Master 1: CRYPTIS INFORMATIQUE

Algorithme et programmation avancée

Rapport: Textures

Réalisé par:

DRIDI Thomas
GNANZOU Abrahame (M2)
SYLLA Ousmane

Année universitaire 2019-2020

1 Introduction

Le projet consiste à remplir une surface carrée avec un échantillon d'image donné. Le remplissage doit se faire de sorte à minimiser le critère d'erreur entre les blocs en utilisant la programmation dynamique.

2 Permuteur

La classe **Permuteur** permet de générer une permutation aléatoire à l'aide de la fonction **random_shuffle** de la bibliothèque standard. On fait appel à la fonction **suivant()** afin de renvoyer l'élément suivant dans la permutation en cours. Une fois tous les éléments de la permutation renvoyés, on génère une nouvelle permutation. À noter qu'au $i_{ème}$ appel, la fonction **suivant()** renvoie l'entier à la i-$1_{ème}$ position dans la permutation.
Le code source des fonctions de la classe sont dans le fichier **permuteur.cpp**.

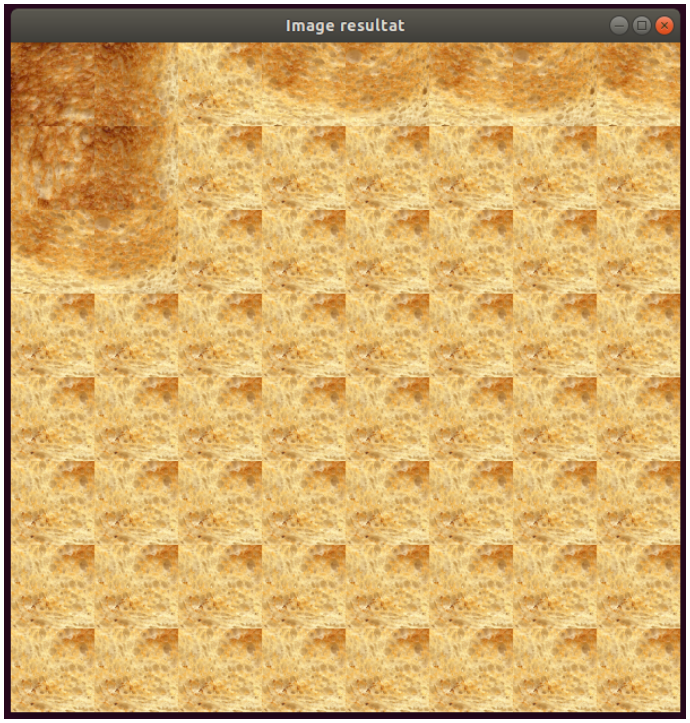

Figure 1: "toast.tif" avec permuteur

3 Raccordeur_recursif

Cette classe hérite de la classe **Raccordeur** et permet de calculer le raccord optimal entre deux blocs et le coût associé à ce raccord.

3.1 Théorie

La fonction **Calculer_Raccord** de la classe **Raccordeur_recursif** doit dans un premier temps calculer de façon récursive les éléments de la coupe optimale entre deux blocs avant de remplir le tableau coupe avec les indices des éléments qui minimisent le critère d'erreur. Le critère d'erreur se calcule avec la formule suivante:

$$E = \sum_{i,j} ||B^2_{i,j} - B^1_{i,j}|| = \sum_{i,j} e_{i,j}$$

Les $e_{i,j}$ sont définies comme la norme 2 des écarts sur les canaux RGB des pixels des deux blocs. Donc les $e_{i,j}$ représente la différence de couleur entre les pixels de deux blocs, plus e est petit et plus la couleur des pixels est proche. E représente la différence totale de couleur entre deux blocs c'est-à-dire le coût du recouvrement.
On utilise la formule suivante afin de déterminer la coupe optimale :

$$E_{i,j} = e_{i,j} + min(E_{i-1,,j-1}, E_{i-1,,j}, E_{i-1,j+1})$$

Nous allons voir sur l'exemple ci-dessous comment ce calcul s'effectue sur une matrice donnée.

e= | | | | |
|---|---|---|---|
| 1 | 3 | 2 | 1 |
| 2 | 1 | 2 | 3 |
| 1 | 3 | 4 | 2 |
| 2 | 4 | 3 | 1 |
| 4 | 3 | 1 | 2 |

donne donc $(E_{i,j})$=

1	3	2	1
3	2	3	4
3	5	6	5
5	7	8	6
9	8	7	8

Ici, le minimum sur la dernière ligne se trouve en 3e colonne, on remonte donc la coupe optimale à partir de cette position. Ainsi, le tableau coupe contiendra les positions {4;3;4;4;3}.
Nous pouvons déduire, de cette formule de calcul de coupe optimale, la récurrence sur la complexité.

$$T(n,m) = \begin{cases} \Theta(1) & \text{si } n = 1 \\ \Theta(1) + T(n-1,m) & \text{si } n > 1 \text{ et } m = 1 \\ \Theta(1) + T(n-1,m-1) + T(n-1,m) & \text{si } n > 1 \text{ et } m = 2 \\ \Theta(1) + T(n-1,m-2) + T(n-1,m-1) + T(n-1,m) & \text{sinon.} \end{cases}$$

On en déduit :

$$T(n,m) \geq 3T(n-1,m)$$

d'où

$$T(n,m) \geq 3^n$$

Ainsi, T(n,m) est une complexité exponentielle. En effet, en utilisant l'approche naïve, la complexité d'un problème de taille n*m est égale à 3 fois la taille du sous-problème de taille (n-1)*m

3.2 Implémentation de la solution recursive

Dans la classe **RaccordeurRecursif**, nous avons écrit une fonction appelée **coupe_recursive()** qui prend en paramètre deux entiers et implémente la formule de récurrence. Pour éviter les calculs redondants, nous avons initialisé un tableau de taille égale à la matrice donnée en paramètre. Ce tableau est rempli au fur et à mesure par les $E_{i,j}$ tout en vérifiant que le calcul n'était pas encore fait.
La fonction **coupe_recursive()** est appelée dans **CalculerRaccord**. Cette dernière permet de calculer le coût et de récupérer les indices de la coupe optimale à partir de la matrice obtenue à l'aide de la fonction **coupe_recursive()**. Les indices sont stockés dans le tableau coupe donné en paramètre de **CalculerRaccord**.
Le code source des fonctions se trouve dans le fichier **raccordeur_recursif.cpp**.

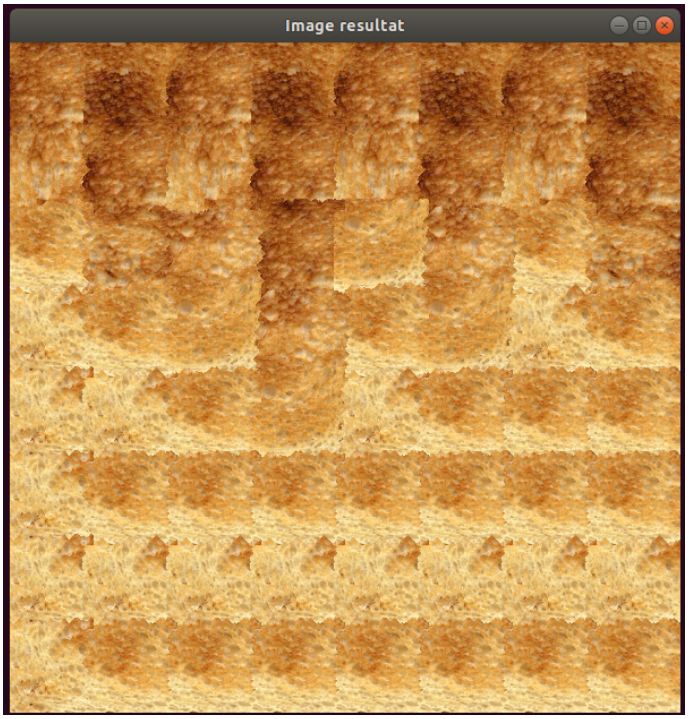

Figure 2: "toast.tif" avec raccordeur récursif

4 Raccordeur_itératif

Cette classe a un principe similaire à la classe **Raccordeur_recursif** et s'appuie sur la même partie théorique. La différence entre les deux réside dans le calcul des éléments de la coupe (c'est-à-dire les $E_{i,j}$). Contrairement au cas récursif, dans l'approche itérative on commence par recopier la première ligne de la matrice e dans la matrice $(E_{i,j})_{i,j\in N}$, puis on calcule les $E_{i,j}$ en tenant compte des effets de bords. Ensuite, on cherche le minimum sur la derniere ligne de $(E_{i,j})_{i,j\in N}$ à partir duquel on remonte dans la matrice en trouvant les autres éléments de la coupe optimale pour remplir le tableau coupe.
Le code sources des fonctions se trouve dans le fichier **raccordeur_iteratif.cpp**.

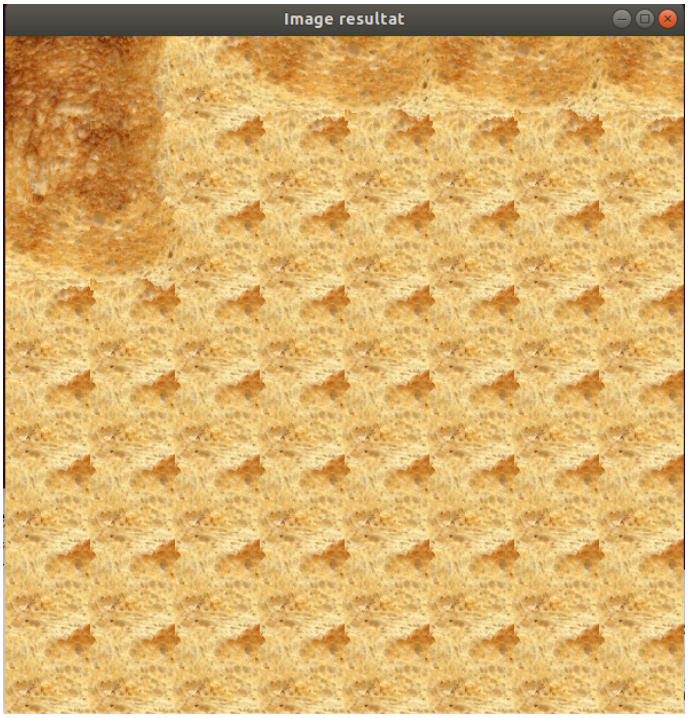

Figure 3: "toast.tif" avec raccordeur itératif

5 Temps de calcul

Pour comparer la performance des approches récursive et itérative, nous avons lancé les deux algorithmes avec la commande time de linux et avons obtenu les resultats suivants:
Le temps d'exécution du **programme Récursif** est de **0,260s**.

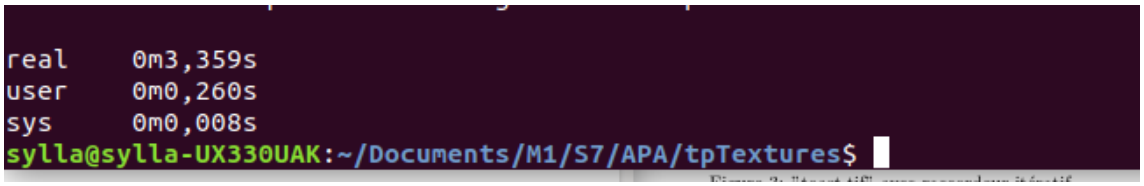

Figure 4: Temps d'exécution de l'algorithme récursif

Celui du **programme Itératif** est de **0,284s**.

Figure 5: Temps d'exécution de l'algorithme itératif

À noter que les deux programmes ont tout deux été testés sur l'image **toast.tif**. Nous constatons que l'approche récursive est légèrement meilleure en terme de performance que l'approche itérative.