



ÉCOLE  
**CENTRALE** LYON

TC S6 INF tc2

Compte Rendu

---

## Compression d'image

---

*Auteurs :*

M. Antoine ZURCHER  
M. Thibault TISCHHAUSER

*Encadrants :*

M. Alexandre SAIDI  
M. Mohsen ARDABILIAN

Version du  
14 mai 2021

# **Sommaire**

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Structure d'arbre explicite</b>	<b>2</b>
<b>3</b>	<b>Structure d'arbre implicite</b>	<b>2</b>
<b>4</b>	<b>Coût temporel et spatial</b>	<b>3</b>
<b>5</b>	<b>Nouveau critère de compréhension</b>	<b>4</b>

## 1 Introduction

Ce document présente le compte rendu du TD N°3 d'algorithmes et structures de données sur le sujet de la compression d'image. La compression d'image à pour but de réduire la redondance de donnée d'une image afin d'en réduire son poids.

## 2 Structure d'arbre explicite

En première approche la compression de l'image se fera à l'aide d'un arbre explicite : chaque noeud de l'arbre (classe arbre dans le code python) correspond à un carré, dont la couleur est la couleur moyenne de ce carré sur l'image. Si cette couleur est homogène alors c'est un noeud final, sinon le noeud possède 4 fils qui correspondent au carré du père divisé en 4 carrés. Afin de déterminer si un noeud est homogène ou non nous utilisons l'écart type des couleurs et le comparons à un seuil limite, nous reviendrons plus tard dans ce rapport sur le sujet de ce critère.

L'implementation de cette méthode dans python correspond à la fonction arbre qui permet de construire l'arbre, et à peindre\_arbre permettant d'afficher l'image compressée.

## 3 Structure d'arbre implicite

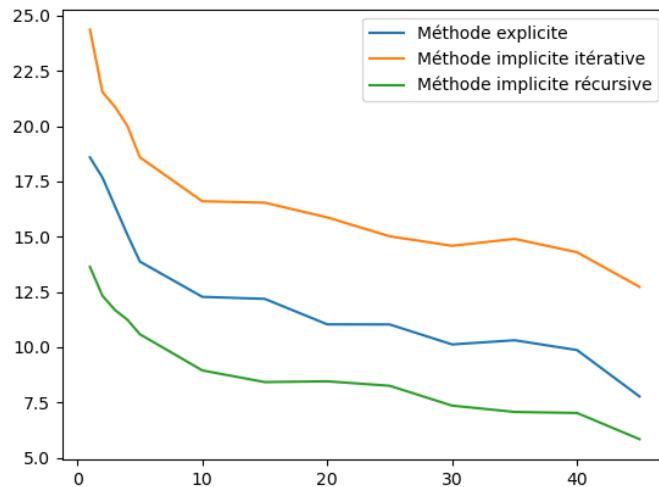
Pour réduire la place prise par la structure mise en place précédemment, on ne crèera qu'une seule structure avec les noeuds placé dans une liste. Ainsi nous utilisons une nouvelle classe de noeud : Noeud2, qui ne contient plus les informations sur ses fils. Ainsi pour un noeud en position  $i$  dans la liste, ses enfants seront placés dans la liste aux indices  $4i, 4i + 1, 4i + 2, 4i + 3$  et son parent se trouve à l'indice :  $\text{int}(i // 4)$

L'implementation de cette nouvelle construction d'arbre à été réalisée de 2 manière différentes :

La première construit la liste de manière itérative et est implémenté dans le code sous le nom : arbre2 et peindre\_arbre2 La seconde construit la liste de manière récursive et est implémenté dans le code sous le nom : arbre3 et peindre\_arbre3

## 4 Coût temporel et spatial

Pour pouvoir récupérer le temps d'exécution d'une fonction, il suffit d'utiliser le package Time et de récupérer le temps avant et après l'appel de la fonction avec la méthode `time()`. Ainsi on obtient la figure 1

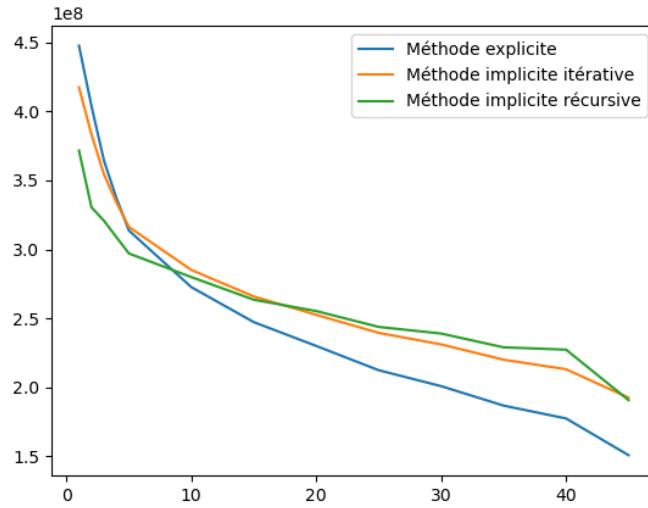


**FIGURE 1 –** Temps nécessaire en fonction de la valeur du seuil

Sur cette figure, on remarque en bleu le temps d'exécution de la compression normale qui est maximal pour un petit seuil et minimale si la compression détruit l'image.

La deuxième méthode utilise une méthode non récursive et prend bien plus de temps. Quand à la troisième méthode, elle est plus rapide que les autres.

De plus on peut regarder la prise de place mémoire avec les modules `os` et `psutil` et la commande `:psutil.Process(os.getpid())`, le résultat se retrouve dans la figure 2

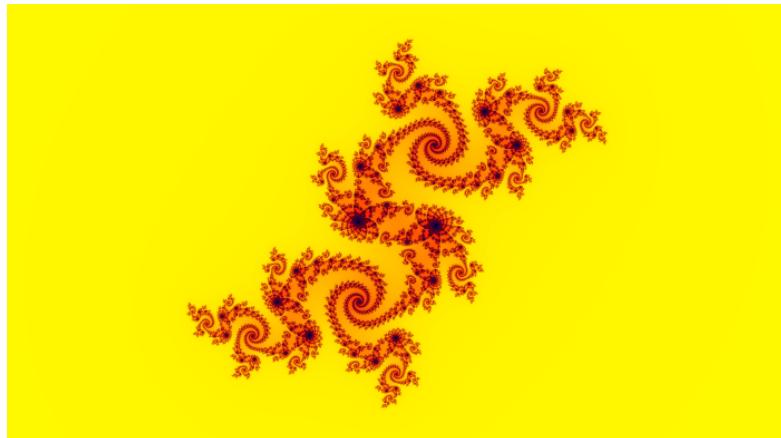


**FIGURE 2** – mémoire nécessaire à la compression d'une image en fonction de la valeur de seuil

On remarque alors que la première méthode utilise légèrement moins de mémoire que les deux autres pour des grande valeurs. On remarque aussi que les deux autres méthodes sont moins efficace pour l'utilisation de la mémoire.

## 5 Nouveau critère de compréhension

Deux nouveaux critères ont été implémentés pour déterminer si un noeud est homogène ou non. Le premier considère de laisser une plus grande netteté sur le rouge et le second est de laisser une plus grande netteté au centre. Nous allons voir le résultats de ces différents critères pour l'image de base suivante :



**FIGURE 3 – Ensemble de Julia**

Les différents tests ont été réalisés sur cette image (au format png) mais aussi sur l'Image8 donnée en exemple afin de mieux percevoir les effets des différentes méthodes.

Compression de base :



**FIGURE 4 – Homogénéité de base**

La compression à coupé le bout en bas à gauche de la fractale, cependant malgré cela il est difficile de faire la différence entre les deux images sans zoomer.

---

De plus l'image de base fait 121 Ko et l'image compressée en fait 98 Ko.

Compression plus nette sur le rouge :



**FIGURE 5 –** Homogénéité rouge

Le résultat est meilleur que précédemment, cependant il reste des petits bouts de la fractale qui ont disparu. L'image fait maintenant 96.7 Ko.

Compression plus nette au centre :



**FIGURE 6 –** Homogénéité centre

L'image résultante fait maintenant 99.4 Ko, le résultat est bon au centre mais les bord de la fractale sont moins bon, cette compression est plus utile pour des images dont le sujet principale est au centre et où les bords sont vides.

Compression avec les deux nouveaux critères :



**FIGURE 7 –** Homogénéité rouge et centre

L'image résultante fait maintenant 98 Ko. Les bords de la fractales sont

toujours légèrement coupé par la compression, cependant le résultat au centre est très acceptable.