Génération procédurale de textures pour enrichir les détails surfaciques

Arthur Cavalier

Université de Limoges

ÉCOLE DOCTORALE « Sciences et Ingénierie pour l'Information, Mathématiques »
FACULTÉ DES SCIENCES ET TECHNIQUES
Laboratoire XLIM - ASALI-SIR

THÈSE

pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ DE LIMOGES
Discipline / Spécialité : Informatique Graphique
présentée et soutenue par
Arthur Cavalier
le 12-12-2019

Génération procédurale de textures pour enrichir les détails surfaciques

Thèse co-encadrée par Guillaume Gilet
et Djamchid Ghazanfarpour

JURY

Président du jury :	
M. Loïc BARTHE	Professeur de l'Université Paul Sabatier
Rapporteurs :	
M. Jean-Michel DISCHLER	Professeur de l'Université de Strasbourg
M. Christophe RENAUD	Professeur de l'Université Littoral Côte d'Opale
Examinateurs :	
M. Djamchid GHAZANFARPOUR	Professeur de l'Université de Limoges
M. Guillaume GILET	Maître de conférences de l'Université de Limoges

C'est au moment où l'on triche pour le beau que l'on est artiste.
- Max Jacob -

À Raymond Bonneau et Roland Cavalier.

Remerciements

Ce manuscrit comporte effectivement une page de remerciements s'adressant aux personnes qui ont été plus ou moins victimes ou témoins de cette thèse de doctorat. Aussi, ne voulant pas abuser du temps, certes précieux, des lecteurs qui se seraient malencontreusement perdus sur *theses.fr* ou sur HAL, mes remerciements seront concis.

Je commencerai bien entendu par remercier tous les membres de mon Jury de thèse pour avoir accepté de s'infliger la lecture de ce manuscrit ainsi que d'évaluer mon travail de recherche.

Je tiens, en suivant, à remercier l'ensemble de mes collègues de l'équipe ASALI-SIR de l'institut XLIM et de l'équipe pédagogique de l'université de Limoges pour les discussions et débats autour de la pédagogie et du monde scientifique que nous avons pu échanger.

Ce travail n'aurait pas pu aboutir sans l'aide de mes collègues et amis du couloir des doctorants et stagiaires du laboratoire XLIM et je tiens à remercier tout particulièrement Xavier Chermain, François Deves, Thibault Tricard et Théo Jonchier pour toutes les discussions aussi bien techniques que théoriques et, bien entendu, toutes les bêtises que nous avons pu nous raconter à longueur de journée.

Je remercie également ma famille pour son soutien inconditionnel et enfin, Marine, sans qui tout cela n'aurait pas eu lieu et qui m'a soutenu et encouragé tout au long de ma thèse, dans les bons comme dans les mauvais moments.

Résumé

Dans le sillage de l'augmentation de la puissance graphique des machines grands publics, le domaine de la synthèse d'images réalistes nous propose de se plonger dans des mondes virtuels toujours plus détaillés. Les artistes sont alors sollicités pour remplir et animer ces scènes virtuelles complexes. Il en résulte un temps de création prohibitif, un coût mémoire grandissant et des difficultés pour rendre de manière correcte et efficace cette profusion de détails. De nombreux outils de génération procédurale de contenu ont alors été proposés pour aider les studios à gérer ces problèmes.

Dans cette thèse, nous nous sommes intéressés à la synthèse de détails mésoscopiques à la volée pour ajouter facilement du détail à la surface des modèles 3D. En se concentrant sur la synthèse procédurale de texture, nous avons proposé des améliorations pour rendre correctement des textures modifiant non seulement la couleur de la surface d'un objet mais feignant aussi le relief de la surface en temps réel. Nous avons travaillé sur la synthèse de motifs structurés contrôlables dans le but de proposer une méthodologie permettant de rendre des textures de grande qualité à la volée sans défauts d'aliassage. Nous avons étendu aussi la génération de texture à la synthèse de détails géométriques mésoscopiques, en synthétisant à la volée des cartes de normales venant perturber le calcul de l'éclairage de la surface pour y faire apparaître des aspérités.

Mots clefs : Informatique Graphique, textures procédurales, bruits procéduraux.

Abstract

With the increasing power of consumer machines, Computer Graphics is offering us the opportunity to immerse ourselves in ever more detailed virtual worlds. The artists are thus tasked to model and animate these complex virtual scenes. This leads to a prohibitive authoring time, a bigger memory cost and difficulties to correctly and efficiently render this abundance of details. Many tools for procedural content generation have been proposed to resolve these issues.

In this thesis, we focused our work on on-the-fly generation of mesoscopic details in order to easily add tiny details on 3D mesh surfaces. By focusing on procedural texture synthesis, we proposed some improvements in order to correctly render textures that modify not only the surface color but faking the surface meso-geometry in real-time. We have presented a methodology for rendering high quality textures without aliasing issues for controllable structured pattern synthesis. We also proposed an on-the-fly normal map generation to disturb the shading calculation and to add irregularites and relief to the textured surface.

Keywords: Computer Graphics, procedural texturing, procedural noise.

Table des matières

Table des figures

Chapitre 1

Introduction générale

Chapitre 1

Introduction générale

L'informatique graphique connaît un intérêt de plus en plus grand au fil des décennies. Cet essor, poussé par l'accès et la puissance grandissante des capacités de calcul des ordinateurs, a permis des avancées significatives dans la visualisation des données, la modélisation géométrique, la simulation, le rendu ou encore l'intelligence artificielle.

Rien que dans l'industrie de divertissement (cinéma, jeux vidéos), ces avancées ont permis de générer des images toujours plus complexes au fil des années. Les avancées dans le domaine du rendu basé physique et la recherche d'images de plus en plus réalistes poussent les studios à générer des mondes virtuels de plus en plus grands pour faciliter l'immersion du spectateur ou du joueur. Les artistes doivent alors concevoir, habiller et animer ces mondes en y ajoutant des détails à toutes les échelles, des détails macroscopiques (terrains, ...) au détails les plus fins (contenus d'une texture, les feuilles d'un arbre, etc.). C'est cette profusion de détails qui pose des problèmes à la fois en temps de conception et en complexité de rendu. En effet, une fois dans le monde virtuel, pouvoir percevoir correctement l'ensemble des détails fins (distinguer les brins d'herbe d'un champ, voir une vague se briser, ...) et des détails "grossiers" (le champ d'herbe dans son ensemble, le mouvement général du courant, ...) reste encore un problème ouvert en Informatique Graphique.

Pour régler le premier problème, de nombreux outils ont été proposés afin de faciliter la création et l'édition des contenus que les artistes doivent produire. Un exemple parmi tant d'autre est la génération procédurale. En lieu et place de la modélisation manuelle des détails, des logiciels de modélisation automatisée proposent à l'utilisateur des outils mathématiques et informatiques permettant la création automatique de contenu en s'appuyant sur des règles que l'utilisateur peut définir. Les méthodes procédurales sont généralement des morceaux de codes, des algorithmes qui précisent les caractéristiques d'un contenu généré par ordinateur. Ce type de méthode se différentie de la modélisation traditionnelle en s'appuyant davantage sur le code, un algorithme, plutôt que par de la donnée. Pour contrôler l'apparence finale du contenu généré, l'utilisateur peut alors définir les rêgles de génération, dans le cas d'une grammaire générative comme les *L-systèmes*, fournir un contenu à reproduire en entrée, pour les méthodes générant un contenu basé sur un exemple, ou encore gérer des paramètres, dans le cas d'un algorithme de génération.

Selon le contenu souhaité, la complexité visuelle et la variété des détails peuvent être compliquées à gérer pour un algorithme déterministe. En effet, certains phénomènes naturels comme les nuages ou les reliefs d'une montagne par exemple, ne peuvent être reproduits fidèlement que par simulation physique.

Depuis les années quatre-vingts, l'utilisation de l'aléatoire contrôlé (ou du bruit plus généralement) s'est généralisée afin de modéliser plus facilement des motifs non structurés. Avec l'introduction du bruit de Perlin, les méthodes basées sur des fonctions procédurales de bruits ont été proposées dans le but de générer des motifs non structurés, modéliser des phénomènes naturels complexes ou encore d'ajouter du détail dans une texture, un volume, une surface, etc.

Les *bruits procéduraux* sont alors devenus un outil puissant, largement utilisé en rendu, permettant de perturber un motif périodique simple (cf. Figure 1.1) ou bien de représenter directement des motifs non structurés comme le montre la Figure 1.2 :

- Le relief d'une montagne, représenté par un champ de hauteur issu de la somme de plusieurs octaves d'un bruit de Perlin [Per85].
- La forme d'un nuage, sculpté par la combinaison [Sch16] de bruits de *Worley* [Wor96] et de bruits de Perlin [Per85] en 3D.
- La simulation du grain d'une pellicule.
- Ou encore une perturbation permettant de paver l'espace avec une texture sans qu'aucune répétition ne soit visible [HN18].

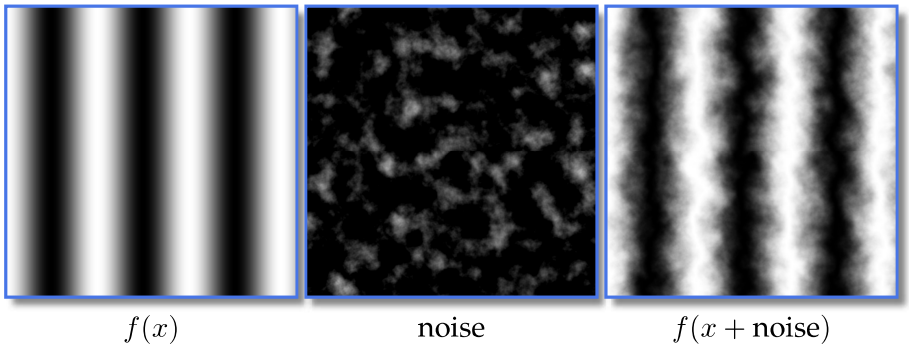


FIGURE 1.1 – En partant d'une fonction *cosinus* classique nous obtenons un motif répétitif basique (un cosinus, à gauche). Lorsqu'un bruit procédural (bruit de Perlin, au centre) est utilisé comme perturbation, nous obtenons une image présentant un motif plus complexe (à droite).

Dans cette thèse, nous nous sommes intéressés à un contenu particulier, la texture. Historiquement, pour des raisons de puissance de calcul, les modèles 3D étaient limités en terme de complexité géométrique. Le plaquage de texture, méthode qui consiste à plaquer un motif sur une surface, a permis d'ajouter du détail sur ces objets pour un coût moindre.

En plus de la modélisation de l'objet, les artistes ont alors eu la lourde tâche de peindre des motifs naturels sur des surfaces de plus en plus grandes en prenant soin de masquer, non sans mal, les répétitions et les défauts. Des méthodes de synthèse de textures ont alors été proposées pour automatiser la génération de motifs complexes.

FIGURE 1.2 – La modélisation de phénomènes non structurés (reliefs d'une montagne, forme d'un nuage, grain de pellicule, la texture recouvrant le sol, etc.) par des fonctions de bruit procédurales permettent de passer d'une image toute simple (en haut) à une image présentant toute une variété de détails, le tout calculé à la volée.

En regardant de plus près la Figure 1.3, nous remarquons la grande variété d'apparences présentes dans les textures naturelles. Nous y distinguons des textures dites *régulières* dont la répétition du motif est évidente et bien marquée (tissus, agencement régulier de briques, etc.) et des textures *stochastiques* pour lesquelles nous ne distinguons aucun motif régulier (nuages, vagues, etc.).

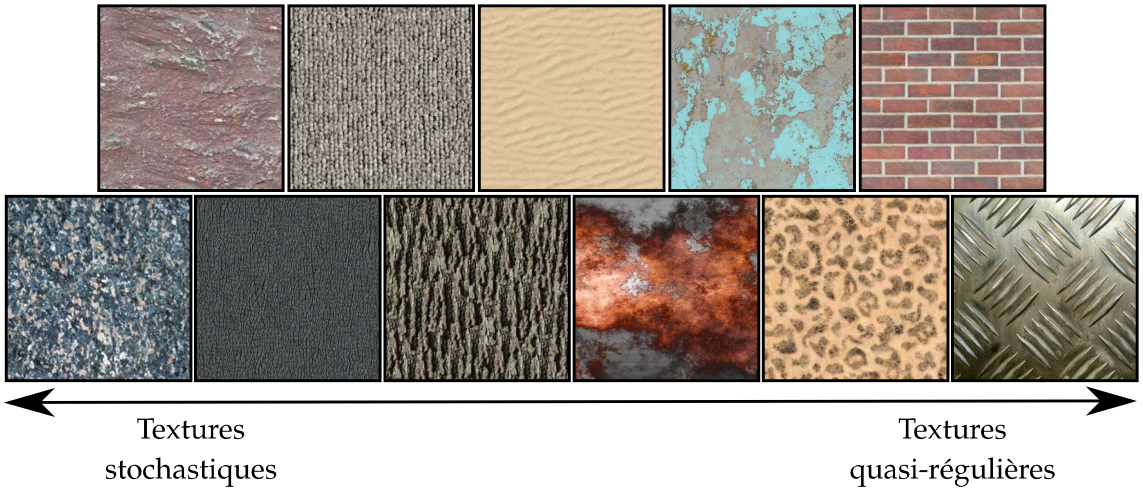


FIGURE 1.3 – Variété des motifs présents dans les textures d'objets allant du plus aléatoire au plus régulier.

Les textures "quasi-régulières" présentent généralement un motif particulier distribué au choix aléatoirement ou régulièrement dans l'espace. De nombreuses méthodes ont exploité l'idée de distribuer des contenus discrets (petites textures) ou procéduraux (fonctions continues) pour remplir l'espace de texture. Les méthodes par *distribution d'objets* comme [LD05, Gla04] s'appuient sur une distribution de points (généralement une distribution de poisson) pour y placer des primitives. Les motifs à pois sont un bon exemple de représentant de cette catégorie de méthodes.

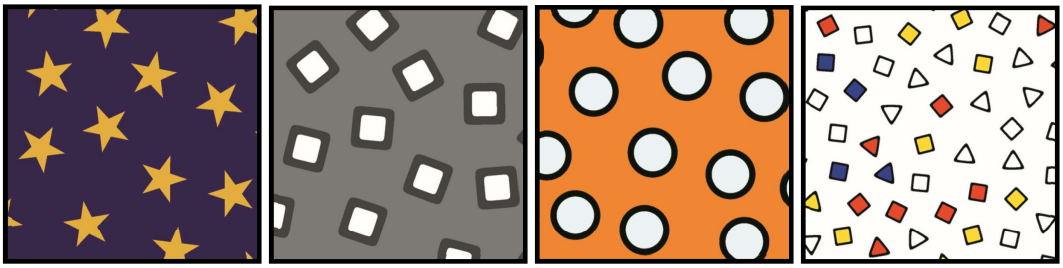
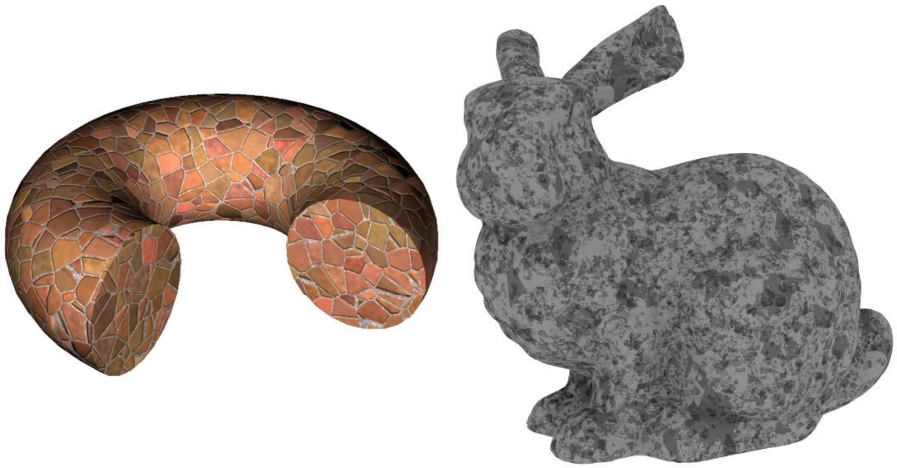


FIGURE 1.4 – Exemple de texture synthétisée par distribution d'objets [LD05].

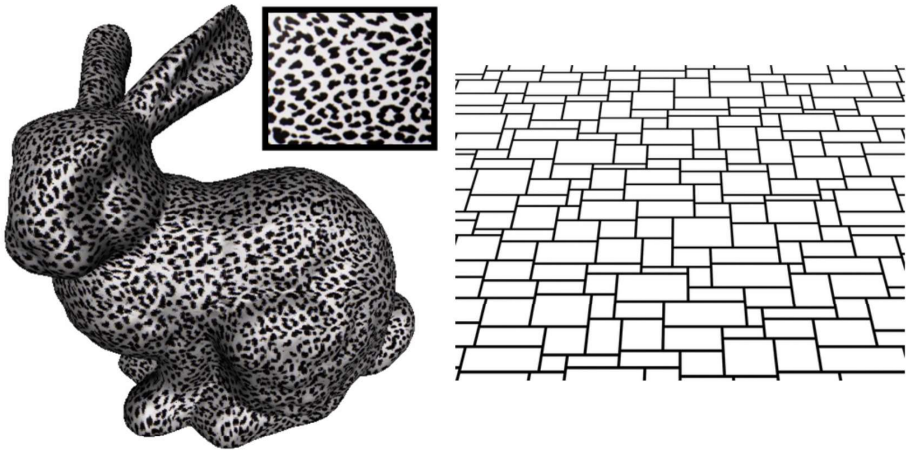
Les *fonctions de base de texture* ou *Texture basis functions* sont, quant à elles, définies comme des fonctions générant un motif simple qui sert de primitive de base pour concevoir des motifs plus complexes. La méthode la plus représentative de cette catégorie est la primitive cellulaire de Worley [Wor96, JDJS19]. En se basant sur la distance séparant des points distribués aléatoirement dans l'espace, cette méthode permet de synthétiser une grande variété de motifs quasi-réguliers allant du papier froissé aux murs de pierres.



[Wor96] [JDJS19]

FIGURE 1.5 – Exemples de textures synthétisées par *Texture basis functions*.

Les méthodes basées sur le tuilage de surface permettent aussi de générer des motifs similaire en pré-calculant des arrangements de tuile pour remplir l'espace. Toutes les méthodes basées sur le *Wang tiling* [CSHD03, Wei04, DJSJ19] sont représentatives de cette famille.



[FL05] [DJSJ19]

FIGURE 1.6 – Exemples de textures synthétisées par *tiling*.

Le *tuilage/pavage stochastique* place aléatoirement des sous-parties d'une image d'exemple dans l'espace. La complexité de ce genre de méthode repose sur la problématique de mélange des *patchs* tirés [LLX+01, Qui15] et du contenu du patch [VSLD13].

Enfin, comme présenté précédemment, les *fonctions de bruit* ont été utilisées comme perturbation et générateur de motifs non structurés semblables aux textures "stochastiques". C'est sur cette famille de méthodes que ce travail de thèse se concentre.

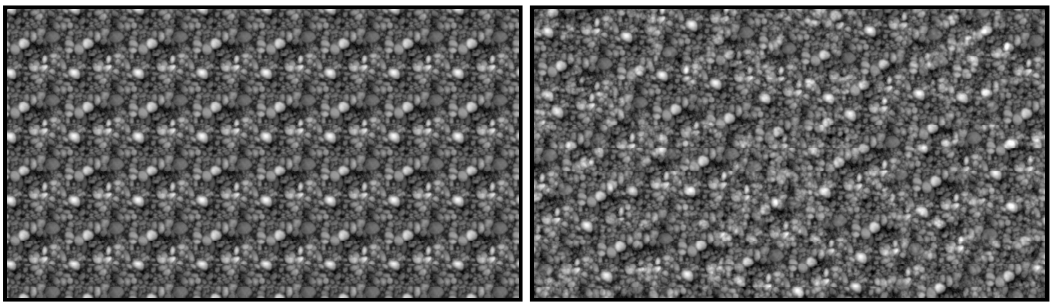
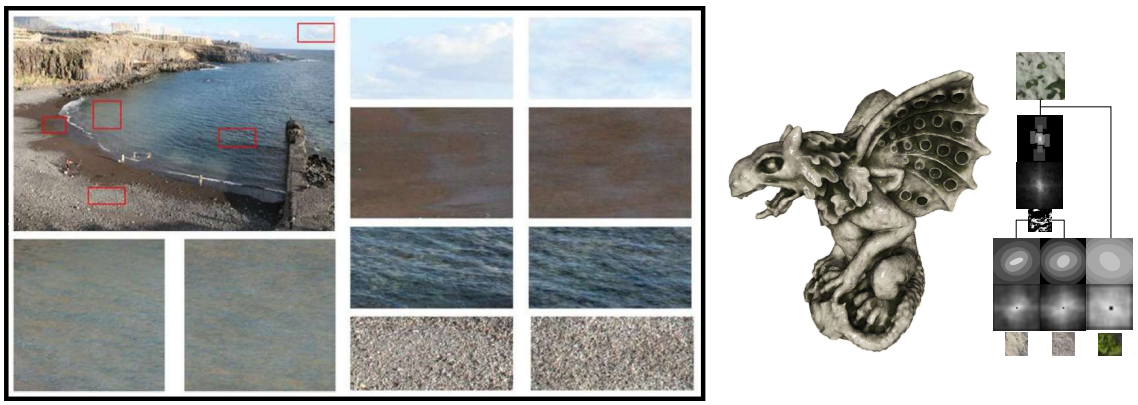


FIGURE 1.7 – Exemple de texture synthétisée par *pavage stochastique* (à droite) pour éviter la répétition de la texture (à gauche) [Qui15].

Objectifs de la thèse

Cette thèse avait pour objectif initial de proposer un modèle de représentation hiérarchique pour les textures composites afin d'étendre le spectre des textures représentable avec une représentation procédurale. Cette représentation, s'inspirant des travaux de Galerne *et al.* [GGM10] et de Gilet *et al.* [GDG12], définit une texture composite comme combinaison de motifs spatiaux contrastés (définis dans les noeuds d'une arborescence) permettant de partitionner l'espace de texture pour venir évaluer différents motifs non structurés aléatoires (définis dans les feuilles de l'arbre).



[GGM10] [GDG12]

FIGURE 1.8 – Étude des textures composites

Avec l'image de gauche de la Figure 1.8, issue de [GGM10], Galerne *et al.* montrent que des sous parties d'une image peuvent être synthétisées par des bruits à phase aléatoires. Dans la deuxième partie de cette figure, issue du *Multiple Kernel Noise*, Gilet *et al.* [GDG12] proposent de combiner plusieurs textures synthétisées pour obtenir ce type de motifs. Les textures contrastées dans les noeuds de l'arbre, appellées "composantes structurelles", sont obtenues par perturbation. Quant aux "micro-textures", elles sont synthétisées par *convolution éparse*.

En se basant sur une formulation similaire, des primitives contrôlables et filtrables contrastées seraient utilisées comme composantes structurelles et définies dans les noeuds d'un arbre. Des textures procédurales stationnaires, les micros-textures évoquées plus haut, seraient ensuite évaluées pour remplir la texture. Ces dernières seraient définies

dans les feuilles de cet arbre de fonctions procédurales. Evaluer une texture composite reviendrait à parcourir cette hiérarchie par pixel. En partant des fonctions définies en haut de l'arbre pour retrouver la position du texel au sein de la texture, en arrivant aux fonctions du bas de l'arbre définissant le contenu de cette texture.

L'évaluation d'une telle représentation comporte plusieurs défis :

- Évaluer efficacement et correctement les pixels se trouvant dans une zone de transition entre deux contenus procéduraux, ce qui entraînerait le parcours de deux sous-arbres dans cette hiérarchie et dont la complexité augmenterait récursivement sur des arbres profonds.
- Dans un contexte de rendu filtré, l'évaluation serait effectuée non pas pour un point de la surface, mais sur le domaine de l'empreinte du pixel. En effet, la requête d'une évaluation filtrée doit prendre en compte un ensemble de points localisé dans une zone dépendante de la forme du pixel une fois projeté sur la surface.

En plus de cette problématique d'évaluation correcte et filtrée de cette texture composite, le manque de primitives structurelles filtrables contrastées dans la littérature sur les fonctions de bruits limite la capacité à générer des motifs complexes au travers de cette méthode. Les travaux de cette thèse se sont donc concentrés sur ce problème en premier.

Recherche de primitives contrastées filtrables

Dans cette thèse, nous nous sommes focalisés sur la recherche de primitives filtrables contrastées comme première brique pour ce type de représentation. En s'appuyant sur les avancées récentes des méthodes de bruits par *convolution éparse* (ou *Sparse Convolution Noise*) et notamment sur les méthodes de bruits par *noyaux ponctuels* (ou *Spot Noise*), nous avons défini une méthodologie pour améliorer des fonctions de texture procédurale basées sur les bruits par convolution éparse en tirant avantage des noyaux de convolution.

La contribution majeure de la thèse est l'amélioration apportée à la représentation de texture par bruit à *noyaux ponctuels localement contrôlés* (ou **LCSN** pour *locally controlled spot noise*). En retravaillant la paramétrisation du noyau de convolution utilisé nous avons pu proposer une méthode de fitrage analytique anisotrope similaire à [GZD08] et [LLDD09]. Nous avons proposé une méthode de synthèse de détails mésoscopiques (représentés par une *carte de normales*) à la volée sur une surface. En se replaçant dans un contexte de rendu, nous avons étudié la problématique de l'antialiassage des cartes de normales pour la synthèse de textures basée sur les méthodes par noyaux ponctuels.

Description du manuscrit

Dans cette thèse, nous commencerons par la traditionnelle section de définitions et de notations, dans laquelle nous rapellerons les principes fondamentaux sur le bruit, les méthodes procédurales, la théorie du signal reposant sur la transformation de Fourier, l'aliassage et le contexte de rendu dans lequel nous évaluerons la texture. Après un état de l'art sur les fonctions de bruits pour la synthèse de motifs non structurés, nous pré-

senterons notre travail sur le bruit par *noyaux ponctuels localement contrôlés* publié dans le journal *Computer & Graphics*. Enfin, nous présenterons une synthèse et des perspectives sur les méthodes par *convolution éparse*.

Dans un souci de reproductibilité des résultats, nous fournissons en plus des démonstrations mathématiques, des implémentations GLSL disponibles sur *shadertoy* et présentes sous forme de lien dans le manuscrit [1].

1. Si le manuscrit est lu en version imprimée, il suffit de copier la clé du lien en fin de l'url suivante : https://www.shadertoy.com/view **/clé**

Chapitre 2

Etat de l'art

Sommaire

Chapitre 2

Etat de l'art

Dans ce chapitre, nous introduisons les définitions et les notations que nous allons utiliser dans le reste du manuscript. Après un rappel sur le contexte de rendu temps réel dans lequel nous nous plaçons, nous y présentons ensuite un état de l'art des méthodes de bruits procéduraux. Nous invitons par ailleurs le lecteur à consulter le livre de Ebert *et al.* [EM03] de 2002 et l'état de l'art de Lagae *et al.* [LLC+10] de 2010 pour une vue d'ensemble des travaux publiés dans ce domaine entre les années 80 et 2010. Pour finir, nous replaçons le travail de cette thèse dans le contexte de cet état de l'art.

2.1 Définitions et notations

Le bruit

Après l'introduction du bruit de Perlin en 1985, Ken Perlin et Eric Hoffert [PH89] introduisent une définition du bruit en Informatique Graphique comme étant *l'approximation d'un bruit blanc limité en fréquence à une seule octave* permettant d'introduire de l'aléatoire dans un signal numérique sans sacrifier la continuité et le contrôle. C'est une primitive définie dans le domaine fréquentiel servant de brique de base pour représenter et/ou filtrer un spectre de puissance. Dans son état de l'art de 2010, Lagae *et al.* proposent une définition un peu plus étendue du bruit comme *un processus stationnaire normal (dans le sens Gaussien)* dont le seul paramètre est sa densité spectrale de puissance (ou *Power Spectral Density* (**PSD**)). Cette dernière est soit éditée directement soit le résultat d'une somme d'instances indépendantes de bruits limités en fréquence (ou passe-bande).

Un bruit procédural

Un *bruit procédural* est, comme son nom l'indique, une méthode procédurale permettant de générer un bruit. Une fonction procédurale de bruit présente idéalement les propriétés suivantes [LLC+10] :

- une fonction procédurale de bruit est **compacte** en mémoire puisqu'elle est entièrement définie sous forme d'algorithme et stockée sous forme de morceaux de code et de paramètres.

- une fonction procédurale de bruit est **continue**, **multi-résolution** et **non-périodique**. En d'autres termes, une fonction de bruit peut s'étendre indéfiniment et couvrir des régions toujours plus vastes de l'espace considéré et générer des bruits à toutes les résolutions sans raccords apparents et sans répétitions.
- une fonction procédurale de bruit est **accessible de manière arbitraire**. Elle est évaluable en tout point de l'espace pour un coût constant, indépendant des évaluations précédentes et peut donc être aisément parallélisable.
- une fonction procédurale de bruit est **paramètrable** et peut générer toute une variété de motifs selon les paramètres utilisés en entrée. Ces paramètres influencent directement ou non le spectre de puissance du motif généré.

Le spectre de puissance et Fourier

Le spectre de puissance d'un signal est obtenu en prenant le carré du module de la *Transformée de Fourier* du signal. Nous rappellons ici que la transformation de Fourier est l'outil de référence en traitement du signal, notamment utilisé pour l'analyse harmonique, cette transformation (inversible) sépare le signal en un spectre d'amplitudes et un spectre de phases, permettant par exemple de reproduire un signal périodique par une somme de fonctions sinusoïdales (séries de Fourier).

En informatique, c'est sous sa forme discrète qu'elle est utilisée majoritairement. Un signal discret s (une image par exemple) composé de N valeurs (pixels) se retrouve représenté, après transformation, par une ensemble S de N coefficients de Fourier, représentant complexe d'une amplitude A (en prenant le module de S) et d'une phase ϕ (en prenant l'argument de S) comme montré en Figure 2.1.

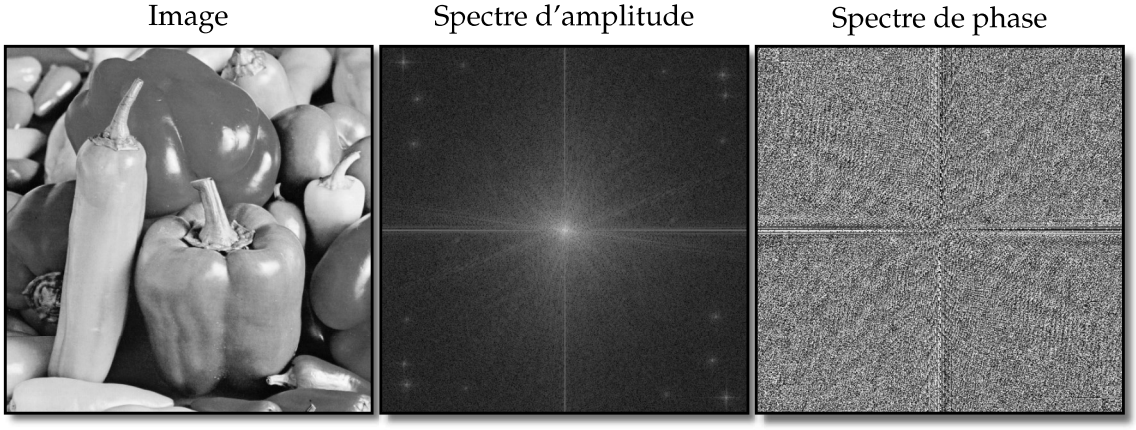


FIGURE 2.1 – La transformation de Fourier d'un image en entrée (à gauche) donne un spectre d'amplitude (au mileu) et un spectre de phase (à droite).

Dans ces deux spectres obtenus, le spectre d'amplitude présente l'amplitude de la fréquence f et le spectre de phase présente la localisation spatiale de l'image au travers d'une phase phi. L'idée que la localisation et la composante structurelle de l'image sont en grande partie représentées par la phase peut être visualisée lorsque nous inversons les phases de deux images et que nous tentons de reconstruire une image par transformée inverse de Fourier (cf. Figure 2.2).

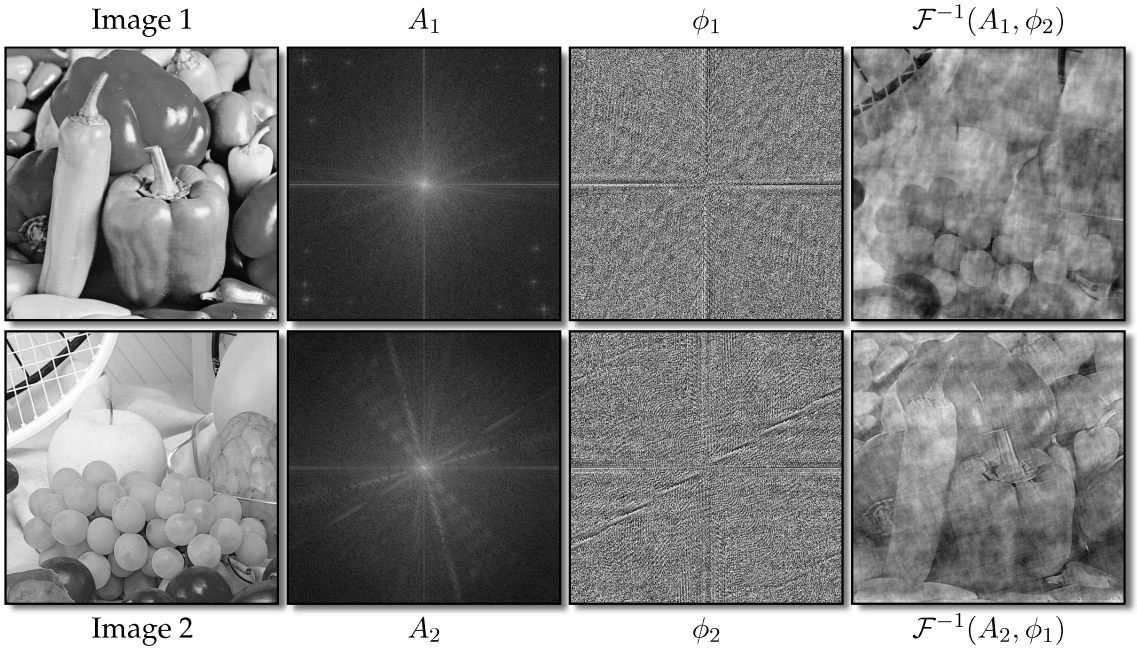


FIGURE 2.2 – Spectres d'amplitude et de phase de deux images. En reconstruisant les deux images par transformée de Fourier inverse en ayant échangé les spectres de phase, nous remarquons que la "structure" apparente des images est échangée.

Pour la synthèse de bruits, il est important de noter qu'il est possible de définir un *bruit à phase aléatoire* à partir de la transformée de Fourier d'une image. En partant d'une image en entrée, nous pouvons récupérer le spectre d'amplitude et le spectre de phase après application de la transformée de Fourier. Si nous remplaçons le spectre de phase par de l'aléatoire (c'est à dire une image dont la valeur de chaque pixel est tirée aléatoirement) et que nous appliquons la transformation inverse en combinant le spectre de fréquence obtenu précédemment avec cette image nous obtenons un bruit homogène partageant (par définition) le même spectre de puissance que le signal d'entrée (cf. Figure 2.3).

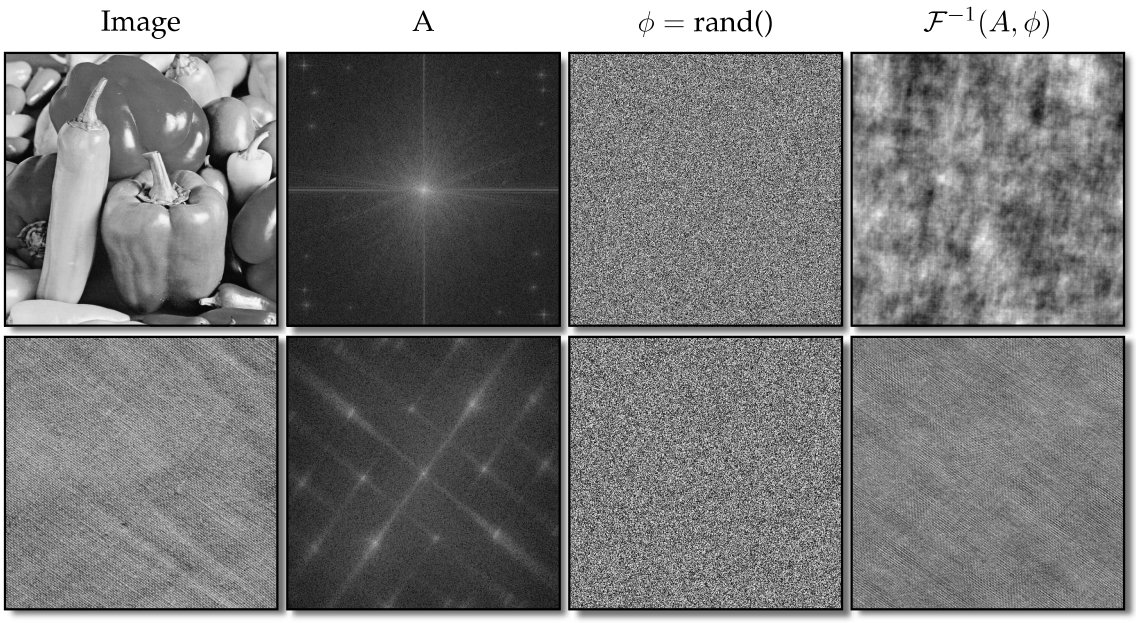


FIGURE 2.3 – En remplaçant le spectre de phase par de l'aléatoire, nous obtenons, après transformation inverse, un bruit homogène. Notons que pour la texture utilisée dans la ligne du bas, le bruit généré est très similaire. Cette texture étant entièrement définie par son spectre d'amplitude, elle fait partie de la catégorie des textures "Gaussiennes" [GLLD12]

Dans ce document, nous étudions principalement des fonctions continues et nous avons donc recours à sa forme continue. Dans la littérature, la fréquence est notée f ou ν et exprimée en cycles de période par unité de distance (Hz si l'unité est la seconde par exemple), mais nous préférons utiliser la notation de fréquence angulaire notée ω exprimée en radians par unité de distance. Les deux notations sont reliées par la relation $\omega = 2\pi f$.

Pour exprimer la relation entre une fonction continue f avec sa transformée de Fourier $\hat{f}$, nous notons le couple $f(\mathbf{x}) \longleftrightarrow \hat{f}(\omega)$, où la fonction $f(\mathbf{x})$ est définie dans le domaine spatial et la fonction $\hat{f}(\omega)$ est définie dans le domaine spectral.
La transformée de Fourier en dimension D est alors définie comme :

$$\hat{f}(\omega) = \int_{\mathbb{R}^D} f(\mathbf{x}) e^{-i\omega^T \mathbf{x}} d\mathbf{x} \tag{2.1}$$

Et son inverse par :

$$f(\omega) = \frac{1}{(2\pi)^D} \int_{\mathbb{R}^D} \hat{f}(\mathbf{x}) e^{i\mathbf{x}^T \omega} d\omega \tag{2.2}$$

Il est à noter une propriété importante en lien avec le produit de convolution de deux fonctions f et g : la transformée de Fourier du produit de convolution entre f et g est le produit de leurs transformées de Fourier respectives

$$(f * g)(\mathbf{x}) \longleftrightarrow \hat{f}(\omega)\hat{g}(\omega) \tag{2.3}$$

Inversement, la transformée de Fourier du produit de deux fonctions f et g est la convo-

lution de leurs transformées de Fourier respectives.

$$f(\mathbf{x})g(\mathbf{x}) \longleftrightarrow (\hat{f} * \hat{g})(\omega) \tag{2.4}$$

L'aliassage

L'aliassage (*aliasing* en anglais), ou repliement de spectre, est un défaut récurrent en informatique graphique et en traitement du signal. L'aliassage correspond à un phénomène introduisant, dans un signal échantillonné, des fréquences indésirables. Ce phénomène est généralement dû à une mauvaise reconstruction du signal échantillonné.

Lorsque l'on veut reconstruire un signal continu il est d'abord nécessaire de l'échantillonner. Á partir de ces échantillons discrets, il est possible de reconstruire une fonction continue en effectuant une opération de convolution avec un filtre de reconstruction.

Or, l'étape d'échantillonnage doit dépendre du contenu fréquentiel du signal en entrée afin d'obtenir une reconstruction correcte. En effet, si la fréquence du signal échantillonné est plus grande que la fréquence d'échantillonnage, le signal reconstruit ne reproduira pas le signal en entrée. Le théorème de Shannon-Nyquist stipule qu'il faut une fréquence d'échantillonnage supérieure au double de la fréquence maximale présente dans le signal en entrée. Pour régler le problème d'aliassage, une première solution est d'augmenter la fréquence d'échantillonnage pour respecter le théorème d'échantillonnage énoncé précédemment. Un autre solution consiste à appliquer un filtre passe-bas sur le signal en entrée pour en retirer les hautes fréquences afin de réduire la fréquence nécessaire pour discrétiser le signal (cf. Figure 2.4).

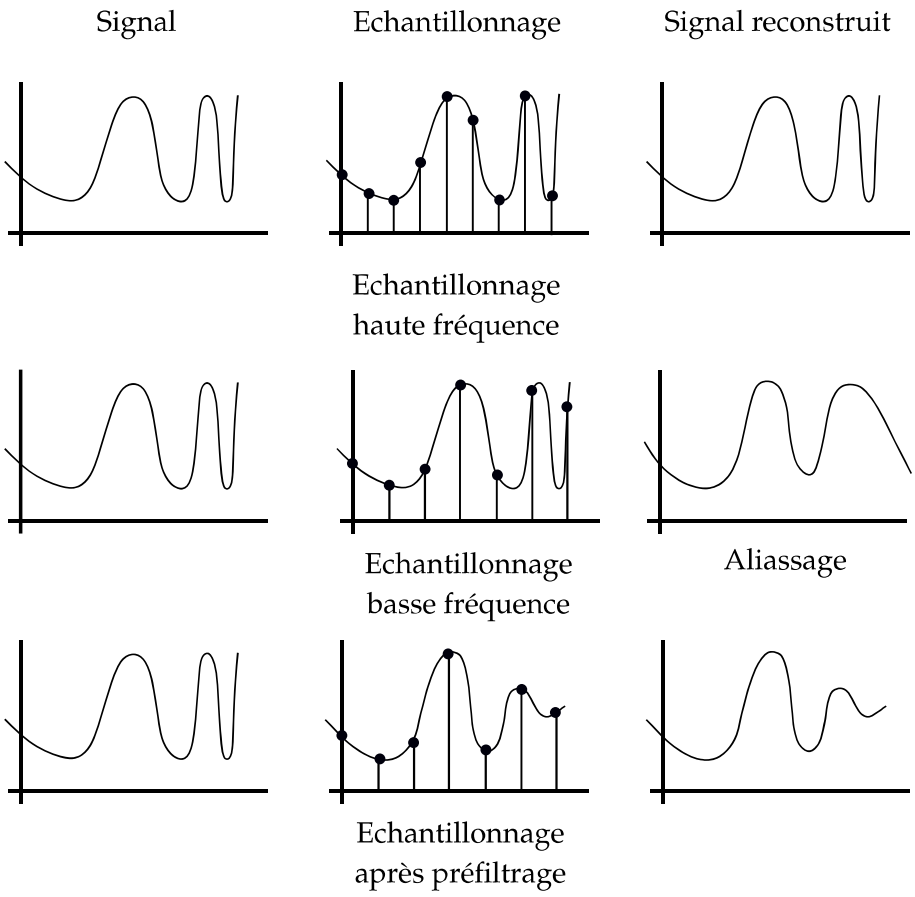


FIGURE 2.4 – La reconstruction d'un signal est effectuée avec une fréquence d'échantillonnage respectant le théorème de Shannon-Nyquist (en haut) : le signal est fidèlement reconstruit. Avec une fréquence d'échantillonnage plus basse que la fréquence maximale du signal, le signal est mal reconstruit : aliassage. Si l'on applique un filtre passe-bas sur le signal, nous retirons les hautes fréquences du signal ayant provoqué l'aliassage avec la fréquence d'échantillonnage basse fréquence. (Schémas issus de [Hec89]).

2.2 Le rendu filtré

Dans cette thèse, nous cherchons à synthétiser de manière correcte une texture procédurale au moment du rendu d'un objet 3D en temps réel. Pour y parvenir, nous devons prendre en compte les différentes échelles de détails [Hei14] venant influencer et définir l'apparence finale de notre objet texturé.

Nous travaillons avec une fonction continue représentant l'intégralité de la scène que nous voulons représenter. Si nous souhaitons obtenir une image de bonne qualité, nous devons évaluer par pixel l'intégrale de cette fonction sur le support du pixel.

En repartant du théorème d'échantillonnage présenté précédemment, une première solution est d'échantillonner finement le support du pixel. C'est ce qu'un moteur de rendu effectue quand on lance plusieurs rayons dans un pixel ou quand on effectue le rendu dans une résolution plus grande que celle attendue. On appelle cela le *supersampling* ou sur-échantillonnage.

Dans un contexte de rendu temps-réel, cette méthode peut devenir très vite impraticable en temps de calcul (complexité de la géométrie à parcourir pour un rayon, nombre de lumières à évaluer, etc.). De plus, il est impossible de définir une fréquence d'échantillonnage permettant de reconstruire les signaux qui ne sont pas limités en fréquence, c'est à dire un signal qui n'admet pas de fréquence maximale [Hec89].

L'approche correcte en synthèse d'image est d'adapter le contenu fréquentiel de la scène, notre fonction, à notre fréquence d'échantillonnage, liée à la résolution de notre image. C'est ce que l'on appelle le *préfiltrage*. L'idée est d'appliquer un filtre passe-bas avant l'étape d'échantillonnage. Le signal à échantilloner est alors limité en fréquence et peut être reconstruit fidèlement. En informatique graphique, l'action d'adapter le contenu fréquentiel de la scène est communément appellé le *niveau de détail* ou **LoD** (pour *level of details*).

Dans cette thèse, nous prendrons en compte trois échelles de détails pour notre pixel:

L'échelle microscopique

À cette échelle, le détail est représenté de manière statistique en utilisant une fonction de distribution de normales dans une fonction de distribution de réflectance bidirectionnelle (ou **BRDF** pour *Bidirectional Reflectance Distribution Function*). Cette fonction modélise l'interaction entre la lumière et la surface. Dans cette thèse, nous utilisons le modèle à micro-facettes de Cook-Torrance [WMLT07].

$$f(\omega_o, \omega_i) = \frac{D(\omega_h)F(\omega_o, \omega_h)G(\omega_o, \omega_i, \omega_h)}{4(\omega_o \cdot \omega_g)(\omega_i \cdot \omega_g)} \tag{2.5}$$

où ω_o et ω_i représentent respectivement les directions d'**o**bservation et d'**i**ncidence, ω_g représente la normale géométrique et ω_h représente le vecteur à mi-chemin caractérisé par :

$$\omega_h = \frac{\omega_o + \omega_i}{||\omega_o + \omega_i||} \tag{2.6}$$

La fonction $G(\omega_o, \omega_i, \omega_h)$ correspond au facteur d'atténuation géométrique, le terme de Fresnel $F(\omega_o, \omega_h)$ modélise l'absorption de la lumière par la surface et la fonction $D(\omega_h)$ correspond à la fonction de distribution de normales. Cette fonction de distribution représente le nombre de micro-facettes orientées dans la direction ω_h Nous utilisons au choix, une distribution de Beckmann ou une distribution de Trowbridge-Reitz (GGX).

Bien que des travaux récents en informatique graphique tendent à modéliser les interactions multi-rebonds (ou *multiple scattering*) nous nous limiterons, dans cette thèse, à une représentation simple rebond dans la surface.

L'échelle macroscopique

Cette échelle correspond à la représentation de l'objet sous forme de maillage triangulaire. Nous faisons l'hypothèse que la géométrie reste basse fréquence, ce qui n'est évidemment pas toujours le cas. Dans le cas où cette hypothèse n'est pas respectée, la géométrie présente soit des discontinuités sous le pixel, soit la géométrie sous pixel n'est pas localement planaire ce qui entraîne que des lobes spéculaires puissent être plus petit que la taille du pixel et donc ne puissent pas être capturés correctement lors du rendu. Dans le cas où la surface est très spéculaire, ce dernier phénomène va provoquer de l'aliassage spéculaire qui se manifeste par des scintillements indésirables.

Pour limiter ce problème, il est possible de filtrer le lobe spéculaire en utilisant la méthodologie décrite par [TK19], suivant les travaux de Kaplanyan *et al.* [KHPL16] et Tokuyoshi [Tok17]. Cette méthode consiste à représenter la déformation géométrique sous pixel sous forme de filtre passe-bas appliqué sur la fonction de distribution de normales D.

L'échelle mésoscopique

C'est à cette échelle que la texture synthétisée va directement impacter l'apparence de l'objet. Plusieurs types de textures sont généralement utilisés par les *pipelines* des moteurs de rendu pour modifier le visuel de l'objet.

Une première texture à considérer est la texture d'albedo définissant la couleur de la surface. En faisant l'hypothèse que la couleur de la surface est indépendante de sa géométrie, cette texture peut être filtrée par mipmapping (dans le cas d'une texture discrète) ou analytiquement en effectuant la convolution de la texture procédurale par l'empreinte du pixel projetée dans l'espace de texture. Nous nous placerons dans les situations où cette hypothèse est vraie, c'est à dire que la couleur de la surface ne dépend pas de la hauteur ou de la pente de la surface par exemple.

Un autre canal de texture régulièrement utilisé est la carte de normales modifiant la géométrie locale de la surface. Cette carte est utilisée pour perturber la normale géométrique ω_g de la surface dans le but de feindre des détails mésoscopiques. Cependant, cette texture, représentant de la "fausse" géométrie, ne prend pas en compte l'occlusion, la visibilité et l'auto-ombrage. Il est possible alors de percevoir des normales faisant dos à la caméra et à la lumière.

Cette texture ne peut en revanche pas être filtrée naïvement par du *mipmapping* ou

par convolution. En effet, la texture représente par *texels* une pente géométrique. Le *mip-mapping* sert généralement à évaluer la moyenne des intensités de la texture étudiée sous l'empreinte de pixel. Si cette opération est effectuée sur une carte de normales, la pente décrite va être "moyennée" au fur et à mesure que l'empreinte de pixel grandit, provoquant un "lissage" de la surface.

Le filtrage de carte de normales est toujours une problématique de recherche active en informatique graphique [BN12]. La solution avancée par de nombreux travaux est de représenter cette géométrie mésoscopique sous forme de fonction de distributions de normales D et de les préfiltrer. L'enjeu de ces méthodes est de proposer des fonctions de distributions compactes, précises et interpolables linéairement. Ces approches peuvent être séparées en deux familles : les représentations simple-lobe et les répresentations multi-lobes.

Les représentations *simple-lobes* vont approximer la distribution de normales par un lobe unique. Ce lobe est calculé en combinant linéairement les moments d'ordres deux des représentations des pentes [ON97, OB10, DHI+13]. Ces méthodes ne vont cependant pas capturer les transitions dures (arrêtes franches de la carte de normale). Les méthodes *multi-lobes* permettent d'approximer des distributions plus complexes [HSRG07] mais requierent un coût mémoire supplémentaire.

Pour conclure, nous considérons, dans un premier temps, que nous pouvons générer procéduralement des textures d'albedo filtrables dès lors qu'elles sont indépendantes de la géométrie de la surface. Dans le cas où l'on veut synthétiser des détails géométriques par une carte de normales, la question du filtrage doit être prise en compte. Dans la suite de cet état de l'art, nous étudierons, avec attention, les capacités de filtrage des différentes fonctions de bruit procédurales.

2.3 Les fonctions de bruits dans la synthèse de texture

Comme montré en introduction de ce document, le bruit est très utilisé en informatique graphique comme outil de création de contenu [EM03]. Que ce soit pour représenter des nuages, des vagues, des reliefs, le grain d'une pellicule de photo, ou d'autres phénomènes complexes présentant des motifs non structurés, les fonctions de bruit sont encore très étudiées aujourd'hui. Dans cette section, nous faisons l'inventaire des travaux sur les fonctions de bruits dans deux des trois grandes familles d'approches: les bruits par interpolations, les bruits explicites et les bruits par convolution éparse qui seront détaillés dans la section suivante.

2.3.1 Les bruits par interpolation (ou *Lattice Gradient Noise*)

Comme leur nom l'indique, ces méthodes sont basées sur l'interpolation de valeurs aléatoires ou des gradients sur une grille régulière. L'exemple le plus représentatif de cette catégorie est le bruit de Perlin car historiquement le premier introduit. Le bruit de Perlin [Per85, Per02] est basé sur l'interpolation de *spline* sur des gradients tirés aléatoirement sur les 8 plus proches sommets d'une grille régulière. Ces gradients sont obtenus en utilisant une fonction de hachage sur les coordonnées entières des sommets de la grille. Pour l'anecdote, cette méthode a été historiquement développée pour le film *Tron* par Ken Perlin afin de rajouter un grain sur les images de synthèse beaucoup trop "propres" par rapport aux images tournées en pellicule à l'époque. Elle a depuis été largement adoptée par l'industrie du fait de sa simplicité.

Dans le cours SIGGRAPH de [OHHM02], Perlin a aussi proposé une amélioration de sa méthode avec le bruit de Simplex [Per01] où il troque la grille carrée pour une grille de simplexe, limitant ainsi le nombre d'évaluation de gradient par pixel à 3 (pour de la 2D), ou à 4 (pour de la 3D), améliorant les performances. Il propose aussi de changer la fonction d'interpolation par une spline quintique d'Hermite afin d'améliorer la qualité du bruit généré. Dans les améliorations amenées par la suite, Kensler *et al.* [KKS08] proposent une table de permutation différente pour chaque dimension, un noyau de reconstruction différent pour améliorer le spectre obtenu en sortie et une méthode de projection pour améliorer l'évaluation du bruit sur une surface 2D en utilisant un bruit solide.

De nombreuses variantes ont été proposées [EM03], comme d'interpoler des valeurs tirées aléatoirement (*Value Noise*), d'effectuer une convolution discrète (*Lattice Convolution Noise*) ou encore de combiner valeurs et gradients pendant l'évaluation (*Value-Gradient Noise*). Après évaluation, nous obtenons une *octave* de bruit que l'on peut ensuite combiner avec d'autres pour obtenir des motifs plus complexes. Le livre *Texturing & Modelling: A Procedural Approach* [EM03] présente plusieurs méthodologies pour synthétiser des terrains (champs de hauteur), des formes ou des textures en utilisant ce genre de méthode comme primitive. Le *fractional brownian motion* ou **fBM**, par exemple, correspond à une somme de plusieurs octaves de fréquences croissantes d'un même bruit pour en définir un motif *autosimilaire* ou *fractal*. Ce genre de motif est généralement utilisé pour représenter des phénomènes naturels comme la montagne ou les nuages de la Figure 1.2. La Figure 2.5 synthétise cette catégorie de méthode et montre deux méthodes d'interpola-

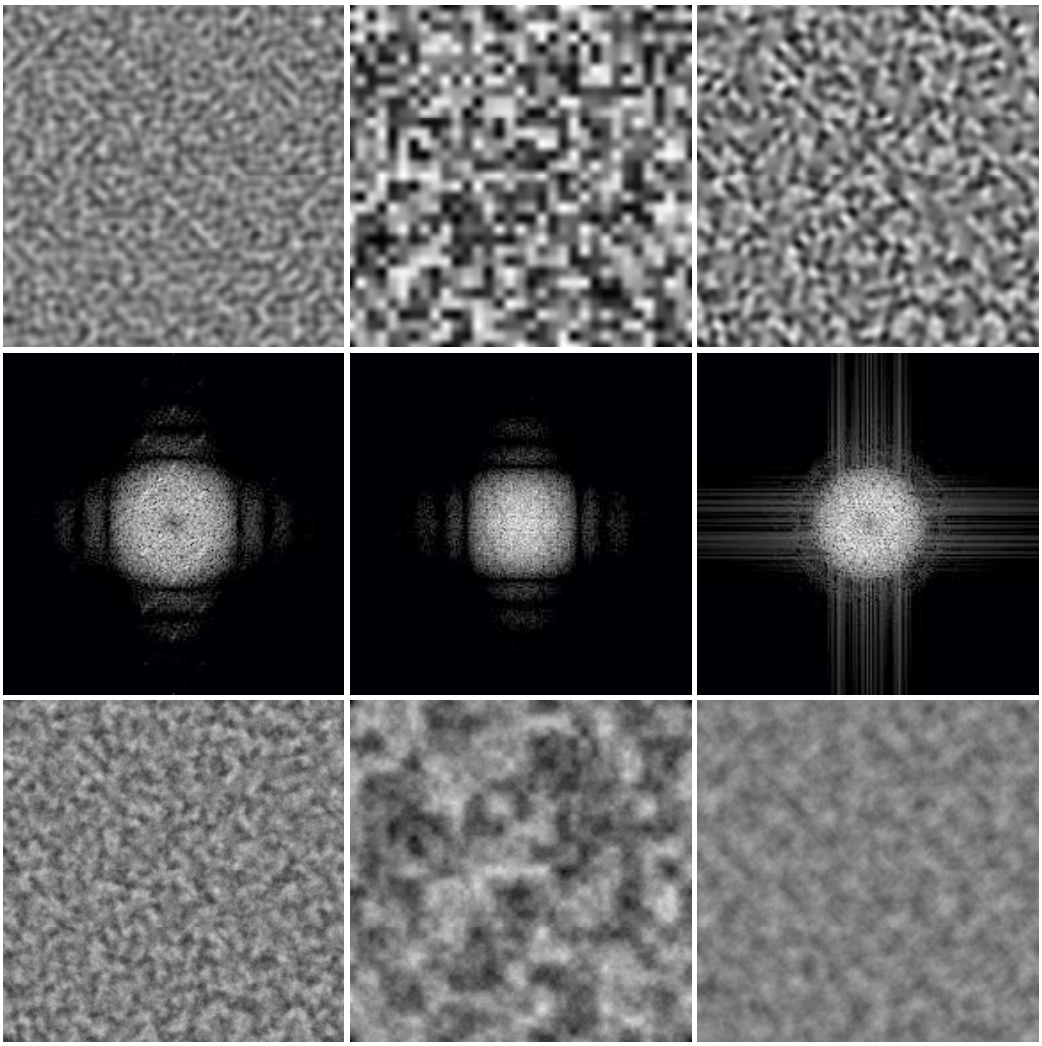


FIGURE 2.5 – Présentation de variantes de bruit par interpolation. Basée gradient (à gauche), basée valeurs (au milieu) sur une grille régulière et basée gradient sur une grille triangulaire (à droite). Du haut vers le bas,une octave évaluée, le spectre de cette octave et une somme d'octaves par **fBM**.

tion différentes : une basée gradient sur une grille régulière (1^{ere} colonne) et sur une grille de simplexe (3^{eme} colonne) et une interpolation basée sur les valeurs (2^{eme} colonne). Pour chaque méthode, elle montre une octave évaluée, le spectre de l'octave évaluée et un **fBM**.

En définissant la grille sous-jacente et le schéma d'interpolation dans des dimensions supérieures, les fonctions de bruit par interpolation peuvent facilement définir un *bruit solide*. Au lieu d'utiliser une paramétrisation en deux dimensions de la surface pour évaluer une fonction de bruit $f(x, y)$, il suffit d'échantillonner la fonction de bruit 3D $f(x, y, z)$ à la position de la surface.

Pour éviter l'aliassage pendant l'évaluation, ces méthodes ont recours au *tronquage fréquenciel* ou *frequency clamping*. L'idée est de retirer les octaves trop hautes fréquences en utilisant la fréquence d'échantillonnage des coordonnées de textures en espace image. Cette fréquence d'échantillonnage est calculée en utilisant les dérivées partielles des coordonnées de textures en espace image (dans un rendu par *rasterisation*) ou par lancer de rayon différentiel. L' image générée n'a plus de défaut d'aliassage mais présente une transition brutale vers une couleur moyenne puisque l'information est perdue quand l'octave est retirée de la somme lors de l'évaluation (cf. Figure 2.6).

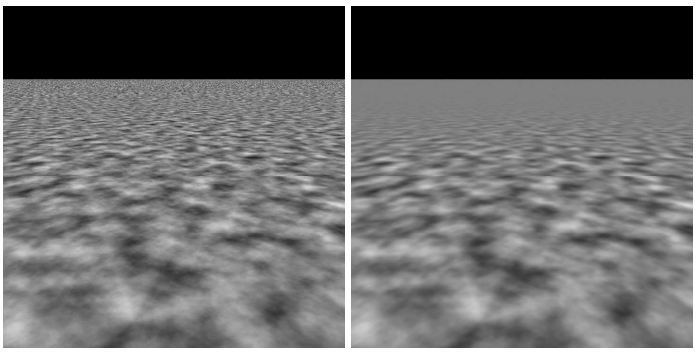


FIGURE 2.6 – Filtrage d'un **fBM** (à gauche) par *Frequency clamping* (à droite).

Des approches ont été proposées pour représenter, contrôler et guider localement les bruits par interpolation en s'appuyant sur des simulations physique. C'est le cas du *Flow Noise* introduit par Ken Perlin et Fabrice Neyret [PN01] et qui repose sur une déformation de la texture par un champs de vecteur (aussi appellé *advection* de texture). Malheureusement, ce type de méthode peut introduire des déformations trop importantes du motif d'entrée (modifiant ainsi le spectre final du motif). Des corrections ont été apportées à ce type de modèle, comme celles de Yu *et al.* [YNBH10] illustrées en Figure 2.7. Le modèle a aussi été étendu aux fluides incompressibles avec le *Curl Noise* par Bridson *et al.* [BHN07] en utilisant des champs de vélocité turbulents.

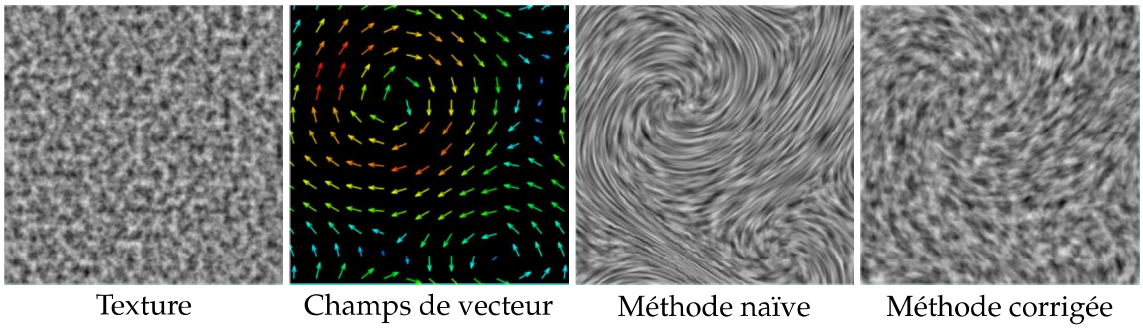


FIGURE 2.7 – Méthode de Yu *et al.* [YNBH10] où, à partir d'une texture en entrée (ici un bruit de Perlin) et d'un champs de vecteur représentant la vitesse ils génèrent une texture animée (tout à droite) conservant les propriétés statistiques de la texture d'entrée et sans déformations. [Figure de [YNBH10]]

2.3.2 Les bruits "explicites"

Nous présentons maintenant une catégorie de méthodes de génération de bruit que l'on nomme "bruits explicites". Ces méthodes se basent sur le précalcul de données utilisées lors de l'évaluation de la fonction de bruit. Ces méthodes ne sont pas à proprement parlé procédurales dans leur définition mais restent pertinentes dans les problématiques et améliorations qu'elles apportent aux fonctions de bruits procédurales en échangeant coût mémoire contre qualité du bruit généré.

Le premier représentant de cette catégorie est le *Wavelet Noise*, ou bruit d'ondelettes. Introduit par Cook et DeRose [CD05] en 2005, cette méthode propose de corriger des défauts des méthodes par interpolation comme le bruit de Perlin. En effet, le bruit de Perlin n'est que faiblement limité en fréquence, forçant un compromis entre défaut d'aliassage et perte de détails pendant l'évaluation (à cause du *frequency clamping*). De plus, [CD05] observent que l'évaluation d'un bruit solide pour texturer la surface ne garantit pas du tout que le spectre obtenu soit limité en fréquence.

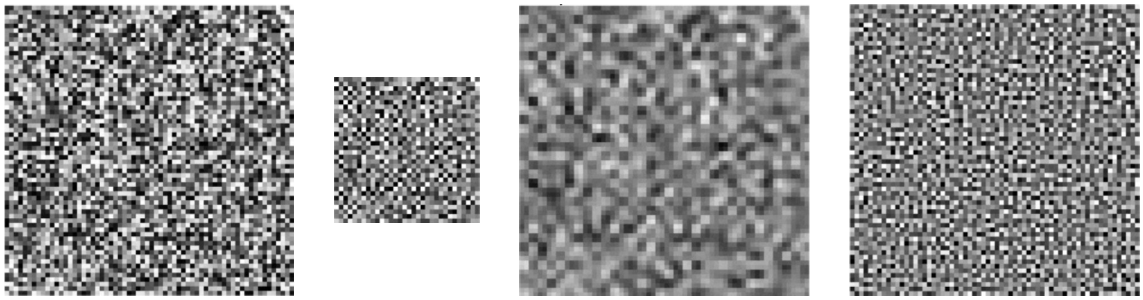


FIGURE 2.8 – Processus de génération d'un *Wavelet Noise*. [CD05] proposent de venir soustraire à image aléatoire R une image sous-échantillonée $R\downarrow$ puis sur-échantillonnée $R\downarrow\uparrow$ de R pour obtenir une tuile de coefficients N. [Figure de [CD05]]

Le but de la méthode est, comme le bruit de Perlin, de générer un bruit multi-resolution $M(\mathbf{x})$ comme une somme pondérée (par un poids w_b) d'octave de bruits $N(\mathbf{x})$. Cependant la construction et l'évaluation de $N(\mathbf{x})$ diffèrent d'un bruit par interpolation classique. Une décomposition en ondelettes est utilisée pour précalculer des octaves de bande b de fréquences limitées.

$$M(\mathbf{x}) = \sum_{b}^{B} w_b \, N(2^b \mathbf{x})$$

Le principe, illustré en partie en Figure 2.8, consiste à calculer une tuile N de coefficients du bruit en partant d'une tuile R remplie par de l'aléatoire uniforme. Une passe de sous-échantillonnage est réalisée pour obtenir une image $R \downarrow$ ayant une résolution de moitié moindre que la tuile utilisée. Cette image est ensuite sur-échantillonnée $R \downarrow\uparrow$ pour retrouver la même taille que la tuile R. La tuile de coefficients N est ensuite obtenue par soustraction de $R - R \downarrow\uparrow$. L'opération de soustraction permet de retirer de R les fréquences qui sont représentables à une résolution réduite de moitié. Ce qui reste dans N correspond à la bande de fréquence limitée uniquement à la résolution de R (c'est à dire la partie du signal de R qui n'est pas représentable à une résolution réduite de moitié).

Lors de l'évaluation, après avoir déterminé la résolution du bruit que l'on peut évaluer et la taille du support de la fonction de base utilisé (ici, une spline), $N(x)$ est calculé à l'aide de la tuile N contenant les coefficients de la *spline* au point $\mathbf{x}$. La contribution du bruit $N(\mathbf{x})$ disparait progressivement pour respecter la théorème d'échantillonnage afin de limiter les défauts d'aliassage (cf. Figure 2.9).

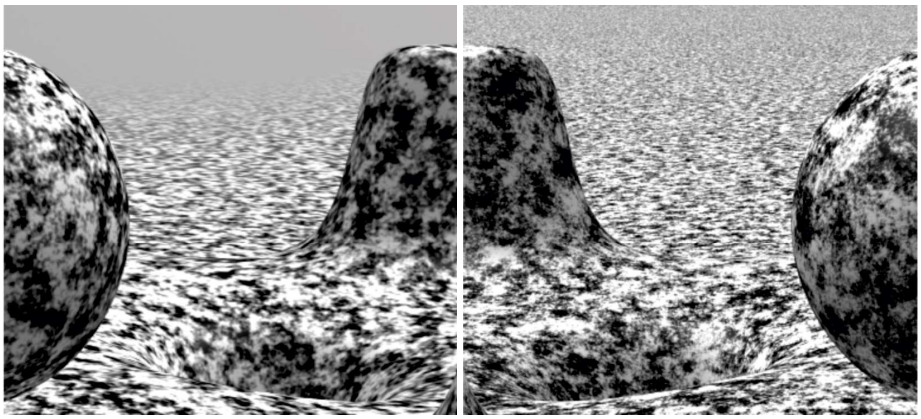


FIGURE 2.9 – Comparaison entre l'évaluation d'un bruit de perlin (à gauche) filtré par *frequency clamping* avec un bruit d'ondelette (à droite). Notons la différence de niveau de détails entre les deux images. [Figure de [CD05]]

Le filtrage est toujours effectué par *frequency clamping* mais la limitation des bandes de fréquences étant meilleure, les détails sont mieux conservés. Cependant, le *frequency clamping* ne considère pas l'anisotropie de l'empreinte de pixel.

Un autre représentant de cette famille est le bruit anisotrope, ou *Anisotropic Noise*, introduit par Goldberg *et al.* [GZD08] en 2008. Cette méthode a pour principe de générer un bruit en décomposant le spectre de puissance par des sous-bandes de fréquences orientées. Une fois ces bandes de fréquences extraites, une transformée de fourier inverse est réalisée avec une phase aléatoire afin d'obtenir des textures de bruits (cf. Figure 2.10). En choisissant correctement les sous-bandes de fréquences selon l'empreinte de pixel projetée, l'anisotropie est prise en compte.

En stockant une sous-bande par canal de texture sur des formats RGBA 32-bit, huit sous-bandes peuvent être stockées par deux textures discrètes. Ces textures sont ensuite

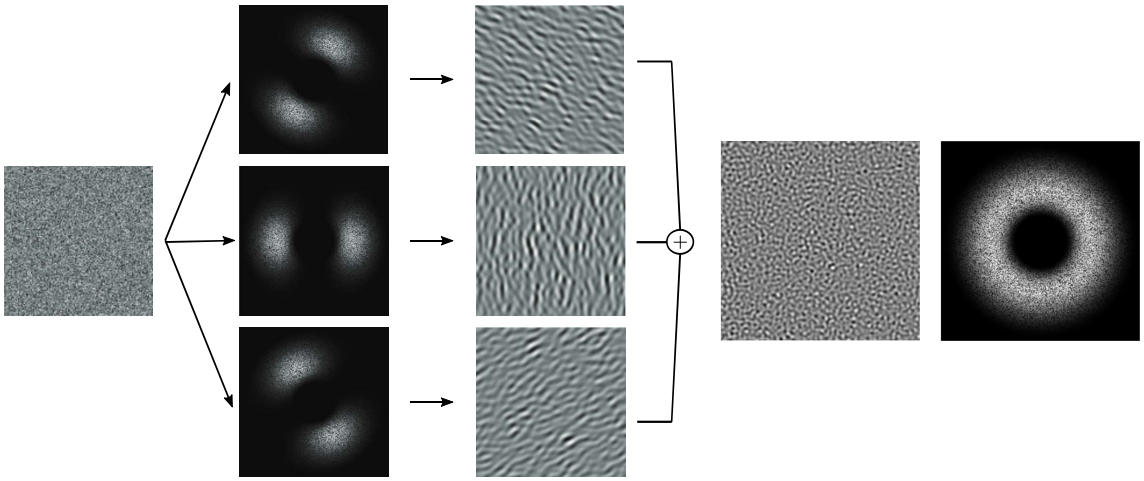


FIGURE 2.10 – Évaluation d'un bruit par synthèse spectrale. Après transformation inverse de Fourier de ces trois sous-bandes de fréquences orientées extraites (à gauche), nous obtenons trois bruits qui sont stockés sous forme de textures (au milieu). Ces textures sont ensuite sommées lors de l'évaluation afin d'obtenir un bruit isotrope (à droite).[Figures de [GZD08]]

sommées lors de l'évaluation dans un *pixel shader* en considérant l'orientation de la sous-bande et de l'empreinte de pixel dans le domaine spectral (cf. Figure 2.11).

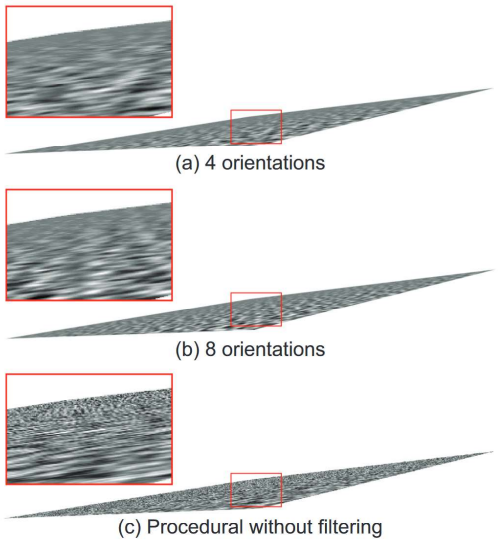


FIGURE 2.11 – Comparaison de l'évaluation procédurale d'un bruit anisotrope avec 4 sous-bandes de fréquences orientées (a), 8 sous-bandes de fréquences orientées (b) et sans filtrage (c). [Figure de [GZD08]]

2.4 La synthèse par convolution éparse

Dans cette section, nous présentons les travaux portant sur une catégorie de fonctions de bruits particulière : les bruits par convolution éparses. Nous y présentons les travaux fondateurs du *Sparse Convolution Noise* et du *Spot Noise* , les différentes variantes qui ont été proposées ainsi que les approches travaillant sur la synthèse par l'exemple. Nous finissons par les travaux les plus récents s'étant intéressés aux défauts inhérents de ces bruits procéduraux.

2.4.1 Des fondements au bruit de Gabor

Dans une série d'articles publiés entre 1984 et 1989, Lewis introduit le bruit par convolution éparse [Lew84, Lew86, Lew89]. Cette approche propose de construire une fonction de bruit comme étant la convolution d'une distribution de points issue d'un processus de Poisson γ par un noyau arbitraire k.

$$N(\mathbf{x}) = \int_{\mathbb{R}^2} \gamma(\mathbf{t}) k(\mathbf{x} - \mathbf{t}) \, d\mathbf{t} \tag{2.7}$$

Ce processus de Poisson est utilisé pour générer des impulsions δ avec des intensités a_i et des positions $\mathbf{x}_i$ tirées aléatoirement.

$$\gamma(\mathbf{x}) = \sum_i a_i \, \delta(\mathbf{x} - \mathbf{x}_i) \tag{2.8}$$

Comme en atteste le delta de Dirac δ, ce processus n'est pas défini en tout point de l'espace. La distribution de points qui en résulte est dite "éparse", c'est à dire qu'elle implique une contrainte de distance entre deux points d'évaluations (donnant son nom à cette catégorie de méthodes). Les noyaux k choisis sont généralement des noyaux à support fini ou tronqué, leur contribution étant nulle après une certaine distance.

Pour évaluer un bruit par convolution éparse pour un pixel donné, il suffit alors d'évaluer uniquement les noyaux dont le support recouvre le dit pixel (cf. Figure 2.12). En pratique, cette recherche de points est accélérée par une grille régulière dont la taille d'une cellule dépend de la largeur du support du noyau utilisé. La formulation du *Sparse Convolution Noise* peut alors être approximée comme étant la somme des N noyaux de convolutions centrés aux impulsions x_i pondérée par w_i

$$N(\mathbf{x}) \approx \sum_i^N w_i \, k(\mathbf{x} - \mathbf{x}_i) \tag{2.9}$$

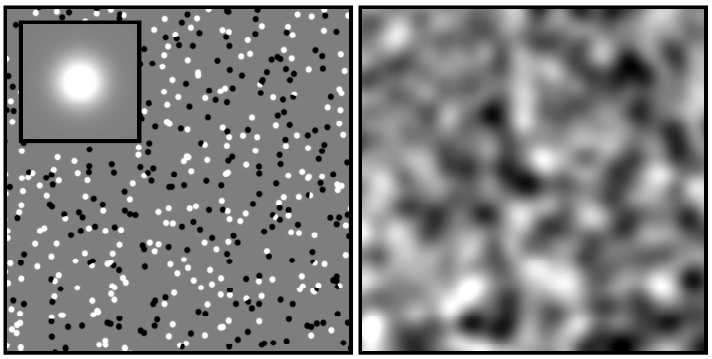


FIGURE 2.12 – En effectuant la convolution de la distribution de points (à gauche) par un noyau arbitraire (ici une gaussienne, en vignette), nous obtenons le motif de droite.

Sur les même principes que le *Sparse Convolution Noise* , le *bruit par noyau ponctuel* ou *Spot Noise* , introduit par Van Wijk en 1991 [vW91], est initialement proposé comme un outil de visualisation scientifique et de synthèse de textures stochastiques. En suivant la formulation décrite en équation (2.9), la texture est générée en effectuant la somme pondérée d'un *spot*. Van Wijk observe que si le noyau présente un motif structuré, la texture générée incorpore un motif similaire (cf. Figure 2.13).

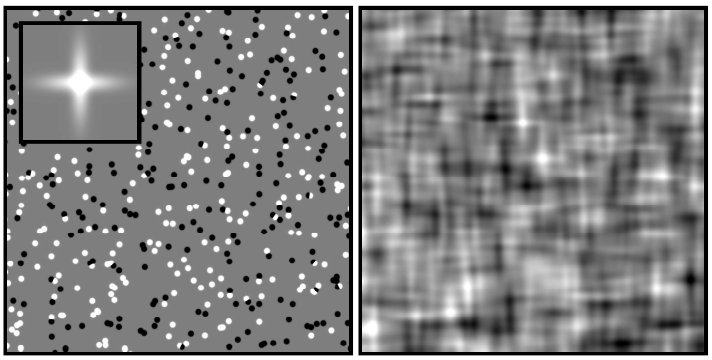


FIGURE 2.13 – La forme décrite par le noyau (en vignette) se retrouve dans la texture générée [vW91].

Ces approches partagent le fait que le contrôle du motif final passe par le contrôle de la fonction de noyau. Différents travaux ont donc été proposés afin d'étudier des noyaux particuliers pour les méthodes de *Sparse Convolution Noise* . Lagae *et al.* [LLDD09] proposent en 2009 un bruit par convolution éparse s'appuyant sur un noyau de Gabor. Ce noyau est défini comme le produit d'une fonction gaussienne et d'une harmonique :

$$\begin{aligned}
\text{Gaussian}(x, y; K, \alpha) &= K\ exp(-\pi\alpha^2(x^2 + y^2)) \\
\text{Harmonic}(x, y; F_0, w_0) &= cos(2\pi F_0(xcos(w_0) + ysin(w_0))) \\
\text{Gabor}(x, y; K, \alpha, F_0, w_0) &= \text{Gaussian}(x, y; K, \alpha)\text{Harmonic}(x, y; F_0, w_0)
\end{aligned}$$

où K correspond à l'amplitude de l'enveloppe gaussienne, α sa largeur, F_0 et w_0 respectivement une fréquence et un angle pour orienter l'harmonique. Le bruit de Gabor est

donc par extension la somme pondérée de noyaux de Gabor (cf. Figure 2.14) :

$$\text{GaborNoise}(x, y) = \sum_{i}^{N} w_i \, \text{Gabor}(x - x_i, y - y_i; K, \alpha, F_0, w_0) \tag{2.10}$$

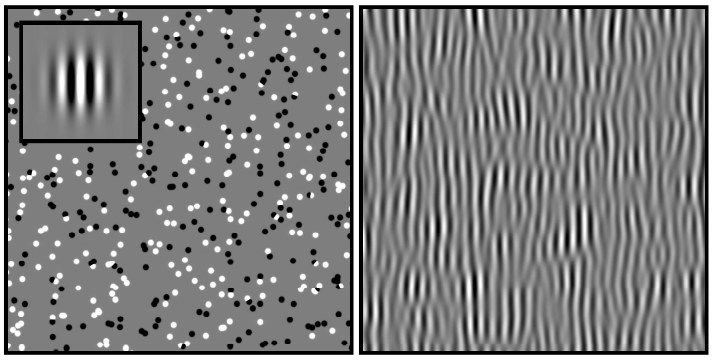


FIGURE 2.14 – Le bruit de Gabor [LLDD09].

La formulation analytique du noyau est l'atout majeur de cette fonction de bruit puisque la transformée de Fourier de ce noyau correspond à une somme de gaussiennes dans l'espace fréquentiel. En effet, en utilisant le théorème de convolution présenté en début de ce chapitre, la transformée de Fourier en deux dimensions du noyau de Gabor correspond au produit des transformées de l'harmonique et de la fonction gaussienne. La transformée de Fourier de la fonction harmonique correspond à la somme de deux delta de Dirac symétriques par rapport à l'origine du domaine spectral et celle de l'enveloppe gaussienne correspond à une gaussienne. Pour un lobe de Gabor, nous obtenons alors la somme de deux gaussiennes placées symétriquement par rapport à l'origine du domaine spectral (comme le montre la figure 2.15).

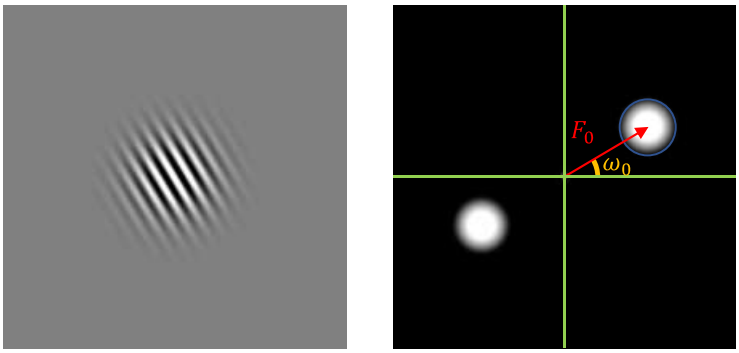


FIGURE 2.15 – La transformée de Fourier d'un noyau de Gabor (à gauche, dans le domaine spatial) donne une somme de gaussiennes symétriques (à droite, dans le domaine spectral).

La transformée de Fourier d'un noyau de Gabor telle qu'énoncée dans [LLDD09] est :

$$\hat{Gabor}(\omega_x, \omega_y) = \frac{K}{2\alpha^2}\left[e^{\frac{-\pi}{2\alpha}\left[(\omega_x - F_0 cos(w_0))^2 + (\omega_y - F_0 sin(w_0))^2\right]} + e^{\frac{-\pi}{2\alpha}\left[(\omega_x + F_0 cos(w_0))^2 + (\omega_y + F_0 sin(w_0))^2\right]} \right] \tag{2.11}$$

Ce spectre analytique permet alors une visualisation directe du contenu fréquentiel du motif généré mais aussi le filtrage analytique de ce bruit. En effet, [LLDD09] présentent aussi une méthodologie pour filtrer analytiquement le bruit de Gabor. Le but est d'évaluer la convolution de l'empreinte de pixel projetée dans l'espace de texture avec la somme de noyaux de Gabor. L'empreinte projetée est représentée comme une gaussienne anisotrope en suivant les travaux de Heckbert [Hec89].

Or, comme énoncé précédemment, il se trouve que la convolution de deux fonctions dans le domaine spatial devient un produit une fois dans le domaine spectral. De plus, une gaussienne restant une gaussienne après transformation de Fourier, le filtrage analytique d'un noyau de Gabor correspond au produit d'une gaussienne avec la transformée de Fourier du noyau de Gabor (une somme de gaussiennes (cf. équation(2.11)). Les gaussiennes admettant une forme close sous produit, le spectre reste une somme de gaussiennes. Il est alors possible de déduire les paramètres d'un noyau de Gabor ayant le spectre recherché à partir de l'expression analytique évaluée. L'évaluation filtrée du bruit de Gabor revient alors à sommer des noyaux de Gabor préfiltrés par une empreinte de pixel gaussienne en utilisant une forme close.

La *Gabor Noise* introduit le concept de "*setup-free surface noise*" en proposant une méthode pour se passer de paramétrisation de la surface. Pour une position $\mathbf{p}$ et une normale $\mathbf{n}$, les auteurs proposent d'évaluer leur texture procédurale en projetant localement les impulsions du processus de Poisson en trois dimensions sur le plan déterminé par $\mathbf{p}$ et $\mathbf{n}$. En orientant aléatoirement le repère local de la surface, nous obtenons un motif isotrope. Respectivement, en alignant le repère local sur un champ de vecteurs, nous obtenons un motif anisotrope. Cependant, en s'appuyant sur la normale géométrique de la surface, des discontinuités peuvent apparaître entre deux faces du modèle. Les auteurs préconisent alors d'utiliser des normales douces entre les sommets pour briser ces discontinuités.

Par la suite, plusieurs améliorations ont été apportées sur ce modèle. Lagae *et al.* [LLD11] améliorent la synthèse de motif isotrope et proposent une méthodologie pour synthétiser des motifs variant spatialement. Lagae *et al.* [LD11] augmentent le noyau de Gabor d'une phase ϕ afin d'étendre la synthèse de texture non plus surfacique mais solide.

$$\text{Harmonic}(x, y; F_0, w_0, \phi) = cos(2\pi F_0(xcos(w_0) + ysin(w_0) + \phi))$$
$$\text{PhasedGabor}(x, y; K, \alpha, F_0, w_0, \phi) = \text{Gaussian}(x, y; K, \alpha)\text{Harmonic}(x, y; F_0, w_0, \phi)$$

En ajoutant une phase ϕ tirée aléatoirement, Lagae *et al.* retirent le paramètre w_i du

bruit de Gabor, devenant ainsi :

$$\text{GaborNoise}(x, y) = \sum_{i}^{N} \text{PhasedGabor}(x - x_i, y - y_i; K, \alpha, F_0, w_0, \phi) \tag{2.12}$$

Le but est d'évaluer le bruit de Gabor en trois dimensions et de définir la texture de la surface comme étant la coupe 2D de ce bruit solide. Le bruit de Gabor solide est évalué sur une grille en trois dimensions permettant de retrouver les noyaux de Gabor 3D proches de la surface à texturer. Une coupe 2D du noyau 3D est calculée, filtrée analytiquement et ajoutée à la contribution du point évalué (cf. Figure 2.16).

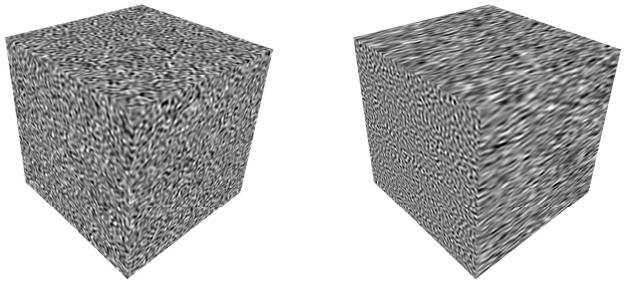


FIGURE 2.16 – Lagae *et al.* [LD11] propose un bruit solide avec un filtrage anisotrope de grande qualité sans discontinuités entre les faces du modèle que ce soit pour un motif isotrope (à gauche) ou un motif anisotrope (à droite).

Le bruit de Gabor offre un contrôle précis de son contenu fréquentiel, la possibilité de se passer de la paramétrisation de la surface, et d'obtenir une texture sans défauts d'aliassage. Cependant, le contrôle utilisateur reste malgré tout non-intuitif. Dans les faits, l'artiste doit définir le spectre de puissance manuellement, par le calcul ou en reproduisant un exemple pour obtenir un bruit. Différentes approches ont alors été proposées pour améliorer le contrôle de ce genre de méthodes.

En 2010, Bénard *et al.* [BLV+10] présentent une évaluation du *Gabor Noise* un peu différente pour faire du rendu stylisé. Ce *NPR Gabor noise*, pour *Non Photorealistic Gabor noise*, sépare la distribution de point, réalisée sur la surface du modèle à texturer, de l'évaluation du bruit de Gabor évaluée en espace écran (cf. Figure 2.17).

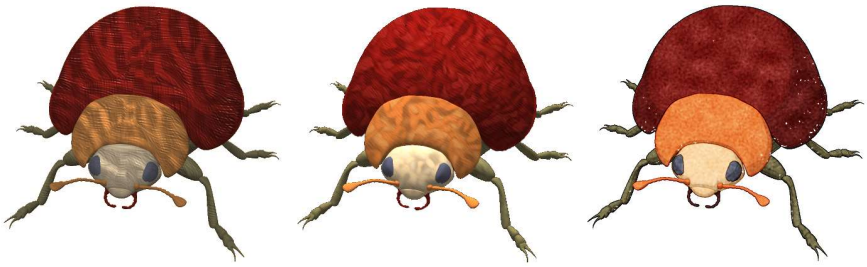


FIGURE 2.17 – Bénard *et al.* [BLV+10] présente une méthode d'évaluation du *Gabor Noise* pour effectuer du rendu stylisé.

Charpenay *et al.* [CSM14] proposent d'influencer les paramètres d'un bruit de Gabor en utilisant des informations spatiales. L'idée est de définir des *cartes de contrôle* en utilisant des textures discrètes ou des fonctions continues (cf. Figure 2.18). Ces cartes sont ensuite utilisées comme paramètres du bruit de Gabor pour offrir un outil de contrôle spatial aux artistes.

FIGURE 2.18 – Le motif de la carte de contrôle (en bas) vient influencer les paramètres d'orientation, de fréquence et d'amplitude des noyaux de Gabor et offre un meilleur outil d'édition pour un artiste (Figure [CSM14])

En 2016, Pavie *et al.* [PGDG16] proposent *le bruit par noyaux ponctuels localement contrôlés* introduisant un noyau de *Spot Noise* particulier. Ce noyau est défini comme une somme de gaussiennes elliptiques définies spatialement. La paramétrisation du noyau est alors entièrement géométrique et intuitive et étendable à la 3D [PGD+16] (cf. Figure 2.19). Le contrôle sur l'apparence finale de la texture est obtenu en construisant le noyau et en utilisant une distribution de points contrainte et contrôlable. Nous détaillons cette méthode en Chapitre 3.

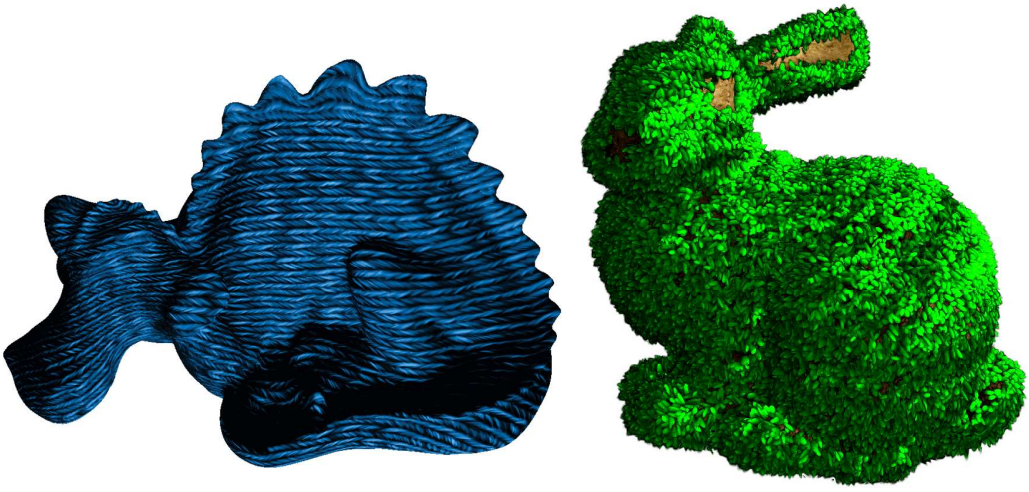


FIGURE 2.19 – Pavie *et al.* [PGDG16,PGD+16] propose un noyau pour de la synthèse de texture basée sur du *Spot Noise* . Le noyau (2D à gauche, 3D à droite) est représenté par une somme de gaussiennes et l'apparence est contrôlée par une distribution contrainte.

2.4.2 Le bruit par l'exemple

Une autre approche proposée pour améliorer le contrôle sur l'apparence finale est d'extraire directement les paramètres d'un bruit à partir d'un exemple fourni. En étudiant le spectre de puissance d'une texture d'exemple, les paramètres d'une fonction de bruit sont déduits de manière automatique.

Ghazanfarpour et Dischler [GD95] proposent de créer des textures 3D automatiquement en analysant le spectre d'une texture 2D et d'effectuer une somme de cosinus parametrée par cette phase d'analyse. La méthode effectue d'abord la transformée de Fourier $S(x, y)$ de l'exemple 2D, puis sélectionne les termes du spectre et de la phase dont l'amplitude est supérieure à un certain seuil pour paramétrer une somme de cosinus $s'(x, y)$

$$s'(x, y) = A_0 + \sum_{i}^{T} A_i \, cos(2\pi(f_i x + g_i y) + \phi_i)$$

où T est le nombre de termes sélectionnés, A_0 la moyenne des valeurs de la texture d'entrée, enfin, A_i,f_i,g_i et ϕ_i respectivement l'amplitude, les fréquences et la phase retenues pour le terme i. Toujours en utilisant le spectre S de l'exemple d'entrée, une perturbation est calculée en filtrant un bruit blanc. Cette perturbation est ensuite combinée avec s' pour obtenir une texture 3D $t(x, y, z)$ avec un aspect plus "naturel" (cf. Figure 2.20).

FIGURE 2.20 – Résultat de la méthode proposée par Ghazanfarpour et Dischler [GD95]

La méthode est peu robuste aux textures d'exemple bruitées et n'utilise qu'une seule texture 2D pour définir l'entièreté de la texture 3D. L'analyse conjointe de deux ou trois textures peut être réalisée pour synthétiser un texture 3D de meilleure qualité [GD96].

Plus récemment, Galerne *et al.* [GGM10] décomposent des textures par transformée de Fourier et remplacent la phase par de l'aléatoire pour créer des bruits homogènes en conservant le même spectre de puissance de la texture d'entrée. Une étude des modèles

de bruits à phase aléatoire et de *Spot Noise* est réalisée pour proposer une extension à la synthèse de textures en couleurs et une étape de pré-traitement de l'exemple permettant d'éviter les défauts visuels dus à la non-périodicité des textures d'exemples.

Lagae *et al.* [LVLD10] proposent une méthode pour générer des textures stochastiques en effectuant une somme pondérée de *wavelet noise* isotropes sur plusieurs échelles dont les poids sont calculés depuis une image en entrée. Pour gérer la synthèse de texture en couleur, une analyse en composantes principales et une correspondance d'histogramme sont utilisées. Bien que la méthode conserve le spectre de puissance de l'exemple, elle échoue à reproduire des motifs structurés ou anisotropes.

Gilet *et al.* [GDS10] présentent une méthode s'appuyant sur le bruit de Gabor pour décrire également les textures anisotropes. Le spectre de puissance d'un exemple est discrétisé en quelques valeurs pour ensuite ajuster des ellipses dans le spectre et y associer un bruit de Gabor. Cette méthode utilise une approche de synthèse de texture en couleurs similaire à [LVLD10]. Gilet et Dischler réutiliseront cette méthode de synthèse la même année pour leur travail sur la synthèse de détails volumétriques à partir d'une image [GD10].

Gilet *et al.* [GDG12] proposent une autre décomposition du spectre de puissance sous forme de sous-ensemble rectangulaires. Cette décomposition est ensuite utilisée pour ajuster les paramètres de noyaux de Gabor ou des sinus cardinaux comme montré en Figure 2.21.

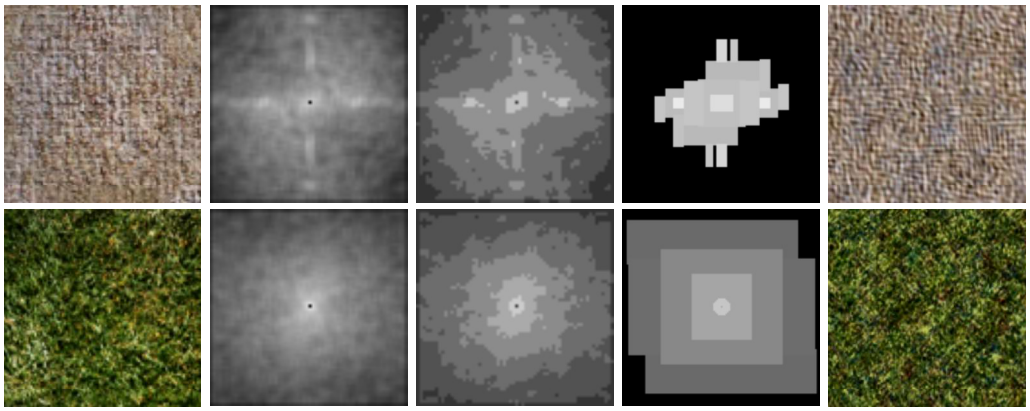


FIGURE 2.21 – À partir d'un exemple en entrée et de son spectre de puissance (1^{ere} et $2^{\text{ème}}$ colonnes), Gilet *et al.* [GDG12] décomposent le spectre en bandes de fréquences rectangulaires ($4^{\text{ème}}$ colonne) pour paramétrer au choix des noyaux de Gabor ou des sinus cardinaux et synthétiser une texture semblable (dernière colonne).

En effet, le choix du sinus cardinal pour la synthèse est justifié par le fait que sa transformée de fourier correspond à la fonction rectangulaire :

$$\text{sinc}(x) = \frac{sin(\pi x)}{\pi x} \longrightarrow \hat{\text{sinc}}(\omega) = \text{rect}(\frac{\omega}{2\pi})$$

La décomposition du spectre peut être aussi utilisée pour la perturbation de motifs. De façon similaire à Ghazanfarpour *et al.* [GD95] et avec l'idée de turbulence introduite par Perlin [Per85], Gilet *et al.* proposent une méthode de synthèse par perturbation (illustrée en Figure 2.22). L'utilisateur choisit tout d'abord une texture d'exemple et définit un motif régulier $P(\mathbf{x})$ (par du dessin vectoriel par exemple). Après analyse du spectre

de puissance, la texture résultat $T(\mathbf{x})$ est calculée comme la somme de $P(\mathbf{x})$ et du bruit $\lambda\, n_S(\mathbf{x})$ généré, λ étant un facteur d'amplitude de la perturbation introduite.

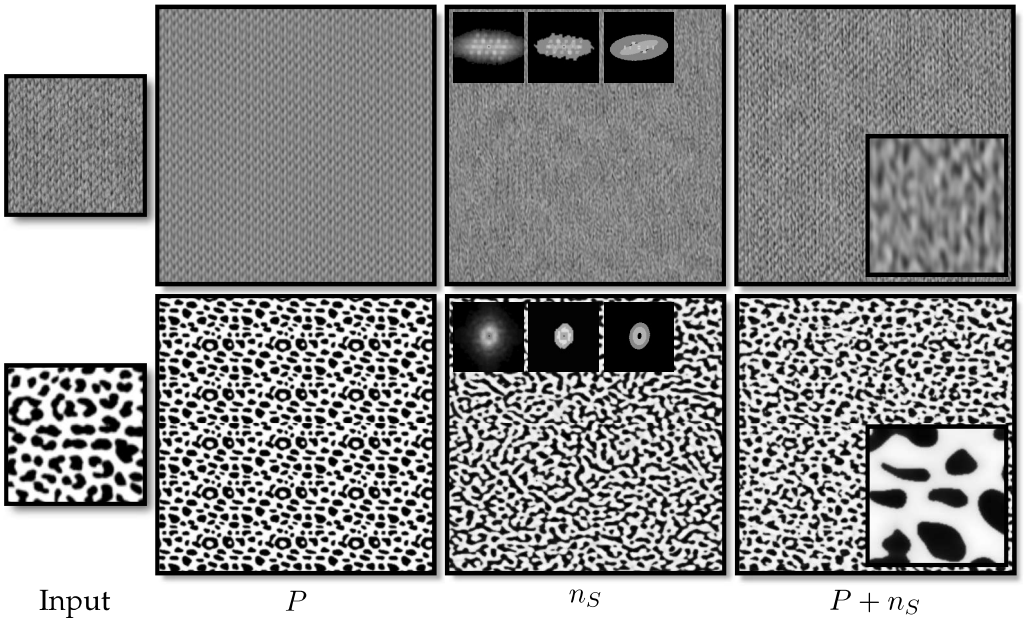


FIGURE 2.22 – Deux exemples de perturbations synthétisées par l'exemple avec la méthode de [GDG12]. L'utilisateur définit un motif régulier P et la méthode génère un bruit basé sur le spectre de puissance

L'idée de synthétiser une texture composite en définissant plusieurs couches/calques de manière hiérarchique est évoquée dans ce travail. En synthétisant des contenus basés sur le spectre de puissance on obtient les "micro-textures" qui vont venir habiller une segmentation binaire (laissée au contrôle de l'utilisateur) de la texture "structurante", un exemple est présenté en Figure 2.23. Cependant aucune méthode de génération automatique n'est proposée, la séparation multi-échelle des différentes "micro-textures" de la texture composite étant un problème ouvert.

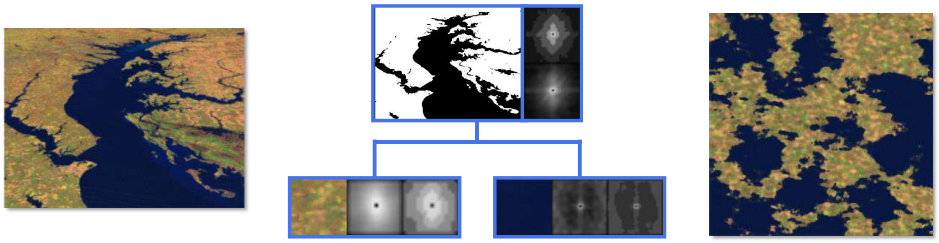


FIGURE 2.23 – Gilet *et al.* [GDG12] présentent l'idée de décrire des textures composites (à gauche) comme une hiérarchie de textures procédurales (milieu) et de venir les combiner ensuite (à droite).

Galerne *et al.* présentent le bruit de Gabor paramétré par l'exemple [GLLD12]. Les auteurs identifient une catégorie de textures dites "gaussiennes" qui, à l'instar du bruit, ont la particularité d'être entièrement définies par leur spectre de puissance. Une nouvelle méthode de décomposition du spectre de puissance robuste et automatique est proposée pour obtenir une somme de gaussiennes multivariées à différents niveaux de largeurs.

L'estimation des paramètres est réalisée en effectuant une optimisation convexe [KKL+07]

résolue en utilisant la méthode proposée par [BT09]. Pour chaque largeur de noyaux estimée, une somme de bruits de Gabor est réalisée par une évaluation multi-grilles. Les paramètres des noyaux sont obtenus par échantillonnage préférentiel pour obtenir un compromis entre nombre de noyaux évalués par pixel et qualité visuelle de l'exemple. Les textures "gaussiennes" générées sont ainsi continues et présentent un spectre de puissance reconstruit fidèlement (cf. Figure 2.24).

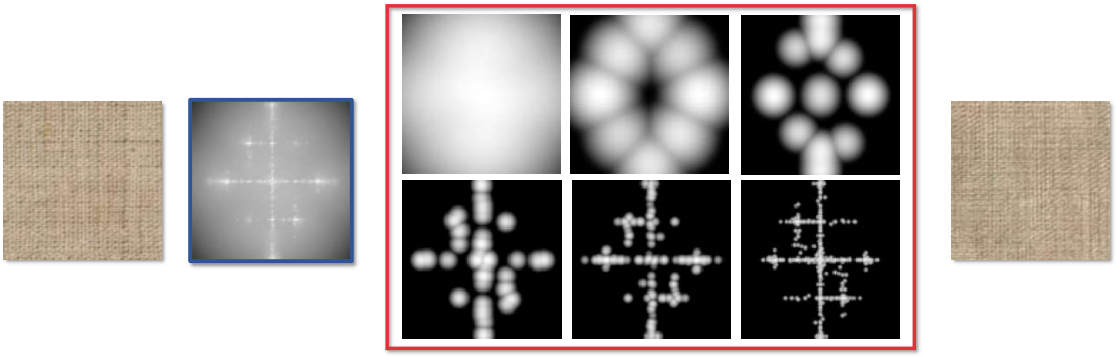


FIGURE 2.24 – Galerne *et al.* [GLLD12] proposent une méthode automatique qui, à partir d'un exemple "Gaussien" (à gauche), discrétise le spectre de puissance (en bleu) en plusieurs somme de gaussiennes (en orange) pour synthétiser un texture "gaussienne" (à droite).

En 2014, Gilet *et al.* [GSV+14] présentent un nouveau modèle analytique de bruit : le bruit à phase aléatoire locale (ou **LRPN** pour *Local Random Phase Noise*). Bien que reposant sur le principe du bruit à convolution éparse, la distribution de points aléatoire est évincée au profit d'une évaluation sur une grille régulière. En lieu et place d'un noyau de Gabor, c'est une somme de cosinus qui est proposée le tout multipliée par une fonction de fenêtrage w :

$$\text{LRPN}(\mathbf{x}) = \sum_{i}^{I} w(\frac{||\mathbf{x} - \mathbf{x}_i||}{\Delta}) \sum_{j}^{J} A_{ij} cos(2\pi f_{ij} \cdot \mathbf{x} + \phi_{ij}) \tag{2.13}$$

où A_{ij}, f_{ij} et ϕ_{ij} sont respectivement l'amplitude, la fréquence et la phase du cosinus et w une fonction de fenêtrage de largeur Δ centrée au point $\mathbf{x}_i$. Pour cette dernière, l'utilisation d'une fonction de fenêtrage de Kaiser-Bessel est proposée. Cette formulation est plus adaptée que la distribution de points pour une évaluation sur carte graphique puisque elle limite l'évaluation par pixel à 9 noyaux de J cosinus.

Ce modèle analytique, illustré en Figure 2.25, peut être paramétré par l'exemple. Comme pour [GLLD12], une phase d'analyse est réalisée sur CPU où le spectre de puissance est calculé, ensuite partitionné et chaque partition est ensuite resubdivisée par un algorithme de k-moyennes (ou *k-means*). Après transfert des données sur GPU, chaque partition est échantillonnée pour retrouver A_{ij}, f_{ij} et tirer une phase ϕ_{ij} aléatoire. Des bruits à phase aléatoire locaux sont ensuite évalués (cf. Figure 2.26):

$$\text{GaussTex}(\mathbf{x}) = \sum_{S} \text{LRPN}_S(\mathbf{x}) \tag{2.14}$$

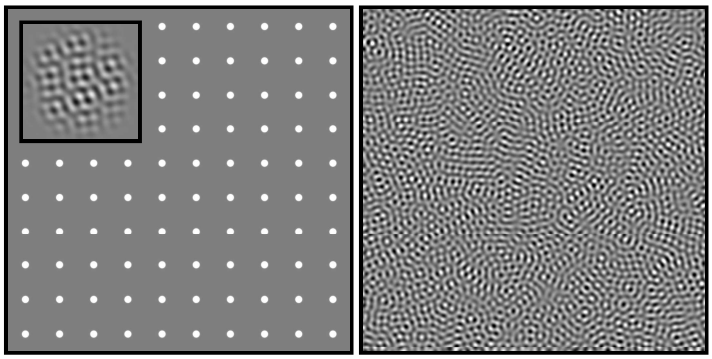


FIGURE 2.25 – Gilet *et al.* [GSV+14] utilisent cette fois une distribution régulière de points pour évaluer un noyau définit comme la somme de cosinus.

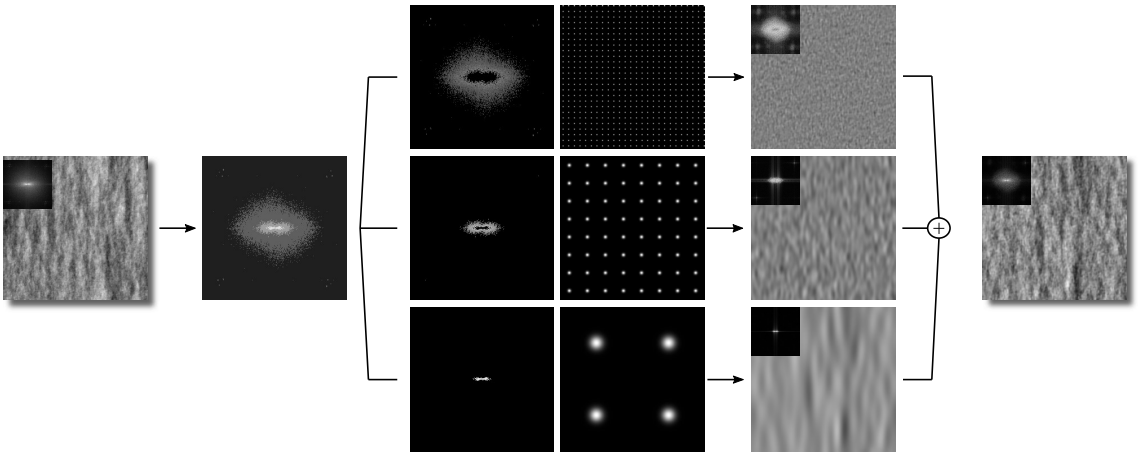


FIGURE 2.26 – À partir du spectre de puissance d'une texture d'exemple (à gauche), Gilet *et al.* le décomposent en plusieurs strates puis évaluée sur plusieurs échelles pour enfin être recombinée pour obtenir une texture "gaussienne" (à droite).

Une nouvelle approche est aussi proposée pour étendre la synthèse par l'exemple à des motifs plus complexes que des textures "gaussiennes". En conservant les phases $\phi(f)$ et les amplitudes $A(f)$ de l'exemple d'entrée selon un certain seuil d'énergie (cf. Figure 2.29), un motif périodique régulier est reconstruit (cf. Figure 2.27).

$$\text{StructuredTex}(\mathbf{x}) = \text{LRPN}_R(\mathbf{x}) + \sum_S \text{LRPN}_S(\mathbf{x}) \tag{2.15}$$

$$\text{où } \text{LRPN}_R(\mathbf{x}) = \sum_i w(\frac{||\mathbf{x} - \mathbf{x}_i||}{\Delta_R}) \sum_f A(f) cos(2\pi f \cdot \mathbf{x} + \phi(f)) \tag{2.16}$$

Ce type d'approche réussit à reproduire finement les caractéristiques de l'exemple. Cependant, en conservant les phases, le signal devient périodique et des répétitions apparaissent pour $\text{LRPN}_R(\mathbf{x})$. Ces répétitions sont brisées en ajoutant de la turbulence et en plaçant les noyaux aléatoirement comme le montre la figure 2.28.

La méthode converge plus vite qu'un bruit de Gabor par l'exemple et peut synthétiser des textures qui ne pouvaient pas l'être par des bruits à phase aléatoire en échange d'une étape de perturbation et de *jittering* de la structure le tout pour des résultats sous

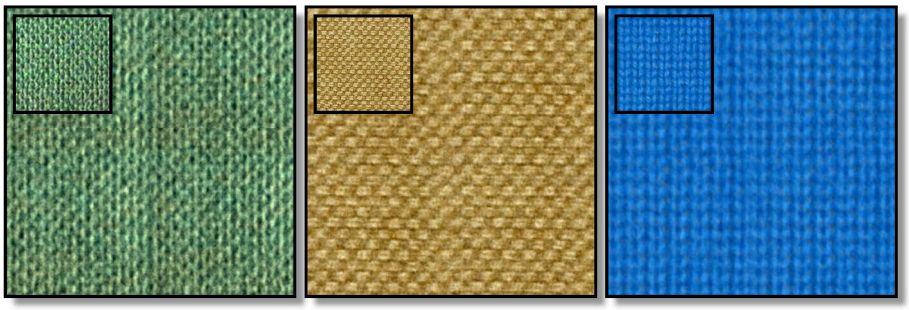


FIGURE 2.27 – En proposant de fixer les phases avec le plus d'énergie, Gilet *et al.* [GSV+14] reproduisent un motif périodique qu'ils viennent ensuite perturber avec leur bruit à phase aléatoire.

la milliseconde pour des textures en 128x128 pour une somme de 30 cosinus en moyenne et 15% de phases convervées. Cependant la capture de la structure au niveau du spectre n'est pas parfaite car elle suppose que la structure est représentée par les couples fréquence/phase de grandes amplitudes. Or, ce n'est pas toujours vrai, une texture représentant un ensemble de graviers par exemple voit sa composante structurelle distribuée sur tout son spectre. De plus, le filtrage proposé est limité à du *frequency clamping* combiné à du *mipmapping* des valeurs de fréquences et de phases à conserver.

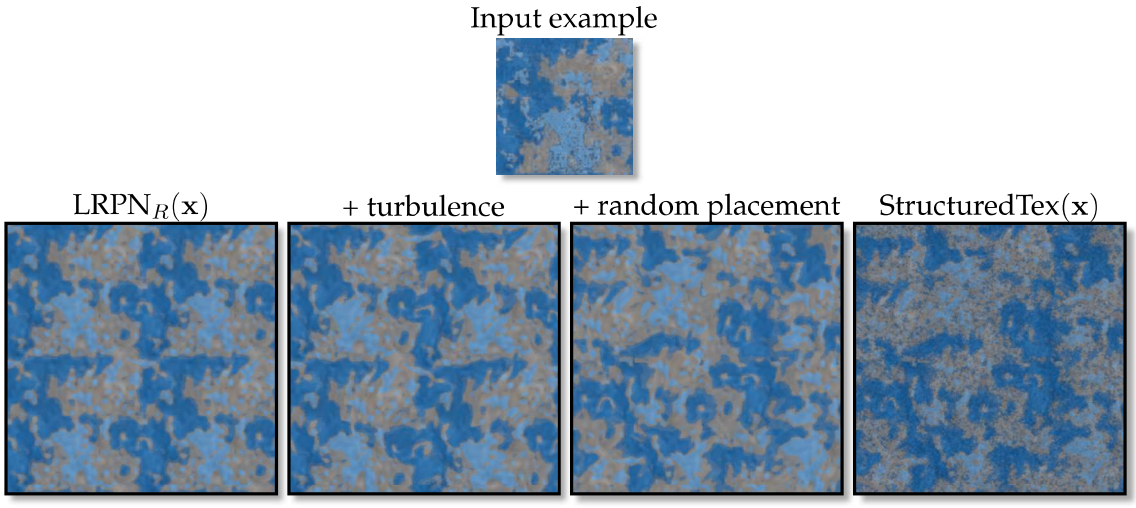


FIGURE 2.28 – En conservant les phases, [GSV+14] la partie structurelle du signal reconstruit redevient périodique. Les auteurs préconisent d'utiliser une perturbation et du placement aléatoire pour venir briser cette répétitions.

Comme évoqué précédemment, l'évaluation des noyaux est rendue efficace si elle est également distribuée sur l'espace d'évaluation (grille régulière, même nombre d'opérations par pixel, etc.). Cependant, l'évaluation peut être encore améliorée en tirant partie des outils que proposent les cartes graphiques comme le *mipmapping hardware*. C'est dans ce contexte qu'est introduit le *Texton Noise* [GLM17].

En se reconcentrant sur les textures dites "gaussiennes", Galerne *et al.* génèrent un noyau discret qu'ils nomment "texton" en analysant le spectre de puissance de l'entrée. Cette texture discrète est ensuite utilisée comme noyau de convolution pour synthétiser une texture par l'exemple à la volée. Le noyau étant discret, il peut tirer avantage

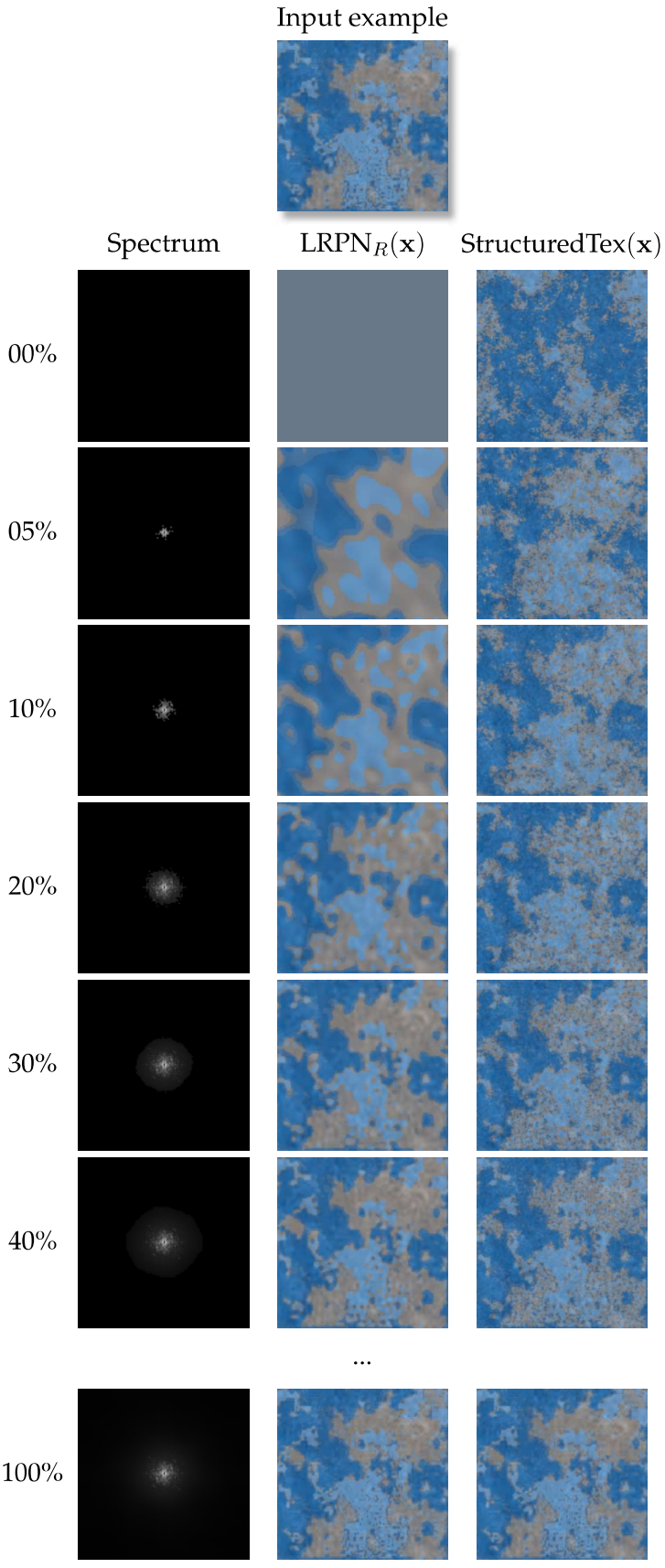


FIGURE 2.29 – La reconstruction du motif est réalisée en conservant les phases ayant le plus d'énergie en s'appuyant sur l'hypothèse que la structure est conservée localement dans la phase.

des fonctionnalités *hardware*, comme le *mipmapping* anisotropique directement pendant la synthèse. Comme pour [LLDD09,GLLD12], l'évaluation consiste à évaluer des noyaux centrés sur une distribution de points. Cette fois, le coût d'évaluation ne dépend plus de la complexité du spectre de l'entrée et ne dépend plus que du nombre de *textons* évalués par pixels. Cette méthode est optimisée pour la synthèse sur carte graphique, reproductible et produit des textures "gaussiennes" en quelques millisecondes pour des textures en HD (1920x1080) en utilisant en moyenne 30 *textons* par pixels.

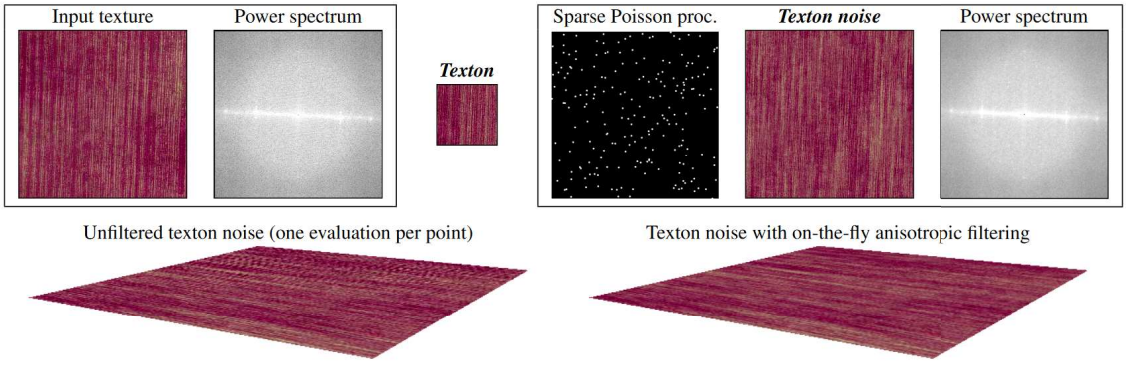


FIGURE 2.30 – En utilisant un noyau discret précalculé à partir de l'exemple d'entrée de manière à reproduire fidèlement son spectre de puissance, [GLM17] peut utiliser avantageusement le filtrage hardware pour générer une texture gaussienne filtrée pour une trentaine d'accès textures sur GPU.

Guingo *et al.* [GSDC17] proposent une méthode d'analyse et de synthèse des textures composites, mais, à la différence de l'approche du **LRPN** , dissocient l'extraction de la structure du spectre. En effet, comme précisé par [GSV+14], la séparation de la structure par le spectre n'est pas évidente et réintroduit des répétitions dans la texture générée. L'idée proposée par Guingo *et al.* est d'analyser l'exemple pour en déduire un masque binaire partitionnant la texture composite. Ce masque est ensuite utilisé pour effectuer une synthèse de texture par pavage. La deuxième phase d'analyse vient calculer deux bruits à phase aléatoire qui viendront habiller le nouveau masque. Le principe de la méthode est illustré en Figure 2.31.

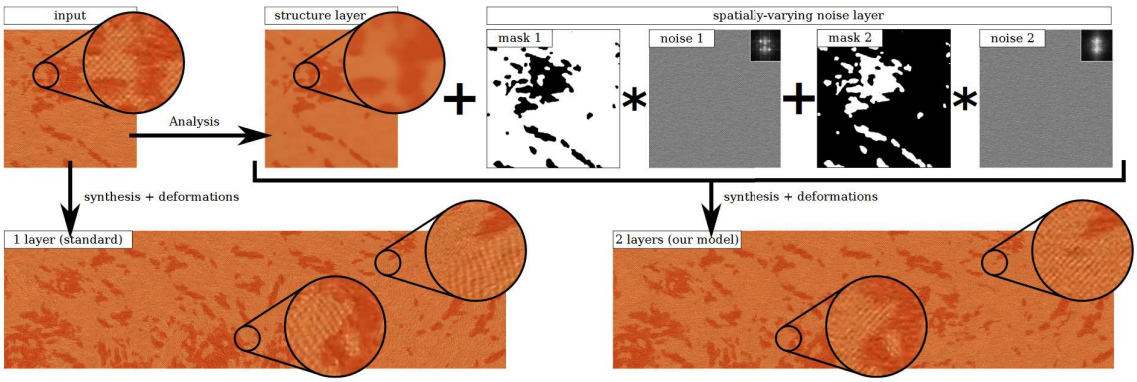


FIGURE 2.31 – Au lieu de chercher la structure dans la phase comme [GSV+14], [GSDC17] propose de reproduire le motif structurel spatialement avec un masque puis de venir y ajouter deux bruits différents pour représenter des textures composites.

2.4.3 Des défauts de contraste

Dans un rapport technique publié par Fabrice Neyret et Eric Heitz en 2016 [NH16], un défaut de qualité présent dans toutes les méthodes basées sur les bruits à phase aléatoire est mis en évidence et étudié : l'*oscillations de contraste*. En effet, si l'on regarde plus attentivement les textures générées par manipulation explicite du spectre de puissance (méthodes basées sur Fourier, bruits par convolution éparses comme le bruit de Gabor et variantes par l'exemple), nous notons la présence d'un signal basse fréquence perturbant le contraste de la texture générée (cf. Figure 2.32). Pourtant le spectre de puissance du signal ne révèle aucune basse fréquence (cf. Figure 2.32 à droite).

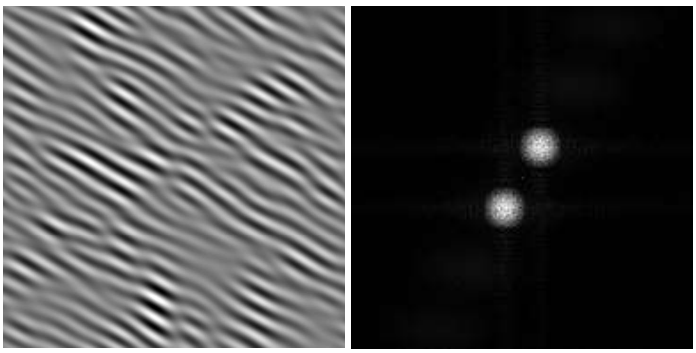


FIGURE 2.32 – Le spectre (à gauche) d'un bruit de Gabor anisotrope (à gauche) ne présente que deux lobes symétriques hautes fréquences. Cependant nous percevons une perte de contraste basse fréquence dans la texture.

Un outil pour étudier et contrôler ce phénomène est proposé : *Le spectre de variance*. Le spectre de variance $\mathcal{F}((s(\mathbf{x}) - \bar{s}(\mathbf{x}))^2)$ permet d'étudier ce signal basse fréquence. Si l'on reprend l'exemple du bruit de Gabor anisotrope, le spectre de variance révèle bien un lobe basse fréquence (cf. Figure 2.33 à droite).

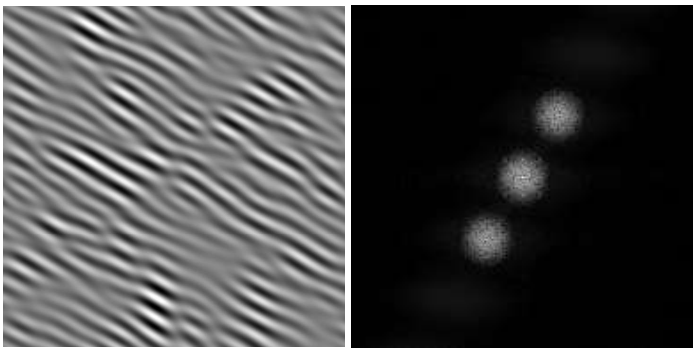


FIGURE 2.33 – Le spectre de variance (à droite) d'un bruit de Gabor anisotrope (à gauche) révèle effectivement un lobe basse fréquence.

Plusieurs manipulations de ce spectre de variance sont proposées pour contrôler le contraste présent dans la texture. En filtrant le spectre de variance, le lobe basse fréquence est amoindri. Une normalisation itérative peut être effectuée pour obtenir de meilleurs résultats.

Une étude directe du processus d'évaluation du bruit de Gabor peut permettre de limiter ces problèmes. Tavernier *et al.* [TNVT19] ont étudié les différentes composantes

(distribution de points, utilisation d'un sinus plutôt que d'un cosinus pour l'harmonique) du bruit de Gabor pour proposer une synthèse de meilleure qualité et rapide.

Tricard et Efremov *et al.* [TEZ+19] ont proposé une méthode pour réécrire le bruit de Gabor sous la forme d'un sinus. En décrivant un champ de phase instantanée en effectuant la somme de phasors, une nouvelle primitive normalisée, par définition, est présentée : le bruit par Phasor (ou *Phasor Noise*) illustré en Figure 2.34.

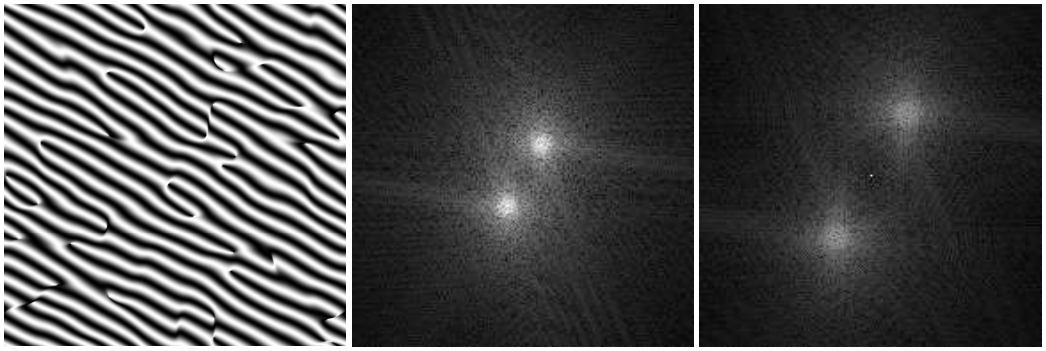


FIGURE 2.34 – Le spectre d'amplitude (au milieur) et le spectre de variance (à droite) d'un phasor bi-lobe (à gauche) de même fréquence que précédemment, ne révèle pas de variation de contraste.

La contrôle de l'apparence finale de cette méthode passe par la fonction de profil. Pour générer un bruit de gabor normalisé, c'est un sinus qui est utilisé. En remplaçant ce sinus par une fonction périodique comme une *square wave*, il est possible de changer et de contrôler totalement les valeurs en sortie de méthodes (cf. Figure 2.35). Cependant, la texture générée peut présenter des singularités dans son motif et la question du filtrage de texture n'est pas considérée dans ce travail.

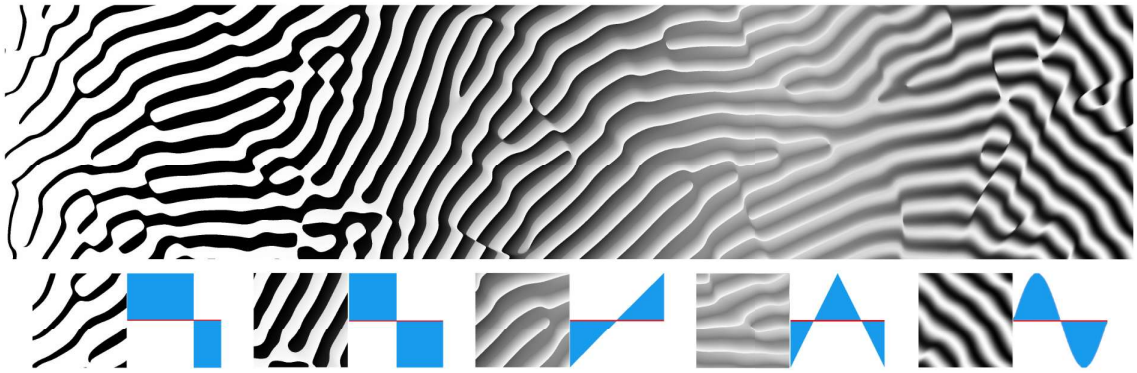


FIGURE 2.35 – L'apparence de la fonction procédurale est contrôlée par la fonction de profil périodique. (Figure [TEZ+19])

Reposer la problématique du mélange des noyaux peut aussi améliorer la qualité des textures générées par l'exemple. Les méthodes basées sur une distribution aléatoire de points pour évaluer les noyaux (comme le pavage stochastique et le *Sparse Convolution Noise*) peuvent générer des zones plus densément fournies que d'autres dans l'espace de texture.

Une distribution régulière et un noyau ayant un support dont la largeur dépend de l'écart des points règle ce problème. Le problème suivant vient alors du mélange

des noyaux. Un mélange linéaire classique (une moyenne) va provoquer une perte de contraste dans la texture finale.

Un mélange préservant la variance va régler le problème pour le mélange des textures gaussiennes. En effet, il se trouve que mélanger des textures gaussiennes en préservant la variance va aussi préserver l'histogramme des valeurs. Les textures ne présentant pas un histogramme Gaussien ne vont pas se mélanger correctement en ne conservant que la variance.

C'est l'observation qui a été réalisée par le travail d'Heitz et Neyret [HN18] de 2018. Une méthode de tiling stochastique par l'exemple est proposée en utilisant une grille régulière de simplexe et des noyaux discrets tirés aléatoirement d'un exemple en entrée. Si l'exemple est une texture gaussienne (histogramme des valeurs Gaussien) les noyaux sont mélangés directement en utilisant les poids barycentriques du simplexe à évaluer pour le pixel. Le mélange utilisé est un mélange conservant la variance.

Si la texture ne possède pas un histogramme Gaussien, il est transformé en histogramme Gaussien dans une étape de précalcul. Dans l'article de [HN18], cette transformation est réalisée par du transport optimal, mais cette étape peut être aussi effectuée par transformation d'histogramme par tri (dans le chapitre de GPU Zen 2 [DH19]). Dans tout les cas, une transformation inverse est précalculée et stockée dans une table de correspondance (ou **LUT** pour *lookup table*).

Pendant l'évaluation, les textures transformées sont mélangées en conservant la variance en utilisant la grille de simplexe. Le mélange ne requiert que trois accès texture, un par sommet du simplexe. Ensuite un accès texture est utilisé pour effectuer la transformation inverse de l'histogramme. Comme précisé précédemment, cette dernière opération et la transformation d'histogramme peuvent être omises si la texture en entrée présente déja un histogramme Gaussien.

Les résultats obtenus, montrés en Figure 2.36, sont des textures stochastiques générées de manière très efficace sur la carte graphique. Les performances de l'étape d'évaluation sont meilleures d'un ordre de grandeur de x20 en comparaison avec le *Texton Noise*, qui tirait avantage des accès textures GPU, et jusqu'à x70 en comparaison avec le *Local Random Phase Noise* (pour de la synthèse par l'exemple conservant la phase), qui utilisait aussi une grille régulière pour évaluer des noyaux de manière plus efficace qu'un [LLDD09]. Avec cet ordre de grandeur de gain de performance, cette méthode peut être utilisée dans des applications temps-réels critiques. De plus, comme pour le *Texton Noise*, la méthode dispose du filtrage par mipmapping hardware.

Cette méthode est très performante pour synthétiser des textures stochastiques et un peu plus structurées (avec des résultats similaires au *Local Random Phase Noise*) mais reste confrontée aux mêmes limitations que les méthodes de synthèses par pavages et tiling stochastiques. Si l'exemple présente une structure trop large (le ciment d'un mur de brique, le spaghetto d'un plat de spaghetti, etc.) la méthode ne réussira pas à générer un résultat correct (cf. Figure 2.36 dernière ligne). Il en va de même si la texture en entrée n'est pas facilement pavable (*tileable*).

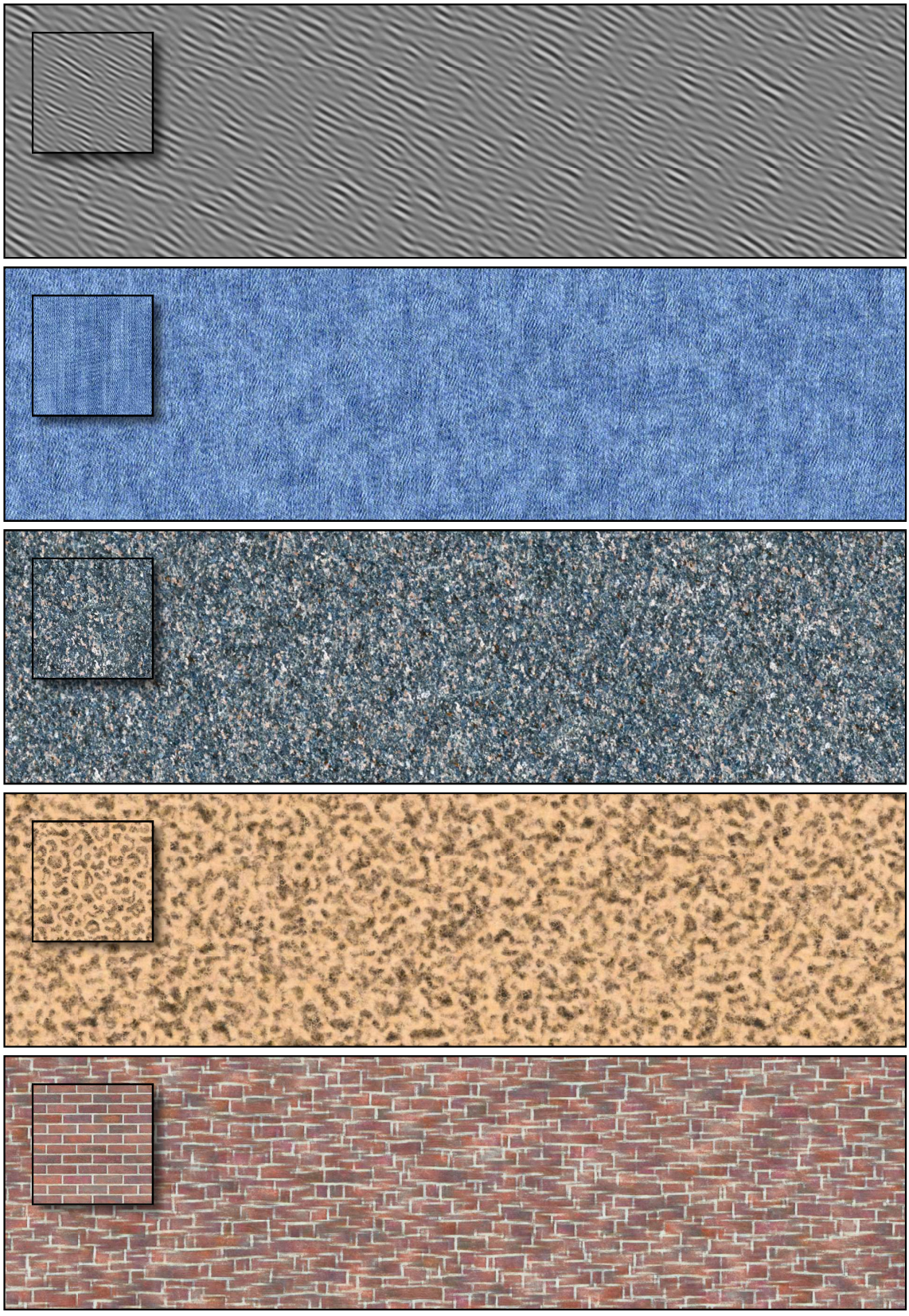


FIGURE 2.36 – Exemple de textures synthétisées par [HN18].

2.5 Synthèse et placement du travail de la thèse

2.5.1 Le bruit et le filtrage

Si nous prenons du recul sur les capacités de filtrage des différentes familles de bruit, le *Sparse Convolution Noise* présente davantage de possibilités que les autres familles. En effet, les méthodes par interpolation sont généralement filtrées par *frequency clamping* et ne considèrent que l'isotropie de l'empreinte de pixel. Les bruits explicites améliorent la qualité du filtrage par *frequency clamping* puisque leur limitation en fréquence est mieux définie [CD05]. Ces méthodes peuvent aussi gérer l'anisotropie de l'empreinte de pixel [GZD08] mais avec la contrepartie du coût mémoire pour couvrir les orientations de l'empreinte de pixel. La qualité du filtrage des méthodes par convolution éparse dépend fortement du noyau choisi. Si ce noyau possède une formulation analytique particulière, le bruit peut être filtré analytiquement dans le domaine spatial, en calculant la convolution avec l'empreinte de pixel, ou dans le domaine spectral, en calculant le produit des transformées de Fourier de l'empreinte projetée et du noyau [LLDD09]. L'empreinte de pixel doit alors être choisie avec attention pour garantir l'anisotropie (gaussienne [Hec89], parallélogramme [ZK16]). Enfin, dans le cas où le noyau choisi est discret ([GLM17], [HN18]) le filtrage anisotrope est délégué au *mipmapping hardware*.

2.5.2 Le bruit et le contrôle utilisateur

Si nous nous intéressons maintenant au contrôle que l'utilisateur possède sur l'apparence finale de la texture, il est majoritairement opéré dans le domaine spectral. Les fonctions de bruit par interpolation vont laisser à l'utilisateur la possibilité de choisir la taille de la bande de fréquence et la manière de les combiner par *fBM*. Pour les méthodes explicites, l'utilisateur doit réeffectuer le précalcul des textures de bruits pour ensuite les combiner. Pour les méthodes par convolution éparse, c'est avec des paramètres spectraux (fréquence, amplitude, phase) que l'utilisateur doit définir l'apparence [LLDD09, LD11] même si le bruit de Gabor laisse le contrôle spatialement sur la fonction de fenêtrage. L'artiste doit, dans ces cas là, pour représenter une apparence, concevoir manuellement un spectre de puissance pour définir un motif.

Les approches basées sur l'exemple ammenuisent cette problématique en déduisant les paramètres directement d'un exemple en entrée, mais l'apparence finale dépend directement de la qualité de l'exemple choisi. La plupart des méthodes présentées dans cet état de l'art restent limitées à des catégories de textures particulières [GD96,GDG12,GLLD12, GLM17]. Cependant, la diversité des textures générées peut être étendue par perturbation [GD96, GDG12] ou par une analyse différente de l'exemple en entrée (conservation de la phase [GSV+14], conservation de l'histogramme [HN18, DH19]). Enfin, si l'utilisateur veut un résultat différent, il doit fournir un exemple différent.

Certaines méthodes se sont intéressées à des manières d'influencer les fonctions de bruits par des paramètres défini spatialement ([CSM14]), ou des manières de les évaluer et les combiner pour obtenir des motifs particuliers ([BLV+10]). Une définition purement spatiale du noyau de convolution comme le *Spot Noise* [vW91] ou le bruit par *noyaux ponctuels localement contrôlés* [PGDG16], permet d'utiliser des paramètres plus intuitifs

dans le domaine spatiale.

Enfin, dans les travaux récemment publiés [NH16, TNVT19, TEZ+19], l'étude du processus d'évaluation des méthodes de bruits peut permettre de dégager des nouvelles manières d'influencer l'apparence finale du motif généré.

2.5.3 Placement de la thèse

Comme expliqué en introduction de cette thèse, notre objectif idéal est de définir une arborescence de fonctions de bruits pour représenter des motifs composites. Nous avons tout d'abord étudié les primitives qu'utiliserait une telle arborescence en prenant en compte à la fois le contrôle utilisateur et les capacités de filtrage de ces primitives. Nous nous sommes donc intéressés, dans un premier temps, à la définition de primitives contrastées filtrables.

Nous avons, au travers d'un travail publié à *Computer and Graphics* en Octobre 2019, travaillé sur une amélioration de la méthode proposée par Pavie *et al.* [PGDG16]. En effet, le bruit par *noyaux ponctuels localement contrôlés* est défini de manière purement spatiale. Le noyau utilisé est une somme de gaussiennes elliptiques définies par des paramètres géométriques (translation, rotation, mise à l'échelle). L'artiste peut alors définir l'apparence de la texture en construisant le noyau par somme de gaussiennes et en contraignant la distribution des noyaux.

En reformulant la formulation analytique du noyau, nous avons proposé une définition filtrée de cette méthode. Nous avons aussi étudié les dérivées partielles de cette méthodes afin d'évaluer à la volée une perturbation géométrique de la surface tout en essayant de limiter les défauts d'aliassage. En s'inspirant de [CSM14], nous avons aussi proposé une méthode de synthèse de motifs variant spatialement en utilisant des cartes de contrôles. Enfin, en utilisant les motifs contrastés ainsi générés nous avons synthétisé des textures composites.

Chapitre 3

Synthèse de détails surfaciques par *Spot Noise* localement contrôlé

Sommaire

Chapitre 3

Synthèse de détails surfaciques par *Spot Noise* localement contrôlé

Ce chapite présente les résultats d'un travail ayant fait l'objet d'une publication au journal *Computer and Graphics* en Octobre 2019.

3.1 La synthèse de détails surfaciques par *Spot Noise*

3.1.1 Introduction

Introduit par van Wijk en 1991, le *Spot Noise* [vW91] est une variante des bruits par convolution éparse et consiste à évaluer une texture stochastique en effectuant la somme d'un noyau à des positions aléatoires dans l'espace de texture. Ce modèle fut initialement proposé pour aider à la visualisation de champs de vecteur et permet de générer une large gamme de textures en changeant le noyau utilisé, comme le montre la Figure 3.1.

Le noyau en question reste arbitraire et peut tout aussi bien être défini spatialement ou synthétisé par série de Fourier. Contrairement aux méthodes de bruits par convolution éparse (comme le bruit de Gabor [LLDD09]) où le contrôle est défini via des paramètres spectraux (fréquence, angle, phase), un noyau défini spatialement peut être contrôlé par des paramètres plus intuitifs pour l'artiste. C'est dans cette optique que Pavie *et al.* présentent en 2016 le bruit par noyaux ponctuels localement contrôlés [PGDG16] (ou **LCSN** pour *locally controlled spot noise*). Ce *Spot Noise* possède la particularité de proposer une définition du noyau comme étant une somme de fonctions gaussiennes elliptiques. Cette définition permet un contrôle intuitif de l'apparence finale de la texture comme chacune des gaussiennes du noyau peut être éditée en utilisant des paramètres purement géométriques (à savoir translation, rotation et mise à l'échelle). Quant au contrôle sur la composante structurelle du motif et l'apparence générale de la texture, il s'effectue en contraignant la distribution des jets des noyaux (comme montré en Figure 3.1).

Nous rappellons ici la formulation mathématique du *Spot Noise* localement contrôlé comme

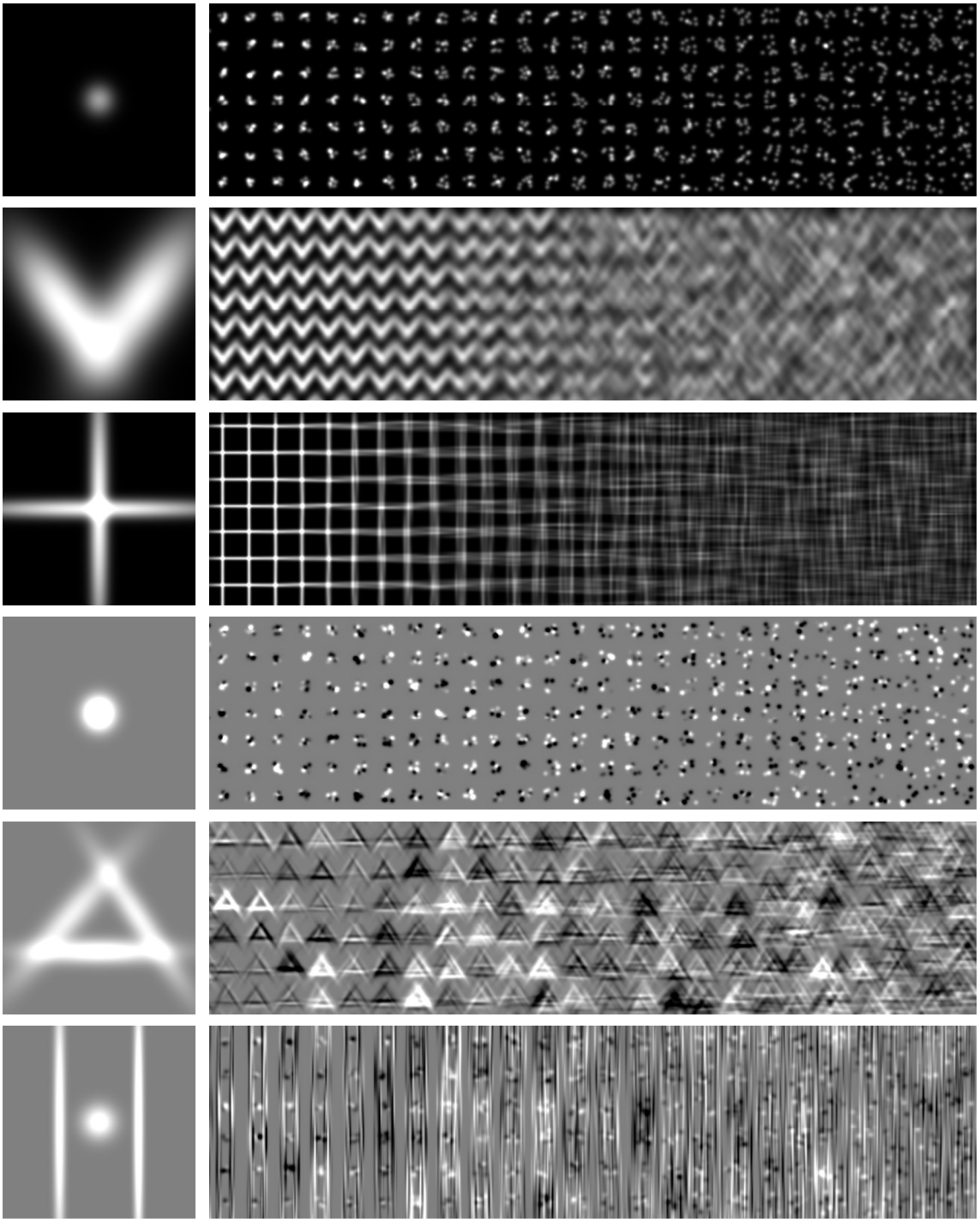


FIGURE 3.1 – Le modèle de *Spot Noise* introduit par Pavie *et al.* [PGDG16] consiste à effectuer une somme de noyaux définis comme étant eux même une somme de fonctions gaussiennes (à gauche).
En contrôlant la distribution des impulsions, ce *Spot Noise* peut générer une grande variété de motifs allant du plus stochastique au plus régulier (à droite).
De plus il permet d'obtenir des motifs très contrastés (3 lignes du haut).

étant la somme pondérée w_i de M noyaux k évalués aux positions tirées aléatoirement $\mathbf{p_i}$

$$\text{lcsn}(\mathbf{p}) = \sum_{i=1}^{M} w_i k(\mathbf{p} - \mathbf{p}_i) \tag{3.1}$$

Où $\mathbf{p}$ est le représentant en coordonnées homogènes du point $\mathbf{x}$ de l'espace de texture (i.e. $\mathbf{p} = (\mathbf{x}, 1)^T$). Chaque fonction de noyau k est elle-même définie comme la somme de N gaussiennes elliptiques de dimension D avec une orientation, une échelle et une translation arbitraire :

$$k(\mathbf{p}) = \sum_{j=1}^{N} \lambda_j e^{-\frac{1}{2}\mathbf{p}^T \mathbf{V}_j^{-1} \mathbf{p}} \tag{3.2}$$

où λ correspond à l'amplitude de la gaussienne et $\mathbf{V_j}$ est une matrice de rang $(\mathbf{D}+1) \times (\mathbf{D}+1)$ que l'on peut exprimer comme ${\mathbf{V_j}}^{-1} = (\mathbf{M_j R_j S_j})^{-T} (\mathbf{M_j R_j S_j})^{-1}$. Ici $\mathbf{M_j}$,$\mathbf{R_j}$ et $\mathbf{S_j}$ sont respectivement les matrices de translation, de rotation et de mise à l'échelle utilisant les coordonnées homogènes. Pour obtenir des motifs structurés, le *Spot Noise* localement contrôlé repose sur une distribution uniforme de points contrainte par une fonction de Kronecker $\delta\left(\xi(\mathbf{p_j}) < d(\mathbf{p_j})\right)$ où d est un champ scalaire représenté par une probabilité. Ce paramètre permet à l'utilisateur de contrôler la densité de noyau dans une région donnée de la texture et peut être défini en utilisant une fonction analytique ou discrète arbitraire.

En pratique, comme pour les méthodes de bruits par convolution éparse [LLDD09, GSV+14, GLM17], cette fonction de texture procédurale est évaluée sur une grille régulière pour trouver les noyaux contribuant à l'intensité du pixel évalué. Pour évaluer la couleur d'un pixel, nous sommons la contribution de tous les noyaux de la cellule courante et des cellules voisines afin de garantir la continuité de la texture. La distribution de points est généralement un échantillonnage stratifié dans la grille en utilisant une graine aléatoire dépendante de la case considérée.

3.1.2 Limitations

Comme pour les méthodes de bruits par convolution éparse, le coût d'une telle fonction de texture procédurale est dépendant du nombre de noyaux évalués par pixel. Dans le cas du **LCSN** , la complexité du noyau, c'est à dire du nombre de gaussiennes qui le composent, rajoute un surcoût constant à l'étape d'évaluation et l'utilisation des coordonnées homogènes n'est pas discutée. De plus, le filtrage de ce *Spot Noise* n'est pas traité dans ce travail laissant des défauts d'aliassage dans la texture finale après évaluation.

3.1.3 Objectifs

L'objectif de ce travail était d'améliorer ce modèle de *Spot Noise* . De plus, comme présenté dans le chapitre précédent, les aspects visuels les plus attrayants présentent des variations subtiles dans leur manière d'interagir avec la lumière et ne sont pas limités à l'albedo. Nous avons donc voulu générer un autre canal de texture permettant d'ajouter

des détails à la volée à l'échelle mésoscopique sur la surface de l'objet texturé : une carte de normales procédurale.

Le reste de ce chapitre présente donc nos contributions principales :

- Une formulation simplifiée du noyau préservant la paramétrisation intuitive et géometrique du **LCSN** .
- Une méthode de filtrage anisotropique analytique afin d'obtenir des résultats de haute qualité pendant le rendu.
- Une méthode de génération de cartes procédurales de normales de manière analytique tout en limitant les problèmes d'aliassage.

3.2 Nouvelle formulation du noyau

Comme montré dans la section précédente, la méthode proposée par Pavie *et al.* [PGDG16] est basée sur une double somme : la somme des noyaux étant eux même définis comme une somme de Gaussiennes. Cependant, le besoin des coordonnées homogènes dans la paramétrisation originale ne semble pas obligatoire et justifié dans leur modèle. En effet, la *forme* elliptique de la Gaussienne est entièrement indépendante de l'opération de translation, mais dépend directement de la mise à l'échelle et de son orientation. Une fonction Gaussienne 2D classique (comme une loi normale non normalisée) avec une amplitude λ, un vecteur de moyenne μ et une matrice de covariance $\mathbf{\Sigma}$ peut être utilisée à la place sans perte de généralité :

$$g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma}) = \lambda e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)} \tag{3.3}$$

Nous obtenons alors:

$$\text{lcsn}(\mathbf{x}) = \sum_{i=1}^{M} w_i \sum_{j=1}^{N} g(\mathbf{x} - \mathbf{x_i}; \lambda_j, \mu_j, \mathbf{\Sigma_j}) \tag{3.4}$$

L'isocontour du noyau est toujours déterminé par $(\mathbf{x} - \mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \mu)$ mais ne nécessite plus la représentation en coordonnées homogènes, conservant donc la dimension $\mathbf{D}$. La matrice de covariance $\mathbf{\Sigma}$ peut être exprimée comme étant la combinaison en deux matrices $\mathbf{R}$, matrice de rotation, et $\mathbf{S}$, matrice de mise à l'échelle, avec la relation $\mathbf{\Sigma} = \mathbf{RSSR}^T$. La translation du noyau est ainsi changée d'une multiplication matricielle $\mathbf{M}$ à une soustraction du vecteur μ.

Ce léger changement dans la formulation conserve les paramètres géométriques que l'utilisateur peut contrôler, à savoir : un angle de rotation, un vecteur de mise à l'échelle et un vecteur de translation. Cette définition peut aussi être utilisée dans des dimensions supérieures (comme montré sur la Figure 3.2) et entraîne un léger gain de performances (ce point est discuté en section 3.6).

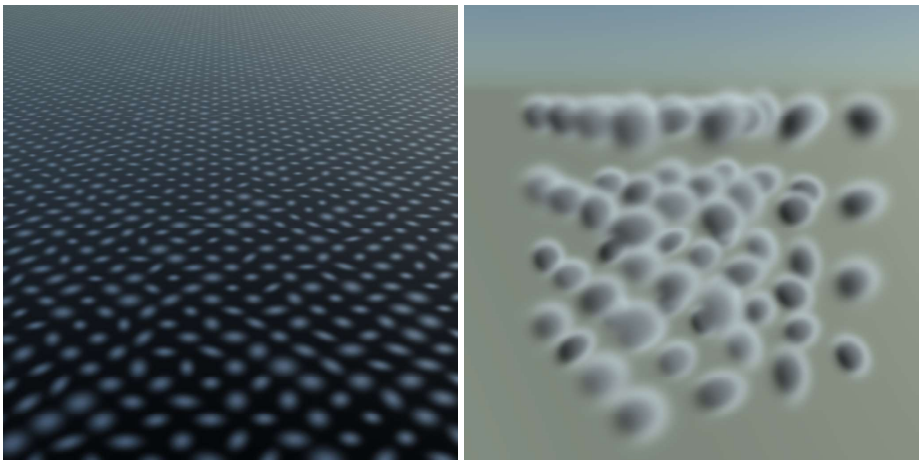


FIGURE 3.2 – En changeant légèrement la formulation du noyau, nous pouvons toujours aussi bien représenter des Gaussiennes en 2D sur une surface (à gauche) qu'en 3D dans un volume (à droite).

3.3 Filtrage Analytique

Pour préfiltrer notre noyau de *Spot Noise* nous représentons notre empreinte de pixel par une gaussienne elliptique similaire à Heckbert [Hec89] (voir Figure 3.3). En effet, une empreinte de pixel gaussienne offre souvent la possibilité d'obtenir des formes closes lors des calculs de filtrage si la fonction de noyau possède une formulation analytique particulière comme pour le Gabor Noise [LLDD09]. Cette gaussienne est exprimée dans l'espace de texture comme une gaussienne normalisée centrée sur le pixel projeté avec une matrice de covariance pouvant être décomposée à l'aide de la jacobienne $\mathbf{J}$ des coordonnées de textures et σ comme largeur de l'empreinte de pixel en espace image :

$$k_P(\mathbf{x}) = \frac{1}{2\pi\sqrt{|\mathbf{J}\mathbf{J}^T|}} e^{-\frac{1}{2}\left[\mathbf{x}^T(\mathbf{J}\mathbf{J}^T)^{-1}\mathbf{x}\right]} \tag{3.5}$$

avec

$$\mathbf{J} = \sigma \begin{pmatrix} \frac{du}{dx} & \frac{du}{dy} \\ \frac{dv}{dx} & \frac{dv}{dy} \end{pmatrix}$$

En pratique, la matrice $\mathbf{J}$ est construite en utilisant les dérivées partielles en espace image des coordonnées de textures (paramétrisant la surface de l'objet) si l'image est rasterisée. Dans le cas d'un rendu par lancer de rayon, ces dérivées partielles peuvent être évaluées par lancer de rayon différentiel.

Pour préfiltrer la fonction procédurale, il est nécessaire d'évaluer la convolution de l'empreinte de pixel projetée avec la fonction procédurale. Il faut évaluer l'intégrale du produit du filtre avec la fonction procédurale. Cette dernière étant, dans notre cas, une double somme de gaussiennes pondérées, nous obtenons par distribution du produit et séparation de l'intégrale que le bruit préfiltré correspond à la somme des noyaux préfiltrés. Et, par extension, à la somme des gaussiennes préfiltrées. Les gaussiennes admettant des formes closes sous convolution (cf. 3.3.1), nous calculons analytiquement la somme

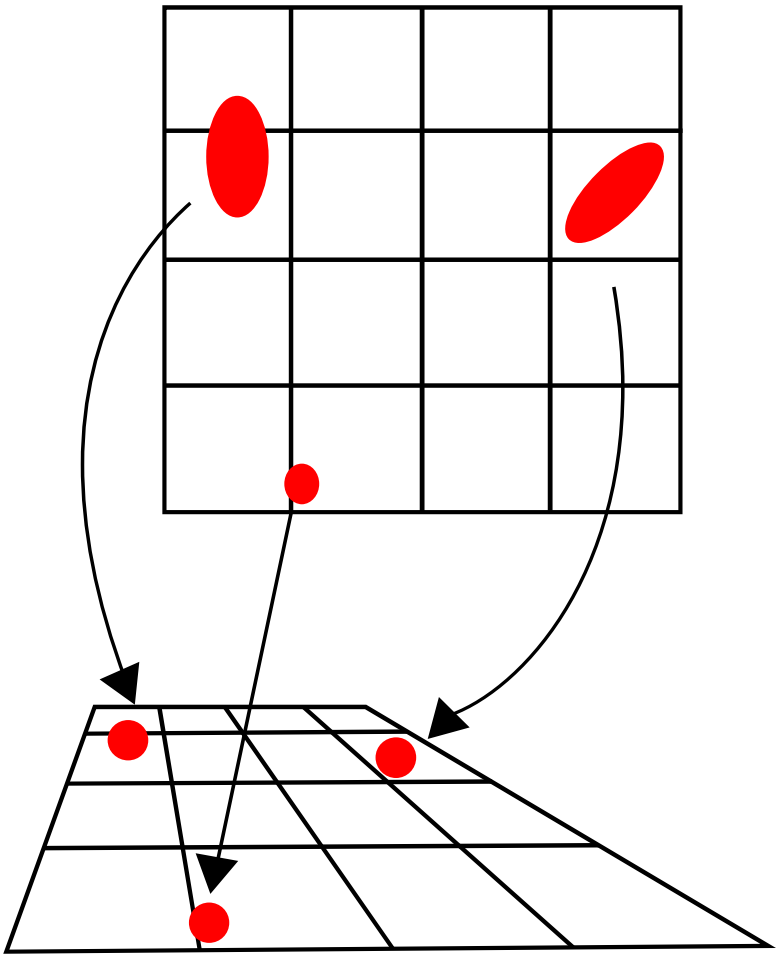


FIGURE 3.3 – Illustration de l'empreinte de pixel gaussienne. Une gaussienne isotrope est utilisée pour définir un pixel en espace image (en bas). Après projection, l'empreinte du pixel devient alors une gaussienne anisotrope (en haut) (Schéma issu de [Hec89]).

de gaussiennes préfiltrées pour obtenir notre *Spot Noise* filtré (voir Figure 3.4) :

$$g(\mathbf{x};\lambda_1,\mu_1,\mathbf{\Sigma_1}) \circledast g(\mathbf{x};\lambda_2,\mu_2,\mathbf{\Sigma_2}) = g(\mathbf{x};\lambda_3,\mu_3,\mathbf{\Sigma_3}) \tag{3.6}$$

avec

$$\begin{aligned}
\mathbf{\Sigma_3} &= \mathbf{\Sigma_1} + \mathbf{\Sigma_2} \\
\mu_3 &= \mu_1 + \mu_2 \\
\lambda_3 &= (2\pi)^{D/2} |(\mathbf{\Sigma_1}^{-1} + \mathbf{\Sigma_2}^{-1})^{-1}|^{1/2} \lambda_1 \lambda_2
\end{aligned}$$

Comme montré sur la Figure 3.4, nous obtenons des résultats sans défauts d'aliassage (implémentation *shadertoy* : **/3dcGW7**)

Cependant, dans le cadre pratique, des défauts subsistent avec cette approche. En effet, dans certains cas, l'empreinte de pixel projetée peut dégénérer et prendre des formes extrêmes selon le point de vue, la scène et les paramètres choisis. L'empreinte de pixel

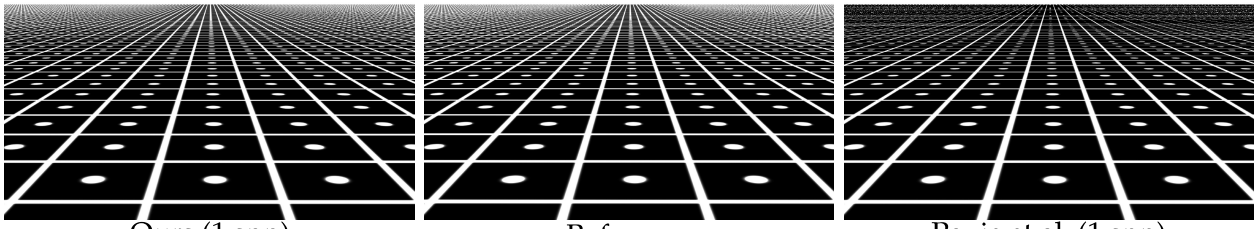


FIGURE 3.4 – Nous comparons notre méthode de filtrage (à gauche) avec la méthode non filtrée de Pavie *et al.* (à droite). Nous avons choisi un motif régulier car ce sont les plus susceptibles de générer de l'aliassage. En préfiltrant analytiquement le *Spot Noise* , nous obtenons un résultat similaire à la référence sur-échantillonée.

projetée recouvre alors plus de cellules de la grille régulière que le nombre de cases considérées pendant l'évaluation. Il en résulte un défaut de sous-estimation de la convolution (voir les régions sombres sur la première ligne de la Figure 3.5). Ce problème apparaît dans des situations extrêmes telles que les angles rasants, quand les coordonnées de textures sont mises à l'échelle de manière disproportionnée, quand la grille sous-jacente possède une trop grande résolution dans l'espace de texure et dans les points de fuite.

Évaluer correctement la convolution nécessite d'augmenter le nombre de cellules à considérer pendant l'évaluation selon l'empreinte de pixel projetée, ce qui présente l'inconvénient d'ajouter du temps de calcul pour les pixels concernés. De plus, quand l'empreinte dégénère en une ellipse plate et allongée, le nombre de cellules nécessaires au calcul correct de la convolution devient impraticable pour une évalution à la volée en temps réel.

Cependant, pour de la texture en couleur, une hypothèse peut être faite dans la mesure où la couleur du pixel va tendre vers la couleur moyenne des couleurs dans l'empreinte. Une première solution pratique est de contraindre l'empreinte pixel sur le voisinage considéré. Cela fonctionne assez bien dans la plupart des cas pratiques. Cependant dans les cas des angles très rasants et quand les coordonnées sont mises à l'échelle de manière trop importante, des problèmes persistent. La meilleure solution d'ordre pratique pour évaluer à la volée la texture sans ces défauts est d'utiliser deux types de représentation et de faire une transition douce entre ces dernières. En précalculant une tuile représentant un voisinage complet considéré lors d'une évaluation, nous sommes capables de paver l'espace avec une représentation *mipmappée* de notre *Spot Noise* . Un algorithme de tuilage aléatoire peut être utilisé afin de briser la répétition de la texture si une seule tuile est calculée. On passe alors d'une représentation par filtrage analytique lorsque cela est possible (c'est à dire en dehors des cas extrêmes cités plus haut) à une représentation par tuile précalculée et *mipmappée* lorsque ces problèmes apparaissent. Ce choix de représentation reste une solution pratique, le problème de filtrer efficacement une texture procédurale entrièrement à la volée restant une problématique de recherche dans le domaine.

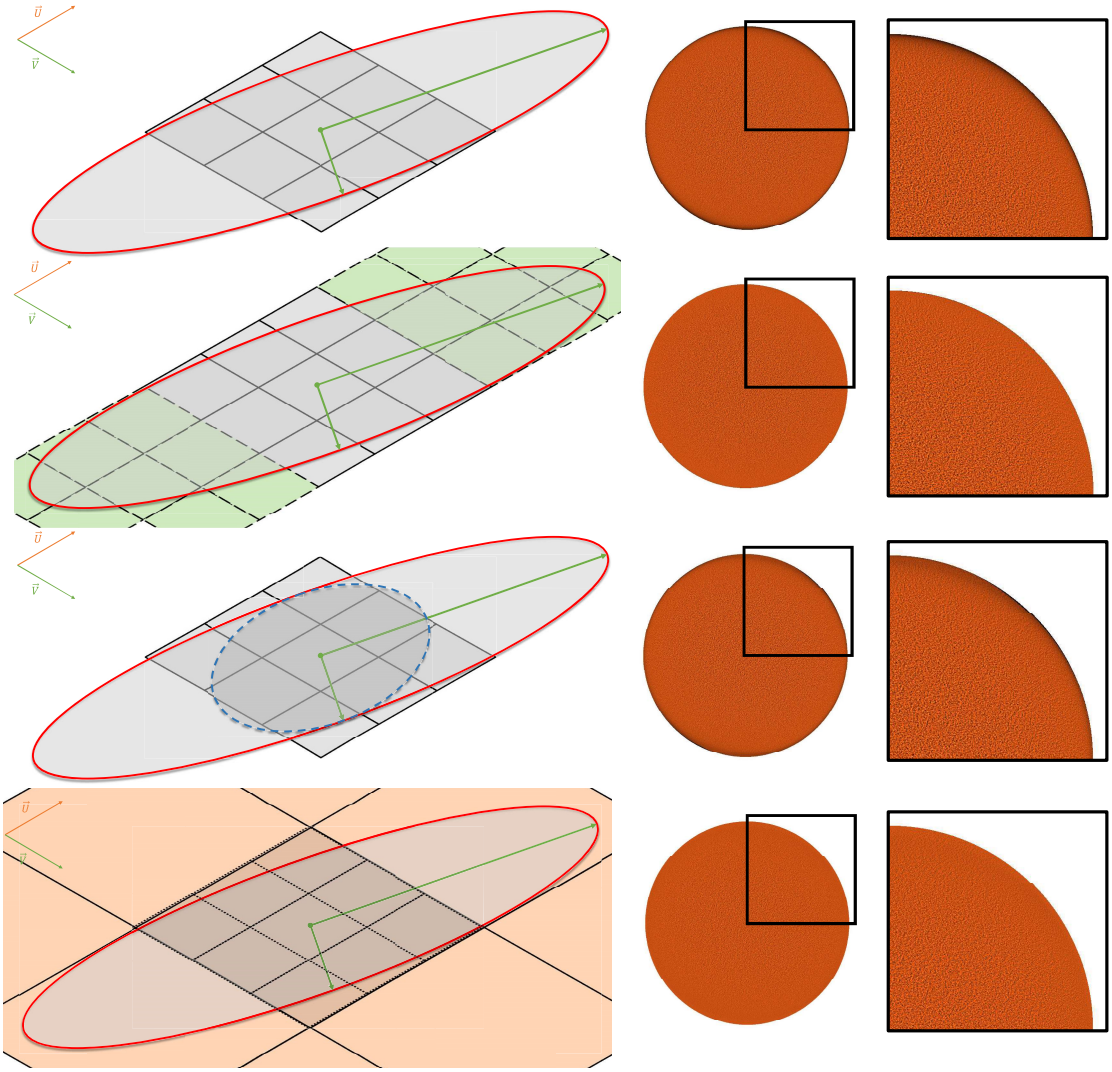


FIGURE 3.5 – Ligne du haut: Dans des cas extrêmes, l'empreinte de pixel (en rouge) peut devenir plus grande que le voisinage considéré pendant l'évaluation (en gris). Si les cellules manquantes ne sont pas prises en compte, la convolution n'est pas calculée correctement (les régions sombre sur les bordures de la sphère).
Deuxième ligne: La solution correcte est de considérer toutes les cellules intersectant l'empreinte de pixel lors du calcul. Cependant, cela devient rapidement impraticable pour du temps réel.
Troisième ligne: En pratique, l'empreinte de pixel est généralement limitée au voisinage considéré (en bleu), ce qui réintroduit de l'aliassage dans certains cas.
Denière ligne : Une meilleure solution pratique est de précalculer une tuile representative du voisinage et faire une transition douce entre le filtrage analytique et cette représentation *mipmappée*.

3.3.1 Preuve du filtrage analytique

Dans cette sous-section, nous développons la preuve que les fonctions gaussiennes admettent bien des formes closes sous convolution. Afin d'obtenir l'équation (3.6), nous devons caluler la convolution I de deux fonctions gaussiennes:

$$I = \int_{\mathbf{R}^D} g(\mathbf{t}; \lambda_1, \mu_1, \mathbf{\Sigma_1}) g(\mathbf{x} - \mathbf{t}; \lambda_2, \mu_2, \mathbf{\Sigma_2}) d\mathbf{t} \tag{3.7}$$

Nous rappellerons ici les formes closes nécessaires pour la suite des calculs, en commençant par l'intégrale d'une fonction gaussienne puis du produit de deux gaussiennes. Nous présenterons ensuite l'intégrale du produit de deux gaussiennes, ainsi que la résolution, par substitution, de l'équation (3.7).

Intégrale d'une gaussienne.
Intégrer une gaussienne multivariée g sur $\mathbf{R}^D$ donne :

$$\int_{\mathbf{R}^D} g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma}) d\mathbf{x} = \lambda \cdot (2\pi)^{D/2} |\mathbf{\Sigma}|^{1/2} \tag{3.8}$$

Produit de deux fonctions gaussiennes.
Le produit de deux gaussiennes multivariée donne une autre gaussienne multivariée [PP12]:

$$g(\mathbf{x}; \lambda_1, \mu_1, \mathbf{\Sigma_1}) \cdot g(\mathbf{x}; \lambda_2, \mu_2, \mathbf{\Sigma_2}) = g(\mathbf{x}; \lambda_3, \mu_3, \mathbf{\Sigma_3}) \tag{3.9}$$

où

$$\begin{aligned} \mathbf{\Sigma_3} &= (\mathbf{\Sigma_1}^{-1} + \mathbf{\Sigma_2}^{-1})^{-1} \\ \mu_3 &= \mathbf{\Sigma_3}(\mathbf{\Sigma_1}^{-1}\mu_1 + \mathbf{\Sigma_2}^{-1}\mu_2) \\ \lambda_3 &= g(\mu_1; \lambda_1\lambda_2, \mu_2, \mathbf{\Sigma_1} + \mathbf{\Sigma_2}) \end{aligned}$$

Intégrale du produit de deux gaussiennes multivariée.
En appliquant les équations (3.8) et (3.9), le résultat de cette intégrale est:

$$\begin{aligned} &\int_{\mathbf{R}^D} g(\mathbf{x}; \lambda_1, \mu_1, \mathbf{\Sigma_1}) g(\mathbf{x}; \lambda_2, \mu_2, \mathbf{\Sigma_2}) d\mathbf{x} \\ &\quad = g(\mu_1; \lambda_1\lambda_2, \mu_2, \mathbf{\Sigma_1} + \mathbf{\Sigma_2}) \int_{\mathbf{R}^D} g(\mathbf{x}; 1, \mu_3, \mathbf{\Sigma_3}) d\mathbf{x} \\ &\quad = g(\mu_1; \lambda_1\lambda_2(2\pi)^{D/2} |\mathbf{\Sigma_3}|^{1/2}, \mu_2, \mathbf{\Sigma_1} + \mathbf{\Sigma_2}) \end{aligned}$$

Convolution de deux gaussiennes multivariée.
Enfin, en substituant l'équation (3.7) dans l'équation (3.9), nous obtenons alors :

$$\begin{aligned}
&\int_{\mathbf{R}^D} g(\mathbf{t};\lambda_1,\mu_1,\mathbf{\Sigma_1})g(\mathbf{x}-\mathbf{t};\lambda_2,\mu_2,\mathbf{\Sigma_2})d\mathbf{t} \\
&\quad = \int_{\mathbf{R}^D} g(\mathbf{t};\lambda_1,\mu_1,\mathbf{\Sigma_1})g(\mathbf{t};\lambda_2,\mathbf{x}-\mu_2,\mathbf{\Sigma_2})d\mathbf{t} \\
&\quad = g(\mathbf{x};\lambda_1\lambda_2(2\pi)^{D/2}|\mathbf{\Sigma_3}|^{1/2},\mu_1+\mu_2,\mathbf{\Sigma_1}+\mathbf{\Sigma_2})
\end{aligned}$$

3.4 Synthèse de carte de normales

Le *normal mapping* est une technique communément utilisée pour feindre la présence de détails géométriques pendant le calcul d'éclairage sans la générer explicitement. Nous avons proposé dans ce travail de générer ce type de texture en tirant avantage de la formulation de notre *Spot Noise* .

La création de cartes des normales depuis des fonctions procédurales a déjà été formalisée par le passé. Ce processus consiste généralement à considérer le résultat de la texture procédurale comme un champ de hauteur sur la surface et d'en calculer les dérivées partielles [Bli78, Mik10]. Les dérivées peuvent être calculées par différences finies, ce qui nécessite d'évaluer plusieurs fois la fonction procédurale par pixel et qui va drastiquement augmenter le temps d'évaluation de la texture. Une manière alternative consiste à les calculer analytiquement.

3.4.1 Dérivées Partielles

Nous définissons alors notre *Spot Noise* comme un champ de hauteur dans le plan tangent de la surface et nous utilisons la formulation du noyau afin de calculer la pente du champ de hauteur dans l'espace de texture. En appliquant la règle de la somme sur la formulation du *Spot Noise* , nous calculons la pente finale en ajoutant les dérivées partielles de chaque noyau, et par extension, ajoutons les dérivées partielles de chaque gaussienne. Cela nous permet d'évaluer directement les dérivées partielles de notre *Spot Noise* à la volée :

$$\frac{\partial \mathrm{lcsn}(\mathbf{x})}{\partial \mathbf{x}} = \sum_{i=1}^{M} w_i \sum_{j=1}^{N} \frac{\partial g(\mathbf{x}-\mathbf{x_i};\lambda_j,\mu_j,\mathbf{\Sigma_j})}{\partial \mathbf{x}} \tag{3.10}$$

avec

$$\frac{\partial g(\mathbf{x};\lambda,\mu,\mathbf{\Sigma})}{\partial \mathbf{x}} = g(\mathbf{x};\lambda,\mu,\mathbf{\Sigma})(-\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)) \tag{3.11}$$

3.4.2 Calcul d'éclairage

Lors du calcul de l'éclairage, nous évaluons les dérivées partielles du *Spot Noise* et nous nous retrouvons avec deux scalaires $\frac{\partial \mathrm{lcsn}(\mathbf{x})}{\partial x}$ et $\frac{\partial \mathrm{lcsn}(\mathbf{x})}{\partial y}$. Il existe plusieurs façons

d'utiliser ces dérivées partielles afin de perturber le calcul de l'éclairage. La première est de calculer la normale perturbée $\omega_{\mathbf{p}}$ en utilisant la transformation suivante :

$$\omega_{\mathbf{p}} = \frac{\left(-\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x}, -\frac{\partial \text{lcsn}(\mathbf{x})}{\partial y}, 1 \right)^T}{\sqrt{(\frac{\partial \text{lcsn}(\mathbf{x})}{x})^2 + (\frac{\partial \text{lcsn}(\mathbf{x})}{y})^2 + 1}} \tag{3.12}$$

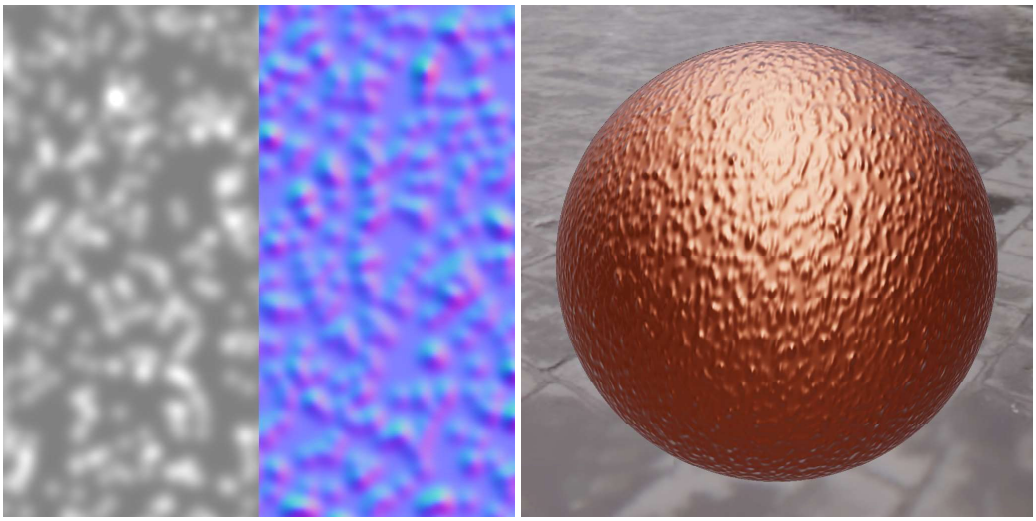


FIGURE 3.6 – En utilisant les dérivées partielles du champ de hauteur (à gauche), nous calculons la carte de normale à la volée (au milieu) pour ajouter des détails géométriques lors du calcul d'éclairage en temps-réel.

Une solution alternative est d'utiliser directement ces valeurs pendant l'évaluation de la **BRDF** en utilisant une distribution "décentrée", trouvable dans les travaux de Olano et Baker [OB10] (équation (1)) et de Dupuy *et al.* [DHI+13] (équations (8) et (10)).

Dans un modèle d'éclairage comme Cook-Torrance, présenté en début de chapitre 2, il est possible d'écrire la fonction D de distribution de normales en trois dimensions en utilisant une fonction de distribution de pentes en deux dimensions. Cette distribution des pentes, nommée P_{22} dans la littérature, peut être "décentrée" en utilisant nos dérivées partielles.

En utilisant le produit du jacobien de la transformation normale/pente $\frac{1}{(\omega_n \cdot \omega_g)^3}$ avec l'inverse de la projection de la normale sur la macrosurface $\frac{1}{(\omega_n \cdot \omega_g)}$, la fonction de distribution de normales $D(\omega_n)$ est construite à partir de la fonction de distribution des pentes $P_{22}(\tilde{n})$:

$$D(\omega_n) = \frac{P_{22}(\tilde{n})}{(\omega_n \cdot \omega_g)^4} \tag{3.13}$$

où ω_g est la normale géométrique, $\tilde{n}$ correspond à la pente associée à la normale $\omega_n = (x_n, y_n, z_n)^T$ et est obtenu par la transformation suivante :

$$\tilde{n} = \begin{pmatrix} -x_n/z_n \\ -y_n/z_n \end{pmatrix}$$

Inversement, en utilisant l'équation (3.12) nous pouvons retrouver ω_n. La fonction de distribution des pentes utilisée pour une distribution de Beckmann anisotrope et décentrée est donnée par une loi normale 2D :

$$P_{22}(\mathbf{x};\mu,\Sigma) = \frac{exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu))}{(2\pi)\sqrt{|\Sigma|}} \tag{3.14}$$

et pour une fonction de distribution des pentes de GGX anisotrope et décentrée est donnée par la formule suivante :

$$P_{22}(\mathbf{x};\mu,\Sigma) = \frac{1}{(2\pi)\sqrt{|\Sigma|}(1+(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu))^2} \tag{3.15}$$

où μ correspond à la moyenne de la distribution, dans notre cas $\mu = (\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x}, \frac{\partial \text{lcsn}(\mathbf{x})}{\partial y})^T$, et la matrice de covariance Σ est définie par :

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

où σ_x et σ_y contrôle respectivement la rugosité de la surface en x et en y et ρ correspond au facteur de corrélation pour gérer l'anisotropie. Dans la littérature, ces paramètres de rugosités sont notés α_x et α_y. La relation entre ces deux notations est la suivante : $\alpha_i = \sqrt{2}\sigma_i$.

3.4.3 Problématique du filtrage des cartes de normales

Re-précisons tout d'abord que le filtrage correct et efficace des cartes de normales est toujours un problème ouvert en Informatique Graphique. En effet, les textures de normales discrètes sont généralement naïvement filtrées en utilisant le *mipmapping* pendant le calcul de l'éclairage, ce qui a l'inconvénient de changer l'apparence de l'objet après filtrage, causant un éclairage faussé du matériau.

Le *mipmapping* de texture fonctionne avec des textures de couleurs puisqu'il faut intégrer les intensités sous l'empreinte de pixel. La couleur moyenne qui résulte de cette opération à un sens pour la suite du rendu. En effectuant la même opération sur une carte de normales, nous obtenons la normale moyenne de la surface. Or, cette dernière ne va pas donner une représentation fidèle de la complexité de la surface sous-pixel. La figure 3.7, issue de [BN12], illustre ce phénomène.

Afin de filtrer efficacement ce genre de textures, il faut transformer l'information de perturbation géometrique en rugosité de surface. Pour cela, plusieurs approches ont été proposées [BN12]. Le but de ces méthodes est de trouver une représentation interpolable et *moyennable* des normales sous forme de rugosité. Prenons par exemple le cas des approches de filtrage basées sur le calcul des moments d'ordre 2 comme le LEAN [OB10] ou le LEADR [DHI+13] mapping. Ces méthodes définissent la matrice de covariance de la fonction de distribution de normales en utilisant les moments d'ordre 2 précalculés et *mipmappés* pour définir la rugosité de la surface (la taille du lobe spéculaire). La pente moyenne est ensuite utilisée comme décentrage de la fonction de distribution de normales.

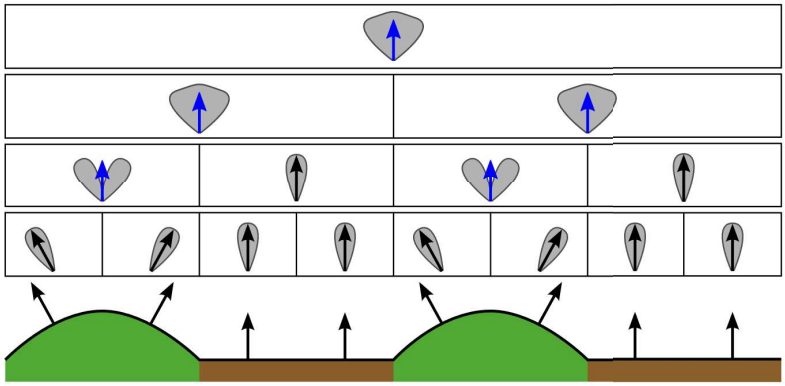


FIGURE 3.7 – En utilisant naïvement le *mipmapping* de la carte de normales (flèches noires), nous obtenons rapidement une normale moyenne de la surface (flèche bleue) qui ne permet pas à elle seule de représenter fidèlement la surface. Des approches ont été proposées pour préfiltrer la distribution des normales (lobes gris sur le schéma). Figure issue de [BN12].

Dans notre cas, une telle approche ne peut être évaluée à la volée. En effet, cela nécessiterait de calculer $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x}^2$, $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial y}^2$ et $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x}\frac{\partial \text{lcsn}(\mathbf{x})}{\partial y}$. Comme montré dans l'équation (3.10), les dérivées partielles sont calculées en effectuant la somme pondérée des dérivées partielles des gaussiennes. Calculer analytiquement les moments de deuxième ordre requiert d'évaluer le carré de ces sommes, faisant apparaître des termes croisés, rendant une évalution à la volée trop complexe au regard des performances visées.

Cependant, il se trouve que les dérivées partielles (premières et secondes) des gaussiennes admettent des formes closes sous convolution (voir la sous-section 3.4.3.1). Nous proposons tout de même d'obtenir la pente moyenne du *Spot Noise* à la volée sous l'empreinte de pixel. En remplaçant les dérivées partielles dans l'équation (3.12) ou en les utilisant directement dans une fonction de distribution de normales non centrée, nous obtenons malgré tout une approximation de la surface perturbée sous-jacente et nous réduisons ainsi les défauts d'aliassage (implémentation *shadertoy* : **/Wdc3W7**).

3.4.3.1 Filtrage des dérivées partielles premières du *Spot Noise*

Dans cette sous-section, je développe la preuve que les dérivées partielles premières d'une fonction Gaussienne 2D admettent bien des formes closes sous convolution. Nous partons tout d'abord des équations décrivant les dérivées partielles d'une Gaussienne (équation (3.11)).

$$\frac{\partial g(\mathbf{x};\lambda,\mu,\mathbf{\Sigma})}{\partial \mathbf{x}} = g(\mathbf{x};\lambda,\mu,\mathbf{\Sigma})(-\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu))$$

Nous posons alors un certain nombre de notations afin d'alléger les équations suivantes. Pour une matrice de covariance $\mathbf{\Sigma}_i$, un vecteur moyenne μ_i et un vecteur arbitraire $\mathbf{x}$, nous notons leurs éléments comme suit:

$$\mathbf{\Sigma}_i^{-1} = \begin{bmatrix} a_i & b_i \\ b_i & c_i \end{bmatrix} \qquad \mu_\mathbf{i} = \begin{pmatrix} \mu_{ix} \\ \mu_{iy} \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

Afin de réduire encore la taille des expressions, nous utiliserons la notation suivante pour les fonctions gaussiennes 2D:

$$g_i(\mathbf{x}) = g(\mathbf{x}; \lambda_i, \mu_\mathbf{i}, \mathbf{\Sigma_i})$$

En repartant de l'équation (3.11), nous obtenons le système d'équation suivant :

$$\begin{aligned}\frac{\partial g_i(\mathbf{x})}{\partial \mathbf{x}} &= -\begin{bmatrix} a_i & b_i \\ b_i & c_i \end{bmatrix} \begin{pmatrix} x_i - \mu_{ix} \\ y_i - \mu_{iy} \end{pmatrix} g_i(\mathbf{x}) \\ &= -\begin{pmatrix} a_i(x_i - \mu_{ix}) + b_i(y_i - \mu_{iy}) \\ b_i(x_i - \mu_{ix}) + c_i(y_i - \mu_{iy}) \end{pmatrix} g_i(\mathbf{x})\end{aligned}$$

Nous obtenons donc les deux dérivées partielles (respectivement en x et en y) ci-dessus. Nous pouvons aussi les écrire sous une forme utilisant le produit scalaire :

$$\begin{aligned}\frac{\partial g_i(\mathbf{x})}{\partial x} &= -(a_i, b_i)^T \cdot (\mathbf{x} - \mu_\mathbf{i}) g_i(\mathbf{x}) \\ \frac{\partial g_i(\mathbf{x})}{\partial y} &= -(b_i, c_i)^T \cdot (\mathbf{x} - \mu_\mathbf{i}) g_i(\mathbf{x})\end{aligned}$$

La preuve nécessite de connaître l'intégrale 2D du produit d'une gaussienne avec respectivement x ou y :

$$\begin{aligned}\int_{\mathbf{R}^2} x \cdot g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma}) d\mathbf{x} &= \mu_x \lambda (2\pi) |\mathbf{\Sigma}|^{1/2} \\ \int_{\mathbf{R}^2} y \cdot g(\mathbf{x}; \lambda, \mu, \mathbf{\Sigma}) d\mathbf{x} &= \mu_y \lambda (2\pi) |\mathbf{\Sigma}|^{1/2}\end{aligned}$$

Convolution d'une dérivée partielle de gaussienne avec une fonction gaussienne : Nous commençons par calculer l'intégrale du produit d'une dérivée partielle de gaussienne 2D $g_1(\mathbf{x})$ selon x et d'une autre gaussienne $g_2(\mathbf{x})$:

$$\begin{aligned}\int_{\mathbf{R}^2} \frac{\partial g_1(\mathbf{x})}{\partial x} g_2(\mathbf{x}) d\mathbf{x} &= \int_{\mathbf{R}^2} -\Big((a_1, b_1)^T \cdot (\mathbf{x} - \mu_\mathbf{1})\Big) g_1(\mathbf{x}) g_2(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathbf{R}^2} \Big((a_1, b_1)^T \cdot (\mu_\mathbf{1} - \mathbf{x})\Big) g_1(\mathbf{x}) g_2(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathbf{R}^2} \Big((a_1, b_1)^T \cdot \mu_\mathbf{1} - (a_1, b_1)^T \cdot \mathbf{x}\Big) g_1(\mathbf{x}) g_2(\mathbf{x}) d\mathbf{x}\end{aligned}$$

Nous rappellerons que le produit de deux gaussiennes g_1 et g_2 donne une autre gaussienne g_3, ce qui permet d'obtenir:

$$\begin{aligned}\int_{\mathbf{R}^2} \frac{\partial g_1(\mathbf{x})}{\partial x} g_2(\mathbf{x}) d\mathbf{x} &= \int_{\mathbf{R}^2} \Big((a_1, b_1)^T \cdot \mu_\mathbf{1} - (a_1, b_1)^T \cdot \mathbf{x}\Big) g_3(\mathbf{x}) d\mathbf{x} \\ &= \int_{\mathbf{R}^2} \Big((a_1, b_1)^T \cdot \mu_\mathbf{1}\Big) g_3(\mathbf{x}) d\mathbf{x} - \int_{\mathbf{R}^2} \Big((a_1, b_1)^T \cdot \mathbf{x}\Big) g_3(\mathbf{x}) d\mathbf{x} \quad (3.16)\end{aligned}$$

La première partie de l'équation 3.16 peut être évaluée en utilisant la forme close de l'intégrale d'une gaussienne :

$$\begin{aligned}\int_{\mathbf{R}^2}\Big((a_1,b_1)^T\cdot\mu_{\mathbf{1}}\Big)g_3(\mathbf{x})d\mathbf{x} &= \Big((a_1,b_1)^T\cdot\mu_{\mathbf{1}}\Big)\int_{\mathbf{R}^2}g_3(\mathbf{x})d\mathbf{x}\\ &= \Big((a_1,b_1)^T\cdot\mu_{\mathbf{1}}\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}\end{aligned} \tag{3.17}$$

Pour la seconde partie de l'équation 3.16 nous développons le produit scalaire entre $(a_1,b_1)^T$ et $\mu_{\mathbf{1}}$:

$$\begin{aligned}\int_{\mathbf{R}^2}\Big((a_1,b_1)^T\cdot\mathbf{x}\Big)g_3(\mathbf{x})d\mathbf{x} &= \int_{\mathbf{R}^2}\Big(a_1x+b_1y\Big)g_3(\mathbf{x})d\mathbf{x}\\ &= \int_{\mathbf{R}^2}\Big(a_1x\Big)g_3(\mathbf{x})d\mathbf{x}+\int_{\mathbf{R}^2}\Big(b_1y\Big)g_3(\mathbf{x})d\mathbf{x}\\ &= a_1\int_{\mathbf{R}^2}xg_3(\mathbf{x})d\mathbf{x}+b_1\int_{\mathbf{R}^2}yg_3(\mathbf{x})d\mathbf{x}\end{aligned}$$

En utilisant les formes closes introduites précédemment :

$$\begin{aligned}a_1\int_{\mathbf{R}^2}xg_3(\mathbf{x})d\mathbf{x}+b_1\int_{\mathbf{R}^2}yg_3(\mathbf{x})d\mathbf{x} &= \Big(a_1\mu_{3x}+b_1\mu_{3y}\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}\\ &= \Big((a_1,b_1)^T\cdot\mu_{\mathbf{3}}\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}\end{aligned} \tag{3.18}$$

En substituant 3.17 et 3.18 dans 3.16 nous obtenons :

$$\begin{aligned}&\int_{\mathbf{R}^2}\Big((a_1,b_1)^T\cdot\mu_{\mathbf{1}}\Big)g_3(\mathbf{x})d\mathbf{x}-\int_{\mathbf{R}^2}\Big((a_1,b_1)^T\cdot\mathbf{x}\Big)g_3(\mathbf{x})d\mathbf{x}\\ &\quad= \Big((a_1,b_1)^T\cdot\mu_{\mathbf{1}}\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}-\Big[\Big((a_1,b_1)^T\cdot\mu_{\mathbf{3}}\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}\Big]\\ &\quad= \Big[\Big((a_1,b_1)^T\cdot\mu_{\mathbf{1}}\Big)-\Big((a_1,b_1)^T\cdot\mu_{\mathbf{3}}\Big)\Big]\lambda_3(2\pi)|\Sigma_3|^{1/2}\\ &\quad= \Big((a_1,b_1)^T\cdot(\mu_{\mathbf{1}}-\mu_{\mathbf{3}})\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}\end{aligned}$$

Ce qui résulte :

$$\int_{\mathbf{R}^2}\frac{\partial g_1(\mathbf{x})}{\partial x}g_2(\mathbf{x})d\mathbf{x} = \Big((a_1,b_1)^T\cdot(\mu_{\mathbf{1}}-\mu_{\mathbf{3}})\Big)\lambda_3(2\pi)|\Sigma_3|^{1/2}$$

Afin de retrouver la convolution nous substitutions $\mu_{\mathbf{1}}$ par $\mathbf{x}-\mu_{\mathbf{1}}$

$$\int_{\mathbf{R}^2} \frac{\partial g_1(\mathbf{x}-\mathbf{t})}{\partial x} g_2(\mathbf{t}) d\mathbf{t} = -\Big((a_1,b_1)^T \cdot (\mathbf{x} - (\mu_\mathbf{1} + \mu_\mathbf{3}))\Big)\lambda_3(2\pi)|\mathbf{\Sigma_3}|^{1/2}$$

En suivant le même processus selon y, nous obtenons finalement :

$$\int_{\mathbf{R}^2} \frac{\partial g_1(\mathbf{x}-\mathbf{t})}{\partial y} g_2(\mathbf{t}) d\mathbf{t} = -\Big((b_1,c_1)^T \cdot (\mathbf{x} - (\mu_\mathbf{1} + \mu_\mathbf{3}))\Big)\lambda_3(2\pi)|\mathbf{\Sigma_3}|^{1/2}$$

3.4.4 Les dérivées secondes

Bien que nous synthétisions de la *fausse géométrie*, nous n'avons pas pris en compte, dans ce travail, le fait que nous ajoutions artificellement une information de courbure. Nous avons vu dans le début du Chapitre 2, en section 2.2, que la courbure géométrique sous pixel peut provoquer des défauts d'aliassage si le matériau est spéculaire. Or, cette information de courbure permettrait possiblement d'obtenir une intuition géométrique de l'évolution de notre fonction procédurale sous notre pixel.

La formulation analytique du **LCSN** permet, toujours en appliquant la règle des sommes, d'obtenir les dérivées secondes $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial x^2}$, $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial y^2}$ et $\frac{\partial \text{lcsn}(\mathbf{x})}{\partial xy}$ de notre fonction procédurale. De plus, il est aussi possible d'évaluer des formes closes de ces dérivées partielles secondes si elles sont convoluées par une empreinte de pixel gaussienne. Cette étude des dérivées partielles secondes n'a pas été réalisée et est laissée en perspective. Nous laissons tout de même la formulation de ces dérivées secondes en Annexe de ce document.

3.5 Contrôle Utilisateur

Ce modèle de *Spot Noise* , tel qu'introduit par Pavie *et al.* , vise à être intuitif pour un artiste. En s'inspirant des travaux de Charpenay *et al.* [CSM14], nous montrons comment le *Spot Noise* peut être modifié par l'usage de processus externes (comme une texture ou des fonctions procédurale) dans le but de créer un motif qui varie spatialement.

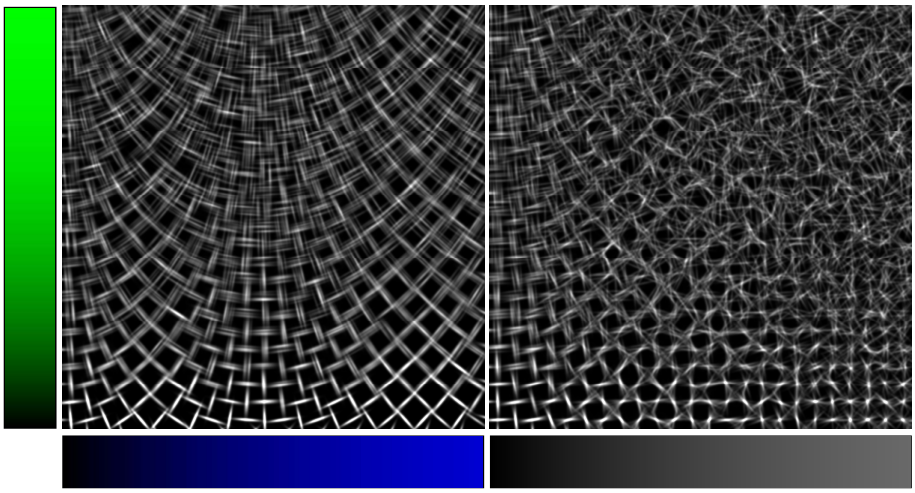


FIGURE 3.8 – Chaque paramètre du *Spot Noise* est directement controlé par une *carte de contrôle* (comme proposé par Charpenay *et al.* [CSM14]). À gauche: le canal vert contrôle la largeur de la distribution des jets des noyaux tandis que le canal bleu influence l'angle de rotation des noyaux. À droite: l'angle de rotation est choisi aléatoirement selon la valeur du canal alpha.

En utilisant des textures discrètes ou procédurales (voir Figure 3.8), nous pouvons modifier les paramètres purement géométrique du *Spot Noise* . Cette approche est similaire à celle proposée par Charpenay *et al.* avec le bruit de Gabor. Nous montrons sur l'image de gauche comment nous modifions l'angle de rotation (qui varie horizontalement) et la translation locale de chaque gaussienne (variant verticalement). De manière similaire, l'image de droite montre une perturbation croissante pour le choix de l'angle de rotation en utilisant une distribution uniforme variant horizontalement (d'une perturbation légère à plus importante). Notons qu'en interpolant les valeurs dans l'espace des paramètres avant de générer le résultat final, nous obtenons une variation douce du motif (implémentation *shadertoy* : **/Ws33W7**).

Cette méthode permet d'offrir à l'utilisateur un outil d'édition simple à intégrer dans un logiciel existant tout en lui offrant un contrôle précis sur l'apparence de la texture sur l'objet à la volée comme montré sur la Figure 3.9

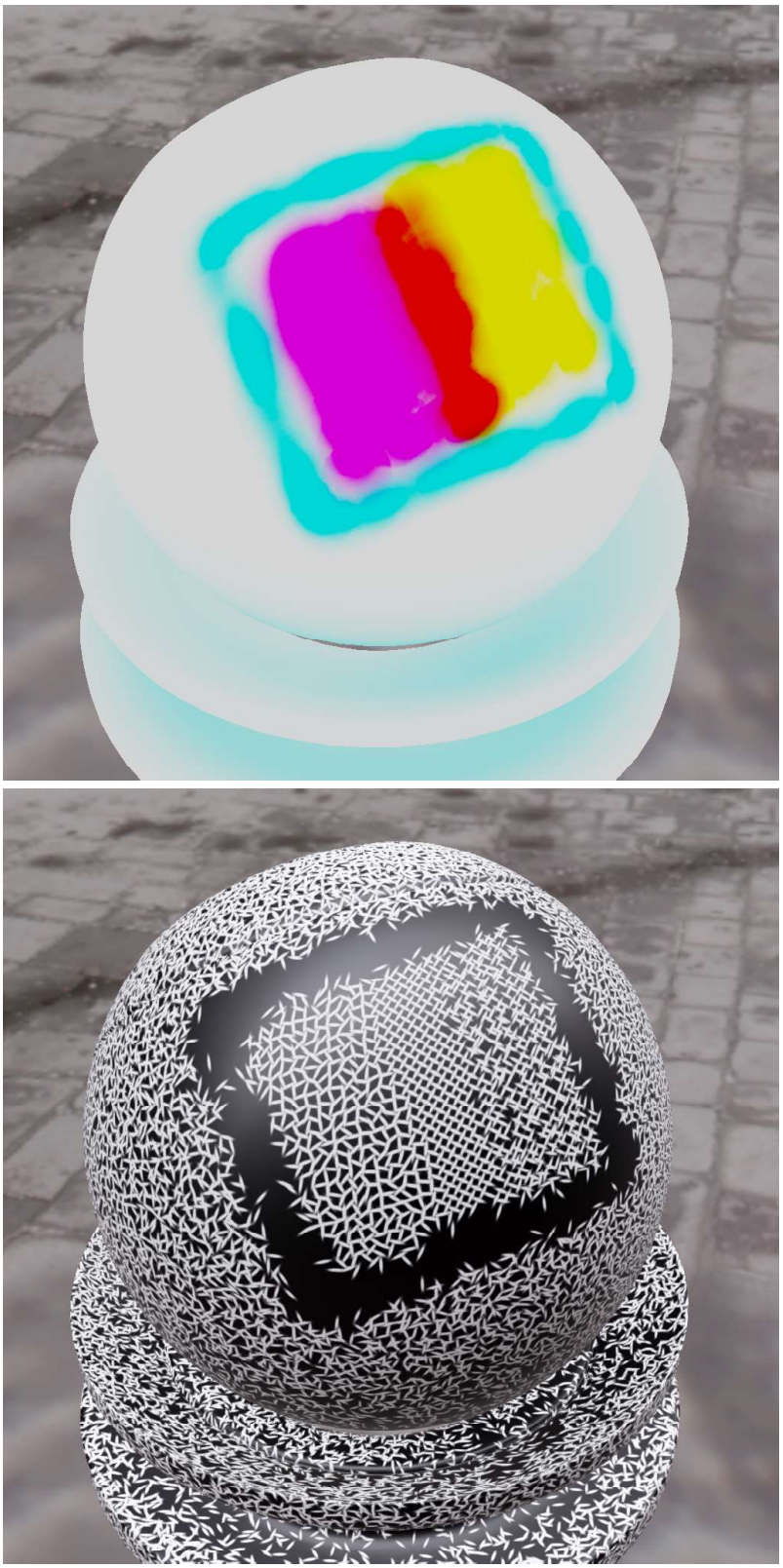


FIGURE 3.9 – L'utilisateur peut peindre ou modifier la carte de contrôle dans son outil d'édition (en haut) et obtenir à la volée le motif résultat (en bas).

3.6 Résultats

Nous commençons par montrer la différence de performances introduites par le léger changement dans la formulation du noyau (cf. Figure 3.10).

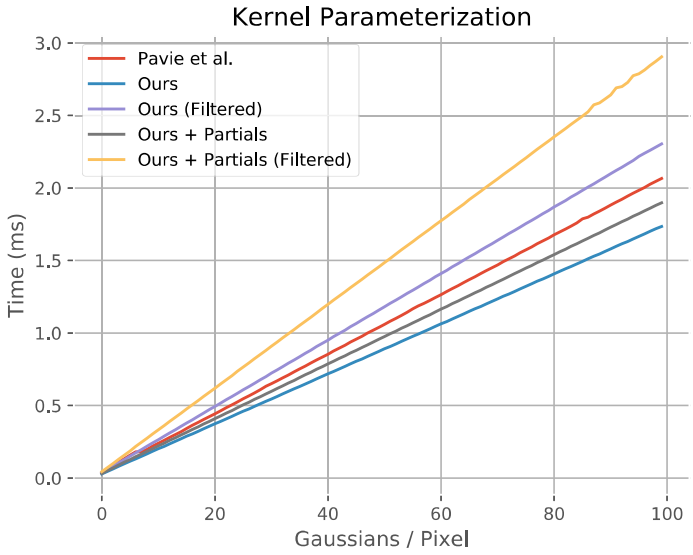


FIGURE 3.10 – Comparaisons entre notre formulation et celle proposée par Pavie *et al.* [PGDG16] . Les mesures ont été réalisées sur des textures ayant une définition de 1920 par 1080 pixels sur une carte graphique *NVidia GTX 2080*.

Pour la même qualité visuelle, c'est-à-dire sans évaluer le filtrage et les dérivées partielles, nous voyons une légère amélioration des performances en retirant l'utilisation des coordonnées homogènes. En évaluant les dérivées partielles, toujours sans filtrage, la méthode reste plus rapide que le modèle de Pavie *et al.* [PGDG16] qui ne génère que l'information de couleur. Cependant, en ajoutant le surcoût du filtrage, les performances diminuent logiquement.

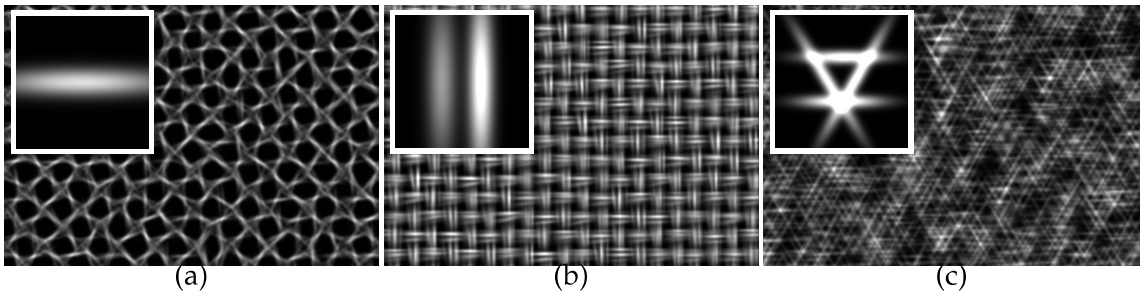


FIGURE 3.11 – Trois motifs générés en utilisant le *Spot Noise* localement contrôlé avec une complexité croissante (ie. le nombre de gaussiennes par noyau). (a) : Une gaussienne par noyaux avec une rotation symétrique pour les cellules impaires et cinq jets par cellules. (b) : Deux gaussiennes par noyaux avec une rotation symétrique pour les cellules impaires et dix jets par cellules. (c) : Quatres gaussiennes par noyaux et dix jets par cellules.

Le tableau 3.1 montre l'étude en terme de performance du modèle de *Spot Noise* (en millisecondes) pour trois motifs à complexité croissante. Les mesures ont été réalisées sur

deux *GPUs* (une Nvidia GTX 1080Ti et une GTX 2080) sur les trois motifs montrés en Figure 3.11 générés en 2048 par 2048. Comme expliqué dans la section 3.1, les performances découlent logiquement du nombre de gaussiennes à évaluer par pixel.

Scene (fig. 3.11)	Gauss/cell	GTX 1080Ti	GTX 2080
(a) Hive	5	5.1 ms	3.3 ms
(b) Fiber	20	15.7 ms	10.7 ms
(c) Sparkles	40	28.8 ms	19.4 ms

TABLE 3.1 – Performance *Spot Noise* en 2D (motifs dans la Figure 3.11) pour des textures en 2048x2048 pixels. Les performances sont directement liées aux nombre de gaussiennes à évaluer par pixel.

La Figure 3.12 montre l'impact de notre méthode de filtrage sur l'image finale pour la génération procédurale de texture couleur et de cartes de normales. Nous notons bien que le filtrage de la carte de normales est, de manière similaire aux cartes discrètes, toujours une approximation générant un calcul d'éclairage faussé (voir [BN12]).

Comme montré en Figure 3.14, ce modèle de *Spot Noise* est conçu pour être facile à utiliser pour un artiste et permet un contrôle intuitif de l'apparence finale. Contrairement aux approches basées exemples, l'utilisateur peut modifier chaque zone de la texture. De plus, le côté aléatoire de l'arrangement des noyaux peut être contrôlé localement en modifiant la densité de probabilité de chacun des paramètres selon l'envie de l'artiste. Par exemple, en Figure 3.14 (b), l'utilisateur peut créer différents types de motifs de manière interactive du plus régulier au plus aléatoire.

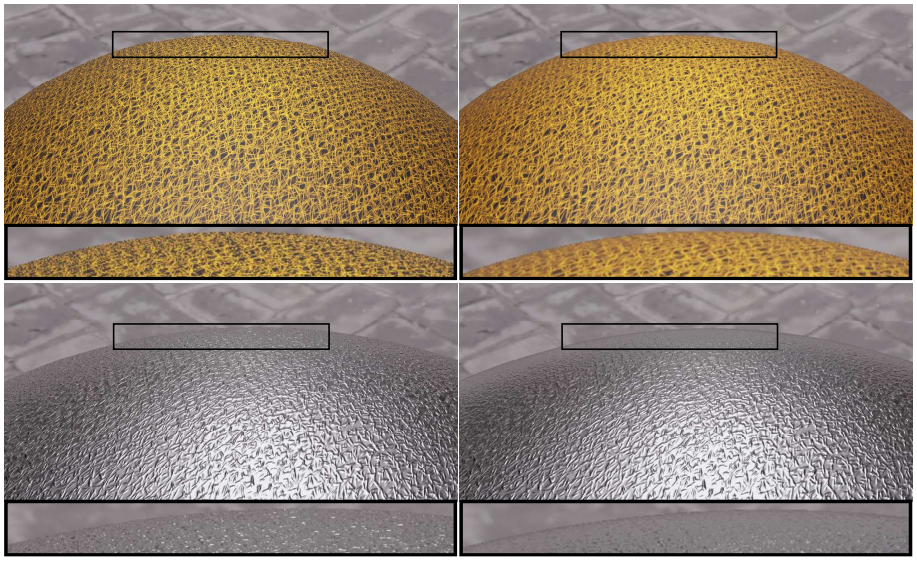


FIGURE 3.12 – Notre filtrage analytique (à droite) prévient des défauts d'aliassage (visible en colonne gauche) pour l'albedo (en haut) et la normale perturbée (en bas).

Le *Spot Noise* peut aussi être utilisé comme un *post-traitement* utilisant aussi bien une image (voir Figure 3.13) qu'un tampon de trame (ou *framebuffer* en Figure 3.14(e)) afin d'obtenir un résultat stylisé.

Enfin, comme mis en évidence par les Figures 3.14 (a),(c) et (d), le *Spot Noise* peut servir de primitive pour un graphe d'édition pour diriger différentes textures procédu-

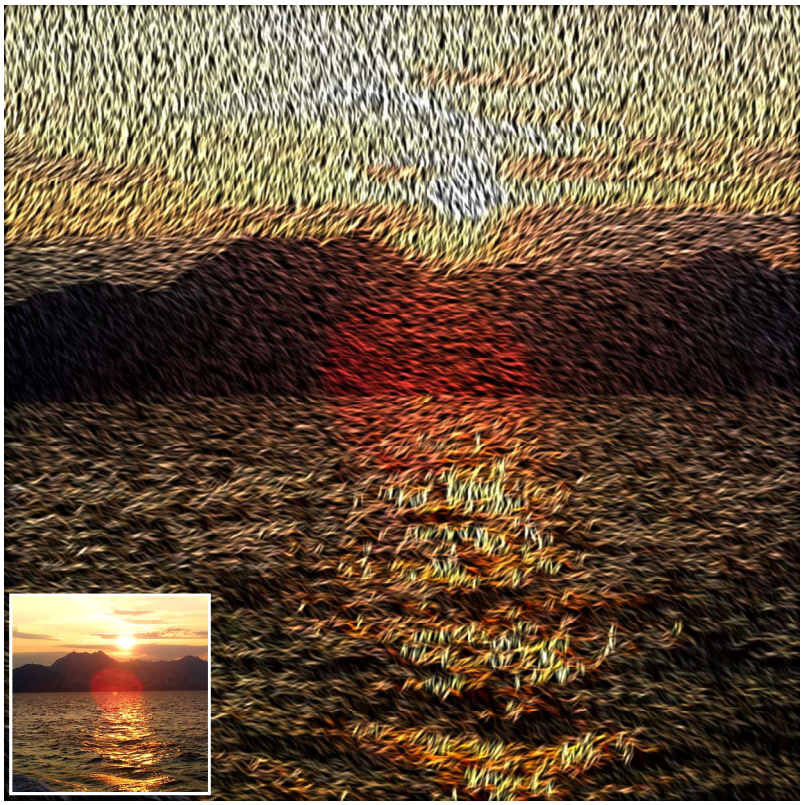


FIGURE 3.13 – Le *Spot Noise* peut être utilisé comme un post-traitement pour obtenir des rendus stylisé sur une entrée arbitraire (ici une photographie).

rales pour obtenir des textures composites. Ici, nous nous servons du motif contrasté comme d'une partition de l'espace conjointement avec deux textures procédurales stationnaires [HN18] pour obtenir une apparence complexe. Nous utilisons l'amplitude du motif pour mélanger de manière linéaire les deux textures procédurales. Nous utilisons ensuite les dérivées partielles évaluées pour perturber le calcul d'éclairage en conséquence et ajouter ce sentiment de relief à la texture finale.

3.7 Discussions et limitations

Comme présenté précédemment, ce modèle de *Spot Noise* génère une grande variété de motifs et d'apparences, allant du plus régulier au plus aléatoire. Cependant, le coût d'évaluation d'un tel modèle est directement dépendant de la complexité du noyau (du nombre de gaussiennes qui le composent) et du nombre de noyaux évalués par pixel.

Ce *Spot Noise* nécessite aussi la présence d'une paramétrisation de la surface. Il serait tout à fait envisageable de proposer un modèle de *Spot Noise* dit "setup-free", c'est à dire pouvant s'abstraire du besoin de coordonnées de textures. Toutefois, une telle approche déformerait et/ou briserait la structure générale si la surface ne possède pas de propriétés particulières.

Le filtrage analytique de ce genre de méthode donne de très bons résultats pour de la texture couleur mais n'est pas suffisant pour filtrer des cartes de normales. Comme

FIGURE 3.14 – Rendus des différents motifs et textures générées en utilisant notre *Spot Noise* . (a),(c) et (d) mettent en évidence la synthèse de textures composite en utilisant conjointement notre *Spot Noise* et deux textures procédurales [HN18] (moyenne du temps de rendu: 6.87ms). (b) montre des motifs influencés par le contrôle local des paramètres en utilisant des cartes de contrôle (avg : 4.6ms). (e) : un rendu stylisé depuis un tampon de trame (*framebuffer*) utilisant notre méthode comme post-traitement (avg : 0.34ms).

expliqué dans la Section 3.4, calculer la pente moyenne sous l'empreinte de pixel ne suffit pas. Il est nécessaire de trouver une représentation de ce *faux relief* sous forme de rugosité de la surface.

De même, des défauts de filtrage beaucoup plus complexes apparaissent lorsque nous utilisons notre *Spot Noise* comme partition de l'espace de texture pour créer des textures composites ou lorsque nous le modifions en utilisant des cartes de contrôles (cf. Section 3.6). Nous avons filtré indépendemment chaque composante, alors que le filtrage correct nécessiterait de prendre en compte l'intégralité de la représentation. Cette problématique reste non traitée.

3.8 Conclusion

Notre objectif principal reste de produire des images de hautes qualités en utilisant des textures procédurale calculée à-la-volée. Dans cette optique, nous avons présenté deux améliorations majeures au *Spot Noise* localement contrôlé à savoir: une méthode de filtrage anisotropique et analytique, prévenant ainsi des défauts d'aliassage quelque soit le motif dépeint, et une méthode analytique d'évaluation de carte de normales afin d'ajouter facilement du détail à la volée pendant le calcul de l'éclairage. De plus cette méthode offre à l'utilisateur un outil de contrôle intuitif sur l'apparence finale de la texture grâce au paramètrage purement géometrique du noyau et du motif.

Afin d'obtenir un grand niveau de réalisme dans les images de synthèse, les apparences virtuelles doivent contenir et révéler de fins détails non-stationnaires à plusieurs échelles. Comme montré dans ce chapitre, une ouverture pour de futurs travaux réside dans l'usage conjoint des méthodes de synthèse de texture procédurale stationnaire [GLM17, HN18] avec des motifs structurés variant spatialement afin de créer des surfaces texturées plus élaborées, tout en considérant prudemment la question du filtrage.

De manière similaire, le lien entre le filtrage correct des cartes de normales et l'apparence finale du matériau est toujours étudié en Informatique Graphique. Filtrer efficacement les cartes de normales générées procéduralement reste une problèmatique non triviale et intéressante pour de futurs travaux.

Conclusion et perspectives

Quels contextes d'utilisation pour les fonctions de bruits ?

À l'heure actuelle, les bruits évalués à la volée sont utilisés sous plusieurs formes variant selon les contraintes du contexte d'utilisation.

En rendu offline par exemple, ils sont utilisés pour ajouter très rapidement du détail à une surface puisque le budget de temps alloué n'est pas contraint. Pour du rendu temps-réel, voir interactif, les bruits sont utilisés par les artistes dans des outils de conception de contenus. Ils sont souvent représentés comme noeuds dans un graphe de texture et peuvent être évalués à la volée pour permettre à l'artiste d'obtenir un retour en temps réel ou interactif de la texture qu'il est en train de concevoir.

Dans les applications temps réel critiques comme le jeu vidéo, les fonctions de bruit (autres que [HN18]) sont généralement précalculées avant le rendu. Pourquoi ? Tout simplement parce que le budget de calcul d'une image dans ces applications est grandement contraint (33 ms pour obtenir 30 images par secondes, 16ms pour obtenir 60, etc.). Ce budget n'est pas seulement alloué au calcul de l'image mais aussi à la mise à jour du contenu (animation, calcul physique, etc.). Le grain ajouté sur une image sera bien plus performant si le bruit utilisé est stocké et lu par un accès texture pendant l'évaluation et l'application d'un *post-process*. Les nuages issus de la combinaison de plusieurs bruits de perlin et de texture cellulaires (**fBM** de [Wor96]) utilisés par [Sch16] sont précalculés pour alléger l'étape d'évaluation très demandante en calcul. Le nuage est obtenu par *raymarching* du volume obtenu et nécessite plusieurs optimisations (*culling* de l'horizon, rendu en basse résolution, reconstruction réalisée de manière temporelle, ...) pour obtenir une image correcte en un temps convenable pour le budget alloué.

Une exception peut être vue dans le domaine de la *demoscene*, où l'évaluation parfois directe des fonctions de bruits est réalisée pour définir un terrain accidenté par **fBM** ou générer un motif sur une surface. Ce choix est réalisé parce que la contrainte mémoire (exécutables de 4Ko, 64Ko, ...) prend le pas sur la contrainte de temps d'évaluation.

La génération de contenu et le niveau de détail.

L'aliassage est encore une problématique très étudiée en Informatique Graphique et définir correctement la bonne représentation géométrie-matériau-lumière pour modéliser une scène selon la résolution souhaitée est un problème toujours ouvert. L'aliassage de la géométrie, l'aliassage spéculaire, l'aliassage des cartes de normales sont autant d'exemples de problèmes pouvant altérer la qualité d'un rendu.

D'un autre côté, la génération de contenu tend à proposer des modèles de plus en plus généraux pour étendre le spectre des contenus concevables. Cependant, le contrôle sur les entrées et les sorties des algorithmes de génération de contenu peut permettre d'étudier plus en détail certaines problèmatiques. En étudiant non seulement les paramètres d'entrées du modèle génératif et les sorties, il est tout à fait envisageable de dégager suffisament de connaissances a priori pour définir le bon contenu pour le bon contexte. Nous pensons que les fonctions de bruits ne font pas exception à cette idée.

Générer d'autres informations pour le rendu filtré.

Nous avons présenté, dans ce manuscrit, différentes utilisations des fonctions de bruits pour modéliser des motifs non structurés. Que ce soit pour définir un volume, définir une texture ou encore appliquer un post-process, les bruits procéduraux se présentent comme un outil puissant de modélisation. En nous plaçant dans un contexte de rendu, nous avons pu voir comment les différentes échelles de détails viennent définir notre rendu final. Le schéma de filtrage défini dans cette thèse pour la texture d'albedo n'est correct que dans les hypothèses suivantes :

- l'information de couleur correspond aux valeurs directement issues du processus d'évaluation de la fonction de bruit. C'est à dire qu'aucune transformation non linéaire n'a été appliquée sur la fonction de bruit, ce qui peut être le cas lorsque l'artiste utilise conjointement des fonctions de bruits avec des cartes de couleurs.
- l'information de couleur est indépendante de la géométrie de la surface.

Nous dirigeons le lecteur vers les travaux suivants [HNPN13a, HNPN13b] prenant en compte le filtrage des transformations non linéaires des propriétés de la surface.

Nous avons aussi proposé une méthode de synthèse de carte de normales en utilisant à notre avantage la formulation du *Spot Noise* et du noyau utilisé. Cependant, notre méthode de filtrage reste incorrecte et génère un résultat similaire à une carte de normales discrètes filtrée par *mipmapping*. Nous pensons que poursuivre dans le filtrage correct et efficace des détails géométriques mésoscopiques est une voie de recherche intéressante. Comme expliqué plus haut, les connaissances a priori que nous pouvons avoir sur les statistiques globales et locales des informations générées à la volée peuvent nous aider à définir de nouvelles méthodes générant correctement et intelligemment du contenu. Nous pensons qu'il est pertinent de continuer et d'étendre les champs d'applications des fonctions de bruits ou de l'aléatoire contrôlé, de façon générale, la génération procédurale de détails pouvant s'appliquer à plusieurs niveaux d'échelles, en atteste le travail de Zirr et Kaplanyan [ZK16].

Etudier les bruits et définir des bonnes pratiques.

Pendant le déroulement de la thèse, deux travaux, respectivement [NH16] et [TNVT19], ont effectué une analyse qualitative des bruits définis spectralement comme le bruit de Gabor [LLDD09]. Le premier travail [NH16] a mis à jour un défaut de contraste jusqu'à lors non pris en compte sur l'oscillation du contraste de ces méthodes. Plusieurs méthodologies ont été proposées dans l'article pour limiter ce défaut quand il n'est pas souhaité. Le deuxième, [TNVT19], a étudié les différentes composantes d'un bruit de Gabor, du choix de la distribution de point, jusqu'au choix de la fonction d'harmonique utilisée (sinus vs cosinus) en effectuant l'étude statistique, non pas uniquement sur le processus stochastique lui-même, mais sur des réalisations de ce processus (instances).

Nous pensons qu'il faudrait poursuivre ces travaux dans le but de dégager un modèle correct de bruit par phase aléatoire. Ce travail synthétiserait 10 ans de recherche sur les bruits par convolution éparse dérivés du bruit de Gabor. Rien que pour la formulation analytique du noyau, en mettant de côté la synthèse par l'exemple, cette dernière a été souvent modifiée selon les contextes d'utilisations [LLDD09, LLD11, LD11, GSV+14, NH16, TNVT19, TEZ+19], comme en atteste l'état de l'art de cette thèse.

Une première piste de reflexion porterait sur la comparaison des cas d'utilisations entre la paramétrisation utilisant une distribution de points et celle utilisant une grille régulière. Dans une optique de performance, la paramétrisation par grille semble plus efficace que distribuer aléatoirement des noyaux dans l'espace, comme en atteste [GSV+14, HN18]. Cependant, la fonction de fenêtrage, permettant à la fois de pondérer le mélange des noyaux et d'obtenir une formulation analytique du spectre, doit avoir une largeur définie correctement afin d'éviter les discontinuités sur les bords des cellules de la grille. Une étude du type de grille à utiliser peut être réalisée. [HN18, Per01] utilisent une grille de simplexe limitant le nombre de noyaux contribuant à l'intensité du pixel à trois, améliorant ainsi les performances de ces bruits. Cependant, dans le cas d'un noyau analytique comme le noyau de Gabor [LLDD09] ou la somme pondérée de cosinus [GSV+14], la grille de simplexe limite la qualité du filtrage analytique. En effet, comme montré en chapitre 3, section 3.3, il peut arriver que l'empreinte de pixel recouvre une zone plus dense que la cellule considérée. Or, comme illustré par la Figure A, l'évaluation sur une grille de simplexe ne va considérer que trois points d'évaluation. Le calcul de la convolution, nécessaire pour réaliser le filtrage analytique, ne prendra en compte qu'une sous-partie du domaine à considérer.

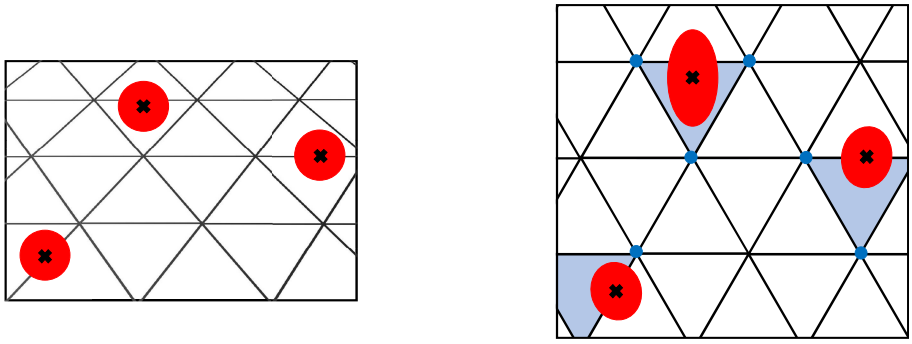


FIGURE A – Comme expliqué pour la grille régulière (Figure 3.5), dans le cas où l'empreinte de pixel est déformée, l'étape d'évaluation ne va prendre en compte la contribution que de trois noyaux, tronquant ainsi le domaine de l'empreinte de pixel à considérer pour obtenir un filtrage correct.

De plus, l'étude de la paramétrisation sous jacente pourrait s'étendre aux grilles non uniformes ou hiérarchique afin de proposer des outils intuitifs de génération de textures variant spatialement.

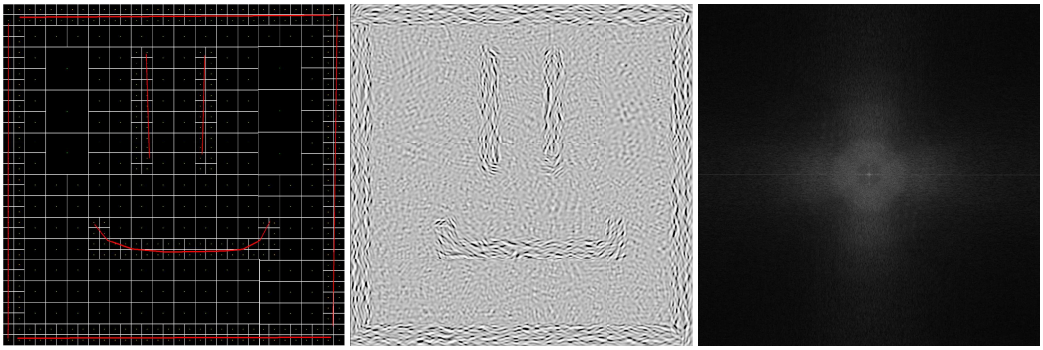


FIGURE B – Un exemple de bruit par phase aléatoire synthétisé sur un arbre hiérarchique (ici un *quadtree* linéaire). Ici, l'utilisateur dessine des segments influençant la création du quadtree (à gauche) et donc du motif généré (au milieu). Le contenu fréquentiel est choisi selon la profondeur du noeud et l'orientation choisie dépend du segment le plus proche.

Enfin, la génération procédurale de la géométrie mésoscopique en utilisant des bruits par phase aléatoire est une piste de recherche intéressante. En effet, ces derniers étant issus d'un processus aléatoire contrôlé, ils permettent, par construction, d'en déduire une distribution statistique Gaussienne. Il est alors possible de définir une fonction de distribution de normales (**NDF**) Gaussienne (Beckmann par exemple) lorsque le support de l'empreinte de pixel dépasse le voisinage local considéré. L'enjeu d'un tel travail serait de définir une représentation permettant une transition douce entre le calcul de l'éclairage perturbé par une carte de normales procédurale et le lobe spéculaire défini par cette distribution statistique quand l'empreinte devient suffisament grande. Il faudrait alors reposer les statistiques locales, liées à une instance issue du processus de génération et non plus lié au processus lui-même, nécessaires à une telle transition entre les deux représentations.

Annexe A

Formes Closes des fonctions gaussiennes

Sommaire

Annexe A

Formes Closes des fonctions gaussiennes

Dans cette annexe, nous présentons un inventaire des formes closes des fonctions gaussiennes. Nous définissons une fonction gaussienne g pour un point $\mathbf{x}$ de l'espace $\mathbf{R}^D$, centrée à la moyenne μ, avec une amplitude λ et définie par la matrice de covariance Σ.

$$g(\mathbf{x}; \lambda, \mu, \Sigma) = \lambda \, exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \, \Sigma^{-1} \, (\mathbf{x} - \mu))$$

En deux dimensions Σ, μ et $\mathbf{x}$ sont définis par :

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} \qquad \mu = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

A.1 Intégrales

L'intégrale d'une gaussienne de dimension D est définie par :

$$\int_{\mathbf{R}^D} g(\mathbf{x}; \lambda, \mu, \Sigma) d\mathbf{x} = \lambda \, (2\pi)^{D/2} |\Sigma|^{1/2}$$

Un gaussienne est normalisée (c'est à dire que son intégrale est égale à 1) si

$$\lambda = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}}$$

Les fonctions gaussiennes possèdent des formes closes pour des intégrales sur le domaine

$\mathbf{R}^2$

$$\int_{\mathbf{R}^2} x\, g(\mathbf{x};\lambda,\mu,\Sigma)d\mathbf{x} = \mu_x\, \lambda\, (2\pi)|\Sigma|^{1/2}$$
$$\int_{\mathbf{R}^2} y\, g(\mathbf{x};\lambda,\mu,\Sigma)d\mathbf{x} = \mu_y\, \lambda\, (2\pi)|\Sigma|^{1/2}$$
$$\int_{\mathbf{R}^2} x^2\, g(\mathbf{x};\lambda,\mu,\Sigma)d\mathbf{x} = (\sigma_x^2 + \mu_x^2)\, \lambda\, (2\pi)|\Sigma|^{1/2}$$
$$\int_{\mathbf{R}^2} y^2\, g(\mathbf{x};\lambda,\mu,\Sigma)d\mathbf{x} = (\sigma_y^2 + \mu_y^2)\, \lambda\, (2\pi)|\Sigma|^{1/2}$$
$$\int_{\mathbf{R}^2} xy\, g(\mathbf{x};\lambda,\mu,\Sigma)d\mathbf{x} = (\rho\sigma_x\sigma_y + \mu_x\mu_y)\, \lambda\, (2\pi)|\Sigma|^{1/2}$$

A.2 Produit

Le produit d'une fonction gaussienne multivariée $g(\mathbf{x};\lambda_1,\mu_\mathbf{1},\Sigma_1)$ avec une autre gaussienne multivariée $g(\mathbf{x};\lambda_2,\mu_\mathbf{2},\Sigma_2)$ donne une autre gaussienne multivariée :

$$g(\mathbf{x};\lambda_1,\mu_\mathbf{1},\Sigma_1)\cdot g(\mathbf{x};\lambda_2,\mu_\mathbf{2},\Sigma_2) = g(\mathbf{x};\lambda_3,\mu_\mathbf{3},\Sigma_3)$$

avec

$$\Sigma_3 = (\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}$$
$$\mu_3 = \Sigma_3(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2)$$
$$\lambda_3 = g(\mu_\mathbf{1};\lambda_1\lambda_2,\mu_\mathbf{2},\Sigma_1+\Sigma_2)$$

A.3 Convolution

La convolution d'une gaussienne multivariée $g(\mathbf{x};\lambda_1,\mu_\mathbf{1},\Sigma_1)$ par une autre gaussienne multivariée $g(\mathbf{x};\lambda_2,\mu_\mathbf{2},\Sigma_2)$ donne une autre gaussienne multivariée :

$$g(\mathbf{x};\lambda_1,\mu_\mathbf{1},\Sigma_1)\circledast g(\mathbf{x};\lambda_2,\mu_\mathbf{2},\Sigma_2) = g(\mathbf{x};\lambda_3,\mu_\mathbf{3},\Sigma_3)$$

avec

$$\Sigma_3 = \Sigma_1 + \Sigma_2$$
$$\mu_3 = \mu_1 + \mu_2$$
$$\lambda_3 = (2\pi)^{D/2}|(\Sigma_1^{-1} + \Sigma_2^{-1})^{-1}|^{1/2}\lambda_1\lambda_2$$

A.4 Dérivées partielles

Les dérivées partielles premières et secondes de dimensions D d'une fonction gaussienne sont données par [PP12] :

$$\frac{\partial g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial \mathbf{x}} = \Big(-\Sigma^{-1}(\mathbf{x}-\mu)\Big) g(\mathbf{x};\lambda,\mu,\Sigma)$$
$$\frac{\partial^2 g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial \mathbf{x}\partial \mathbf{x}^T} = \Big(\Sigma^{-1}(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\Sigma^{-1} - \Sigma^{-1}\Big) g(\mathbf{x};\lambda,\mu,\Sigma)$$

Les dérivées partielles premières de dimensions 2 d'une fonction gaussienne sont données par :

$$\frac{\partial g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial x} = -(a,b)^T \cdot (\mathbf{x}-\mu) g(\mathbf{x};\lambda,\mu,\Sigma)$$
$$\frac{\partial g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial y} = -(b,c)^T \cdot (\mathbf{x}-\mu) g(\mathbf{x};\lambda,\mu,\Sigma)$$

en posant

$$\Sigma^{-1} = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

Annexe B

Dérivées partielles secondes du *Local Spot Noise*

Annexe B

Dérivées partielles secondes du *Local Spot Noise*

En utilisant la règle de la somme pour la dérivation, nous pouvons calculer les dérivées partielles secondes de notre *Spot Noise* . Pour cela nous commençons par développer l'équation de la dérivée partielle seconde d'une gaussienne.
Les dérivées partielles secondes d'une fonction gaussienne sont données par :

$$\frac{\partial^2 g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial \mathbf{x}\partial \mathbf{x}^T} = \left(\Sigma^{-1}(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\Sigma^{-1} - \Sigma^{-1}\right) g(\mathbf{x};\lambda,\mu,\Sigma) \tag{B.1}$$

Nous posons tout d'abord les notations utilisées :

$$\mathbf{\Sigma}_i^{-1} = \begin{bmatrix} a_i & b_i \\ b_i & c_i \end{bmatrix} \qquad \mu_\mathbf{i} = \begin{pmatrix} \mu_{ix} \\ \mu_{iy} \end{pmatrix} \qquad \mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

nous développons la matrice $(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T$ de l'équation B.1 :

$$(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T = \begin{bmatrix} (x-\mu_x)^2 & (x-\mu_x)(y-\mu_y) \\ (x-\mu_x)(y-\mu_y) & (y-\mu_y)^2 \end{bmatrix}$$

En développant $\left(\Sigma^{-1}(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T\Sigma^{-1} - \Sigma^{-1}\right)$ nous obtenons une matrice symétrique dont les éléments sont :

$$\begin{aligned}
\Sigma_{00} &= (a(x-\mu_x) + b(y-\mu_y))^2 - a \\
\Sigma_{01} &= ab(x-\mu_x)^2 + cb(y-\mu_y)^2 + (ac+b^2)(x-\mu_x)(y-\mu_y) - b \\
\Sigma_{10} &= ab(x-\mu_x)^2 + cb(y-\mu_y)^2 + (ac+b^2)(x-\mu_x)(y-\mu_y) - b \\
\Sigma_{11} &= (b(x-\mu_x) + c(y-\mu_y))^2 - c
\end{aligned}$$

Nous obtenons alors les équations des dérivées secondes :

$$\frac{\partial^2 g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial x^2} = \Sigma_{00} g(\mathbf{x};\lambda,\mu,\Sigma)$$
$$\frac{\partial^2 g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial y^2} = \Sigma_{11} g(\mathbf{x};\lambda,\mu,\Sigma)$$
$$\frac{\partial^2 g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial x \partial y} = \frac{\partial^2 g(\mathbf{x};\lambda,\mu,\Sigma)}{\partial y \partial x} = \Sigma_{01} g(\mathbf{x};\lambda,\mu,\Sigma)$$

Ces dérivées partielles admettent une forme close sous convolution par une autre fonction gaussienne. La démonstration est similaire à celles présentées dans ce document. Elle fait intervenir plusieurs séparation d'intégrales et les formes closes de produit de deux gaussiennes et des intégrales liées aux fonctions gaussiennes. Nous mettons en lien le résultat de cette évaluation filtrée : **/3ddXDB**.

Bibliographie

[BHN07] Robert Bridson, Jim Houriham, and Marcus Nordenstam. Curl-noise for procedural fluid flow. In *ACM Transactions on Graphics (ToG)*, volume 26, page 46. ACM, 2007. 2.3.1

[Bli78] James F Blinn. Simulation of wrinkled surfaces. In *ACM SIGGRAPH computer graphics*, volume 12, pages 286–292. ACM, 1978. 3.4

[BLV+10] Pierre Bénard, Ares Lagae, Peter Vangorp, Sylvain Lefebvre, George Drettakis, and Joëlle Thollot. A Dynamic Noise Primitive for Coherent Stylization. *Computer Graphics Forum*, 29(4):1497–1506, June 2010. Session: Procedural Textures and Texture Atlases. (document), 2.4.1, 2.17, 2.5.2

[BN12] Éric Bruneton and Fabrice Neyret. A survey of non-linear pre-filtering methods for efficient and accurate surface shading. *IEEE Trans. Vis. Comput. Graph.*, 18(2):242–260, 2012. (document), 2.2, 3.4.3, 3.4.3, 3.7, 3.6

[BT09] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009. 2.4.2

[CD05] Robert L Cook and Tony DeRose. Wavelet noise. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 803–811. ACM, 2005. 2.3.2, 2.8, 2.9, 2.5.1

[CSHD03] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, July 2003. 1

[CSM14] Victor Charpenay, Bernhard Steiner, and Przemyslaw Musialski. Sampling gabor noise in the spatial domain. In Diego Gutierrez, editor, *Proceedings of the 30th Spring Conference on Computer Graphics - SCCG*, pages 79–82. ACM Press, May 2014. (document), 2.4.1, 2.18, 2.5.2, 2.5.3, 3.5, 3.8

[DH19] Thomas Deliot and Eric Heitz. Procedural stochastic textures by tiling and blending. In Wolfgang Engel, editor, *GPU Zen 2*, chapter 4. Black Cat Publishing, 2019. 2.4.3, 2.5.2

[DHI+13] Jonathan Dupuy, Eric Heitz, Jean-Claude Iehl, Poulin Pierre, Fabrice Neyret, and Victor Ostromoukhov. Linear Efficient Antialiased Displacement and Reflectance Mapping. *ACM Transactions on Graphics*, 32(6), September 2013. 2.2, 3.4.2, 3.4.3

[DJSJ19] Alexandre Derouet-Jourdan, Marc Salvati, and Theo Jonchier. Generating stochastic wall patterns on-the-fly with wang tiles. *Comput. Graph. Forum*, 38:255–264, 2019. 1, **??**

[EM03] David S Ebert and F Kenton Musgrave. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003. 2, 2.3, 2.3.1

[FL05] Chi-Wing Fu and Man-Kang Leung. Texture tiling on arbitrary topological surfaces using wang tiles. In *Proceedings of the Eurographics Symposium on Rendering Techniques, Konstanz, Germany, June 29 - July 1, 2005*, pages 99–104, 2005. **??**

[GD95] Djamchid Ghazanfarpour and Jean-Michel Dischler. Spectral analysis for automatic 3-d texture generation. *Computers & Graphics*, 19(3):413–422, 1995. (document), 2.4.2, 2.20, 2.4.2

[GD96] Djamchid Ghazanfarpour and Jean-Michel DISCHLER. Generation of 3d texture using multiple 2d models analysis. In *Computer Graphics Forum*, volume 15, pages 311–323. Wiley Online Library, 1996. 2.4.2, 2.5.2

[GD10] Guillaume Gilet and J-M Dischler. An image-based approach for stochastic volumetric and procedural details. In *Computer Graphics Forum*, volume 29, pages 1411–1419. Wiley Online Library, 2010. 2.4.2

[GDG12] Guillaume Gilet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Multiple kernels noise for improved procedural texturing. *The Visual Computer*, 28(6-8):679–689, 2012. (document), 1, **??**, 1, 2.4.2, 2.21, 2.22, 2.23, 2.5.2

[GDS10] Guillaume Gilet, Jean-Michel Dischler, and Luc Soler. Procedural descriptions of anisotropic noisy textures by example. In *Eurographics*, 2010. 2.4.2

[GGM10] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Random phase textures: Theory and synthesis. *IEEE Transactions on image processing*, 20(1):257–267, 2010. 1, **??**, 1, 2.4.2

[Gla04] Steve Glanville. Texture bombing. *GPU Gems: Programming Techniques, Tips, and Tricks for*, 2004. 1

[GLLD12] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. Gabor noise by example. *ACM Transactions on Graphics (TOG)*, 31(4):73, 2012. (document), 2.3, 2.4.2, 2.24, 2.4.2, 2.4.2, 2.5.2

[GLM17] B. Galerne, A. Leclaire, and L. Moisan. Texton noise. *Computer Graphics Forum*, pages n/a–n/a, 2017. (document), 2.4.2, 2.30, 2.5.1, 2.5.2, 3.1.1, 3.8

[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. (document), 2.4.2, 2.31

[GSV+14] Guillaume Gilet, Basile Sauvage, Kenneth Vanhoey, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Local random-phase noise for procedural texturing. *ACM Transactions on Graphics (TOG)*, 33(6):195, 2014. (document), 2.4.2, 2.25, 2.27, 2.28, 2.4.2, 2.31, 2.5.2, 3.1.1, 3.8

[GZD08] Alexander Goldberg, Matthias Zwicker, and Frédo Durand. Anisotropic noise. In *ACM Transactions on Graphics (TOG)*, volume 27, page 54. ACM, 2008. 1, 2.3.2, 2.10, 2.11, 2.5.1

[Hec89] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Technical Report UCB/CSD-89-516, EECS Department, University of California, Berkeley, Jun 1989. (document), 2.4, 2.2, 2.4.1, 2.5.1, 3.3, 3.3

[Hei14] Eric Heitz. *Appearance of multi-scale surfaces*. Theses, Université de Grenoble, September 2014. 2.2

[HN18] Eric Heitz and Fabrice Neyret. High-Performance By-Example Noise using a Histogram-Preserving Blending Operator. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):Article No. 31:1–25, August 2018. 1, 2.4.3, 2.36, 2.5.1, 2.5.2, 3.6, 3.14, 3.8, 3.8, 3.8

[HNPN13a] Eric Heitz, Derek Nowrouzezahrai, Pierre Poulin, and Fabrice Neyret. Filtering Color Mapped Textures and Surfaces. In Stephen N. Spencer, editor, *I3D'13 - ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13 Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pages 129–136, Orlando, United States, March 2013. ACM. 3.8

[HNPN13b] Eric Heitz, Derek Nowrouzezahrai, Pierre Poulin, and Fabrice Neyret. Filtering non-linear transferfunctions on surfaces. *IEEE transactions on visualization and computer graphics*, 20(7):996–1008, 2013. 3.8

[HSRG07] Charles Han, Bo Sun, Ravi Ramamoorthi, and Eitan Grinspun. Frequency domain normal map filtering. *ACM Transactions on Graphics (TOG)*, 26(3):28, 2007. 2.2

[JDJS19] Théo Jonchier, Alexandre Derouet-Jourdan, and Marc Salvati. Implementation of fast and adaptive procedural cellular noise. *Journal of Computer Graphics Techniques (JCGT)*, 8(1):35–44, January 2019. 1, **??**

[KHPL16] A. S. Kaplanyan, S. Hill, A. Patney, and A. Lefohn. Filtering distributions of normals for shading antialiasing. In *Proc. High Performance Graphics*, HPG, pages 151–162, 2016. 2.2

[KKL+07] Seung-Jean Kim, Kwangmoo Koh, Michael Lustig, Stephen Boyd, and Dimitry Gorinevsky. An interior-point method for large-scale l_1-regularized least squares. *IEEE journal of selected topics in signal processing*, 1(4):606–617, 2007. 2.4.2

[KKS08] Andrew Kensler, Aaron Knoll, and Peter Shirley. Better gradient noise. In *Tech. Rep. UUSCI-2008-001, SCI Institute*, 2008. 2.3.1

[LD05] Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM transactions on graphics (TOG)*, 24(4):1442–1461, 2005. 1, 1.4

[LD11] Ares Lagae and George Drettakis. Filtering Solid Gabor Noise. *ACM Transactions on Graphics*, 30(3), August 2011. 2.4.1, 2.16, 2.5.2, 3.8

[Lew84] John-Peter Lewis. Texture synthesis for digital painting. *SIGGRAPH Comput. Graph.*, 18(3):245–252, January 1984. 2.4.1

[Lew86] J P Lewis. Methods for stochastic spectral synthesis. In *Proceedings on Graphics Interface '86/Vision Interface '86*, pages 173–179, Toronto, Ont., Canada, Canada, 1986. Canadian Information Processing Society. 2.4.1

[Lew89] J. P. Lewis. Algorithms for solid noise synthesis. *SIGGRAPH Comput. Graph.*, 23(3):263–270, July 1989. 2.4.1

[LLC+10] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony Derose, George Drettakis, David S. Ebert, J.P. Lewis, Ken Perlin, and Matthias Zwicker. A Survey of Procedural Noise Functions. *Computer Graphics Forum*, 29(8):2579–2600, 2010. 2, 2.1

[LLD11] Ares Lagae, Sylvain Lefebvre, and Philip Dutré. Improving gabor noise. *IEEE Transactions on Visualization and Computer Graphics*, 2011. 2.4.1, 3.8

[LLDD09] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. *ACM Transactions on Graphics (TOG)*, 28(3):54, 2009. (document), 1, 2.4.1, 2.14, 2.4.1, 2.4.1, 2.4.2, 2.4.3, 2.5.1, 2.5.2, 3.1.1, 3.1.1, 3.3, 3.8

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001. 1

[LVLD10] Ares Lagae, Peter Vangorp, Toon Lenaerts, and Philip Dutré. Procedural isotropic stochastic textures by example. *Computers & Graphics*, 34(4):312–321, 2010. 2.4.2

[Mik10] Morten S Mikkelsen. Bump mapping unparametrized surfaces on the gpu. *Journal of Graphics, GPU, and Game Tools*, 15(1):49–61, 2010. 3.4

[NH16] Fabrice Neyret and Eric Heitz. Understanding and controlling contrast oscillations in stochastic texture algorithms using Spectrum of Variance. Research report, LJK / Grenoble University - INRIA, May 2016. (document), 2.4.3, 2.5.2, 3.8

[OB10] Marc Olano and Dan Baker. Lean mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 181–188. ACM, 2010. 2.2, 3.4.2, 3.4.3

[OHHM02] Marc Olano, John Hart, Wolfgang Heidrich, and Michael McCool. *Real-time shading*. AK Peters/CRC Press, 2002. 2.3.1

[ON97] Marc Olano and Michael North. Normal distribution mapping. *Univ. of North Carolina Computer Science Technical Report*, pages 97–041, 1997. 2.2

[Per85] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985. 1, 2.3.1, 2.4.2

[Per01] Ken Perlin. Noise hardware. In Olano M., editor, *ACM SIGGRAPH Course Notes*, 2001. 2.3.1, 3.8

[Per02] Ken Perlin. Improving noise. In *ACM transactions on graphics (TOG)*, volume 21, pages 681–682. ACM, 2002. 2.3.1

[PGD+16] Nicolas Pavie, Guillaume Gilet, Jean-Michel Dischler, Eric Galin, and Djamchid Ghazanfarpour. Volumetric spot noise for procedural 3d shell texture synthesis. In *Proceedings of the conferece on Computer Graphics & Visual Computing*, pages 33–40. Eurographics Association, 2016. (document), 2.4.1, 2.19

[PGDG16] Nicolas Pavie, Guillaume Gilet, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Procedural texture synthesis by locally controlled spot noise [c]. In *Wscg*, 2016. (document), 2.4.1, 2.19, 2.5.2, 2.5.3, 3.1.1, 3.1, 3.2, 3.10, 3.6

[PH89] Ken Perlin and Eric M Hoffert. Hypertexture. In *ACM Siggraph Computer Graphics*, volume 23, pages 253–262. ACM, 1989. 2.1

[PN01] Ken Perlin and Fabrice Neyret. Flow Noise. 28th International Conference on Computer Graphics and Interactive Techniques (Technical Sketches and Applications), August 2001. 2.3.1

[PP12] K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012. Version 20121115. 3.3.1, A.4

[Qui15] Iñigo Quilez. Texture repetition, 2015. 1, 1.7

[Sch16] Andrew Schneider. Chapter 4.1 : Real-time volumetric cloudscapes. In W. Engel, editor, *GPU Pro 7: Advanced Rendering Techniques*. CRC Press, 2016. 1, 3.8

[TEZ+19] Thibault Tricard, Semyon Efremov, Cédric Zanni, Fabrice Neyret, Jonàs Martínez, and Sylvain Lefebvre. Procedural phasor noise. *ACM Trans. Graph.*, 38(4):57:1–57:13, July 2019. (document), 2.4.3, 2.35, 2.5.2, 3.8

[TK19] Yusuke Tokuyoshi and Anton S. Kaplanyan. Improved geometric specular antialiasing. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D, pages 18:1–18:9, 2019. 2.2

[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 1–4, Gênes, Italy, May 2019. 2.4.3, 2.5.2, 3.8

[Tok17] Yusuke Tokuyoshi. Error reduction and simplification for shading antialiasing. Technical report, Square Enix Co., Ltd., apr 2017. 2.2

[VSLD13] Kenneth Vanhoey, Basile Sauvage, Frédéric Larue, and Jean-Michel Dischler. On-the-fly multi-scale infinite texturing from example. *ACM Trans. Graph.*, 32(6):208:1–208:10, November 2013. 1

[vW91] Jarke J. van Wijk. Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph.*, 25(4):309–318, July 1991. (document), 2.4.1, 2.13, 2.5.2, 3.1.1

[Wei04] Li-Yi Wei. Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 55–63. ACM, 2004. 1

[WMLT07] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 195–206. Eurographics Association, 2007. 2.2

[Wor96] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM, 1996. 1, 1, **??**, 3.8

[YNBH10] Qizhi Yu, Fabrice Neyret, Eric Bruneton, and Nicolas Holzschuch. Lagrangian texture advection: Preserving both spectrum and velocity field. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1612–1623, 2010. 2.3.1, 2.7

[ZK16] Tobias Zirr and Anton S Kaplanyan. Real-time rendering of procedural multiscale materials. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 139–148. ACM, 2016. 2.5.1, 3.8

If everything you try works, you are not trying hard enough.
- Gordon Moore -