Université de Versailles – Saint-Quentin-en-Yvelines 2023-2024
Licence 1, IN202

Projet – Compression BC1

Version du 21 février

Yann Strozecki – Yann.Strozecki@uvsq.fr

Le but de ce projet est de créer des images au format BC1 qui servent à stocker les textures de manière compressée sur une carte graphique. Nous implémenterons une version simplifiée, mais très proche de la réalité. Pour vous renseigner sur le format BC1 vous pouvez lire `https://www.reedbeta.com/blog/understanding-bcn-texture-compre` `//learn.microsoft.com/en-us/windows/win32/direct3d10/d3d10-graphics-programming-guide-resources-block-c` ou `https://en.wikipedia.org/wiki/S3_Texture_Compression`.

Le projet est à faire en Python, par groupe de 2. Vous présenterez votre travail lors d'une soutenance la semaine du 13 mai, sur les horaires de TD habituels. Des exemples d'images compressées sont donnés avec ce document, pour tester si votre code fonctionne correctement. La maîtrise de votre code et le nombre de questions traitées seront les principaux critères de notations. La qualité du code et des commentaires sera également prise en compte.

La fonction de chargement des images vous est donné dans le fichier *projet.ipynb*. Vous obtenez alors un tableau numpy à trois dimension. Il s'agit en fait d'une matrice dont les éléments sont des triplets d'entiers dans $[0, 255]$ qui représentent les intensités en rouge, vert, bleu (RGB). On vous donne également le code qui permet d'afficher une image, qui vous servira à tester vos fonctions. Pour tester vos fonctions, on vous donne également la fonction PSNR qui calcule la proximité de deux images. Deux images identiques ont un PSNR de 100, des images très proches un PSNR d'environ 40.

Votre code **doit comporter une section test** qui permet de tester toutes les fonctions de votre programme et d'illustrer le fait qu'elles fonctionnent correctement.

1 Manipulation d'image

Pour pouvoir traiter une image, nous avons besoin que ses dimensions soient des multiples de 4. Pour le garantir, il faut faire du remplissage (padding), c'est à dire qu'on va ajouter des lignes et des colonnes de pixel noirs en bas et à droite de l'image. Si on a une image de dimension 15×21, on obtiendra une image de dimension 16×24 avec une ligne supplémentaire et trois colonnes supplémentaires.

Question 1 : Donner la fonction qui réalise ce padding ainsi que celle qui l'élimine et vérifier que l'application de ces deux transformations laissent l'image inchangée.

Dans le format BC1, l'image est décomposée en une grille de blocs de taille 4×4 qui seront traités indépendamment. Nous appellerons ces blocs des *patchs*. Ce découpage peut créer des artefacts carrés caractéristiques de la compression d'une image.

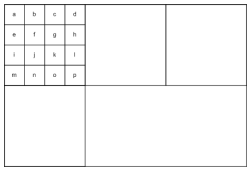

FIGURE 1 – Découpage d'une image en blocs 4×4

Question 2 : Soit une matrice dont les deux dimensions sont divisibles par 4. Donner une fonction qui découpe cette matrice en blocs 4×4 et les stocke dans une liste. L'ordre des blocs correspond à l'ordre de lecture d'une image. Les patchs sont représentés comme une matrice numpy de dimension $(4, 4, 3)$ dont les coefficients sont de type np.uint8, c'est à dire à valeur entière entre 0 et 255.

Question 3 : Donner une fonction qui prend une liste de blocs 4×4 et reconstruit une image. Pour tester que vos fonctions sont correctes, appliquer la fonction de découpage puis de reconstitution et vérifier que l'image n'a pas changé.

2 Traitement d'un bloc

Pour traiter un bloc, on va procéder de la manière suivante. On choisit deux couleurs a et b (des tableaux numpy de trois éléments). Il y a plusieurs manières de choisir ces couleurs en fonction du patch qui seront explicitées après. Ces couleurs a et b seront ensuite stockées en 5 : 6 : 5, c'est à dire 5 bits pour le rouge, 6 bits pour le vert et 5 pour le bleu. À partir de ces couleurs on va construire le tableau de 4 couleurs $[a, 2a/3 + b/3, a/3 + 2b/3, b]$, qu'on appelle la *palette*. Attention, les valeurs obtenues doivent être arrondies à l'entier le plus proche (utiliser la fonction *np.round()*).

Question 4 : Écrire la fonction tronque(n,p) qui enlève les p derniers bits de l'entier n. Ainsi, si n est codé sur 8 bits et que c'est la valeur du canal rouge d'une couleur qui doit être représentable sur 5 bits, on peut faire $n = tronque(n, 3)$ pour ne garder que les 5 bits les plus significatifs. Écrire une fonction qui renvoie la palette à partir des couleurs a et b.

Cette palette va servir à coder les pixels du patch. Chaque pixel va être représenté par la couleur de la palette la plus semblable. Pour comparer deux pixels p et q (des tableaux de taille 3), vous pourrez faire *np.linalg.norm(p.astype(int) - q)* qui renvoie la distance euclidienne entre ces deux pixels. Plus la distance est petite, plus les couleurs se ressemblent.

Question 5 : Étant donné une palette et un pixel, donner une fonction qui trouve la couleur de la palette (représenté par son indice dans la palette) la plus proche du pixel.

Pour coder le patch, on donnera les deux couleurs, a et b, ainsi que la liste des indices représentant les pixels du patch dans la palette (dans l'ordre habituel des images). Pour représenter a et b on aura besoin de 16 bits chacun (représentation 5 : 6 : 5) et pour chaque indice on aura besoin de 2 bits puisqu'ils sont compris entre 0 et 3. En tout, l'information du code d'un patch sera codé par un entier non signé de 64 bits.

Question 6 : À partir de a, b et du tableau des indices, construire l'entier représentant le patch. On écrira les bits de l'entier du bit de poids faible vers le bit de poids fort et le tableau des indices dans l'ordre habituel des images. L'écriture en base 2 de l'entier est la suivante :

$$I[3,3]I[3,2]I[3,1]I[3,0]I[2,3]\dots I[0,1]I[0,0]b[2]b[1]b[0]a[2]a[1]a[0]$$

3 Trouver les meilleurs couleurs

On donne ici plusieurs idées pour trouver les deux couleurs a et b à partir d'un patch. Vous implémenterez au moins la première et une autre de votre choix.

1. Pour chaque canal rouge vert et bleu, vous calculerez le minimum et le maximum rencontré dans un patch. Avec les valeurs minimales, vous formez une première couleur et avec les valeurs maximales, vous formerez une seconde couleur. Attention à mettre ces couleurs au bon format en utilisant la fonction tronque.

2. On calcule la moyenne et l'écart type avec les fonctions **np.mean** et **np.std** et les couleurs sont égales à la moyenne moins l'écart-type et la moyenne plus l'écart-type.
3. On tire des paires de couleurs au hasard et on retient la paire de couleur qui donne la meilleure qualité. La manière de tirer les couleurs au hasard peut changer la qualité des résultats. Attention, cette méthode est lente.
4. On part d'une paire de couleur obtenue par une des méthodes précédentes et on essaye de modifier un des canaux rouge vert ou bleu de 16 (en positif ou négatif). Retenez ensuite la meilleure modification. On peut appliquer cette technique plusieurs fois, et faire varier la valeur 16, mais attention cette méthode est lente.

Pour évaluer la qualité de la compression d'un patch, on pourra calculer sa distance au patch original en calculant la norme euclidienne de la différence des patchs. Vous pourrez utiliser la fonction **np.linalg.norm** ou la coder à la main pour accélérer votre code.

4 Écriture dans un fichier

Pour simplifier cette partie, nous allons écrire l'information sous forme d'un fichier texte. Pour ouvrir un fichier en mode écriture, vous pouvez faire *f = open(path, "w")*. Ensuite, pour écrire une chaîne de caractère, vous pouvez faire *f.write("maligne\n")*. Si vous avez une variable numérique k, pour obtenir la chaîne de caractères représentant cette valeur, il suffit de faire *str(k)*.

Les questions suivantes détaillent les différentes parties de votre fonction d'écriture dans un fichier. Attention à respecter scrupuleusement les instructions, autrement nous ne pourrons pas lire les fichiers que vous aller produire.

Question 7 : Pour commencer, vous écrirez deux lignes contenant les informations de votre image. La première ligne contiendra le type du fichier : "BC1". La deuxième ligne contiendra les dimensions de l'image dans l'ordre hauteur puis largeur, séparées par un espace, par exemple "200 300".

Question 8 : Vous écrirez la valeur du code des patchs en tant que chaîne de caractère. Pour chaque nouvel entier écrit (correspondant à un patch), vous sauterez une ligne.

5 Décompression

Il s'agit d'écrire les fonctions qui à partir d'une image compressée par votre code permet de recréer l'image en RGB qu'on pourra ensuite afficher. Quand vous aurez fait cette section, vous pourrez beaucoup plus facilement déboguer le code de la compression (en visualisant le résultat). Il est donc pertinent de faire cette section en premier, même si elle est placé à la fin pour des raisons de clarté du sujet. Si vous n'arrivez pas à faire cette section, nous vous encourageons à faire décompresser les fichiers que vous avez produit par le décodeur d'un de vos camarades pour vérifier qu'ils sont corrects. Vous pourrez appliquer votre fonction de décompression aux fichiers compressés fournis avec le sujet pour vérifier sa correction.

Question 9 : Écrire une fonction qui lit un fichier BC1 et qui créé la liste des entiers représentant chaque bloc.

Question 10 : Écrire une fonction qui transforme un entier en une paire de couleurs et un tableau d'indices. Donner une autre fonction qui à partir de ces informations calcule le patch correspondant. Utilisez ces fonctions pour reconstruire l'image sous forme d'un tableau de pixels.

6 Améliorations

Cette partie est **complètement facultative** et vous propose quelques pistes d'amélioration de votre compresseur BC1.

— Implémenter des méthodes plus efficaces pour trouver la meilleure palette pour un patch. Vous pouvez soit trouver de nouvelles méthodes, soit accélérer les méthodes sugérées.

— Implémenter l'écriture et la lecture en mode binaire de votre image pour minimiser sa taille sur le disque.

— Appliquer une compression sans perte au fichier BC1 obtenu pour tirer parti des répétitions de couleurs.

— Implémenter le second mode de BC1, en utilisant le fait qu'on peut stocker les deux couleurs de la palette dans un ordre arbitraire.

— Créer une interface Tkinter ergonomique pour tester votre travail.

7 Modalités pratiques

Merci de respecter les consignes suivantes :

— le projet est à faire en binôme ;

— le projet est à rendre sur ecampus, au plus tard le **lundi 13 mai à 9h00** ;

— une soutenance aura lieu la semaine du 13 mai pendant votre séance de TD.

— vous devez déposer un fichier `XY_NOM1_Prenom1-NOM2_Prenom2.zip` qui est le zip du dossier `XY_NOM1_Prenom1-NOM2_Prenom2` contenant votre projet.
`XY` sont les initiales de votre chargé de TD (`CZ`,`LD`, `FN`, `RH`, `YS`).

— Un outil de détection de plagiat sera utilisé sur vos codes.

Le retard de la remise du projet entraîne 1 point de moins par heure de retard.