

Rapport de projet 2

Génération d'une image avec des critères simple et sécurité de transfère des images

Table des matières

1 Génération d'une image mosaïque avec critère simple	2
1.1 Base d'images	2
1.1.1 Choix d'une base d'images	2
1.1.2 Formatage de la base d'images	2
1.2 Génération d'une image mosaïque	3
1.2.1 Découpe de l'image en bloc	3
1.2.2 Parcours et calcul des critères dans la base	3
1.2.3 Génération de l'image mosaïque	3
1.2.4 Génération d'une image mosaïque de haute qualité	5
2 Sécurité de transimisson des images	6
2.1 Chiffrement de Vernam	6
2.2 Methodes des 3 passes	6

1 Génération d'une image mosaïque avec critère simple

Introduction

Dans le cadre du projet de programmation de l'UE HAI809I, nous devons générer des images mosaïques en appliquant des critères avancés. Cette partie présente les avancées réalisées dans ce domaine durant la première semaine de mars.

Un critère de bloc est une mesure permettant de caractériser l'ensemble des pixels qu'il contient, par exemple la moyenne ou la médiane de leurs valeurs.

Afin de générer une image mosaïque, nous pouvons suivre le TP8 disponible sur ce site : https://www.lirmm.fr/~wpuech/enseignement/master_informatique/Analyse_Traitement_Image/TP/TP8/TP8_images_Master1.pdf

1.1 Base d'images

1.1.1 Choix d'une base d'images

La première étape consiste à trouver une base d'images contenant plusieurs milliers d'éléments (environ dix mille au minimum).

Nous avons retenu une base d'images disponible sur le site de Stanford : <http://vision.stanford.edu/aditya86/ImageNetDogs/>.

Cette base contient 20 705 images de différentes races de chiens. En plus de contenir un grand nombre d'images, celles-ci sont au format jpg, en couleur, et de tailles différentes (bien que n'excédant jamais 1000×1000 pixels). Ces caractéristiques nous permettent d'utiliser cette base aussi bien pour la génération d'image en pgm, qu'en ppm et pour des imagettes de différentes tailles.

1.1.2 Formatage de la base d'images

Afin de pouvoir utiliser cette base d'images dans le cadre de notre travail, nous devons y apporter quelques modifications. En effet, nous travaillons sur des images au format PPM ou PGM, il faut donc convertir toutes les images de la base. On en profite également pour modifier la taille des images (avec un format choisi au préalable) dans le but qu'elles aient toutes la même taille.

La méthode qui s'occupe de ce travail est `convertir_pgm()`, elle utilise la librairie openCV afin de traiter les images.

1.2 Génération d'une image mosaïque

1.2.1 Découpe de l'image en bloc

La première étape du traitement est donc de découper notre image en bloc. C'est ici que le critère entre en jeu : nous devons choisir une méthode pour uniformiser la couleur des pixels de chaque bloc. L'approche la plus simple est de le faire par moyenne. Nous avons donc créé une méthode `découpe()` qui prend en entrée une image PGM et renvoie une image PGM de même taille, mais avec des blocs de couleur uniforme :

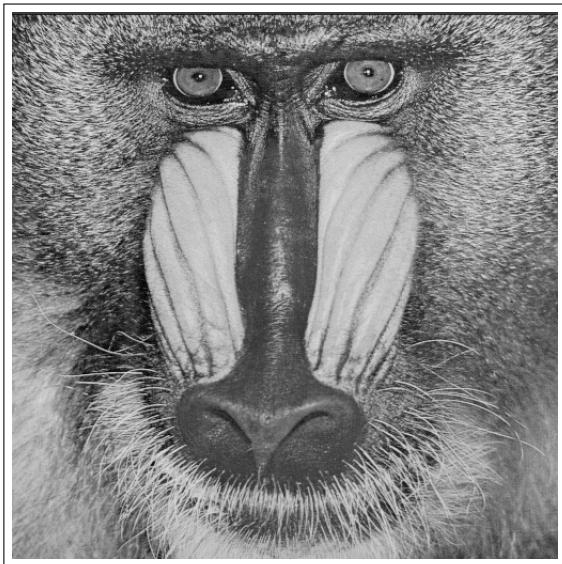


FIGURE 1 – Baboon.pgm originale (taille 512x512)

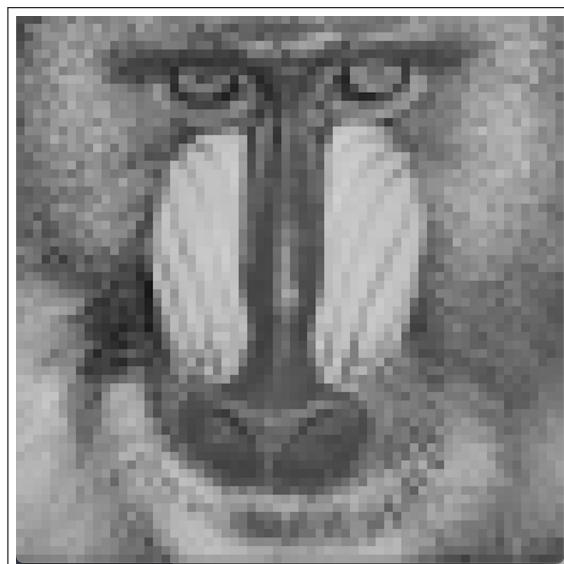


FIGURE 2 – Baboon.pgm découpée par bloc de taille 8x8

Il est possible de choisir n'importe quelle taille de bloc, il est préférable qu'elle soit un multiple de 2. Plus la taille d'un bloc est grande, plus on perd en qualité de l'image originale. Un bloc de petite taille entraînera l'effet inverse.

1.2.2 Parcours et calcul des critères dans la base

Il faut effectuer une opération similaire dans la base d'images. Il faut cette fois-ci calculer le critère, mais sur l'ensemble de l'image. Pour chaque bloc, il faut choisir l'image de la base dont le critère minimise au mieux la distance avec celui du bloc. Afin de ne pas avoir à régénérer ces données, nous les enregistrons dans un fichier annexe.

1.2.3 Génération de l'image mosaïque

Nous utilisons ensuite les données générées par la fonction précédente pour créer l'image mosaïque finale.

Pour chacun des blocs, nous récupérons la valeur du bloc, et pour chacune des images, nous

regardons quel critère est optimal. Une fois trouvée, nous utilisons à nouveau openCV afin de redimensionner l'image à la taille du bloc, puis nous l'écrivons à la place du bloc.

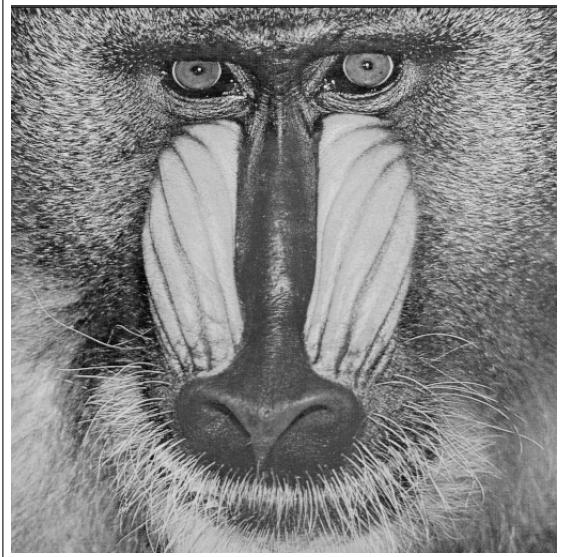


FIGURE 3 – Baboon.pgm originale (taille 512x512)

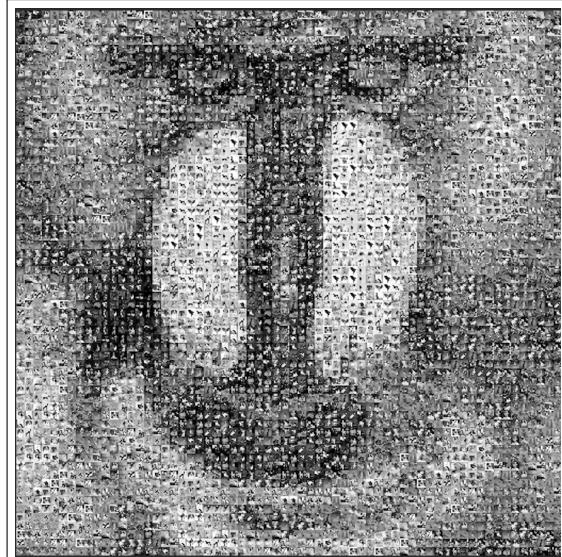


FIGURE 4 – Mosaique de Baboon.pgm bloc de taille 8x8

Le résultat obtenu est une image dont les contours rappellent l'originale, mais lorsqu'on zoomé, on voit qu'elle est composée de nombreuses petites imagettes. Contrairement à précédemment, un bloc de grande taille augmente la qualité des imagettes. Ici, un bloc de taille 8 rend nos imagette indéchiffrables.

1.2.4 Génération d'une image mosaïque de haute qualité

Afin d'assembler le meilleur des deux mondes, nous avons modifié le code précédent afin de faire en sorte d'écrire une image de sortie qui a la taille du nombre de bloc * la taille d'une imagette. Cette méthode nous permet d'obtenir une image de sortie dont les contours sont similaires à l'image mosaïque précédente, mais qui permet également d'observer les imagettes en zoomant sur l'image :



FIGURE 5 – Mosaique de Baboon.pgm haute définition (taille 8192x8192)



FIGURE 6 – Zoom sur la Mosaique de Baboon.pgm de haute définition

Cette version haute définition présente une principale qualité : nous pouvons diminuer la taille des blocs tout en gardant des imagettes de bonne qualité. Cependant, il y a un défaut conséquent : la taille. L'image mosaïque générée est d'une taille démesurée par rapport à l'image originale, si nous prenons l'exemple de `baboon.pgm` :

Titre de l'image	Taille de l'image	Taille en octets
Baboon.pgm	512x512	262 Ko
Baboon_mosaïque.pgm	512x512	262 Ko
Baboon_mosaïque_HD.pgm	8192x8192	67,1 Mo

TABLE 1 – Tableau des images et leurs tailles

2 Sécurité de transimisson des images

Introduction

L'un de nos objectifs sur ce projet est de proposer à l'exemple de certaines banques d'images ou moteurs de recherches, une façon protéger la propriété intellectuelle d'une image tout en en proposant un aperçu accessible.

2.1 Chiffrement de Vernam

Le chiffrement à masque jetable, dit "Chiffre de Vernam", est un schéma de chiffrement s'appuyant sur l'opérateur logique `XOR`. En effet, une clé binaire `key`, de la taille du message à chiffrer est générée aléatoirement, dans le cas d'une image, cette clé est de taille `sizeof(octet) * imgSize`. L'image chiffré `cipherImg`, est alors calculé en faisant `key XOR imgIn`. Puis l'image décodée `imgOut` est calculée en faisant `key XOR cipherImg`.

2.2 Méthodes des 3 passes

Dans le cadre de ce projet, nous serons amenés à utiliser ce chiffrement afin de transférer nos images en toute sécurité sur le réseau. Cependant, le chiffrement de Vernam étant un chiffrement symétrique un problème reste en suspens, celui du partage de la clé privée servant à encodé et décodé l'image. Une solution trouvée et mise en place après multiple recherche a été la Méthode 3 passes :

Prenons Alice et Bob respectivement expéditrice et destinataire de l'image. Alice génère une clé `alicKey`, et chiffre l'image avec celle-ci. Alice transfère l'image résultante `cipherImg1` à Bob. Bob génère sa propre clé `bobKey`, puis chiffre `cipherImg1` avec celle-ci et transfère l'image résultante `cipherImg2` à Alice. Alice déchiffre `cipherImg2` avec sa clé et obtient une image `cipherImg3` qu'elle transfère à Bob. Bob décode l'image finale avec sa clé.

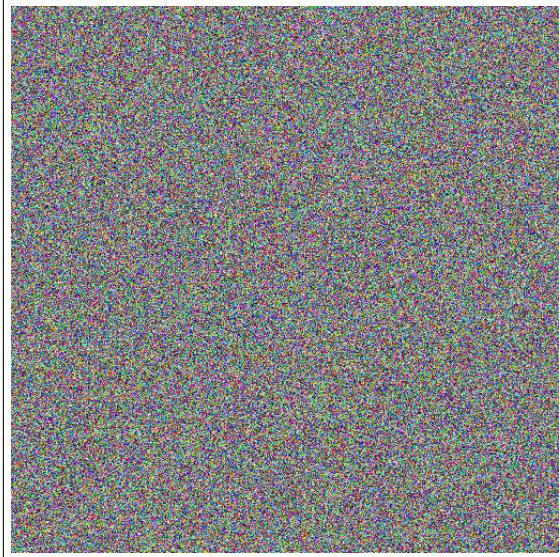


FIGURE 7 – aliceKey

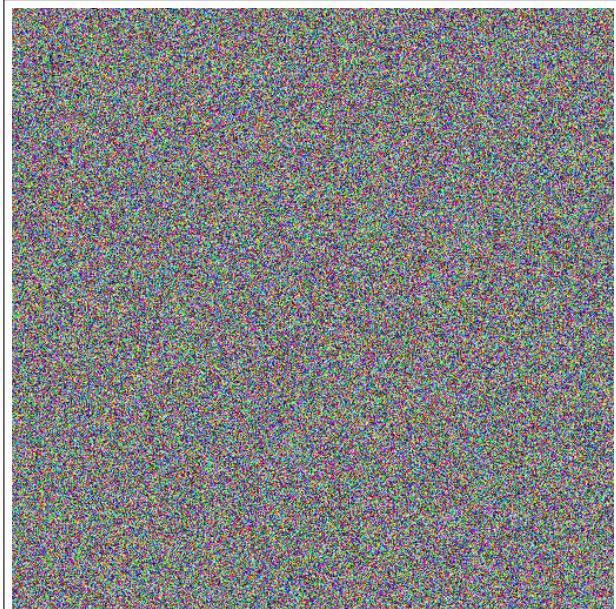


FIGURE 8 – aliceKey

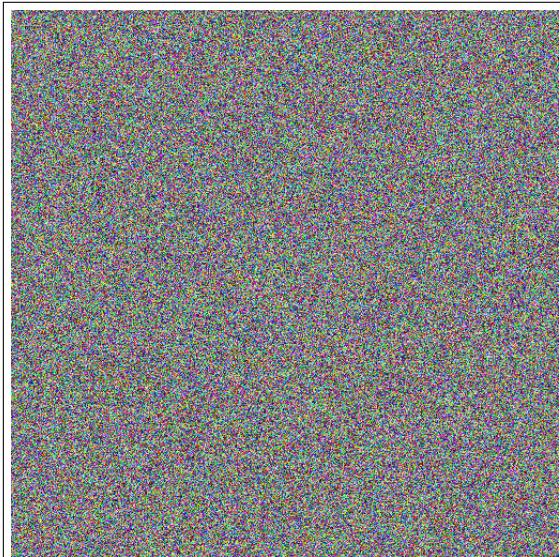


FIGURE 9 – cipherImg1

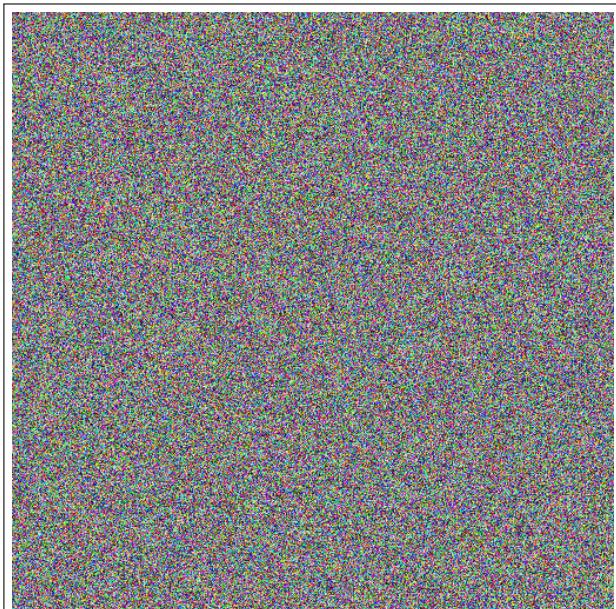


FIGURE 10 – cipherImg2

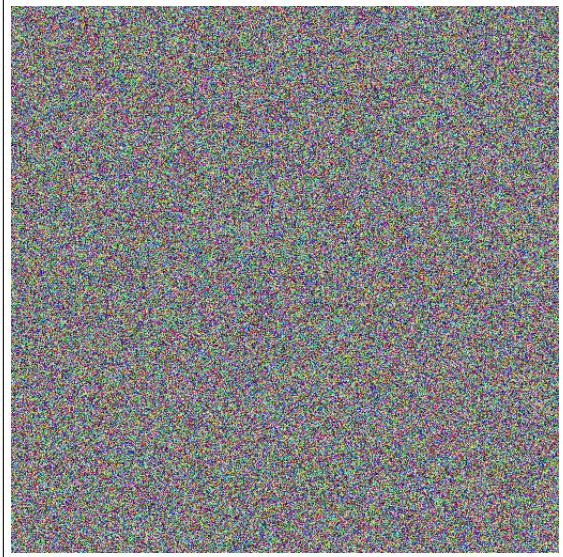


FIGURE 11 – cipherImg3

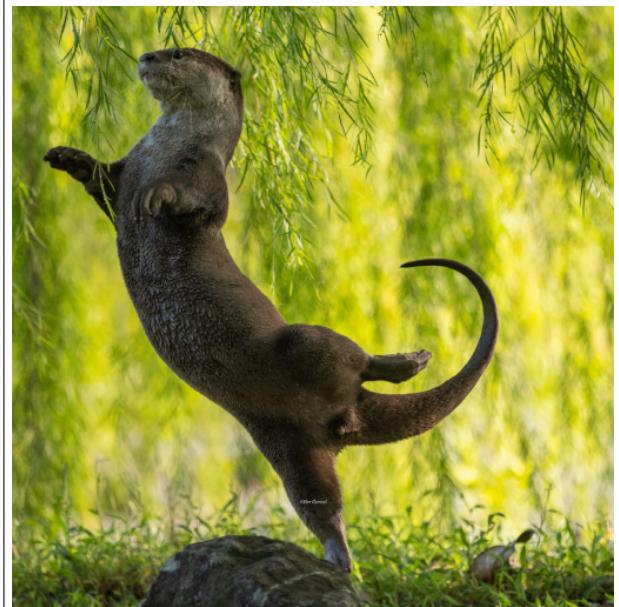


FIGURE 12 – decodeImg