INFO-F105 : Troisième projet (ASM)

Langages de programmation I

Année académique 2019–2020

Résumé

Ceci est l'énoncé du troisième projet du cours « Langages de programmation I ». Il s'agit de vous y familiariser avec les appels à fonction en langage d'assemblage, dans les conditions vues au cours et aux séances d'exercices. Veillez à le réaliser soigneusement et à respecter scrupuleusement les instructions présentées ci-dessous.

Introduction

Pour ce troisième projet, nous vous demandons d'implémenter le zigzag scan.

JPEG (acronyme de Joint Photographic Experts Group) est une norme qui définit le format d'enregistrement et l'algorithme de décodage pour une représentation numérique compressée d'une image fixe.

L'image est découpée en blocs de 64 (8×8) pixels. L'information de luminosité est conservée telle quelle (8×8 pixels dans l'image d'origine) et pour les couleurs, les blocs sont sous-échantillonnés (8×8, 16×8 ou 16×16 pixels dans l'image d'origine).

Il est alors appliqué à chaque bloc la transformée de DCT (Discrete Cosine Transform, variante de la transformée de Fourier). Pour ça il faut voir un bloc comme une fonction à deux variables (les indices de position dans la matrice) qui va pouvoir être décomposée en une somme de fonctions cosinus oscillantes à des fréquences différentes par la transformée DCT. Chaque bloc est ainsi décrit en une carte de fréquences et amplitudes au lieu de pixels et coefficients de couleur/luminosité.

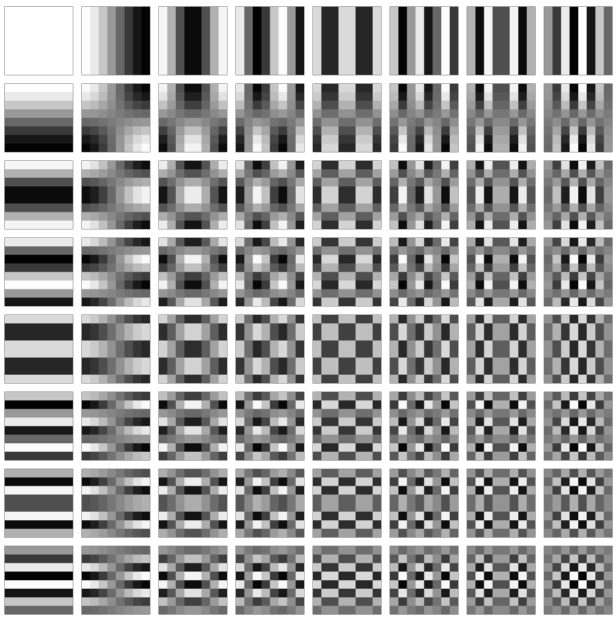


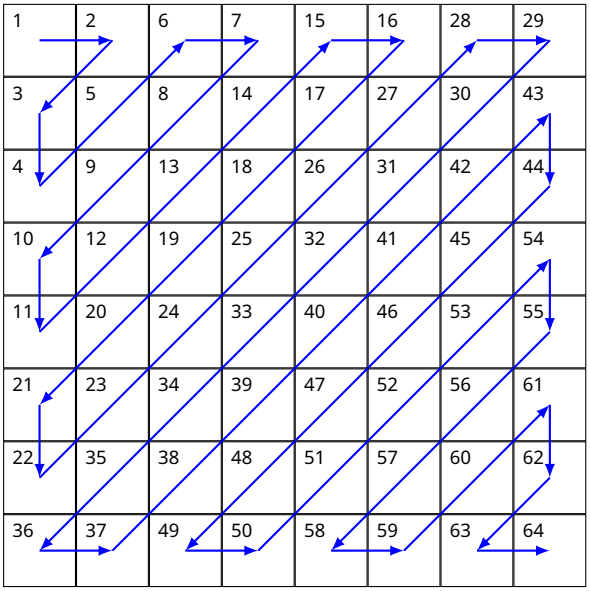
SCHÉMA 1 – carte de fréquences : 64 fonctions cosinus utilisées par la transformée DCT

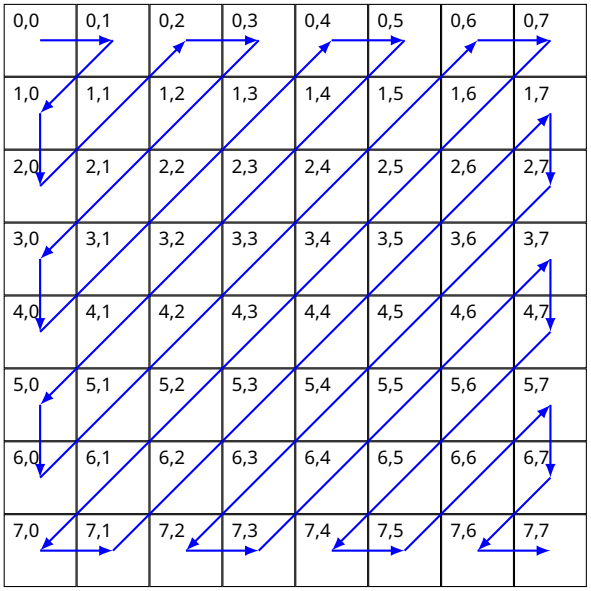
Ensuite, a lieu l'étape de quantification qui consiste à diviser la matrice résultante de la transformée DCT par une autre, appelée matrice de quantification, et qui contient (8×8) coefficients fixes qui sont choisis lors de l'implémentation. Voici un exemple de matrice obtenue après l'étape de quantification :

$$\begin{bmatrix} -23 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -21 & 4 & 2 & 0 & 0 & 0 & 0 & 0 \\ 6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

La quantification ramène plusieurs coefficients à 0. Ce sont surtout les coefficients en bas à droite qui finissent à 0 alors que ceux en haut à gauche sont généralement préservés. Ceci s'explique par le fait que les hautes fréquences contribuent peu à l'image.

C'est alors qu'arrive l'étape qui concerne ce projet, le scan en zigzag. Le scan en zigzag a pour but de représenter notre bloc 8×8 sous la forme d'un vecteur dont les premières valeurs correspondent aux coefficients des fréquences les plus représentées (les basses fréquences) et à la fin se trouvent les coefficients des basses fréquences qui sont le plus souvent à 0. Le scan en zigzag commence donc au coin supérieur gauche $(0, 0)$ et se termine au coin inférieur droit $(7, 7)$ de la table. Le parcours s'effectue comme le montre la figure suivante.





Le dessin de gauche montre l'ordre de visite des cases de la table et le second permet de voir le parcours au niveau des indices de la table.

Modalités de réalisation

Pour ce projet,vous disposez du code C++ ci-dessous :

PROGRAMME 1 – Début de la fonction main en C++

```
1 int main()
2 {
3   std::int8_t source[64] = {
4       1,  2,  6,  7, 15, 16, 28, 29,
```

```
     3,  5,  8, 14, 17, 27, 30, 43,
     4,  9, 13, 18, 26, 31, 42, 44,
    10, 12, 19, 25, 32, 41, 45, 54,
    11, 20, 24, 33, 40, 46, 53, 55,
    21, 23, 34, 39, 47, 52, 56, 61,
    22, 35, 38, 48, 51, 57, 60, 62,
    36, 37, 49, 50, 58, 59, 63, 64
};

std::int8_t dest[64] = {};
//------------------ Your code starts here ------------------
```

Il est à constater que la table est fixe et qu'il ne faut donc pas la lire dans un fichier ou autre. Cette table fixe, qui n'est pas une matrice typique provenant d'une étape de quantification, est utilisée pour ce projet afin de vous permettre de voir tout de suite si le parcours est fait correctement.

Il vous est demandé d'écrire le parcours en zigzag dans la fonction `zigzag_scan` qui sera écrite en assembleur dans le fichier `zigzag_scan.asm` et de faire appel à cette fonction dans le `main()`. Voici la signature de la fonction texttttzigzag_scan :

PROGRAMME 2 – Signature de la fonction zigzag_scan en C++

```
extern "C" std::int8_t* zigzag_scan(std::int8_t* table_source, std::
        int8_t* array_dest);
```

Cette fonction prend en argument un pointeur contenant l'adresse du début d'une table (`source`) et un pointeur contenant l'adresse du vecteur de destination (`dest`). Ils contiennent tous les deux 64 éléments d'une longueur de 8 bits. En ce qui concerne la valeur retour de la fonction `zigzag_scan()`, il s'agira de l'adresse de destination fournie en tant qu'argument. Dans la fonction en assembleur, vous devez insérer dans votre code une ligne du type commentaire pour indiquer clairement où se situe la fin du prologue. De même, une autre ligne de commentaire devra être placée pour indiquer clairement où se situe le début de l'épilogue.

Pour ce projet, l'ensemble de votre code C++ sera stocké dans un seul fichier `zigzag_print.cpp`. Celui-ci contiendra également une autre fonction qu'il vous est demandé d'écrire en C++ qui a pour signature :

PROGRAMME 3 – Signature de la fonction print_array) en C++

```
extern "C" void print_array(std::int8_t* array, std::uint32_t length
        );
```

Cette fonction a pour but d'afficher dans le *terminal* les éléments se trouvant dans le vecteur de bytes vers lequel `array` pointe par ordre croissant des adresses de ce dernier. Le deuxième argument `length` permet d'indiquer la taille du vecteur qui se trouve à l'adresse fournie par le premier argument. À l'affichage, chaque élément sera séparé par un espace. Cette fonction sera appelée uniquement dans la fonction écrite en assembleur (à la fin théoriquement).

Afin de pouvoir compiler les différentes parties du code, vous devez également écrire un makefile. Dans le makefile, les différents flags requis afin d'afficher un maximum de "warnings" seront fournis au compilateur g++.

Ce makefile a pour but de produire un fichier exécutable en format 32 bits nommé `zigzag_print`. Il faudra donc compiler le fichier `zigzag_scan.asm` en appelant le compilateur nasm qui produira le fichier objet `zigzag_scan.o`. Il faudra aussi compiler `zigzag_print.cpp` (qui contient la partie C++ de votre projet) en utilisant g++ et réaliser l'édition des liens en incluant `zigzag_scan.o`. Si ce fichier objet n'a pas déjà été créé, le makefile devra le créer dans la foulée. Il devra être possible de créer le fichier objet `zigzag_scan.o` sans être obligé de compiler le fichier `zigzag_print.cpp`.

Exemple d'exécution

Voici un exemple d'exécution du projet à fournir :

```
di-pc0:projet3 user$ ./zigzag_print
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 ...
di-pc0:projet3 user$
```

Il va de soit que la suite à afficher doit aller jusqu'à 64 mais par manque de place je n'affiche, ici dans l'énoncé, que les 32 premiers éléments.

Consignes

Pour ce projet, deux fichiers sont fournis. Le template de `zigzag_scan.cpp` contenant les seuls `include` autorisés et le début de la fonction `main()`. Et un makefile avec la liste des flags qu'il faut fournir à g++.

Si les signatures des fonctions ne sont pas scrupuleusement respectées (en ce inclus les noms des arguments donnés) ou si les fichiers n'ont pas tous les noms demandés, le projet sera sanctionné d'un 0/20.

Consignes pour la remise du projet

À respecter scrupuleusement !

1. Le projet est à remettre sur l'UV uniquement.
2. Un seul fichier compressé (`.zip` ou équivalent) doit être remis sous le format `<votre_matricule>.zip`[a]. Ce fichier `.zip` doit contenir un dossier qui aura pour nom votre matricule `<votre_matricule>`. Ce dossier contiendra les trois fichiers que vous devez soumettre pour ce projet.
3. Votre code doit être ***commenté***.
4. Date limite : le dimanche 19 avril 2020 ***avant 24 :00***.

a. exemple : si mon matricule est 123456, je remets le fichier `123456.zip`

Les projets en retard sont considérés comme non-remis.

Pour toute question concernant ce projet, veuillez vous adresser à Xavier Barthel [1].

1. e-mail : `<xavier.barthel@ulb.ac.be>`