POLYTECH NICE-SOPHIA

Projet MAM 3 S5 : Compression d'images par transformée de Fourier

Salma Himmichi

Ghaieth Aloui

Oumaima El Hemer

Du 18 au 22 Décembre 2023

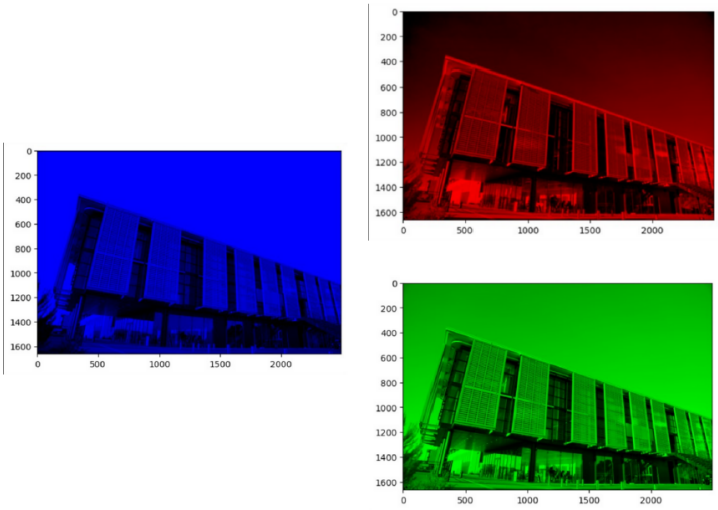
Table des matières

A Présentation du projet :

1 Introduction :

Une image numérique, représentation visuelle d'une scène ou d'un objet, est souvent exprimée en termes de pixels et de couleurs. Le modèle de couleur RGB (Rouge, Vert, Bleu) est fréquemment utilisé pour décrire la composition des couleurs dans une image, où chaque pixel est défini par trois composantes qui définissent les trois couleurs primaires : rouge, vert et bleu. La compression d'images, processus visant à réduire la taille d'un fichier image tout en préservant une qualité visuelle acceptable, est cruciale pour la gestion efficace des données visuelles.

Prenons par exemple notre école :



La compression d'images par la transformée de Fourier est un processus mathématique permettant de convertir l'information spatiale(pixel) d'une image en information fréquentielle. Souvent intégrée dans des algorithmes de compression d'images tels que le PNG, cette méthode offre la possibilité d'ajuster le niveau de compression afin d'atteindre un équilibre entre la taille du fichier résultant et la qualité visuelle de l'image compressée, cependant c'est une compression avec perte, vue que certaines informations sont perdues lors du processus.

Dans la suite, nous explorons les principes de cette méthode et élaborons l'algorithme informatique permettant l'application judicieuse de cette technique de compression d'images.

2 Méthodologie et objectifs :

2.1 Objectif :

On cherche à obtenir une image nette tout en réduisant sa taille le maximum possible.

2.2 Principe et étapes :

Le principe est comme suit :

Après la lecture de l'image, on sépare les matrices relatives au rouge, vert et bleu, puis , on discrétise l'image en plusieurs blocs de pixel 8x8, ensuite on effectue la compression en appliquant la transformée de Fourier discrète à chaque bloc de couleur.

Cette transformation, une variante de la transformée de Fourier classique, change la base de visualisation, et ainsi, on passe des blocs de pixels à des blocs de fréquences.

Cependant, ce changement de base, n'est pas encore une compression réelle et complète de l'image, il est mathématiquement représenté par :

$$D_{k,l} = \frac{1}{4} C_k C_l \sum_{i=0}^{7} \sum_{j=0}^{7} M_{ij} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cdot \cos\left(\frac{(2j+1)l\pi}{16}\right)$$

Où M est la matrice du bloc original relative à chaque couleur (RBG) et D est la matrice du bloc résultat.

Et puisqu'on ne travaille qu'avec les fonctions paires, nous n'utilisons que des cosinus et pas de sinus (DCT).

Ck et Cl, sont des coefficients de normalisation qui garantissent que la DCT soit une transformation orthogonale, (c'est-à-dire une transformation linéaire qui conserve les longueurs et les angles).

Après l'obtention de la matrice D relative à chaque couleur, la quantification est une étape cruciale nous permettant de conserver uniquement les informations les plus importantes, à savoir les basses fréquences, tandis que les hautes fréquences, représentant les bruits, seront éliminées.

C'est à ce stade que survient une perte d'information, contribuant au taux d'erreur.

On note qu'il existe différentes façons pour éliminer les hautes fréquences, à part la division par la matrice de quantification, on pourrait appliquer une fonction de filtrage de type passe-bas (il est possible par exemple de comparer par le biais d'une courbe l'influence de la valeur de f dans le filtre passe bas sur le nombre de zéro obtenu et ainsi sur le degré de compression résultant) ou encore d'autres méthodes plus avancées.

Mais, plus l'on souhaite conserver des fréquences basses, plus l'information supprimée est importante, améliorant ainsi la compression mais augmentant l'erreur.

Le taux de compression est alors calculé et affiché en fonction du nombre de fréquences/informations omises.

La procédure se poursuit par une étape de décompression de chaque bloc de couleur individuellement, qu'on recomposera par la suite pour construire l'image compressée, afin d'évaluer le pourcentage d'erreur entre l'image originale et l'image compressée.

On pourrait aussi afficher la nouvelle image, afin de comparer visuellement sa fidélité à l'image originale et pourquoi pas effectuer des zooms, pour regarder et évaluer de plus près la précision.

B Plan de travail et carnet de bord :

1 Organisation du travail :

Dans notre groupe, on a réparti le travail en deux phases majeures : Une première phase d'étude théorique et de révision des compétences nécessaires à la programmation, à savoir la syntaxe python et un peu d'algèbre linéaire, étant donné qu'on convertira chaque image à une matrice tri-dimensionnelle, dont les éléments sont les intensités des couleurs de base (RBG).

Une deuxième étape d'aspect purement pratique, qui consiste en l'implémentation du code nous permettant d'effectuer la compression de l'image et d'évaluer sa qualité en 4 étapes : Initialisation, compression, décompression et post-processing.

2 Carnet de bord

Jour 1 : (travail collectif)
- Compréhension théorique de la transformée de Fourier et sa relation avec le processus de compression.
- Compréhension de l'objectif du projet.
- Révision de la syntaxe de python.
- Participation active à la discussion sur la répartition des rôles.
- Création d'un espace de travail collectif sur Visual code avec un live Share afin d'offrir à tout le monde la possibilité de suivre et détecter/corriger les erreurs commises.
- Importation des librairies nécessaires : matplotlib. pyplot (pour le traitement d'image), Numpy (matrices) et math (fonction / calcul) et time (pour calculer le temps d'exécution).

Jour 2 :
- Développement de la première partie de l'algorithme. (Salma et Ghaeith)

- Tests unitaires pour s'assurer de la validité de l'implémentation. (Ghaeith)
- Calcul analytique de la matrice de passage. (Salma)
- Définition des fonctions à programmer pour implémenter la fonction de compression. (Salma)
-Elaboration de la fonction de décompression (Oumaima et Salma)
-Développement de la partie décompression sur python (Oumaima)
- 1ère partie du rapport. (Salma)

Jour 3 :
- Développement de la deuxième partie de l'algorithme. (Travail collectif sur Visual code studio)
- Intégration préliminaire du code et premiers tests d'intégration. (Ghaeith)
- Calcul du taux de compression et évaluation de l'erreur. (Oumaima)
-Développement de la partie post-processing (Oumaima)
-2ème partie du rapport. (Salma)

Jour 4 :
- Finalisation des tests d'intégration. (Ghaeith)
- Identification et correction des bugs. (Salma et Oumaima)
- Optimisation du code pour améliorer les performances, et diminuer la complexité temporelle d'exécution. (Tout le monde)
- Préparation des slides pour la présentation des résultats. (Salma et Oumaima)
-Finalisation du rapport. (Salma)

Jour 5 :
-Réalisation de la soutenance orale de 15 min.
-Relecture du rapport. (Salma et Oumaima)

3 Les problèmes rencontrés

Au départ, nous avons rencontré divers problèmes lors des différentes étapes du processus. Initialement, nous avons omis de recomposer l'image après la décompression, ce qui a généré un problème. De plus, la sortie présentait des restes rouges indésirables. Un autre défi était lié au temps d'exécution initial, qui était de l'ordre de 10 à 12 secondes. Nous avons progressivement ajusté plusieurs lignes du programme au fil des essais, parvenant finalement à réduire ce temps à 3 ou 4 secondes, bien que ce dernier chiffre reste élevé. C'est la meilleure performance que nous avons pu atteindre après plusieurs itérations de modifications du code.

C Explications théoriques du processus de compression

1 Initialisation

1.1 Lecture de l'image et extraction du tableau des intensités

Au début du processus d'initialisation, nous débutons en extrayant les données d'une image préalablement enregistrée, ce qui se traduit par l'obtention d'un tableau contenant les pixels de l'image à compresser.

On obtient donc une matrice dont les dimensions correspondent à la résolution de l'image, et dont les éléments sont des triplets représentant les valeurs RGB d'un pixel. Pour réaliser cette opération, nous faisons appel à la fonction imread de la bibliothèque matplotlib. pyplot.

1.2 Extraction des matrices relatives à chaque couleur

red=M[:, :, 0] , green=M[:, :,1] , blue=[:, :, 2]

1.3 Tronquage de l'image et recadrement des intensités

On ajuste les dimensions de l'image en les divisant par des multiples de 8 pour les axes x et y, en utilisant l'opérateur "//", qui réalise une division entière et arrondit au plus proche entier inférieur. La valeur obtenue est ensuite multipliée par 8 afin de garantir que les dimensions finales sont des multiples de 8. Les dimensions ainsi obtenues sont sauvegardées dans les variables "n" et "m". Par la suite, l'image est tronquée à ces dimensions en conservant uniquement ses 3 matrices couleurs.

Les intensités de l'image sont converties en entiers compris entre 0 et 255. Pour ce faire, chaque élément du tableau de pixels est multiplié par 255 afin de mettre à l'échelle les valeurs d'intensité de l'image de la plage de 0 à 1 à la plage de 0 à 255. Ensuite, le type de données est converti en entier à l'aide de la fonction "astype(int)". Et afin de ramener les valeurs d'intensité à une plage de -128 à 127, on soustrait 128 de chaque élément du tableau des pixels.

1.4 La matrice de passage Pm

Calcul de la matrice de passage

On a : $D = PM^tP$ et ona :

$$D_{k,l} = \frac{1}{4} C_k C_l \sum_{i=0}^{7} \sum_{j=0}^{7} M_{ij} \cos\left(\frac{(2i+1)k\pi}{16}\right) \cdot \cos\left(\frac{(2j+1)l\pi}{16}\right)$$

avec

$$C_k = \begin{cases} \frac{1}{\sqrt{2}} & si \quad k = 1 \\ 1 & si \quad k > 1 \end{cases}$$

d'autre part : Soit $A = PM$ alors

$$A_{k,l} = \sum_{i=1}^{n} P_{k,i} m_{i,l} \qquad (*)$$

$$D_{k,l} = \sum_{j=1}^{n} A_{k,j} P_{l,j} \qquad (**)$$

donc on remplace (*) dans (**)

$$D_{k,l} = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} P_{k,l} \cdot m_{i,j} \right) P_{l,j}$$

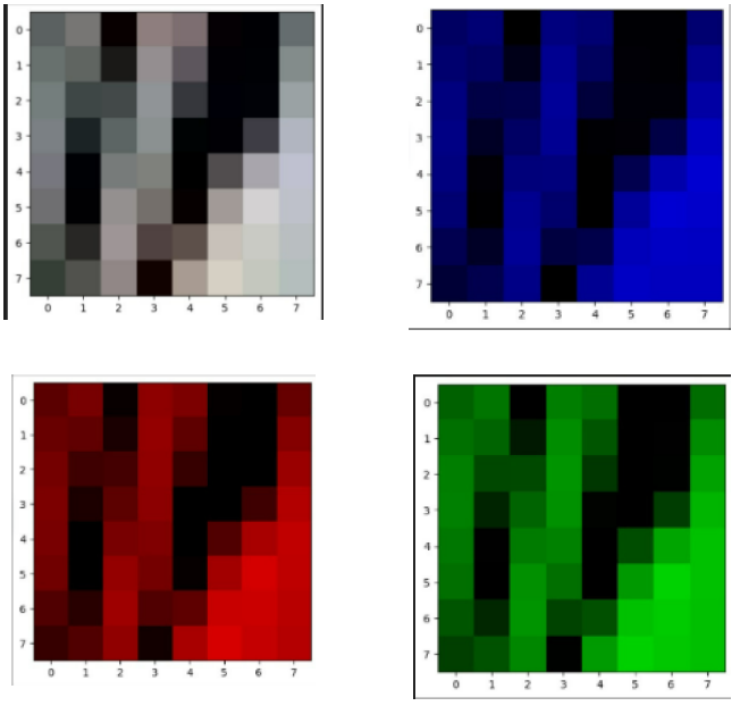
par identification

$$P_{k,i} = \frac{1}{2} C_k \cdot \cos\left(\frac{(2i+1)k\pi}{16} \right)$$

$$P_{l,j} = \frac{1}{2} C_l \cos\left(\frac{(2j+1)l\pi}{16} \right)$$

1.5 Décomposition en des blocs 8*8

Cette fonction prend en entrée une matrice M (les matrices Red, Blue et Green), ainsi que deux entiers, n représentant le nombre de lignes de la matrice M, et k représentant le nombre de colonnes de cette dernière. Elle initialise une liste vide appelée "Bloc" destinée à stocker les matrices 8x8 qui seront extraites de la matrice M. La fonction utilise ensuite une double boucle imbriquée pour itérer à travers les lignes et les colonnes de la matrice M par pas de 8. Chaque sous-matrice de taille 8x8 ainsi obtenue, est ajouté à la liste "Bloc". En conclusion, la fonction retourne la liste "Bloc" qui contient les sous-matrices de taille 8x8 extraites de la matrice M (on effectuera cette décomposition pour chaque matrice couleur extraite de la matrice originale).



2 Compression et Quantification » et/ou « Compression et Filtrage » et Recomposition de l'image

2.1 Compression et Quantification

La fonction def compression(M,Pm,Q) prend en paramètre une matrice M (Mred, Mgreen ,Mblue relative à chaque couleur) , la matrice de passage Pm et la matrice de quantification Q.

Cette fonction nous retourne le résultat de la DCT appliquée à chaque matrice de couleur, en utilisant la formule générale D=P*M*transpose(P). Elle utilise np.matmul de la bibliothèque numpy pour effectuer le produit matriciel de P et M, puis le produit par transposée de P obtenue à l'aide de np.transpose().

Ensuite il faut diviser chaque terme de la matrice D résultat du produit par la matrice de quantification Q, et astype(int) pour ne garder que la partie entière du quotient matriciel.

2.2 Compression et Filtrage

On reprend le même processus de calcul de la matrice D, mais cette fois-ci au lieu de diviser terme à terme la matrice D par la matrice de quantification Q , on effectue un filtrage passe-bas , pour ne garder que les basses fréquences . On définit donc une fonction def filtrepassebas(M,freq), qui prend en paramètre une matrice M et un entier freq , représentant la fréquence à ne pas dépasser , elle crée ensuite une matrice D de taille 8x8 remplie de zéros . Elle parcourt

ensuite les termes de la matrice d'entée M et si la somme des indices i+j vérifie la condition i+j<freq , elle affecte à M[i,j] la valeur D[i,j]. Elle retourne enfin la matrice D après filtrage des hautes fréquences.

3 Décompression et recomposition

3.1 Fonction Décompression

Cette fonction prend en paramètres une matrice D, la matrice de passage Pm, et la matrice de quantification Q. Tout d'abord, elle réalise une multiplication terme à terme des matrices D et Q pour obtenir la matrice M. Subséquemment, elle effectue une transformation de cosinus discrète inverse (DCT) sur la matrice M en utilisant la matrice Pm. Ce processus est réalisé à l'aide des fonctions de multiplication matricielle de numpy, soit np.matmul(M, Pm) et np.matmul(np.transpose(Pm), M). Enfin, la fonction retourne la matrice M ainsi obtenue.

3.2 Fonction Recomposition

Cette fonction prend en paramètre une liste "Bloc" contenant des matrices de 8x8, ainsi que deux entiers n et m représentant respectivement le nombre de lignes et de colonnes de la matrice recomposée. Elle crée une matrice M de dimensions n x m, initialement remplie de zéros. Ensuite, à l'aide d'une boucle, elle parcourt les lignes de la matrice recomposée par pas de 8. Un compteur "c" est utilisé pour itérer à travers les éléments de la liste "Bloc", et un compteur "cl" pour parcourir les colonnes de la matrice recomposée.

À chaque itération de la boucle, la fonction sélectionne la matrice numéro c dans la liste "Bloc" et remplit la matrice recomposée M avec les valeurs de cette matrice. Enfin, la fonction retourne la matrice recomposée M, obtenue en regroupant les matrices de 8x8 contenues dans la liste "Bloc".

3.3 Fonction réglage de pixel : Pixel setting

La fonction 'Pixel setting(M) est conçue pour ajuster les valeurs de chaque pixel dans une matrice tridimensionnelle M. Elle parcourt les dimensions de la matrice M, itérant à travers les canaux de couleur (représentés par l'indice i), les lignes (indice j) et les colonnes (indice k). Pour chaque valeur de pixel, la fonction vérifie si elle est inférieure à -128 ; si tel est le cas, elle la fixe à -128. De même, si la valeur est supérieure à 127, elle est ajustée à 127. En d'autres termes, la fonction s'assure que les intensités de chaque pixel sont limitées à la plage des valeurs acceptables pour un type de données d'entier signé sur 8 bits, soit de -128 à 127. En fin de compte, la matrice M modifiée est renvoyée, reflétant les ajustements effectués pour garantir que les valeurs des pixels restent dans une plage valide.

4 Evaluation de l'erreur /Post- Processing :

4.1 Fonction compressing$_error(M, N)$

Ce programme Python évalue l'erreur de compression d'une image après compression canal par canal (Rouge, Vert, Bleu). Il utilise la norme Euclidienne

pour mesurer la différence entre les pixels de l'image originale et de l'image compressée. Les erreurs individuelles sont agrégées en une seule valeur, normalisée en fonction des normes des canaux de l'image originale, et multipliée par 100 pour obtenir un pourcentage représentant la qualité relative de la compression par rapport à l'image originale.

4.2 Le taux de compression

Ce code additionne le nombre de valeurs non nulles (pixels non nuls) dans chaque canal (Rouge, Vert, Bleu) des matrices MR, MG et MB. La variable 'coeffdiff0' est utilisée pour stocker cette somme cumulative. Ensuite, le taux de compression est calculé en utilisant la formule suivante :

$$Taux\_de\_compression = \left(1 - \frac{coeffdiff0}{n \times m \times 3}\right) \times 100$$

où n et m représentent les dimensions de chaque canal de l'image (par exemple, largeur et hauteur), et le facteur 3 provient des trois canaux de couleur (Rouge, Vert, Bleu). Cette formule mesure le pourcentage de pixels non nuls par rapport au nombre total de pixels dans l'image originale, fournissant ainsi une estimation du taux de compression en fonction du nombre de pixels conservés après le processus de compression. Un taux de compression plus élevé indique généralement une meilleure compression.

4.3 Enregistrement de l'image après le traitement

Pour sauvegarder l'image traitée, la fonction "plt.imsave" est utilisée, tandis que "plt.imshow()" est employée pour afficher l'image traitée et "plt.show()" pour l'afficher dans une fenêtre.

D Résultats et remarques

1 Résultats

On a appliqué l'algorithme élaboré sur deux images :

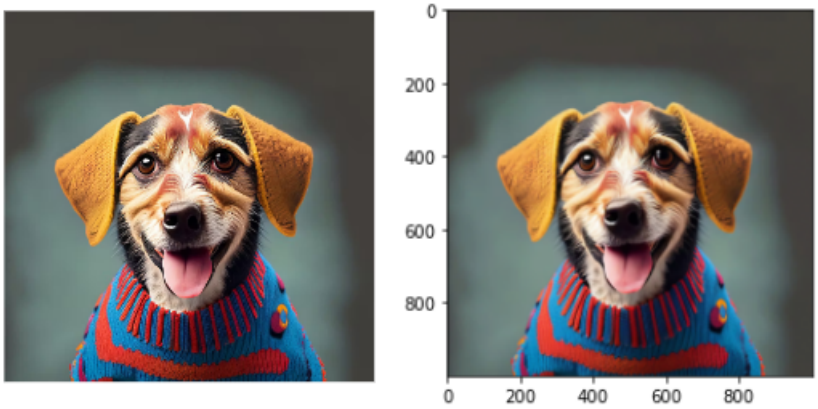
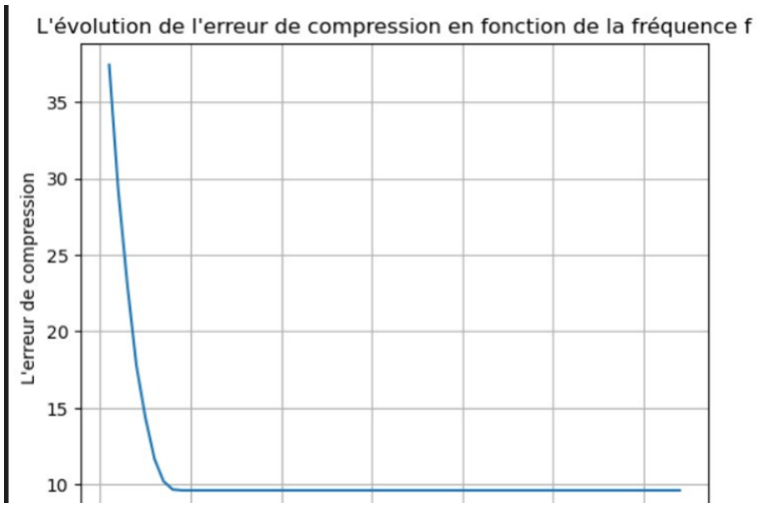


image1

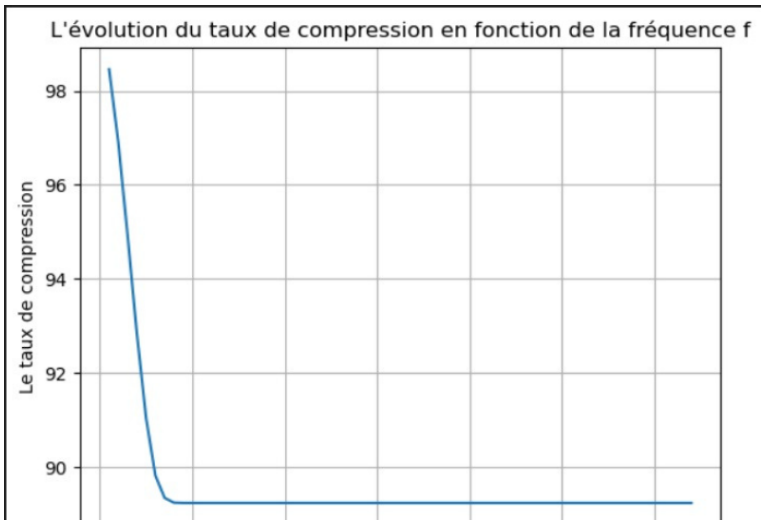
L'erreur de compression est : 14,930734125366127
Le taux de compression est : 92.9799
Le temps d'exécution est : 4.557123899459839

image2
L'erreur de compression est : 11.681131457258106
Le taux de compression est : 89,82018518518518
Le temps d'exécution est : 3.718756914138794

On a tracé le graphe de l'évolution de l'erreur globale et du taux de compression en fonction de la fréquence f qui varie dans l'intervalle [0,16]



graphe d'évolution de l'erreur de compression



graphe d'évolution du taux de compression

Et on a remarqué qu'ils deviennent constants après une certaine valeur de la fréquence de coupure.

2 Remarques

2.1 Remarque 1 : Le taux de compression

Les taux de compression des images varient en fonction de la complexité des détails. Les images détaillées nécessitent souvent une compression plus forte, surtout si elles comportent des motifs complexes. Les détails fins rendent la compression sans perte de qualité plus difficile, entraînant un taux de compression potentiellement plus faible. En revanche, les images avec des zones uniformes ou des dégradés simples peuvent être compressées plus efficacement avec une perte de qualité minimale. Les algorithmes de compression exploitent les redondances et les motifs pour réduire la quantité d'informations nécessaire à la représentation des images.

2.2 Remarque 2 : Le temps d'exécution

L'utilisation de la bibliothèque 'scipy.fftpack' a montré qu'elle rend le processus de compression plus rapide. Cela souligne que les fonctions de transformation de cosinus discrètes de cette bibliothèque sont plus efficaces que l'approche précédemment utilisée dans le programme. L'utilisation de bibliothèques spécialisées offre souvent des fonctionnalités améliorées et des performances meilleures, ce qui aide à rendre le traitement d'opérations complexes, comme la compression d'images, plus efficace et plus rapide.

2.3 Remarque 3 : L'évolution des processus de compression + références

La technologie QOI Compressing : un nouveau projet prometteur, apparu en novembre 2023 Le projet QOI suscite de l'attention dans le domaine de la

compression d'images en raison de ses performances notables. Le format QOI, développé par Dominic Szablewski, se distingue par sa simplicité, sa légèreté (avec seulement 300 lignes de code en C), son efficacité, et son utilisation d'un seul thread sans SIMD, effectuant une seule passe. Les objectifs du projet incluent une approche où chaque pixel n'est traité qu'une seule fois. Les résultats annoncés indiquent des gains significatifs de vitesse par rapport à des bibliothèques bien établies telles que libpng ou stb, avec une compression 20x à 50x plus rapide et une décompression 3x à 4x plus rapide. Ces performances sont d'autant plus impressionnantes compte tenu de la similarité des tailles de fichier obtenues. Il est important de noter que, bien que le projet soit déjà fonctionnel, le développeur indique qu'il n'est pas encore finalisé. Néanmoins, la disponibilité du code source sous licence MIT encourage probablement les développeurs à expérimenter et à contribuer au projet. Pour ceux qui cherchent à améliorer les performances de compression d'images, le format QOI semble être une option prometteuse à explorer.
Références : https ://phoboslab.org/log/2021/11/qoi-fast-lossless-image-compression
https ://youtu.be/EFUYNoFRHQI ?si=T$_{P}aWLhgaGdT5rDV$

E Conclusion

En mettant en pratique les connaissances théoriques fournies dans le sujet sur les techniques de compression et de décompression d'images par la transformée de Fourier, en particulier la DCT (Transformée de cosinus discrète), ce projet nous a permis d'utiliser des algorithmes de traitement d'images. L'objectif était de réduire la taille d'une image tout en maintenant une qualité acceptable. Ce processus a également constitué un défi, nous confrontant à des difficultés et nous enseignant l'optimisation des codes ainsi que la résolution d'erreurs, et surtout ça nous a appris de faire attention aux détails des codes , afin de pouvoir détecter les erreurs qu'on a du rencontrer lors de nos tentatives de programmation.