RAPPORT DE MINI PROJET EN ARCHITECTURE DES ORDINATEURS

ASCII ART EN LANGAGE ASSEMBLEUR

MIPS

Réalisé par FAYE Mohamet Cherif et KO Roqyun

Encadré par Monsieur Olivier MARCHETTI

EISE-3, Polytech Sorbonne

2017-2018

Table des matières

INTRODUCTION

Dans le cadre de notre formation en informatique, ainsi qu'en architecture des ordinateurs, on est amené à réaliser un projet en langage assembleur en vue de mettre en pratique les connaissances acquises sur ce concept ainsi que les améliorer pour une meilleure prise en main.

1. SUJET DU MINI PROJET

Notre projet dénommé ASCII ART a pour but d'écrire un programme assembleur qui à partir d'une image bitmap en 16 niveaux de gris produit une image texte .Le principe consiste à remplacer des blocs de l'image d'origine par des caractères ayant le niveau gris équivalent.

Ci-dessous un exemple du résultat attendu à l'issu de ce projet.

Illustration 1: Logo polytech

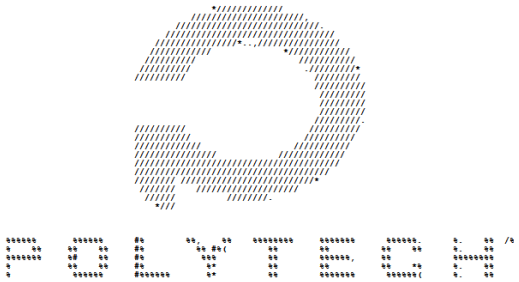

Illustration 2: Logo réseau polytech en texte (Source : https://manytools.org/hacker-tools/convert-images-to-ascii-art/)

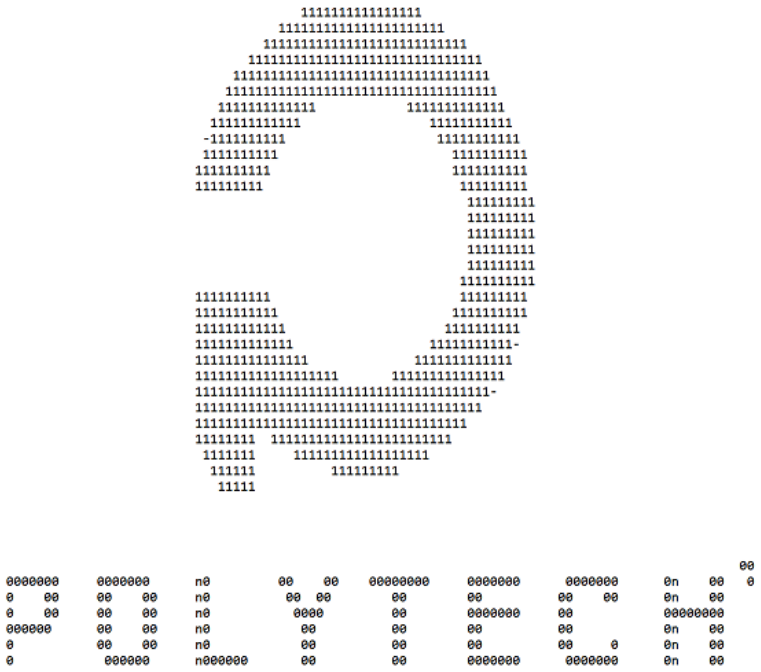

Illustration 3: Notre résultat

Dans ce projet, nous n'allons étudier que le fichier BMP Version 3 de 4 bits sans compression. Puis, nous allons utiliser la même notation que celle de Microsoft MSDN pour les structures des entêtes. Souhaitant garder la cohérence, nous avons codé le programme en C en anglais.

2. ANALYSE DU CAHIER DE CHARGES

Après une étude profonde du sujet, on a vu que pour que ce dernier puisse être réalisé suivant le cahier des charges des fonctionnalités attendues on devait faire appel à plusieurs concepts tels que :

- Ouverture d'un fichier BMP.
- Analyse du fichier BMP en 4 parties :
  - Entête du fichier
  - Entête de l'image
  - Palette de l'image
  - Codage de l'image
- Vérification
  - Est-ce que l'image est un format BMP ?
  - Est-ce que l'image est sans compression ?
  - Est-ce que le BMP est de 4 bit ?
  - La validité des données fournies.
- Calcul de la luminance d'une couleur.
- Le redimensionnement de l'image.
- Conversion de l'image en texte.
- Écriture de l'image en texte.
- La chaîne de niveaux de gris
  « $@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!lI;:,"^ `'. »

3. DESCRIPTION DES FONCTIONS

Pour décrire les fonctions de notre projet, nous allons utiliser une image BMP de 4 bits agrandi dans

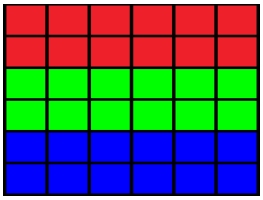

Illustration 4: une image BMP

L'illustration 4. Chaque case représente un pixel. Donc, c'est de la taille 6 x 6 (*hauteur* x *largeur*).

Le décodage de l'image :

```
char**  readBitmapImage(FILE *file, int biWidth, int biHeight);
```

Cette fonction suppose que la position actuelle du fichier ouvert est à « bfOffBits ». C'est-à-dire, les données à la position actuelle représentent une image codée. Il faut vérifier d'abord que l'image n'est pas compressée non plus.

2	2	2	2	2	2	0	0
2	2	2	2	2	2	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Illustration 5: l'image codé sans compression dans le fichier. Les cases blanches seront ignorées lors du décodage.

L'image dans l'illustration est enregisrré de la manière suivante :

Les nombres dans les cases sont les indices de la palette de l'image où :
0 -> rouge, RVB(255, 0, 0) ; 1-> verte, RVB(0, 255, 0)
2 -> bleue, RVB(0, 0, 255).

Chaque ligne est en alignée en mémoire par 4 octets. Soit la hauteur *biHeight* et la largeur *biWidth*, la taille de codage vaut $4 \times \lfloor biWidth \div 8 \rfloor \times biHeight$ (en octet)

Pour une image BMP de 16 couleurs, 4 bits suffisent pour exprimer tous les couleurs de la palette. Donc, tous les pixels sont exprimés en 4 bits. Puisqu'un octet est 8 bits, un octet peut représenter 2 pixels.

Le codage représenté en binaire	Le codage représenté en hexadécimal	Le codage représenté en décimal
0010 0010 0010 0010 0010 0010 0000 0000 0010 0010 0010 0010 0010 0010 0000 0000 0001 0001 0001 0001 0001 0001 0000 0000 0001 0001 0001 0001 0001 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	22 22 22 00 22 22 22 00 11 11 11 00 11 11 11 00 00 00 00 00 00 00 00 00	34 34 34 00 34 34 34 00 17 17 17 00 17 17 17 00 00 00 00 00 00 00 00 00

Ainsi donc, on utilise l'opération bit à bit pour séparer un octet par 2 pixels. Par exemple, pour récupérer les pixels à partir d'une donnée d'un octet 0x2E, on applique la méthode ci-dessous.

1er pixel : palette[(0x22 >> 4] = palette[0x2] = palette[2]
2e pixel : palette[(0x2E & 0x0F)] = palette[0xE] = palette[14]

La fonction *readBitmapImage* est basée sur ces principes pour décoder l'image codée.

- **Le calcul de l'indice de la *chaine niveaux de gris* à partir de la luminance moyenne d'un bloc :**

int densityIndexOfBlock(struct RGBSQUAD palette[], char **bmImage, int biWidth, int biHeight, int blockWidth, int blockHeight, int line, int column);

Cette fonction trouve un pixel à *line*-ème ligne et *column*-ème colonne (on le notera $\boldsymbol{P_0}$) et puis faire la somme des valeurs RVB des pixels autour du P_0 choisi pour calculer la luminance moyenne.

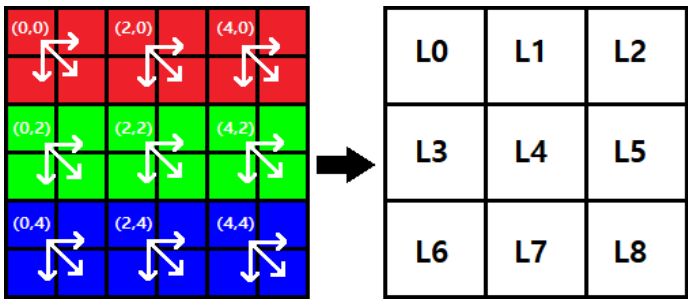

Illustration 6: Le résultat intermédiaire attendu de la fonction densityIndexOfBlock

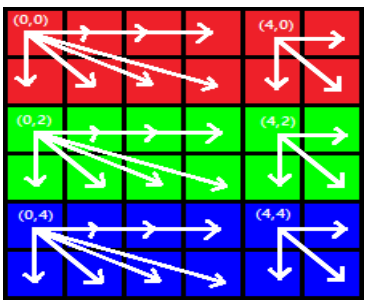

Illustration 7 : La division des blocs dans le cas où blockWidth = 4 et blockHeight = 2

Il y a 2 manières de calculer la luminance: la luminance simple et la luminance relative.

La luminance simple prend la moyenne de RVB. La formule est $L_{simple\_en\%} = \frac{R+V+B}{3\times 255}$

..

On peut résoudre ce problème avec le calcul de la luminance relative. Pour la calculer, nous supposons que la couleur verte est la plus lumineuse, la couleur rouge est suivante et la couleur bleue est la moins lumineuse.

$$L_{relative\_en\%} = \frac{0.2126\times R + 0.7152\times V + 0.0722\times B}{255}$$

La formule est en revanche, dans ce projet, la luminance simple est suffisant car nous supposons que les images lues seront en 16 niveaux de gris (Les valeurs de R, V et B sont, en générale, pareilles pour les couleurs grise).

Enfin, on fait le produit de (**la taille de la *chaîne niveaux de gris* – 1)** par $\mathbf{L_{simple\ en\ \%}}$. Ce produit donne l'indice de la *chaîne niveaux de gris* approprié.

4. LES DIFFICULTÉS RENCONTRÉES

La tâche essentielle afin de réaliser notre projet est l'étude du fichier BMP. La plupart des difficultés proviennent de cette tâche.

Nous avons, donc, étudié les 4 versions du format BMP et choisi la version 3. Ce choix est judicieusement fait pour assurer la compatibilité de notre programme avec les fichiers BMP utilisés en générale. Les versions antérieures sont pratiquement obsolètes et les images de 16 nombres de couleurs sont souvent sauvegardées sous le format BMP version 3 sans compression. Nous avons tiré cette conclusion après avoir utilisé plusieurs logiciels pour sauvegarder les fichiers BMP de 16 couleurs et téléchargé les fichiers BMP de 16 couleurs sur Internet. Nous avons utilisé « hexa-editor » pour analyser les fichiers.

Initialement, nous avons souhaité ouvrir tous les fichiers BMP, quoi que ce soit son type de compression. Cependant, nous avons décidé de supporter seulement les images sans compression. Il y a 4 types d'algorithmes de la compression pour l'image BMP : sans compression, Run-Length-Coding (RLE), JPEG et PNG, celui qui nous intéresse ici est le BMP.

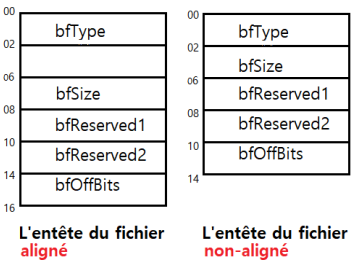

Illustration 8: La différence des structures aligné (16 octets) et non-aligné (14 octets)

La difficulté imprévue a été la lecture de l'entête du fichier. L'entête du fichier n'est pas aligné. Donc, recopier l'entête entièrement dans une structure a retenu une perte critique des données. Il faut d'abord lire 2 octets, la signature du fichier, et puis 12 octets (le reste).

Puis, décrypter les algorithmes de la compression a été la difficulté majeure. Ignorant de la relation entre la taille des images et la taille du codage des images, l'incohérence évidente entre eux nous a forcé à étudier plusieurs exemples des fichiers bitmap pour la résoudre.

Dernièrement, nous avons passé du temps considérable sur la redimensionnement de l'image. On a trouvé 2 méthodes pour redimensionner les images. La première méthode est utiliser la matrice d'une application linéaire. Les variables x et y représente les coordonnées des pixels. Soit k_0 et k_1 coefficients de compression suivant l'axe horizontale et l'axe verticale, respectivement, la formule est :

$$\begin{bmatrix} x \\ y \end{bmatrix} \times \begin{bmatrix} k_0 & 0 \\ 0 & k_1 \end{bmatrix} = \begin{bmatrix} x \times k_0 \\ y \times k_1 \end{bmatrix}$$

5. REMARQUES SUR LA PROGRAMMATION ASSEMBLEUR

On a pu aussi faire quelles remarques sur notre pratique de la programmation assembleur :

- Programmer en assembleur sur ce genre de projet requiert beaucoup de concentration et d'organisation pour gérer cette pléthore de ligne de codes.
- Pour faciliter la réalisation du projet en assembleur, on a dû d'abord coder en C pour ensuite le coder en assembleur, ce qui nous a bien aidés sur la suite du projet.
- la réalisation de ce genre de programme nécessitait aussi des recherches sur le fonctionnement des images ainsi que leurs compositions. Ce qui a été en quelque sorte un challenge comme c'est une notion qui nous a été nouvelle au niveau de la programmation en assembleur d'où même la raison que ce sujet faisait partie de nos choix.
- Durant l'élaboration du code, on a procédé de façon progressive en apportant des améliorations au niveau du code assembleur au fur et à mesure qu'on avançait dans le projet ainsi qu'en apportant plus d'informations explicatives au niveau des commentaires pour faciliter la compréhension des autres qui seront amenés à lire notre code.

CONCLUSION

Ce projet nous a été d'une très grande utilité puisqu'il nous a permis d'appréhender davantage la programmation assembleur dans ses différents aspects, mais aussi de rencontrer un certain nombre de difficultés que nous arriverons à résoudre sans problème dans le futur. De plus ça nous a permis de développer notre connaissance sur l'utilisation des images bitmap ainsi que leur mode fonctionnement.

En outre on peut noter une parfaite cohésion au niveau du travail collectif, qui a conduit à l'aboutissement de ce projet d'une importance capitale dans notre formation d'ingénieur.

SOURCE :

http://www.commentcamarche.net/contents/1200-bmp-format-bmp

http://www.fileformat.info/format/bmp/egff.htm

https://msdn.microsoft.com/en-us/library/dd162938.aspx

https://msdn.microsoft.com/en-us/library/dd183375.aspx

https://msdn.microsoft.com/en-us/library/dd183374.aspx

http://paulbourke.net/dataformats/asciiart/

https://en.wikipedia.org/wiki/Transformation_matrix

https://en.wikipedia.org/wiki/Brightness

https://en.wikipedia.org/wiki/Relative_luminance