

RAPPORT TP1

Synthèse de texture

Barros Mikel-ange

L'objectif du tp était de mettre en place l'algorithme d'Efros-Leung vu en cours. Cet algorithme a pour but de générer des textures de taille variable à partir d'une image de base.

Cet algorithme se découpe en plusieurs étapes.

- Découper l'image d'entrée en N patch de taille $n \times n$
- Choisir un patch aléatoire parmi les N patch et le coller sur l'image de sortie
- Parcourir l'image de sortie et prendre le pixel ayant le plus de voisins remplis et n'étant pas lui-même rempli comme pixel à remplir
- Découper un patch $n \times n$ autour de ce pixel et calculer la distance entre ce patch et les N patches de l'image d'entrée
- Prendre le patch I de l'image d'entrée ayant la distance calculée la plus faible
- Regarder pour tous les patches précédents lesquels vérifient $\text{dist} < (1 + \epsilon) \times \text{dist}_{\min}$, avec dist la distance du patch en cours, epsilon une valeur faible, dist_{\min} la distance du patch I.
- Parmi les patch vérifiant cette condition, en choisir un aléatoire
- Coller le pixel au centre du patch choisi à la position choisit au départ
- Revenir à l'étape trois jusqu'à ce que l'image soit entièrement remplie

Afin d'implémenter cet algorithme, j'ai mis en place trois versions: une version octave non vectorisée, une version octave, vectorisée et une version en c++. Ces trois versions nous permettrons de comparer les temps d'exécution et la qualité des résultats. Cependant, comme l'objectif du tp était d'implémenter l'algorithme sous octave le rapport se concentrera dessus. De plus, seuls les deux versions octaves sont associées à ce rapport.

Figure 1 : les trois algorithmes ont été lancés avec les mêmes paramètres soit, epsilon =0,01 et taille patch=20

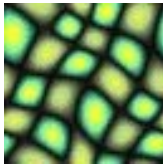
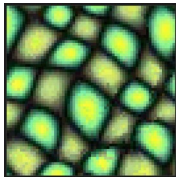
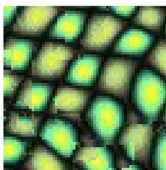
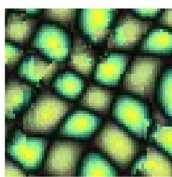
Image de base	C++	Octave	Octave (vectorisé)
			

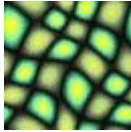
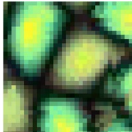
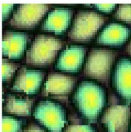
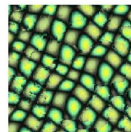








Figure 2 : les trois algorithmes ont été lancés avec les mêmes paramètres soit, epsilon =0,01 et taille patch=20

	32*32	64*64	128*128
Octave (Implémentation 1)	1minutes30	13minutes	8heures
Octave (Implémentation 2)	1minutes	4minutes	20minutes
C++	4secondes	12secondes	40secondes

Comme on peut le voir sur la figure 1 , la version vectorisée de l'algorithme renvoie de moins bons résultats que la version non vectorisée, ce sera donc celle que nous utiliserons pour la suite du rapport. Cependant, la figure 2 nous montre que la version non vectorisée est très longue et nous nous limiterons donc à générer des textures de taille 128*128.

Dans la figure suivante, nous nous intéressons à la génération de texture de différentes tailles afin de voir si l'algorithme fonctionne quel que soit la taille du patch choisit.

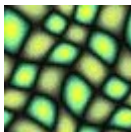
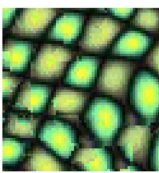
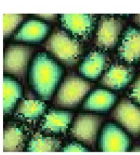
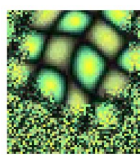
Figure 3 : epsilon =0,01 et taille patch=20

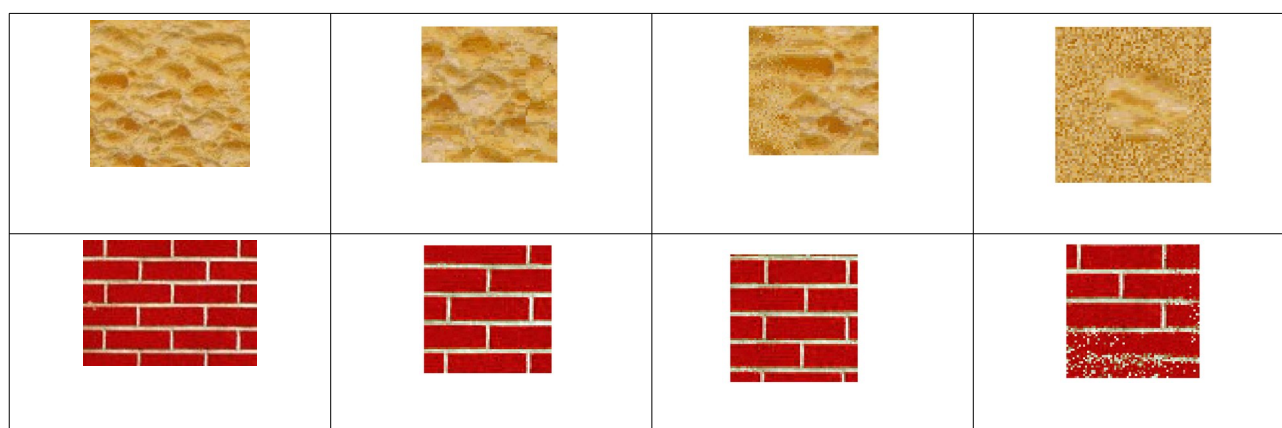
Image de base	32*32	64*64	128*128
			
			
			

Nous pouvons remarquer dans la figure 3, et surtout sur la texture 1, de nombreux artefacts. On peut supposer que cela vient de la taille des patches et du epsilon. En effet, en fonction de ces deux paramètres, nous acceptons plus ou moins de patch dans la dernière partie de notre algorithme. Pour voir si le problème vient de la, nous allons donc faire varier ces deux paramètres et commenter les résultats.

Tout d'abord intéressons-nous au epsilon, ce dernier va déterminer l'erreur acceptable entre 2 de nos patches et donc quels seront les patches que nous garderons ou non.

Figure 4 : la taille du patch est de 20 et la taille de l'image de sortie en 64*64

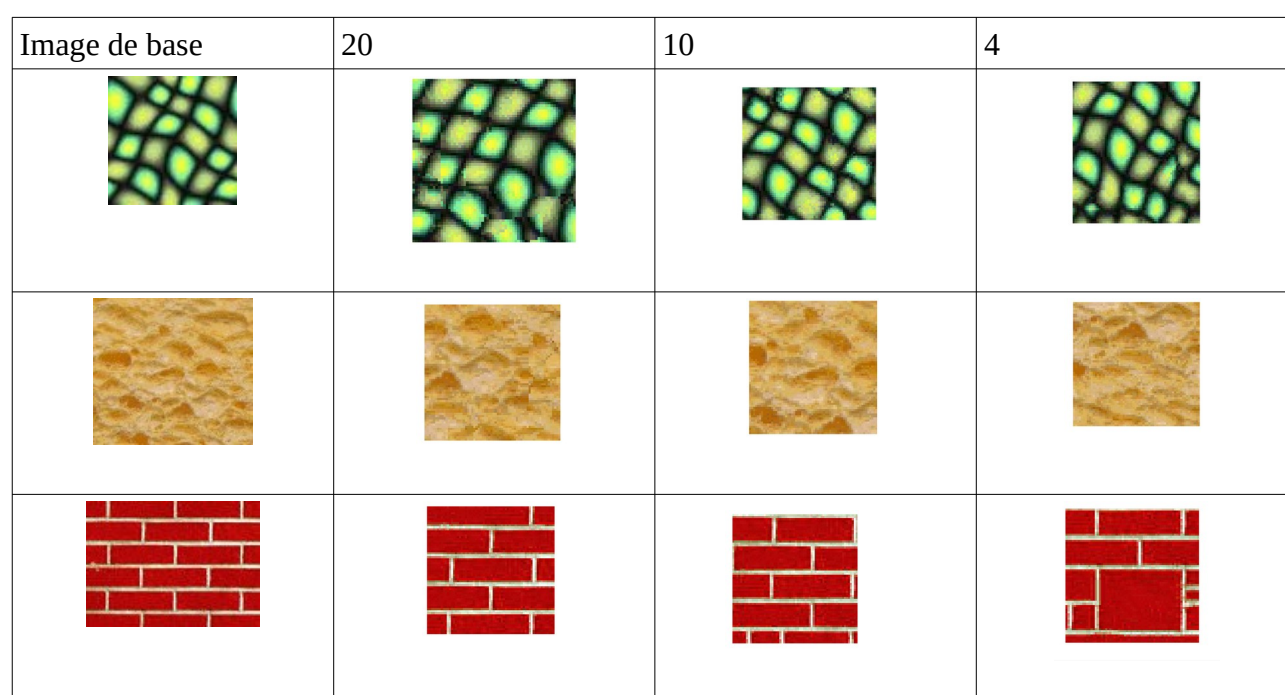
Image de base	0,01	0,1	0,5
			



On peut remarquer que plus epsilon est grand plus l'image contiendra de bruit. Cela semble logique, car epsilon représente le taux d'erreur acceptable pour choisir un patch, plus epsilon est grand donc plus on accepte des patches et plus on aura de chance d'obtenir des patches faux dans la génération de textures.

Intéressons-nous maintenant à l'influence de la taille des patches. Pour cela, nous allons faire comme précédemment avec le epsilon et regarder les résultats obtenus en fonction de ce paramètres.

Figure 5: le epsilon est 0,01 et la taille de l'image de sortie en 64*64



On peut donc remarquer 2 choses importantes:

Si la taille du patch est trop petite, la forme de la texture n'est plus respectée et on obtiendra une texture peu régulière.

Si la taille du patch est trop grande, on a des artefacts qui peuvent apparaître sur l'image.

Ces deux phénomènes peuvent s'expliquer facilement, en effet la taille du patch à une influence direct sur la distance entre deux patch et donc plus le patch est grand plus il est probable que l'on

accepte des patches ayant une distance proche, mais tout de même peu ressemblant au patch recherché. De même, pour les patches de petites tailles, on aura beaucoup de patches accepté et ce même si ils sont bien différents (tant que la partie comparée est ressemblante sur les deux patches) et à cause de l'aléatoire, nous ne pouvons être certain de garder la régularité de la texture sur de petits patches.

Ainsi, l'algorithme utilisé est efficace du moment que l'on arrive à définir un epsilon et une taille de patch correcte. Cependant, comme vu dans la première partie, il peut être très coûteux en tant et ce plus la texture à générer est grosse. Cela peut poser des problèmes, pour la génération de grosses textures ou dans un contexte professionnel. Et ce d'autant plus que l'algorithme ne peut pas être parallélisé, car il dépend des états précédents.