Filtrage de textures procédurales définies par fonctions de transfert

Simon LUCAS

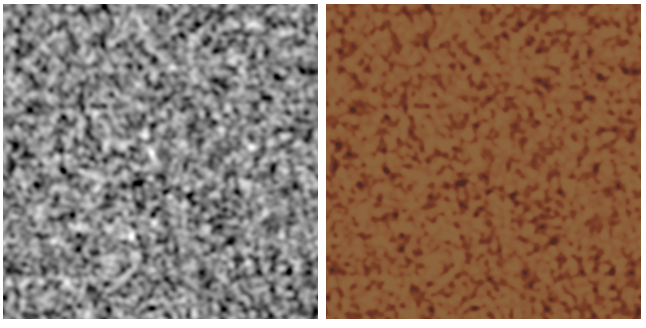
Résumé—Les facteurs essentiels dans le rendu d'effets photo-réalistes sont nombreux et différents, mais une influence décisive est donnée par les propriétés des matériaux qui sont utilisés pour qualifier les objets de la scène que l'on souhaite représenter. Parmi ces facteurs figurent les textures que l'on peut créer à partir d'une image, d'un tableau de pixels ou définir de façon analytique par un algorithme. On parle alors de textures procédurales. Dans le cadre de mon travail d'étude et de recherche (TER), je me suis intéressé à ce type de textures ayant des caratéristiques particulières. Pour ce faire, je suis parti des travaux de Heitz et al. [HNPN13] en implémentant leur méthode de filtrage des textures résultant de la composition d'un bruit scalaire avec une carte de couleur aussi appelée fonction de transfert ou colormap.

1 Introduction

En informatique graphique, les textures sont couramment utilisées. Appliquées sur des surfaces, elles peuvent permettre d'augmenter le réalisme d'une scène [Per85], [PH89] ou bien donner des informations visuelles supplémentaires. Dans les industries du divertissement, particulièrement les domaines du jeu vidéo et du cinéma, les besoins en textures sont immenses. Il se pose alors le problème de la création de ces textures. Beaucoup de textures peuvent être peintes directement par des artistes mais toujours en un nombre limité. Une autre approche pour créer des textures – sans contrainte sur leur nombre – est de les créer de manière procédurale. Comme indiqué par A. Lagae [LLC+10], en informatique, l'adjectif procédural représente les éléments décrits par des algorithmes et non par des données.

Pour créer des textures, il est donc répandu d'utiliser une catégorie de fonctions procédurales appelées fonctions de bruit. Ces fonctions de bruit produisent de manière générale des textures en niveaux de gris. Pour représenter une plus grande variété d'apparences, ces fonctions de bruit sont souvent couplées à des fonctions de transfert (TF) qui permettent d'associer à une intensité de bruit une couleur spécifique telle que $\mathcal{C}(Noise(x \in \mathbb{R}^n)) \rightarrow RGB$. La figure 1 représente une texture de bruit ainsi que son application sur une TF permettant de représenter une texture de liège.

Dans [HNPN13], les auteurs s'intéressent au filtrage de ces textures procédurales. Lors d'une synthèse d'image utilisant des textures, il est nécessaire d'appliquer un filtrage sur les textures afin de réduire l'effet de moiré introduit par un repliement de spectre, phénomène qui introduit, dans un signal échantillonné, des fréquences qui ne devraient pas s'y trouver. L'objet de mon TER consiste à *i)* implémenter cette méthode et *ii)* d'en tester les avantages et limites.



(a) Bruit de Gabor (b) Texture de liège

Fig. 1: Application d'une carte de couleur

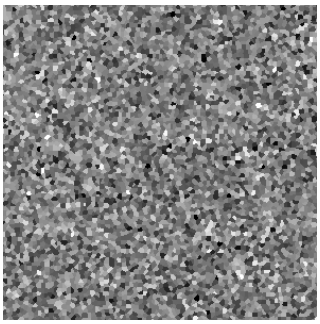
Nous allons en section 2 parler dans un premier temps de la génération de textures procédurales. En section 3, nous présenterons des notions élémentaires sur les textures avant de montrer, en section 4, comment les filtrer d'abord de manière naïve en présentant les limites de l'approche lors de l'utilisation de cartes de couleurs. Nous verrons en section 5 la solution apportée par [HNPN13]. Enfin, en section 6, nous présenterons une implémentation du filtrage de Heitz et al. [HNPN13] garantissant les performances nécessaires pour un rendu temps réel.

2 Génération de bruit

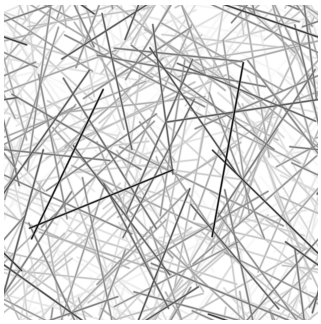
Comme indiqué précédement, les textures procédurales sont générées par des algorithmes. Ces algorithmes doivent être capables de rendre les variations des textures imprévisibles de telle sorte qu'aucune répétition ne soit visible.

Ces fonctions peuvent être utilisées de différentes manières. Elles peuvent être utilisées afin de générer des textures de bruit qui sont ensuite stockées en

• *Simon LUCAS: Université de Strasbourg*
E-mail: simon.lucas@etu.unistra.fr.

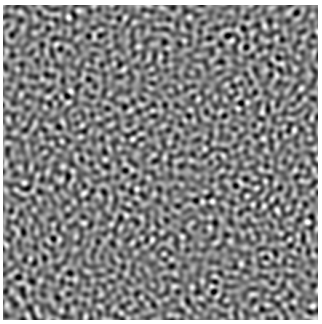


(a) Voronoi noise

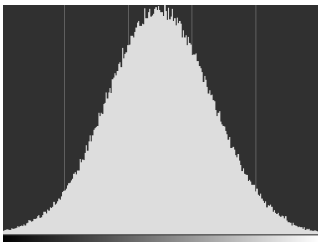


(b) Scratch noise

Fig. 2: Exemples de fonctions de bruit



(a) Cosine noise



(b) Histogram

Fig. 3: Fonction de bruit Gaussien

mémoire. Elles peuvent aussi être directement utilisées à la volée lors d'une synthèse d'image. Les fonctions utilisées à la volée permettent de ne pas avoir à stocker de textures dans la mémoire, et peuvent permettre d'obtenir un échantillonnage de la fonction à une coordonnée précise ce qui n'est pas forcément possible lors de l'utilisation de textures.

Échantillonner les fonctions à la volée a tout de même des inconvénients. Pour un rendu temps réel, certaines fonctions de bruit peuvent prendre trop de temps à calculer. Un autre inconvénient est que certains traitements ne peuvent pas être effectués directement sur les fonctions. Par exemple, le calcul d'une moyenne sur un sous-ensemble du domaine de définition de la fonction de bruit n'est pas toujours calculable analytiquement.

Dans cette section, nous nous intéressons principalement aux bruits générés par une somme de cosinus. Pour autant, il existe une multitude de fonctions procédurales ayant chacunes leurs particularités. On peut noter par exemple le bruit de Gabor (cf. figure 1a), les scratches (cf. figure 2b), ou bien les cellules de Voronoï (cf. figure 2a) ayant chacun un processus aléatoire différent.

Une caractéristique intrinsèque aux fonctions de bruit est leur distribution d'intensité. Cette distribution décrit la probabilité d'obtenir une intensité lors d'un échantillonnage. Le bruit par somme de cosinus a une distribution d'intensité appelée gaussienne car elle a la forme d'une loi normale. De manière discrète, cette distribution se représente sous forme d'histogramme comme le montre la figure 3.

Les illustrations 6 et 10 ont été faites à partir d'un bruit généré par une somme de cosinus évaluée à la volée en utilisant l'équation suivante :

$$noise(x,y) = \mu + \sqrt{\frac{2\sigma^2}{N_c}} \sum_{i=1}^{N_c} \cos(\phi_i + f_i \cos(\theta_i)x + f_i \sin(\theta_i)y) \quad (1)$$

où $noise$ est la fonction de bruit, N_c le nombre de cosinus à additionner, f_i une fréquence aléatoire, ϕ_i une phase aléatoire et enfin θ_i une orientation aléatoire. Les valeurs μ et σ correspondent respectivement à la moyenne et à l'écart-type du bruit. Ces valeurs ont pour but de modifier la distribution d'intensité généralement de telle sorte qu'elle soit centrée sur 0.5 et que les valeurs soient majoritairement comprises entre 0 et 1.

3 Bases et Vocabulaire

Avant parler de filtrage, il est nécessaire de parler un peu plus du problème qui nous est donné. On peut définir les textures comme des fonctions continues, telles que :

$$f(x) = y, \quad x \in \mathbb{R}^2, \quad y \in \mathbb{R}^n \quad (2)$$

Lorsque l'on veut prendre une valeur dans la texture, on va en récupérer un échantillon. Lors d'un rendu, on va donc échantillonner une texture et produire un ensemble de valeurs discrétisées séparées par un intervalle correspondant à la taille de l'empreinte du pixel (cf. section 3.2).

3.1 Traitement du signal

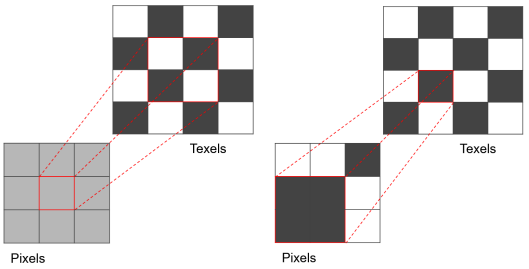
Le traitement d'image peut être considéré comme du traitement du signal. Un théorème important du traitement du signal portant sur la discrétisation est le théorème de Nyquist-Shannon. Il énonce qu'un signal discrétisé peut permettre de reconstruire le signal original continu uniquement si la fréquence d'échantillonnage f_e est supérieure ou égale à deux fois la fréquence maximale du signal original :

$$f_e \geq 2f_{max}. \quad (3)$$

En complément, on appelle la fréquence de Nyquist f_n , la moitié de la fréquence d'échantillonnage. Si la fréquence de Nyquist est inférieure à la fréquence maximale, on parle de sous-échantillonnage. A l'inverse, si elle est supérieure, on parle alors de sur-échantillonnage.

$$f_n > f_{max} \quad (4)$$

On peut en conclure que lors d'un sous-échantillonnage, il y a une perte d'information et le signal est dégradé.



(a) Minification (b) Magnification

Fig. 4: Minification et Magnification

3.2 Empreinte de pixel

Idéalement, lors d'une synthèse d'image, à un pixel de la fenêtre de rendu correspond un texel (texture element) dans la texture. Les scènes 3D ayant des surfaces complexes, des caméras perspectives, du mouvement, l'ensemble de ces caractéristiques va impliquer que plusieurs texels de la texture vont correspondre à un même pixel ou bien un seul texel est la projection de plusieurs pixels. Ces phénomènes sont respectivement appelés minification et magnification (cf. figure 4).

Cet effet est quantifié par la taille de l'empreinte du pixel. La notion d'empreinte de pixel correspond à la projection du pixel sur la surface. Lors d'une minification, la projection d'un pixel va donc correspondre à un ensemble de texels. Si le nombre d'échantillons est inférieur au nombre de texels dans l'empreinte, on se retrouve dans la situation où la fréquence d'échantillonnage est inférieure à la fréquence de la texture. Il y a donc une perte d'information dans le signal, l'intensité du pixel issue des échantillons de la texture sera bruitée par rapport à l'intensité de la texture.

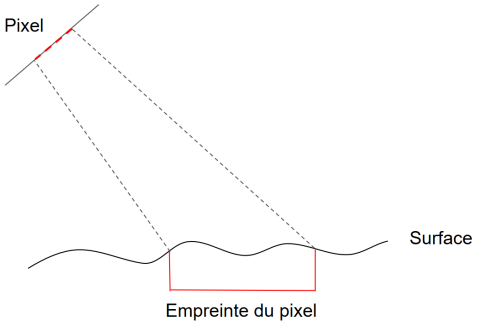


Fig. 5: Empreinte de pixel

Nous allons voir dans la section 4 certaines méthodes permettant de résoudre ce problème de perte d'information lors de minification.

4 Filtrage

Dans cette section, nous allons nous intéresser au filtrage naïf des textures. Comme vu précédemment, lors d'un rendu, il ne suffit pas d'échantillonner une seule valeur de la texture pour avoir l'intensité I du pixel. En effet, l'intensité du pixel doit correspondre à la moyenne des intensités du bruit sur l'empreinte du pixel. L'intensité du pixel I est ainsi définie par une intégrale sur l'empreinte du pixel.

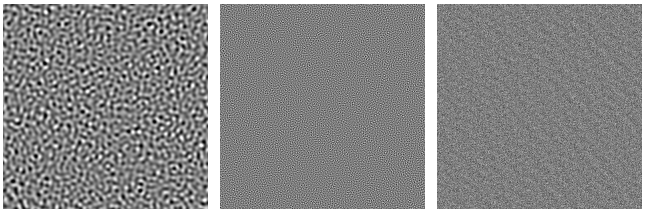
$$I = \frac{\int_{\mathcal{P}} f(x)dx}{\int_{\mathcal{P}} dx} \tag{5}$$

où $\mathcal{P}$ correspond à la surface de l'empreinte du pixel, $f(x)$ à la fonction de bruit et $\int_{\mathcal{P}} dx$ l'aire de la surface.

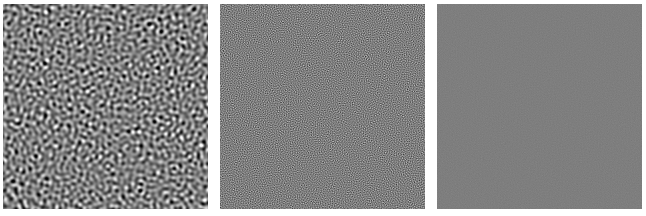
Pour calculer l'équation 5, il est nécessaire de calculer la moyenne des intensités sur un certain nombre d'échantillons de la texture dans l'intervalle correspondant à l'empreinte du pixel. Cette méthode est appelée sur-échantillonnage ou super-sampling. De manière discrète, on peut écrire l'équation 5 ainsi :

$$I = \frac{\sum_{i=1}^{N} f(x_i)}{N} \tag{6}$$

où x_i est la i$^{\text{ème}}$ position dans l'empreinte du pixel sur la surface. Les résultats de l'équation 5 permettent donc de supprimer l'aliasing (cf. figure 6b) engendré par un sous-échantillonnage d'une texture de bruit (cf. figure 6a). Il est à noter que lorsque N est égal à l'ensemble des texels dans $\mathcal{P}$, I est considéré comme la vérité terrain.



(a) Sans filtrage



(b) Avec filtrage

Fig. 6: Filtrage de bruit en niveau de gris.

Dans la figure 6, le ratio entre la taille du pixel et la taille du texel, qui définit la taille de l'empreinte est de $1/1$, puis $1/4^2$, puis $1/16^2$. En haut à droite, le sous-échantillonnage de la fonction de bruit cause un repliement de spectre. En bas à droite, le taux d'échantillonnage est suffisant, supprimant l'aliasing.

D'autres méthodes permettent d'obtenir la valeur moyenne du bruit, on peut noter la méthode du

mipmapping qui n'est d'ailleurs pas utilisée uniquement sur les textures procédurales mais plus généralement pour calculer la moyenne des texels. Le mipmapping a un avantage sur l'équation 6 étant donné qu'il n'y a aucun sur-échantillonnage réalisé à la volée avec le mipmapping. Cela permet de réaliser un filtrage de texture en temps réel.

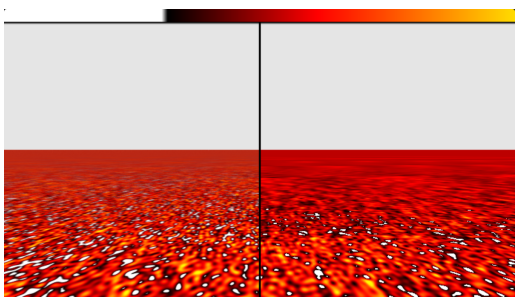
Pour le filtrage des textures transformées par TF, le filtrage naïf est défini par la fonction

$$C_0 \approx C\left(\frac{\int_{\mathcal{P}} f(x)dx}{\int_{\mathcal{P}} dx}\right). \tag{7}$$

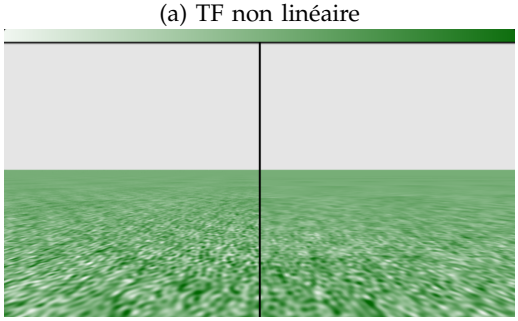
où C est la TF, C_0 le résultat du filtrage de la composée de la texture par la TF. En appliquant ce filtrage, on obtient une couleur définie à partir d'une moyenne d'intensité. Cela est efficace pour une TF linéaire, mais erroné pour les TFs non linéaires (cf. figure 7). En effet, pour que le filtrage soit efficace sur n'importe quelle TF, il ne faut pas faire la moyenne des intensités mais la moyenne des couleurs issues des intensités dans l'empreinte. Ce filtrage est décrit par l'équation :

$$C_0 = \frac{\int_{\mathcal{P}} C(f(x))dx}{\int_{\mathcal{P}} dx}. \tag{8}$$

La résolution de l'équation 8 oblige à effectuer du super-sampling. Nous verrons dans la section 5 la méthode apportée par Heitz et al [HNPN13] qui permet de résoudre cette équation sans passer par un sur-échantillonnage, permettant ainsi d'effectuer ce filtrage en temps réel.



(a) TF non linéaire



(b) TF linéaire

Fig. 7: Comparaison entre le filtrage naïf à droite et la vérité terrain à gauche.

(a) $\sigma_{max} = 0.25$

(b) $\sigma_{max} = 0.5$

(c) $\sigma_{max} = 1.$

Fig. 8: Fonctions de transfert convoluées

5 Solution apportée par Heitz et al

La solution apportée par Heitz et al [HNPN13] permet d'obtenir des solutions exactes lors du filtrage de texture gaussienne en rendu temps réel. Cela est possible car aucune intégration n'est faite pendant le rendu mais aussi parce que les calculs ne sont plus dépendants de la taille de l'empreinte du pixel. En effet, en partant du principe que la distribution des valeurs de la fonction de bruit sur une empreinte est connue, il est possible de réécrire l'équation 8 sous la forme :

$$C_0 = \int_{-\infty}^{\infty} C(v) D_f(\mathcal{P}, v) dv = \langle C, D_f \rangle, \tag{9}$$

où $D_f(\mathcal{P}, v)$ est la distribution d'intensité de la fonction f, $\mathcal{P}$ l'empreinte et v une intensité.

Dans le cas où f est une fonction générée par un processus gaussien, on sait que la distribution d'intensité est définie par une loi normale. On peut alors définir l'équation 9 ainsi :

$$C_0 = \langle C, N(\mu, \sigma^2) \rangle, \tag{10}$$

avec

$$\mathcal{N}(v, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(v-\mu)^2}{2\sigma^2}\right), \tag{11}$$

où μ est la moyenne de la loi normale et σ l'écart-type. La valeur μ correspond à la moyenne sur l'empreinte de la fonction notée $\overline{f_0}$, σ correspond à l'écart-type sur l'empreinte noté σ_0.

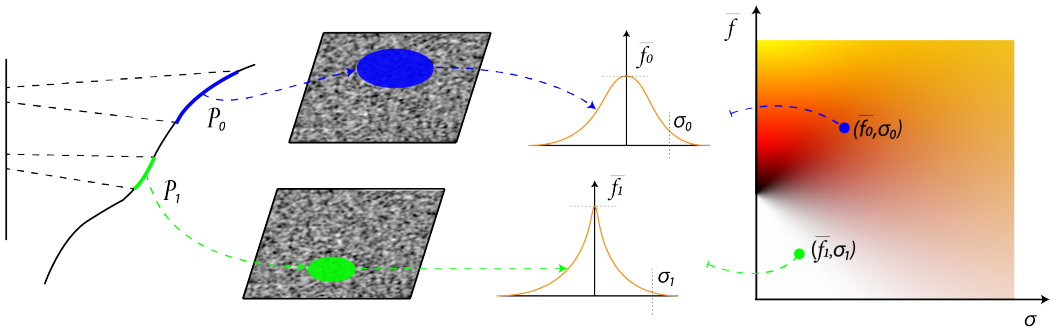


Fig. 9: Récupération de C_0

Nous sommes passés d'une intégration sur une empreinte de pixel à une intégration sur une intensité ayant deux inconnues $\overline{f_0}$ et σ_0. L'avantage est que cette intégration va donc pouvoir être faite dans une phase de pré-traitement. En pratique, cette phase va consister en la création d'un tableau 2D ayant pour ordonnée $\overline{f_0}$ et pour abscisse σ_0 telles que $\overline{f_0} \in [0,1]$ et $\sigma_0 \in [0, \sigma_{max}]$

De manière discrète, on peut noter et évaluer ce tableau ainsi :

$$T(\sigma_0, \overline{f_0}) = \sum_{\substack{i=-3\sigma_0+\overline{f_0}, \\ i+=\frac{6\sigma_0}{n}}}^{3\sigma_0+\overline{f_0}} C(i)\mathcal{N}(i, \overline{f_0}, \sigma_0^2)\left(\frac{6\sigma_0}{n}\right), \quad (12)$$

avec un n grand, au moins supérieur à 100.

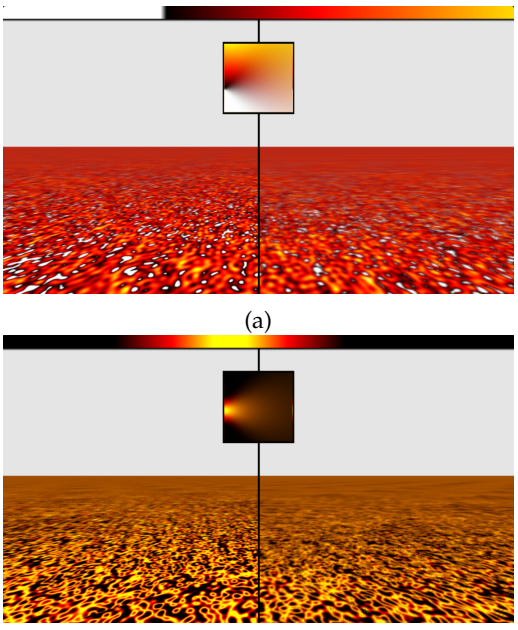
Cela nous permet de générer des TFs convoluées visibles en figure 8. On peut noter que pour un écart-type nul, les couleurs correspondent à la fonction de transfert.

Une fois que cette texture est générée, afin de récupérer la couleur C_0 d'un pixel, il va falloir récupérer la moyenne de l'empreinte ainsi que l'écart-type sur l'empreinte. Récupérer la moyenne peut être fait en utilisant les méthodes décrites dans la section 4 mais pour un filtrage temps réel, il est nécessaire de passer par le mipmapping. L'écart-type est défini par

$$\sqrt{\overline{f_0^2} - \overline{f_0}^2} = \sigma_0. \quad (13)$$

En plus d'appliquer l'algorithme du mipmapping sur la texture de bruit, il faudra aussi l'appliquer sur la texture de bruit au carré. Visuellement, la figure 9 représente l'analyse de l'empreinte $\mathcal{P}_0$ afin de récupérer les coordonnées σ_0 et $\overline{f_0}$ issues d'une surface pour évaluer la couleur C_0 dans la fonction de transfert convoluée.

En appliquant ce filtrage, on obtient les résultats présentés en figure 10. Mathématiquement, la méthode de filtrage de Heitz et al [HNPN13] permet d'obtenir des résultats identiques à la vérité terrain uniquement lorsque la distribution d'intensité dans l'empreinte est égale à la distribution gaussienne. En pratique, le résultat du filtrage diffère un peu de la vérité terrain. D'une part du au fait que la distribution réelle des intensités n'est pas forcément gaussienne sur l'empreinte mais aussi du fait de l'utilisation du mipmapping qui effectue des approximations.

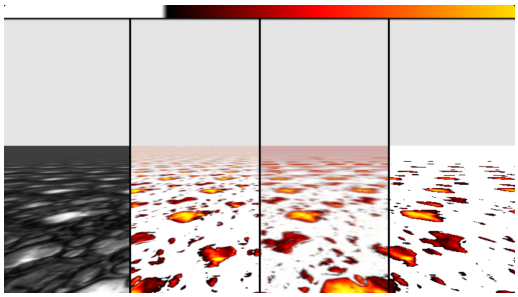


(a)

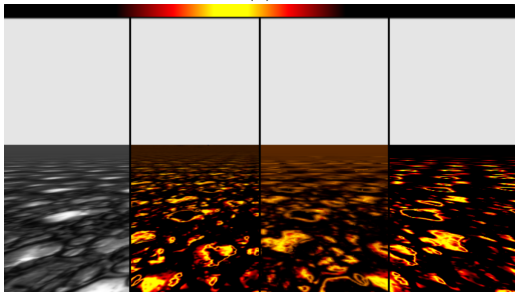
(b)

Fig. 10: Comparaison entre la vérité terrain à gauche et le filtrage de la méthode de Heitz et al [HNPN13] à droite

Comme nous pouvons voir dans la figure 11, en appliquant le filtrage sur une texture ayant une répartition des intensités non gaussienne, on obtient des résultats différents de la vérité terrain. Cette différence est due au fait que la TF convoluée n'est plus valide. La TF a été convoluée à partir d'une loi normale. Si l'on veut appliquer le filtrage sur une texture quelconque, il faut connaître au préalable sa loi de distribution. On notera que le filtrage de Heitz et al [HNPN13] reste préferable au filtrage naif (cf. figure 11).



(a)



(b)

Fig. 11: Comparaison entre la texture à gauche, la vérité terrain au milieu gauche, le filtrage de la méthode de Heitz et al [HNPN13] au milieu droit et le filtrage naïf à droite en utilisant une texture dont la répartition des intensités n'est pas gaussienne.

6 Implémentation de la méthode de filtrage

L'une des ambitions de la méthode de Heitz et al [HNPN13] consiste à appliquer le filtrage en temps réel. Nous verrons dans cette section par quelle méthode d'implémentation ce filtrage peut être réalisé de manière très performante et quasiment gratuitement en comparaison au filtrage naïf.

La génération de la TF convoluée est la seule partie non critique en terme de performance car c'est la seule à être calculable en différé durant une phase de précalcul. La méthode d'implémentation n'est donc pas primordiale, elle peut par exemple implémenter l'équation 12.

Une fois la TF convoluée calculée, le but est de calculer la moyenne $\overline{f_0}$ ainsi que l'ecart-type σ_0 sur l'empreinte le plus efficacement possible. Pour cela, comme évoqué en section 4, il est nécessaire de passer par du mipmapping qui permet de récupérer la moyenne d'une texture tout en gardant des performances élevée. Comme indiqué précédement, l'écart-type est calculable par l'équation 13. Il est donc nécessaire d'avoir accès à la texture de bruit ainsi qu'à la texture de bruit au carré qui doit donc être précalculée. Pour les cartes graphiques, les textures sont stockées dans des tableaux 2D ayant entre 1 et 4 canaux. L'idée pour accélérer au maximum le filtrage est de stocker la texture de bruit dans un premier canal et la texture au carré dans un second canal. Ainsi l'évaluation pour récupérer les moyennes n'est faite qu'une seule fois au lieu de deux. Les moyennes $\overline{f_0^2}$ et $\overline{f_0}$ peuvent donc être obtenues avec une seule évaluation. Une implémentation de cet algorithme issue d'un fragment shader est donnée en annexe 1.

Si l'implémentation est bien réalisée, les coûts de calculs en terme d'évaluation selon les méthodes sont les suivants :

- Filtrage naïf : 1 évaluation de texture par appel au mipmapping + 1 évalution de la TF.
- Filtrage de Heitz et al [HNPN13] : 1 évaluation texture par appel au mipmapping + 1 évaluation de la TF convoluée.

Nous pouvons remarquer que les différences de performance entre les deux méthodes de filtrage vont uniquement dépendre de la vitesse d'évaluation de la TF pour le filtrage naïf et de la TF convoluée pour le filtrage de Heitz et al. [HNPN13].

6.1 Mes implémentations

Durant ce TER j'ai pu implémenter la méthode de filtrage dans la bibliothèque $ASTex$[1]. $Astex$ est une bibliothèque C++ open-source de synthèse et traitement de textures développée à l'Université de Strasbourg par l'équipe IGG (Informatique Géométrique et Graphique). Cela ma permis de visualiser les résultats du filtrage sur des plans en 2D. Cette implémentation a été effectuée durant la première partie du TER lorsque je cherchais à comprendre l'algorithme et à l'implémenter de manière pratique sans vouloir, dans un premier temps, obtenir des performances optimales. J'ai donc commencé par effectuer le filtrage en utilisant du sur-échantillonage.

En parallèle, j'ai voulu visualiser l'action du filtrage dans une scène 3D. Pour cela, j'ai commencé par créer une démonstration sur le site Shadertoy[2] qui est une platforme mettant à disposition un fragment shader afin de réaliser des rendus procéduraux sur laquelle j'avais déjà un peu d'expérience. Cette première démonstration utilise le même algorithme implémenté pour la bibliothèque $Astex$.

Dans un second temps, j'ai implémenté le filtrage de Heitz et al. [HNPN13] tel qu'expliqué précedemement afin d'obtenir des performances optimales. Pour cela, j'ai donc réalisé une seconde démonstration sur le Shadertoy[3] utilisant le mipmapping. J'ai également développé une version OpenGL de cette technique de filtrage bénéficiant de l'efficacité de la rasterisation (cf. figure 12).

Les deux démonstrations faites sur le site Shadertoy dispose d'une option permettant de visualiser les coordonnées σ_0 et $\overline{f_0}$ corespondant au pointeur de la souris. Une implémentations générale de cet algorithme est présentée sur le pseudo-code 1.

1. https://astex-icube.github.io/
2. https://www.shadertoy.com/view/wtcSRH
3. https://www.shadertoy.com/view/WdScDW

Algorithm 1: Pseudo-code de l'algorithme de filtrage

begin
- /* Pré-traitement */
- $t \leftarrow$ texture de bruit;
- $t2 \leftarrow t$ mise au carré;
- $tf \leftarrow$ fonction de transfert;
- $ctf \leftarrow tf$ convoluée (cf. eq 12);
- /* Boucle de rendu */
- **foreach** *pixel* p **do**
  - /* Calcul des moyennes d'intensité sur l'empreinte de p (cf. section 4) */
  - $n \leftarrow$ moyenne de t sur $\mathcal{P}(p)$;
  - $n2 \leftarrow$ moyenne de $t2$ sur $\mathcal{P}(p)$;
  - /* Calcul de l'écart type des intensités sur l'empreinte de p (cf. section 5) */
  - $\sigma \leftarrow \sqrt{n2 - n * n}$;
  - /* Obtention de la couleur filtrée */
  - $C(p) \leftarrow ctf[\sigma][n]$;

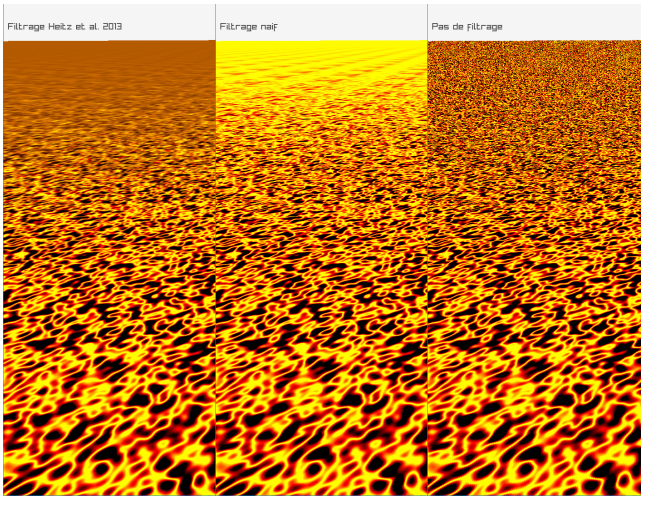


Fig. 12: Implémentation OpenGL du filtrage de Heitz et al [HNPN13]

7 Discussion et Conclusion

La méthode de Heitz et al [HNPN13] utilise une loi normale pour le pré-filtrage de la carte de couleur car elle part du principe qu'une grande partie des fonctions de bruit ont une distribution des intensités (D_f) gaussienne. Les résultats sont donc très proche de la vérité terrain lorsque D_f est gaussienne. Il serait intéressant de voir l'évolution du filtrage si D_f n'est pas gaussienne. Un exemple présenté par Heitz et al [HNPN13] ainsi que la figure 11 semblent suggérer que le filtrage de Heitz et al [HNPN13] reste plus efficace que le filtrage naif. Reste à en déterminer la limite.

Pour réaliser ce TER, j'ai rencontré deux difficultés principales. La première correspond à la découverte de nouvelles notions. Par exemple, la notion d'empreinte de pixel m'était inconnue. La seconde difficulté tenait au contenu de l'article de Heitz et al [HNPN13]. La notion d'implémentation était plus abordable pour moi, informatiquement, que les notions mathématiques largement utilisées dans l'article. En effet, cet article est d'avantage porté sur la résolution du problème par les mathématiques que par des solutions informatiques.

Remerciements

Je tiens à remercier Basile SAUVAGE pour la confiance qu'il a placée en moi en me proposant ce sujet ainsi que Xavier CHERMAIN pour toute l'aide qu'il m'a apportée. Merci aussi à Nathaniel SEYLER avec qui j'ai pu travailler en collaboration sur le sujet.

References

[HNPN13] 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.

[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.

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

[PH89] K. Perlin and E. M. Hoffert. Hypertexture. *SIGGRAPH Comput. Graph.*, 23(3):253–262, July 1989.

Annexe

Listing 1: Fragment shader appliquant le filtrage de Heitz et al [HNPN13]

```
// Input vertex attributes (from vertex shader)
in vec2 fragTexCoord;

// Transfer function convolved
uniform sampler2D tfConv;
// Noise values
uniform sampler2D textureNoise;

// Output fragment color
out vec4 fragmentColor;

void main()
{

  vec2 t = texture(textureNoise, fragTexCoord).xy;
  //  Mean of the squares - square of the mean
  float variance = t.y - t.x * t.x;
  float sigma = sqrt(variance);
  // Filtered color
  vec4 color = texture(tfConv, vec2(sigma,t.x));

  // Calculate final fragment color
  fragmentColor = vec4(color.xyz,1.);
}
```