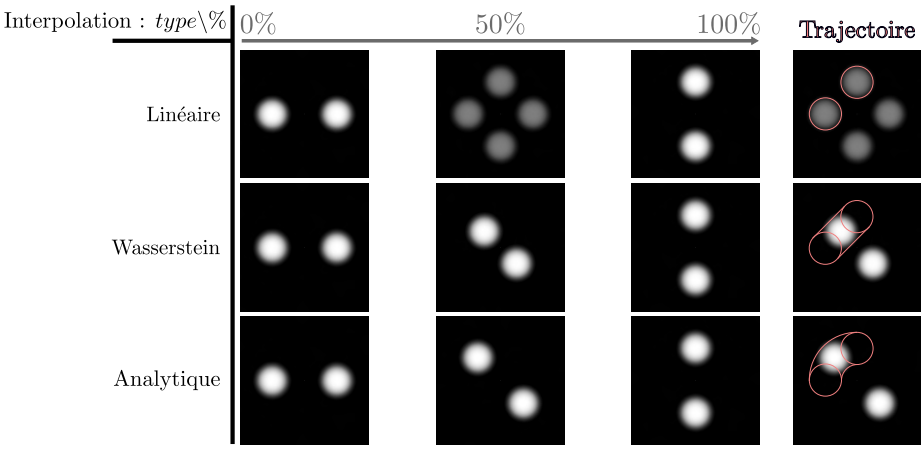


FIGURE 6.2 – Interpolations entre deux spectres représentant des bi-lobes (de haut en bas : interpolation linéaire, de Wasserstein et analytique). L'interpolation analytique est l'interpolation des paramètres du spectre analytique. La trajectoire des lobes au cours de l'interpolation est donnée en rose.

Synthèse spectrale par l'exemple L'adaptation de ces algorithmes pour la synthèse par l'exemple est également difficile. Le problème est que nous ne savons pas encore extraire les spectres des composantes polyphases, car les composantes polyphases n'incluent pas les corrélations avec les composantes polyphases voisines. Autrement dit, le second moment d'ordre 2 $\mathbf{m}_2^2(\mathbf{x}, \tau)$ du processus cyclostationnaire n'est pas égal au second moment d'ordre 2 de la $\mathbf{x}$-ème composante polyphase. Cependant, les deux mesures sont intrinsèquement liées et peuvent être calculées l'une à partir de l'autre [KGE18]. Comme il est possible d'estimer $\mathbf{m}_2^2$, de futures recherches pourraient montrer comment estimer de façon robuste les spectres des composantes polyphases à partir d'un exemple. Comme montré par Galerne et al. [GLLD12] pour le bruit de Gabor, un spectre analytique peut être calculé à partir d'un spectre stationnaire : il serait donc possible d'en déduire des spectres de contrôle et d'effectuer une interpolation analytique pour passer du discret au continu.

Il est important de mentionner l'utilisation d'une unique phase aléatoire dans notre modèle cyclostationnaire : une phase aléatoire identique est définie pour chaque spectre des composantes polyphases, mais l'invariance de la phase aléatoire n'est pas obligatoire. Les phases contrôlent, avec les spectres, les corrélations entre les différentes composantes polyphases. L'important pour préserver la propriété de cyclostationnarité est que les phases soient identiques par congruence, c'est-à-dire que les phases instantanées en $\mathbf{x}$ et en $\mathbf{x} + k\mathbf{t_0} + l\mathbf{t_1}$ soient les mêmes. Ainsi, il peut être possible d'explorer de nouvelles façons de générer d'autres motifs cyclostationnaires en faisant varier la phase en plus de l'amplitude ; c'est d'ailleurs une méthode employée par le *phasor noise* [TEZ+19]

pour pouvoir générer des motifs très contrastés.

6.1.2 Extension à d'autres bruits procéduraux

Nous avons présenté quelques algorithmes stationnaires aux paramètres instanciés par des spectres ou par un exemple, listés dans le tableau 6.1.2, mais d'autres algorithmes basés sur des processus stationnaires pourraient être adaptés.

	Statio. inst.	Statio. par l'ex.	CS inst.	CS par l'ex.
RPN	✓	✓	✓	
LRPN	✓	✓	✓	
Gabor noise	✓	✓	✓	
Spot noise	✓	✓		✓
Texton noise	✓	✓		
HPN		✓		✓
Phasor noise	✓		✓	

TABLE 6.1 – Liste d'algorithmes basés sur des modèles de bruit procédural, et les différentes adaptations. Dans l'ordre, de haut en bas : *random phase noise* [GGM11], *local random phase noise* [GSV+14], bruit de Gabor [LLDD09, GLLD12], *spot noise* [vW91], *texton noise* [GLM17], *high performance noise* [HN18], et *phasor noise* [TEZ+19]. De gauche à droite : stationnaire instancié (sans exemple), stationnaire par l'exemple, cyclostationnaire instancié, et cyclostationnaire par l'exemple.

Nous avons adapté le *spot noise* [vW91] au cyclostationnaire, mais sa version aux performances améliorées, le *texton noise* [GLM17], est plus difficile à adapter. Le *texton noise* est basé sur le « résumé » d'un exemple, qui forme une texture à plus petite taille appelée « texton ». Ce texton contient les informations fréquentielles les plus significatives, et le *spot noise* du texton, appelé *texton noise*, approxime le *spot noise* de l'exemple. La difficulté est qu'un exemple dont on considère le processus sous-jacent l'ayant généré comme étant un processus cyclostationnaire contient beaucoup moins d'informations qu'un exemple dont le processus sous-jacent l'ayant généré est considéré comme étant un processus stationnaire. Dans le cas stationnaire, la quantité d'informations dépend du nombre de texels de la texture, alors que dans le cas cyclostationnaire, elle dépend du nombre de texels de chaque composante polyphase, qui correspond au ratio entre la taille de la texture et la taille des vecteurs de période $\mathbf{t_0}$ et $\mathbf{t_1}$. Le manque d'informations rend donc difficile la simplification d'un exemple en un texton.

6.1.3 Amélioration du transfert d'histogramme

Le transfert d'histogramme présenté par Galerne et al. [GLR17] et adapté au temps réel par Heitz et Neyret [HN18] a été adapté au contexte cyclostationnaire.

Exemples irréguliers Comme mentionné dans la section précédente, les exemples des processus cyclostationnaires souffrent d'un manque d'information, car l'estimation des composantes polyphases $|E^{\mathbf{x}}|$ d'un exemple E est trop petite pour que les paramètres du processus cycloergodique sous-jacent X puissent être inférés précisément. L'estimation des statistiques d'ordre 1 de X, équivalente à l'ensemble des histogrammes des composantes polyphases $E^{\mathbf{x}}$, n'est donc pas toujours précise par rapport aux statistiques d'ordre 1 du processus.

À cause de ça, des artefacts sont observés pour les exemples ayant un degré d'irrégularité géométrique (comme celui de la Figure 4.10). Ces exemples sont mal gérés par nos algorithmes cyclostationnaires et ne sont pas pris en compte dans le modèle stationnaire. Nous avons proposé d'augmenter la précision de l'estimation pour ces exemples en étendant l'estimation pour chaque composante polyphase aux voisins proches pour augmenter la précision de l'estimation, en supposant que la valeur des voisins proches est similaire, et dans la même gamme de valeurs ; cependant, cette hypothèse n'augmente pas significativement la qualité de la synthèse. Une meilleure gestion de ces exemples et de leur irrégularité permettrait potentiellement de résoudre automatiquement le problème.

Préservation des statistiques Enfin, le transfert d'histogramme a un impact limité sur l'apparence de la texture : il ne permet pas de conserver les statistiques d'ordre 2 et supérieures. Cependant, ce défaut est inhérent au modèle de transfert d'histogramme, qui est un transfert des statistiques d'ordre 1. Il est difficile d'imaginer un transfert efficace permettant de transporter des statistiques d'ordre supérieures, car elles sont très difficiles à estimer dans un exemple (car trop larges, et dont une estimation précise est liée aux limites de l'ergodicité) et requièrent la modification de la valeur de plusieurs texels à la fois. On peut cependant supposer qu'un transfert de moments d'ordre 2 et supérieurs, comme la fonction d'autocovariance $\mathbf{mc}_2^2$, serait envisageable. Tartavel et al. [TGP15] proposent de conserver le spectre dans la sortie en remplaçant le spectre de la sortie par le spectre de l'exemple, une approche reprise plus tard [LGX16]. Leur synthèse, qui appartient à la famille des synthèses par optimisation, consiste à successivement optimiser une sortie aléatoire, en effectuant un transfert d'histogramme, un changement de spectre, et en ajustant la fréquence d'utilisation d'atomes représentant des parties de la texture pour qu'elle soit proportionnelle à celle de l'exemple. Cette approche est difficile à mettre en

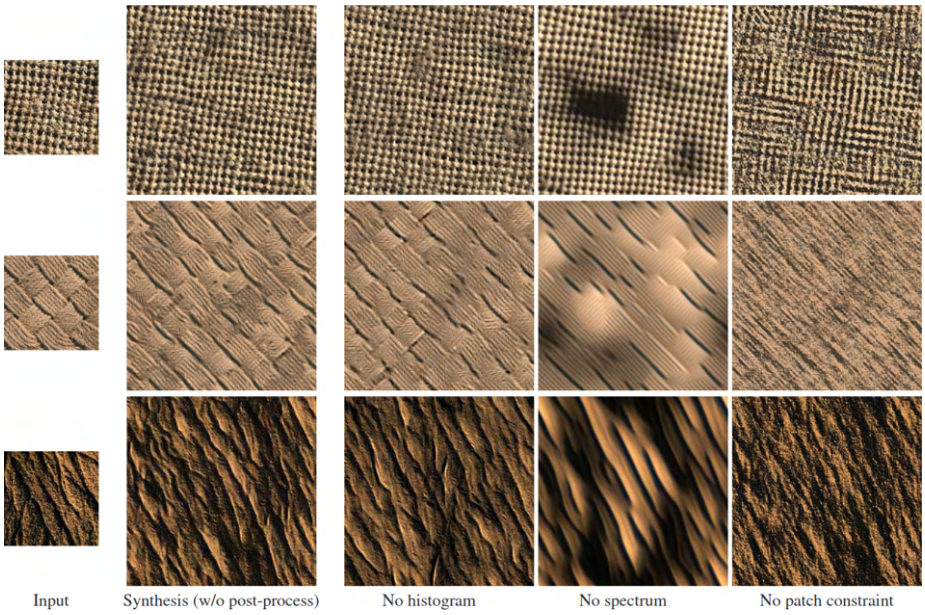


FIGURE 6.3 – Figure de Tartavel et al. [TGP15]. Un exemple est exprimé comme la combinaison linéaire d'un ensemble de patchs appelés « atomes » et leurs poids. Les différentes colonnes représentent, respectivement : l'exemple, la sortie, la sortie sans contraintes d'histogramme, sans contraintes de spectre, et sans contraindre l'utilisation des atomes pour qu'elle corrèle à celle de l'exemple. La contrainte de spectre permet de conserver au mieux le spectre de l'exemple dans la sortie, et donc le second moment d'ordre 2 $\mathbf{m}_2^2$.

place, notamment car le changement de spectre devrait se faire avec le spectre cyclostationnaire et que les atomes doivent être sélectionnés en respectant la cyclostationnarité du processus sous-jacent. Des résultats de leur synthèse sont donnés en Figure 6.3.

Filtrage Notre algorithme pose aussi des problèmes de filtrage. L'implémentation du transfert, sans filtrage, est classiquement effectuée par un tableau stockant T^{-1} pour chaque niveau de détail. Notre transfert requiert un stockage de $T_{\mathbf{x}}^{-1}$ pour chaque sous-échantillon $\mathbf{x}$ de X^-, ce qui augmente l'occupation mémoire de la texture.

L'implémentation pratique et performante de ce transfert filtré reste un problème : elle requiert de filtrer toutes les cartes $T_{\mathbf{x}}^{-1}$, tout en fusionnant les cartes voisines à une résolution inférieure. Certaines pistes peuvent être étudiées : par exemple, on peut passer à un transfert stationnaire à plus basse résolution, ou réduire la précision du transfert à plus basses résolutions.

6.2 Filtrage de la synthèse par échange de contenus

Nous avons montré dans le chapitre 5 une façon de filtrer la synthèse par échange de contenus [VSLD13], qui était dépourvue de filtrage jusqu'à alors. Cette synthèse consiste à répéter périodiquement l'exemple en changeant à la volée les « contenus » de l'exemple. La méthode de filtrage consiste à pré-filtrer ces contenus dans un atlas. La *patch-map*, qui représente les patchs, est aussi pré-filtrée à l'aide d'un opérateur OU bit à bit plutôt qu'une interpolation bilinéaire classique.

6.2.1 Extension à Kolář et al.

La synthèse par patchs interchangeables [KCD16] consiste, de façon similaire à Vanhoey et al. [VSLD13], à définir une texture comme la répétition d'une tuile périodique d'exemple au sein desquels les patchs sont changés aléatoirement. La différence est que deux ensembles différents de patchs non nécessairement disjoints se chevauchant sont définis pour deux types de tuiles, les tuiles « A » et les tuiles « B », qui sont alternées de la façon donnée en Figure 6.4.

Un graphe permet d'identifier les chevauchements entre les patchs ; ainsi, pendant le rendu, sur chaque tuile, les ensembles de patchs ne se chevauchent pas. L'avantage par rapport à la synthèse par échange de contenus [VSLD13] est que la patch-map est différente pour chaque répétition périodique de la texture, et permet encore d'avantage d'éviter les artefacts d'alignement en réduisant la probabilité que le contenu à deux endroits alignés soient les mêmes. Notre méthode de filtrage est séparée en deux parties : le filtrage de la *patch-map*, et celui des contenus. Le filtrage des contenus poserait un problème d'occupation mémoire : comme nous stockons les contenus et que la superficie totale des patchs est plus grande, l'occupation mémoire serait plus large, mais sans rendre leur stockage techniquement impossible. Le filtrage de la patch-map est plus problématique, pour les mêmes raisons qui empêchent le filtrage direct par pré-filtrage de l'exemple : elle est dynamique et change d'une tuile périodique à l'autre, ce qui signifie qu'il faudrait filtrer toutes les combinaisons possibles de patch-maps. Si on suppose que le nombre de patchs reste bas, il est toutefois possible d'utiliser une map bit à bit de façon similaire à notre méthode, mais la gestion de cette patch-map reste à étudier. De plus, à basse résolution, des patchs faits pour ne pas se chevaucher peuvent être amenés à le faire, ce qui risque de poser des problèmes d'implémentation.

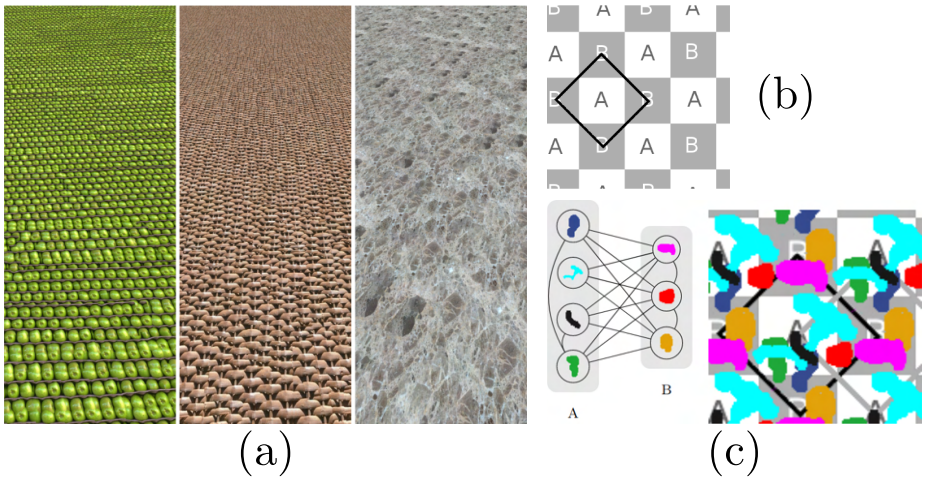


FIGURE 6.4 – Illustrations de Kolàř et al. [KCD16]. Les résultats de la synthèse ne sont pas filtrés (a). Les patchs interchangeables de la tuile périodique en A ne peuvent pas dépasser le carré noir (b), et la synthèse est une composition de deux types de tuiles périodiques A et B, pour lesquelles les patchs potentiels sont différents (c). Un graphe permet de savoir si deux patchs se chevauchent ; s'ils ne se chevauchent pas, ils peuvent être utilisés en même temps, sinon, ils ne peuvent pas l'être.

6.2.2 Amélioration des performances et de la qualité

Comme nous l'avons montré dans la section 5.4, le nombre d'accès texture requis par notre filtrage de la synthèse par échange de contenus [VSLD13] augmente exponentiellement lorsque la résolution diminue, jusqu'à un maximum égal au nombre de patchs fixé par la synthèse. Ainsi, le rendu d'une surface texturée à l'aide de la synthèse par échange de contenus devient lent si la surface est observée de loin.

La raison de cette perte de performances vient des patchs qui se mélangent de plus en plus au fur et à mesure que la résolution diminue, jusqu'à ce que les patchs aient la taille d'un seul texel qui recouvre tout le domaine de la texture. On peut en déduire plusieurs pistes : éliminer l'utilisation de notre filtrage à basse résolution, et éviter au maximum que les patchs ne se mélangent.

Ainsi, il peut être possible de revenir à un filtrage classique à basse résolution, c'est-à-dire en pré-filtrant l'exemple. Des artefacts apparaîtraient au bord des patchs, mais la distance avec la texture est potentiellement suffisamment large pour que ces défauts ne soient pas visibles. On peut également éviter au maximum que les patchs ne se mélangent en alignant les frontières des patchs avec des coordonnées correspondant à diverses puissances de 2 : cependant, cette

méthode contraindrait la découpe des patchs et ne fait qu'avancer le niveau de détail où les mélangent surviennent de façon inéluctable. Les performances peuvent aussi être améliorées en réduisant le degré de filtrage anisotrope (nous avons utilisé 16 échantillons au plus), et en améliorant l'implémentation pour réduire les défauts de cache.

6.3 Processus stochastiques pour la synthèse

6.3.1 Généralisation de la synthèse comme implémentation d'un processus stochastique

Dans le chapitre 3, nous avons émis l'hypothèse que toute synthèse de textures peut être modélisée par un processus stochastique. Un modèle de synthèse de textures peut donc s'exprimer par son ensemble des indices, son ensemble des valeurs, et son espace probabilisé. L'espace probabilisé est lui-même défini par l'ensemble des observables, celui des événements, et la mesure de probabilité des événements. Notre but est donc de déterminer les paramètres du processus stochastique qui est simulé par chacun des algorithmes de synthèse de textures.

Lorsqu'une synthèse de textures est indexée sur un ensemble discret et à valeurs dans un autre ensemble discret, par exemple, comme pour le pavage de Wang [Sta97] ou la synthèse par échange de contenus [VSLD13] (qui revient à un pavage apériodique à tuiles régulières avec de très nombreuses tuiles), les évènements de l'espace probabilisé du processus stochastique sous-jacent peuvent être exprimés par l'ensemble des parties de l'ensemble des observables Ω. Déterminer l'espace probabilisé revient alors à déterminer la probabilité de chaque $Y(\omega)$ pour tout $\omega \in \Omega$, c'est-à-dire que chaque texture ne soit générée. Par exemple, pour la synthèse par échange de contenus, lors de l'étape de rendu, les contenus sont choisis selon une loi uniforme ; chaque texture $Y(\omega)$ a donc la même probabilité d'apparition. Si la synthèse de textures est à valeurs dans un ensemble continu, les statistiques d'ordre de rang k sont des outils utiles pour déterminer la forme de la synthèse : ainsi, les modèles de bruits basés sur la convolution parcimonieuse ont souvent une forme Gaussienne [LLC+10].

Une limite de la définition du processus stochastique derrière une synthèse de textures est qu'elle ne donne pas forcément d'indication sur la méthode de calcul pouvant être utilisée pour la générer : par exemple, le modèle de processus stochastique de l'ADSN est un processus Gaussien stationnaire de second moment d'ordre 2 égal à celui du spot exemple, mais cette définition ne donne pas directement de moyen de calculer l'ADSN. En revanche, les propriétés déduites de l'analyse du modèle peuvent être utiles pour améliorer la synthèse. Ainsi, c'est grâce au modèle sous-jacent des bruits procéduraux (ADSN et bruit à phase aléatoire) que nous avons pu élaborer les modèles cyclostationnaires Gaussiens

présentées dans le chapitre 4. La connaissance que la plupart de ces processus sont stationnaires, Gaussiens et ergodiques a permis d'adapter les algorithmes au cyclostationnaire, Gaussien et cycloergodique. De plus, les mathématiques derrière ces processus ont donné les indications nécessaires à l'implémentation, comme par exemple, la forme que prend p_1 pour le transfert des statistiques d'ordre 1, le calcul de la moyenne $\mathbf{m}_1^1$ et la préservation des moments $\mathbf{m}_2^{1+1}$ utiles pour le *CS-spot noise*.

6.3.2 Généralisation à d'autres processus stochastiques

Nous avons montré qu'il était possible de généraliser des algorithmes basés sur les processus stationnaires pour qu'ils soient basés sur des processus cyclostationnaires. Il existe d'autres sortes de processus : les processus quasi-cyclostationnaires [Nap19], les processus quasi-cyclostationnaires généralisés ou encore les processus spectralement corrélés [Nap12]. L'adaptation à la cyclostationnarité des algorithmes basés sur des processus stochastiques augmente les degrés de liberté de la synthèse, mais complexifie les données d'entrée ; par exemple, l'adaptation des modèles à phase aléatoire, qui utilisent un unique spectre, consiste à définir un spectre pour chaque position ; on peut déjà observer certains exemples de ce type en ligne avec le bruit de Gabor et le *phasor noise*[1]. Il est donc difficile de sortir une synthèse d'un contexte stationnaire et ergodique, et encore plus d'aller au-delà d'un contexte cyclostationnaire et cycloergodique.

6.3.3 Synthèse par l'exemple non ergodique et non cycloergodique

La synthèse par l'exemple forme une famille d'algorithmes de synthèse qui utilisent un exemple pour en déduire la totalité des paramètres de la synthèse, et donc, du processus stochastique sous-jacent. Ce modèle n'est adapté qu'à des situations où ce processus est ergodique ou cycloergodique. Si on suppose qu'on a un processus Y générant un exemple E, alors l'exemple E n'est qu'un membre de l'ensemble des réalisations $Y(\omega)$. Sans l'ergodicité ou la cycloergodicité, et sans informations supplémentaire (par exemple une moyenne variant sur Ω, comme sur la Figure 3.9), il est impossible de déduire tous les paramètres de la synthèse à partir d'une seule réalisation. Ainsi, des modèles de synthèse par l'exemple basés sur des réseaux de neurones [GEB15, LGX16] utilisent un seul exemple, mais apprennent des données a priori à partir d'une base de donnée de textures, ce qui leur permet d'inférer un espace probabilisé.

Pour illustrer cette ouverture, nous proposons le prototype d'algorithme de la

1. https://www.shadertoy.com/view/wtt3RH

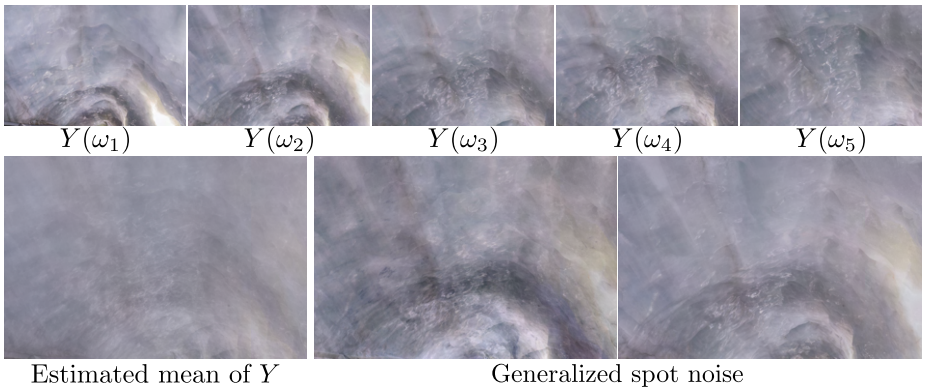


FIGURE 6.5 – Notre proposition de *spot noise* généralisé avec des textures de glace de forme particulière. La moyenne est calculée à partir des différentes réalisations utilisées comme exemples $Y(\omega)$, de ω_1 à ω_5. La convolution ne modifie pas la position de l'exemple (spot), mais l'exemple utilisé. Nous montrons deux textures générées à partir de ce processus.

Figure 6.5 pour généraliser le *spot noise* à plusieurs exemples, pour traiter le cas général où le processus n'est ni stationnaire et ergodique, ni cyclostationnaire et cycloergodique. Le principe du *spot noise* stationnaire est de modifier la position à laquelle les spots sont placés à chaque tir. La raison en est que le processus est jugé stationnaire et ergodique : ainsi, le même exemple à différentes positions est équivalent à une réalisation différente sur la même position. Notre *spot noise* cyclostationnaire exploite la propriété de cyclostationnarité et de cycloergodicité de la même façon : le même exemple à différentes translations de $k\mathbf{t_0} + l\mathbf{t_1}$ est équivalent à une réalisation différente sur la même position. Pour généraliser cet algorithme, on commence par calculer la moyenne générale $\mathbf{m}_1^1$ sans hypothèse stationnaire ou ergodique, mais en moyennant les exemples entre eux ; puis, un ADSN « général » est simulé sans modifier la position, mais en modifiant le spot utilisé. Le résultat obtenu est montré en Figure 6.5. L'avantage de cette approche est que n'importe quel n-uplet d'exemple ayant été générés par un processus Gaussien général peut être bien reproduit par cette approche ; mais elle ne permet pas de passer à une taille non bornée ou à un ensemble d'indices continu.

Notre prototype de *spot noise* généralisé ressemble aux travaux effectués par Liu et al. [LH05], où les différentes tuiles identifiables d'une texture quasi-régulière sont séparées. Chaque tuile est alors la réalisation d'un processus de génération de tuiles général, et pour lequel de nouvelles tuiles sont générées grâce aux tuiles existantes. L'idée d'utiliser plusieurs exemples se retrouve aussi dans la synthèse de terrains par dictionnaire parcimonieux [GDGP16], où un terrain est séparé en plusieurs exemples qui forment les atomes du dictionnaire, eux-mêmes mélangés pour la synthèse.

Bibliographie

[AM80] G Adomian and K Malakian. Stochastic analysis. *Mathematical Modelling*, 1(3) :211–235, 1980.

[BCL15] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. 2015.

[BPC16] Nicolas Bonneel, Gabriel Peyré, and Marco Cuturi. Wasserstein barycentric coordinates : histogram regression using optimal transport. *ACM Trans. Graph.*, 35(4) :71–1, 2016.

[Bur19] Brent Burley. On histogram-preserving blending for randomized texture tiling. *Journal of Computer Graphics Techniques (JCGT)*, 8(4) :31–53, November 2019.

[CB11] Y. Cai and G. Baciu. Detection of repetitive patterns in near regular texture images. In *2011 IEEE 10th IVMSP Workshop : Perception and Visual Signal Analysis*, pages 60–65, 2011.

[CB13] Y. Cai and G. Baciu. Detecting, grouping, and structure inference for invariant repetitive patterns in images. *IEEE Transactions on Image Processing*, 22(6) :2343–2355, 2013.

[CGG19] Arthur Cavalier, Guillaume Gilet, and Djamchid Ghazanfarpour. Local spot noise for procedural surface details synthesis. *Computers & Graphics*, 85 :92 – 99, 2019.

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3) :287–294, 2003.

[DED05] Khalid Djado, Richard Egli, and François Deschênes. Extraction of a representative tile from a near-periodic texture. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '05, page 331–337, New York, NY, USA, 2005. Association for Computing Machinery.

[DGF98] J.M. Dischler, D. Ghazanfarpour, and R. Freydier. Anisotropic solid texture synthesis using orthogonal 2d views. *Computer Graphics Forum*, 17(3) :87–95, 1998.

[DH18] Thomas Deliot and Eric Heitz. Procedural stochastic textures by tiling and blending. *GPU Zen 2 : Advanced Rendering Techniques*, 2018.

[EF01] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 341–346. ACM, 2001.

[EL99] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1033–1038 vol.2, 1999.

[FF02] Dominique Foata and Aimé Fuchs. Processus stochastiques : Processus de poisson, chaînes de markov et martingales. *Dunod, Parigi*, 2002.

[GAD+20] Pascal Guehl, Rémi Allegre, J-M Dischler, Bedrich Benes, and Eric Galin. Semi-procedural textures using point process texture basis functions. In *Computer Graphics Forum*, volume 39, pages 159–171. Wiley Online Library, 2020.

[GDGP16] Eric Guérin, Julie Digne, Eric Galin, and Adrien Peytavie. Sparse representation of terrains for procedural modeling. In *Computer Graphics Forum*, volume 35, pages 177–187. Wiley Online Library, 2016.

[GEB15] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. *Advances in neural information processing systems*, 28 :262–270, 2015.

[GGM11] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Random phase textures : Theory and synthesis. *IEEE Transactions on Image Processing*, 20(1) :257 – 267, 2011.

[GLLD12] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. Gabor noise by example. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)*, 31(4) :73 :1–73 :9, July 2012.

[GLM17] B. Galerne, A. Leclaire, and L. Moisan. Texton noise. *Computer Graphics Forum*, 36(8) :205–218, 2017.

[GLR17] Bruno Galerne, Arthur Leclaire, and Julien Rabin. Semi-discrete optimal transport in patch space for enriching gaussian textures. In *Geometric Science of Information*, volume 10589 of *Lecture Notes in Computer Science*, Paris, France, November 2017.

[GRGH18] Jorge Gutierrez, Julien Rabin, Bruno Galerne, and Thomas Hurtut. On Demand Solid Texture Synthesis Using Deep 3D Networks. working paper or preprint, December 2018.

[GSDC17] Geoffrey Guingo, Basile Sauvage, Jean-Michel Dischler, and Marie-Paule Cani. Bi-layer textures : a model for synthesis and deformation of composite textures. *Computer Graphics Forum*, 36(4) :111–122, 2017.

[GSV+14] Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Local random-phase noise for procedural texturing. *ACM Trans. Graph.*, 33(6) :195 :1–195 :11, November 2014.

[HH09] Michal Haindl and Martin Hatka. Near-regular texture synthesis. In Xiaoyi Jiang and Nicolai Petkov, editors, *Computer Analysis of Images and Patterns*, pages 1138–1145, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[HN18] Eric Heitz and Fabrice Neyret. High-performance by-example noise using a histogram-preserving blending operator. *Eurographics Symposium on High-Performance Graphics 2018*, 2018.

[HPCE16] Noura Hamzé, Igor Peterlík, Stéphane Cotin, and Caroline Essert. Preoperative trajectory planning for percutaneous procedures in deformable environments. *Computerized Medical Imaging and Graphics*, 47 :16 – 28, 2016.

[HWM06] Danny Holten, Jarke J Van Wijk, and Jean-Bernard Martens. A perceptually based spectral model for isotropic textures. *ACM Transactions on Applied Perception (TAP)*, 3(4) :376–398, 2006.

[JR15] Emmanuel Jeandel and Michael Rao. An aperiodic set of 11 wang tiles. *arXiv preprint arXiv :1506.06492*, 2015.

[Jul81] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802) :91–97, 1981.

[KCD16] Martin Kolàř, Alan Chalmers, and Kurt Debattista. Repeatable texture sampling with interchangeable patches. *The Visual Computer*, 32(10) :1263–1272, 2016.

[KGE18] A. Kipnis, A. J. Goldsmith, and Y. C. Eldar. The distortion rate function of cyclostationary gaussian processes. *IEEE Transactions on Information Theory*, 64(5) :3810–3824, 2018.

[KSE+03] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures : Image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3) :277–286, 2003.

[LD06] Ares Lagae and Philip Dutré. An alternative for wang tiles : Colored edges versus colored corners. *ACM Transactions on Graphics*, 25(4) :1442–1459, 2006.

[Lef08] Sylvain Lefebvre. Filtered tilemaps (in shader x6). In Wolfgang Engel, editor, *Shader X6 : Advanced Rendering Techniques*, Shader X6, pages 63–72. Charles River Media, 2008. no note.

[Lew84] John-Peter Lewis. Texture synthesis for digital painting. *SIGGRAPH Comput. Graph.*, 18(3) :245–252, January 1984.

[LGX16] Gang Liu, Yann Gousseau, and Gui-Song Xia. Texture synthesis through convolutional neural networks and spectrum constraints. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3234–3239. IEEE, 2016.

[LH05] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 777–786, New York, NY, USA, 2005. Association for Computing Machinery.

[LLC+10] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, D.S. Ebert, J.P. Lewis, Ken Perlin, and Matthias Zwicker. State of the art in procedural noise functions. In Helwig Hauser and Erik Reinhard, editors, *EG 2010 - State of the Art Reports*. Eurographics, Eurographics Association, May 2010.

[LLD11] Ares Lagae, Sylvain Lefebvre, and Philip Dutré. Improving gabor noise. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[LLDD09] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2009)*, 28(3) :54–64, July 2009.

[LLH04] Y. Liu, W-C. Lin, and J. Hays. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 2004.

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3) :127–150, 2001.

[LNS+15] Siying Liu, Tian-Tsong Ng, Kalyan Sunkavalli, Minh N. Do, Eli Shechtman, and Nathan Carr. Patchmatch-based automatic lattice detection for near-regular textures. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[LPVV17] L. Lettry, M. Perdoch, K. Vanhoey, and L. Van Gool. Repeated pattern detection using cnn activations. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 47–55, 2017.

[LSD21] Nicolas Lutz, Basile Sauvage, and Jean-Michel Dischler. Cyclostationary Gaussian noise : theory and synthesis. In *Eurographics 2021*, Vienna, Austria, May 2021.

[LSLD19] Nicolas Lutz, Basile Sauvage, Frédéric Larue, and Jean-Michel Dischler. Anisotropic filtering for on-the-fly patch-based texturing. In *Eurographics 2019*, Genoa, Italy, May 2019.

[LTL05] Yanxi Liu, Yanghai Tsin, and Wen-Chieh Lin. The promise and perils of near-regular texture. *International Journal of Computer Vision*, 62(1-2) :145–159, 2005.

[Nap12] Antonio Napolitano. *Generalizations of cyclostationary signal processing : spectral analysis and applications*, volume 95. John Wiley & Sons, 2012.

[Nap19] Antonio Napolitano. *Cyclostationary processes and time series : theory, applications, and generalizations*. Academic Press, 2019.

[NMMK05] Andre Nicoll, Jan Meseth, Gero Müller, and Reinhard Klein. Fractional fourier texture masks : Guiding near-regular texture synthesis. *Computer Graphics Forum*, 24 :569 – 579, 10 2005.

[NWT+05] T.-Y Ng, Conghua Wen, Tiow-Seng Tan, Xinyu Zhang, and Young Kim. Generating an ω-tile set for texture synthesis. In *International 2005 Computer Graphics*, pages 177 – 184, 07 2005.

[PBCL09] M. Park, K. Brocklehurst, R. T. Collins, and Y. Liu. Deformed lattice detection in real-world images using mean-shift belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10) :1804–1816, 2009.

[Per85] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3) :287–296, July 1985.

[Per02] Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002.

[PGDG16] Nicolas Pavie, Guillaume Gilet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Procedural texture synthesis by locally controlled spot noise. *Computer Science Research Notes*, 2601 :71–79, 2016.

[PPC10] Nicolas Papadakis, Edoardo Provenzi, and Vicent Caselles. A variational model for histogram transfer of color images. *IEEE Transactions on Image Processing*, 20(6) :1682–1695, 2010.

[RHE11] Diego Lopez Recas, Anna Hilsmann, and Peter Eisert. Near-Regular Texture Synthesis by Random Sampling and Gap Filling. In Peter Eisert, Joachim Hornegger, and Konrad Polthier, editors, *Vision, Modeling, and Visualization (2011)*. The Eurographics Association, 2011.

[Sau88] Dietmar Saupe. Algorithms for random fractals. In *The science of fractal images*, pages 71–136. Springer, 1988.

[SCO17] Omry Sendik and Daniel Cohen-Or. Deep correlations for texture synthesis. *ACM Trans. Graph.*, 36(4), July 2017.

[Sta97] Jos Stam. Aperiodic texture mapping. 08 1997.

[TEZ+19] Thibault Tricard, Semyon Efremov, Cédric Zanni, Fabrice Neyret, Jonàs Martínez, and Sylvain Lefebvre. Procedural phasor noise. *ACM Transactions on Graphics*, 38(4) :Article No. 57 :1–13, July 2019.

[TGP15] Guillaume Tartavel, Yann Gousseau, and Gabriel Peyré. Variational texture synthesis with sparsity and spectrum constraints. *Journal of Mathematical Imaging and Vision*, 52(1) :124–144, 2015.

[TNVT19] Vincent Tavernier, Fabrice Neyret, Romain Vergne, and Joëlle Thollot. Making gabor noise fast and normalized. In The Eurographics Association, editor, *Eurographics 2019 - 40th Annual Conference of the European Association for Computer Graphics*, Eurographics 2019 - Short Papers, pages 37–40, Genoa, Italy, May 2019.

[ULVL16] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks : Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.

[VSLD13] Kenneth Vanhoey, Basile Sauvage, Frédéric Larue, and Jean-Michel Dischler. On-the-fly multi-scale infinite texturing from example. *Transactions on Graphics*, 32(6) :208 :1–208 :10, 2013. (Proceedings of Siggraph Asia'13).

[vW91] Jarke J. van Wijk. Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph.*, 25(4) :309–318, 1991.

[Wei04] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '04, pages 55–63. ACM, 2004.

[WHC+06] Wen-Chieh Lin, J. Hays, Chenyu Wu, Yanxi Liu, and V. Kwatra. Quantitative evaluation of near regular texture synthesis algorithms. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 427–434, 2006.

[WL00] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 479–488, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Wor96] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, 1996.

[XZJ+07] Feng Xue, You-Sheng Zhang, Ju-Lang Jiang, Min Hu, Xin-Dong Wu, and Rong-Gui Wang. Real-time texture synthesis using s-tile set. *J. Comput. Sci. Technol.*, 22(4) :590–596, July 2007.

[YCT04] Yanxi Liu, R. T. Collins, and Y. Tsin. A computational model for periodic pattern perception based on frieze and wallpaper groups. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3) :354–371, 2004.