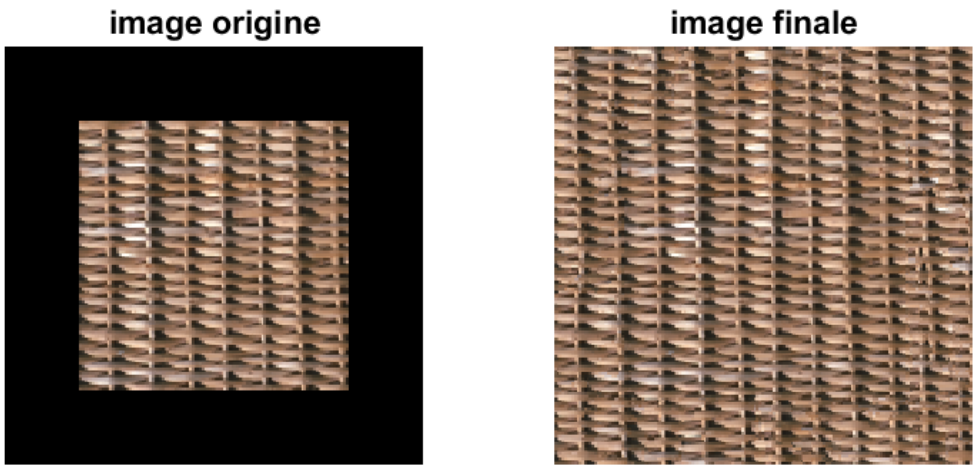
Rapport du TP1 du module de Statistique

Génération de textures

Guillaume DURET (p2021346)

Figure 1



27 Janvier 2021

1 Introduction

L'objectif ce TP est d'étudier la génération de textures, pour commencer les textures peuvent se séparer en plusieurs catégories allant de la texture régulière qui reproduit exactement un motif jusqu'à la texture stochastique ou il n'y aucun lien entre 2 pixels voisin (bruit) comme il est illustré sur la figure 2

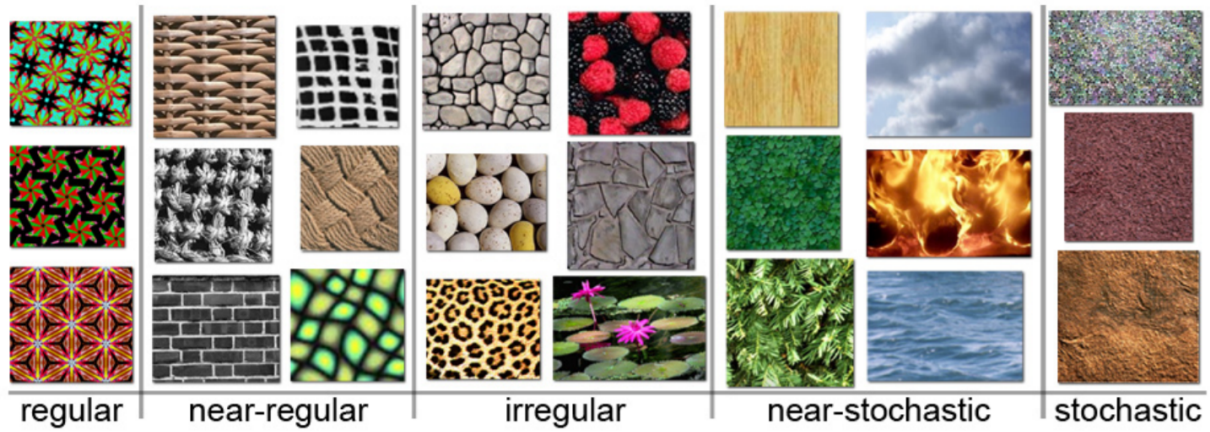


Figure 2: Présentation des différentes types de textures qui peuvent être amené à être généré

L'algorithme utilisé sera l'algorithme de Efros Leung (1999), en premier temps il sera expliqué son fonctionnement et enfin nous commenterons ses résultats et l'influence des paramètres.

2 Commentaires et explications algorithmiques

L'algorithme de Efros Leung est un algorithme qui se base sur les champs aléatoires de Marcov qui permet de dire que chaque pixel dépend de son voisinage. L'idée de l'algorithme de Efros Leung(fig 4) et donc de pouvoir générer des pixels en ayant une texture de base :

Input: image I_{smp}, output size, neighborhood size,

1. Initialize with 3x3 random seed of I_{smp}
2. While output I not filled
   1. Pick a pixel p in B with maximal neighbor pixels
   2. Compute the distance of $w(p)$ to all patches of input I_{smp}
   3. Pick randomly one of the similar neighborhood $w(p')$ in $\Omega'(p)$
   4. Set $I(p) = I_{smp}(p')$ (= Fill p with central value)

Figure 3

Figure 4: Différentes étapes de l'algorithme de Efros Leung

L'image de la texture qu'on possède est appelé I_{smp}, cette image sert de référence, en effet l'algorithme est de : Choisir un pixel à générer de tel sorte qu'il possède un nombre maximum de voisin. Cette étape est nécessaire pour que les pixels générés possèdent un maximum d'information pour que ceux-ci soient le plus fiable possible. En effet les pixels générés seront utilisés pour des pixels plus loin et donc si une erreur se produit elle risque de se répéter sur les pixels suivants. On calcule la distance des voisins de ce point p généré avec les voisins de chaque pixel de I_{smp}. Cette distance permet de calculer la probabilité que le point p soit un pixel q de I_{smp} selon ses voisins. La largeur du patch choisi permet de déterminer le nombre de pixel voisin que l'on prend en compte dans cette distance. La distance utilisée pour ce TP est :

$$d(w(p), w'(q)) = \frac{\sum_i (I(p+i) - I_{smp}(q+i))^2 G_\sigma(i)}{\sum_i (G_\sigma(i))} \tag{1}$$

qui représente une somme de distance en norme L2 entre les pixels du voisinages w du point p et les pixels de chaque voisinage w' de q de I_{smp} (Sum of Squared Difference (SSD)). Il est à noter que cette distance n'est qu'un choix et qu'il est possible d'utiliser d'autre normes. L'étape suivante est de choisir aléatoirement un pixel q de I_{smp} qui vérifie la condition :

$$d(w(p), w'(q)) < (1+\epsilon) dBest \tag{2}$$
$$\text{avec } dBest = d(w, w_{Best}) \text{ avec } w_{Best} = argmin_{w \in I_{smp}}(d(w, w(p)))$$

ϵ est un paramètre qui permet de contrôler si on accepte seulement le meilleur des pixels de I_{smp} (pour $\epsilon = 0$) ou si on accepte des distances proches de la distance minimale sans nécessairement prendre la meilleure distance.

Note importante : Il faut noter que l'algorithme qui a été codé est qui permet d'obtenir les résultats ne correspond pas parfaitement à ce qui a été décrit ci-dessus (dû à une incompréhension quand j'ai réalisé le code...). En effet je ne choisis pas aléatoirement le point p parmi les points vérifiant 2. Ils sont choisi comme présenté ci-dessous en prenant le dernier point vérifiant 2:

```
1               if Dist_patch_med < Min_dist_patch*(1+epsilon)
2                   good_a=a;
3                   good_b=b;
4                   Min_dist_patch = Dist_patch_med;
```

Listing 1: Critère de choix du pixel p

Ce qui renvoi un point de l'ensemble mais il n'est pas choisi aléatoirement ce qui peut réduire l'effet aléatoire de l'algorithme.

3 Expérimentations et variation de paramètres

3.1 Variation taille du patch

Dans cette partie il est étudié l'influence de la taille de patch sur l'algorithme Efros Leung. Il est remarquable que les résultats changent beaucoup selon le type de texture.

Texture stochastique :

Ainsi d'après le résultat des figures 5 et 5 il est observable que les résultats ne varient que très légèrement selon la grandeur du patch. Cela est du au fait que les textures stochastiques possèdent des pixels très peu corrélé entre eux. Ainsi un petit patch est suffisant à avoir de bonne génération de texture.

(a) (b) (c) (d) (e)

Figure 5: Affichage de la texture générée en fonction de la taille du patch qui sont de la gauche à la droite 3x3, 5x5, 9x9, 15x15 et 23x23 avec epsilon = 0

(a) (b) (c) (d) (e)

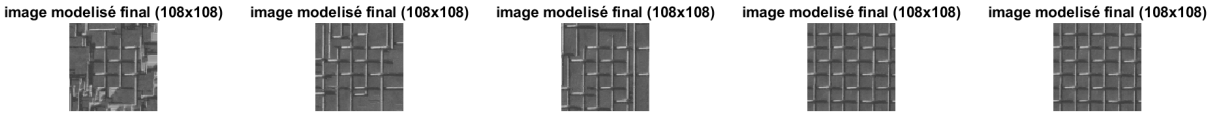
Figure 6: Affichage de la texture générée en fonction de la taille du patch qui sont de la gauche à la droite 3x3, 5x5, 9x9, 15x15 et 23x23 avec epsilon = 0

Texture régulière :

Pour les textures régulière (fig 7 et 8) c'est le contraire, l'influence de la taille du patch influe très fortement sur le résultat. Cela est dû au fait que les textures régulières possèdent des pixels très corrélé entre eux (répétition de motif). Ainsi avec un patch petit la génération de la texture est faussée du fait que le patch n'a pas toute l'information du motif de la texture.

(a) (b) (c) (d) (e)

Figure 7: Affichage de la texture générée en fonction de la taille du patch qui sont de la gauche à la droite 3x3, 5x5, 9x9, 15x15 et 23x23 avec epsilon = 0

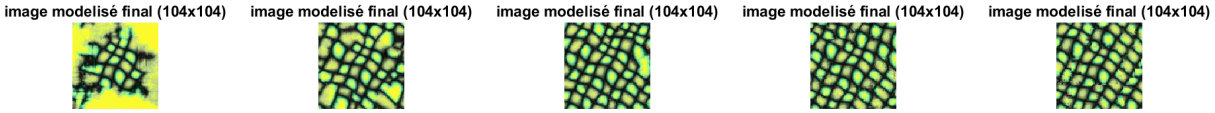


(a) (b) (c) (d) (e)

Figure 8: Affichage de la texture générée en fonction de la taille du patch qui sont de la gauche à la droite 3x3, 5x5, 9x9, 15x15 et 23x23 avec epsilon = 0

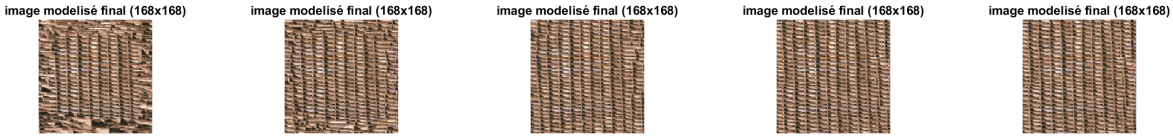
Texture irrégulière :

Dans le cas des autres textures l'impact de la taille du patch est important pour les mêmes raisons que la texture régulière. En effet les textures pseudo-régulière, irrégulière et pseudo-stochastique possèdes des pixels qui sont corrélé entre eux ce qui implique d'avoir un patch assez grand pour pouvoir reproduire les propriétés de la texture.



(a) (b) (c) (d) (e)

Figure 9: Affichage de la texture générée en fonction de la taille du patch qui sont de la gauche à la droite 3x3, 5x5, 9x9, 15x15 et 23x23 avec epsilon = 0



(a) (b) (c) (d) (e)

Figure 10: Affichage de la texture générée en fonction de la taille du patch qui sont de la gauche à la droite 3x3, 5x5, 9x9, 15x15 et 23x23 avec epsilon = 0

3.2 Variation epsilon

Cette partie a pour but d'étudier l'influence du paramètre ϵ. Ce paramètre de déterminer la marge d'acceptation d'un pixel candidat q. En effet plus ϵ est élevé plus le nombre de pixels q candidat est élevés et plus la texture résultante a des chances d'être différente de la texture d'origine. En effet avec $\epsilon = 0$ la texture a tendance à se répéter parfaitement et à toujours choisir le même pixel selon le voisinage. Ce paramètre permet d'insérer des propriétés aléatoires sur la texture résultante.

Texture stochastique :

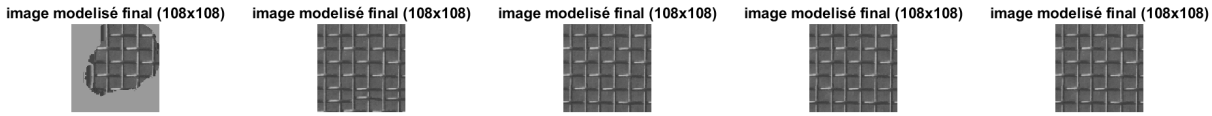
Pour les textures stochastiques, du fait de leur caractères aléatoires l'influence du paramètre ϵ n'influence pas sur le résultat. L'influence du paramètre que l'on observe sur la figure 11 vient du fait que le choix du pixel q n'est pas aléatoire dans mon implémentations de l'algorithme.

(a) (b) (c) (d) (e)

Figure 11: Affichage de la texture générée en fonction du paramètre epsilon sont de la gauche à la droite 0.5, 0.2, 0.1, 0.05 et 0 avec la taille du patch à 15x15

Texture régulière :

Pour les textures régulières, le paramètre ϵ n'améliore pas le résultat. En effet ajouter un paramètre aléatoire à des textures régulière rend le résultat incohérent (en tout cas pour mon implémentation)

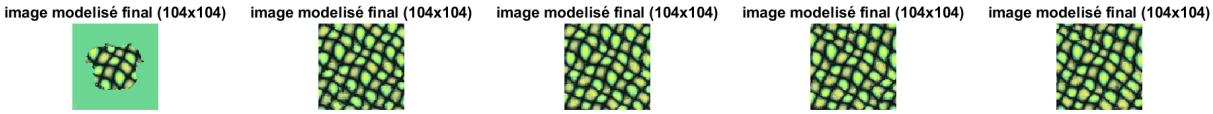


(a) (b) (c) (d) (e)

Figure 12: Affichage de la texture générée en fonction du paramètre epsilon sont de la gauche à la droite 0.5, 0.2, 0.1, 0.05 et 0 avec la taille du patch à 15x15

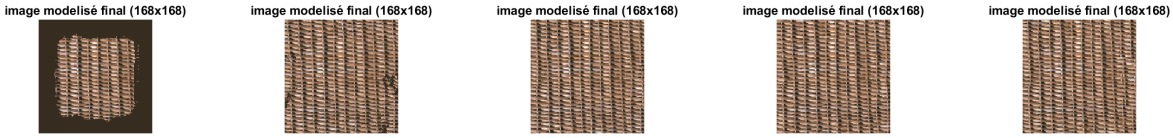
Texture irrégulière :

Pour les textures régulières, le paramètre ϵ prend tout son sens car il permet d'ajouter des irrégularités dans la texture pour pouvoir rendre inhomogène la texture et enlever l'effet de « copier-coller ».



(a) (b) (c) (d) (e)

Figure 13: Affichage de la texture générée en fonction du paramètre epsilon sont de la gauche à la droite 0.5, 0.2, 0.1, 0.05 et 0 avec la taille du patch à 15x15



(a) (b) (c) (d) (e)

Figure 14: Affichage de la texture générée en fonction du paramètre epsilon sont de la gauche à la droite 0.5, 0.2, 0.1, 0.05 et 0 avec la taille du patch à 15x15

4 Amélioration

Paramétrage :

Une façon d'améliorer la vitesse de l'algorithme est de choisir les paramètres de l'algorithme pour caractériser la texture décrite. En effet par exemple pour la texture brique il serait intéressant de choisir un patch rectangulaire pour pouvoir mieux décrire les informations du motif de la brique.

Selon les types de textures d'autres méthodes peuvent être un meilleur choix que cet algorithme. En effet dans le cas de textures régulières il serait beaucoup plus efficace et plus fiable de réaliser un assemblage naïf de

modèle. Pour les textures stochastiques il serait plus efficace de directement générer la texture à l'aide d'une distribution direct qui pourrait être facilement obtenu à partir de la texture de référence (histogramme). Sinon un petit patch donnera d'aussi bon résultat que des grands patchs.

La force de l'algorithme vient pour les autres textures ou les méthodes naïves ne fonctionne pas. De plus l'ajout du caractère aléatoire permet de contrôler le résultat final et ainsi pouvoir générer différentes textures selon les propriétés voulues.

Optimisation :

Un des avantages de cet algorithme qui peut sembler simple est qu'il peut facilement être parallélisable et il est ainsi possible de gagner un temps de calcul considérable.

5 Conclusion

Ce TP a donc permis de voir une méthode de génération de texture avec l'algorithme d'Efros Leung qui permet de générer un pixel en connaissant son voisinage. Cet algorithme a pour avantage d'être simple à mettre en place cependant il a pour inconvénient d'avoir une complexité très élevée. Cet algorithme a aussi l'avantage d'avoir un bon contrôle sur la texture généré de par le choix de la taille du patch de voisinage ou par le caractère aléatoire du paramètre ϵ