

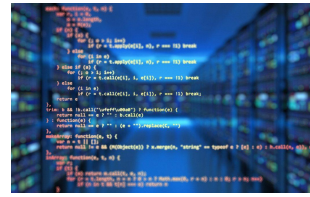
UNIVERSITE
DE LIMOGES



FACULTE DES SCIENCES
ET TECHNIQUES



PROGRAMMATION &
ALGORITHMIQUE



Filière : Master Mathématiques et Applications, ACSYON

Programmation et Algorithmique : Projet ASCII Art

Rédigé par :

Gabriel BLANCHOT

Ayman FAKHOURI

Rami MECHI

Cleque Marlain MBOULOU

Enseignants de l'UE :

François ARNAULT

Pierre DUSART

Année académique 2022-2023

TABLE DES MATIÈRES

1	Prototypes des fonctions utilisées	3
1.1	Introduction aux représentations RVB et HSL	3
1.2	Structures, découpage et conversion	4
1.3	Mise en œuvre de la transformation en ASCII	6
1.4	Affichage du résultat sur écran et dans un fichier texte	7
1.5	ASCII vers Jpeg Noir et Blanc	8
1.6	Pixel art	11
1.7	Programmation Modulaire et Finitions	12
2	Organisation du travail et difficultés rencontrée	15
2.1	Organisation du travail	15
2.2	Difficultés rencontrées	15
	Bibliographie	17




CHAPITRE

1

PROTOTYPES DES FONCTIONS UTILISÉES

1.1 Introduction aux représentations RVB et HSL

Le code couleur RVB se présente sous la forme d'un vecteur de dimension 3 en chiffres compris entre 0 et 255. Chaque composante représente le dosage nécessaire de chacune des couleurs primaires pour obtenir la couleur désirée. Les trois composantes représentent respectivement le dosage du rouge, du vert et du bleu. Par exemple on a :

Nom de la couleur	Aperçu de la couleur	Code couleur RVB
Bleu		(0, 0, 255)
Aigue-marine		(121, 248, 248)
Azur		(0, 127, 255)

Cependant, cette représentation ne sépare pas clairement la luminance, la saturation et la hue. Pour cela il existe une autre représentation des pixels appelée HSL :

- Hue : composante pour la couleur, entre 0 et 1 ;
- Saturation : composante pour la vivacité de la couleur, entre 0 et 1 (par exemple de rouge vif à rouge terne puis gris)
- Luminance : l'intensité lumineuse, entre 0 et 1.

1.2 Structures, découpage et conversion

Nous allons donc construire un programme permettant le passage de la représentation **RVB** vers **HSL**.

Dans un premier temps on définit les structures nécessaires, **HSL** pour construire le pixel **HSL**, **size** pour avoir les dimensions après le découpage, et **Tableau** pour stocker les données de l'image après traitement.

Structures

```
1 struct hsl{
2     float h;
3     float s;
4     float l;
5 };
6 typedef struct hsl hsl;
7
8 struct size{
9     int largeur;
10    int longueur;
11    int composantes;
12 };
13
14 typedef struct size size;
15
16 struct Tableau{
17
18     struct hsl **T;
19 };
20 typedef struct Tableau Tableau;
```

Découpage

Ensuite on construit une fonction que l'on appelle `decoupage` qui va nous permettre d'obtenir la taille de chaque pavé une fois notre image découpée (on obtient le nombre de lignes et de colonnes de chaque pavé) comme le montre l'image suivante :



```

1 size decoupage(int lignes , int colonnes , int composantes , int longue , int
    large){
2 //longue et large indique le nombre de caractères désirées en termes de
    lignes et de colonnes
3     int l;
4     int c;
5     l=floor(lignes/longue);
6     c=floor(colonnes/large);
7     size decoupe;
8     decoupe.longueur = l;
9     decoupe.largeur = c;
10    decoupe.composantes = JCOMPONENTS;
11    return (decoupe);
12 }

```

Conversion

A présent on construit la fonction **rgbToHsl** qui va prendre en paramètre un triplet de valeurs **R,V,B** et qui va retourner la valeur **H,S,L** associé. Pour cela on prend la fonction qui nous a été fournit dans l'énoncé.

```

1 hsl rgbToHsl(float r , float g , float b)
2 //voir code pt1 si besoin
3 return (pixel);

```

Les valeurs de **H,S** et **L** sont contruites de la manière suivante :

Les valeurs R , G , B sont divisées par 255 pour changer la plage de 0..255 à 0..1 :

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

Calcul de la teinte:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right) & , C_{max} = B' \end{cases}$$

Calcul de la saturation:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Calcul de la luminance :

$$L = (C_{max} + C_{min})/2$$

1.3 Mise en œuvre de la transformation en ASCII

Pour notre mise en oeuvre nous avons décidé tout d'abord de construire une fonction de conversion qui va se charger de convertir un `rgbbuffer` passée en entrée en un `hslbuffer` dans un nouveau buffer.

```
1 float *convertisseur(unsigned char *rgbbuffer, int jwidth, int jheight){
2     float *hslbuffer;
3     hslbuffer = malloc(jheight*jwidth*sizeof(hsl));
4     for (int i=0; i<jwidth*jheight*3; i+=3){
5         *(hslbuffer+i)= rgbtohsl(*(rgbbuffer+i), *(rgbbuffer+i+1), *(rgbbuffer+i+2)).h;
6         *(hslbuffer+i+1)= rgbtohsl(*(rgbbuffer+i), *(rgbbuffer+i+1), *(rgbbuffer+i+2)).s;
7         *(hslbuffer+i+2)= rgbtohsl(*(rgbbuffer+i), *(rgbbuffer+i+1), *(rgbbuffer+i+2)).l;
8     }
9     return(hslbuffer);
10 }
```

Ensuite en considérant une image découpée en plusieurs pavés, nous allons construire une fonction qui va calculer la moyenne **H,S,L** pour le pavé passé en entrée (le pavé de la *i*-ème ligne et *j*-ème colonne) et retourner la valeur moyenne de chaque composante. L'image originale est découpée en pavés contigus selon la géométrie de l'image finale (par exemple 80x25 caractères). Pour chaque pavé, on calcule l'intensité lumineuse moyenne et la couleur moyenne. Ces deux paramètres déterminent le caractère qui sera choisi pour la représentation du pavé.

```
1 hsl gethslmoy(int i,int j,size decoupe,unsigned char *rgbbuffer,int jwidth,
2               int jheight)
3 //voir code pt1 si besoin
3 return(result);
```

Où :

i,j représentent les coordonnées du pavé en entrée pour lequel on veut calculer les moyennes (H,S,L);

size decoupe représente un vecteur contenant la largeur et longueur du pavé.

Notons ici que pour se retrouver au pavé de coordonnées (*i,j*) nous avons besoin dans un premier temps pour se trouver dans la bonne ligne de passer à la composante *i * decoupe.longueur * jwidth * 3* ensuite pour se retrouver sur la bonne colonne nous avons besoin de prendre en plus *j * decoupe.largeur * 3* composantes.

Nous construisons dans la suite une fonction qui crée le tableau qui va contenir les moyennes(H,S,L) de chaque pavé de l'image :

```

1 Tableau ASCII(float *hslbuffer, size decoupe, int jwidth, int jheight, int
   longe, int large){
2
3   printf("%d \n", decoupe.longueur);
4   printf("%d \n", decoupe.largeur);
5   Tableau T;
6   T.T = (hsl**) malloc((longe)*sizeof(hsl*));
7   for(int i=0; i<longe; i++){
8       T.T[i] = (hsl*) malloc((large)*sizeof(hsl));
9   }
10
11  for(int i =0; i<longe; i++){
12      for(int j = 0; j<large ;j++){
13          T.T[i][j] = gethslmoy(i, j, hslbuffer, decoupe, jwidth, jheight);
14      }
15  }
16  return(T);
17 }

```

1.4 Affichage du résultat sur écran et dans un fichier texte

Affichage dans le terminal

Pour l'affichage dans le terminal, on crée une fonction "affichage" de manière simple qui consiste à parcourir le tableau et en fonction des valeurs hsl du tableau afficher le caractère correspondant :

```

1 void affichage(char *C[6][7], Tableau T, int longe, int large){
2 //voir code si besoin
3 }

```

cette fonction va utiliser les caractères **ASCII** de la table suivante :

```

1 char *C[6][7] = {
2     {".", "-", "/", "r", "L", "o", "*"},
3     {"!", "_", "|", "c", "C", "a", "&"},
4     {"^", "+", "(", "v", "J", "h", "%"},
5     {"", "<", ")", "u", "U", "k", "$"},
6     {"^", "i", "l", "n", "Y", "b", "#"},
7     {":", "?", "]", "x", "X", "d", "@"}
8 };

```

Chaque ligne correspond à une couleur différente, allant du plus clair au plus foncé.

Écriture dans un fichier texte (.txt)

Le principe d'écriture dans un fichier **.txt** va être le même que celui d'**affichage**. Dans la fonction :

```
1 void fichiertext(char *C[6][7], tableau T, int longe, int large){  
2 //voir code si besoin  
3 }
```

Dans cette fonction nous déclarons un fichier **intext**, qui va être ouvert dans le format **.txt**.

La seule différence avec la fonction **afficher** est la suivante : au lieu d'utiliser la commande **printf**, nous allons utiliser la commande **fprintf** pour écrire dans le fichier texte **intext**, le caractère associé à la moyenne luminance et à la moyenne couleur associé.

Exemple du rendu final



1.5 ASCII vers Jpeg Noir et Blanc

Dans cette section nous partons du fichier txt obtenu dans la partie I qui contient l'image convertie en ASCII. A partir de ce fichier nous allons créer une image en noir et blanc. Pour cela nous décompressons l'image jpeg comme précédemment pour obtenir les dimensions et nous découpons l'image de sorte à obtenir les mêmes dimensions que la partie ASCII. Pour obtenir notre image en noir et blanc nous avons décidé de modifier

notre srcbuffer afin qu'il ait les valeurs de noir et blanc (les nuances), pour cela notre programme comporte trois fonctions.

Une première pour la conversion du caractère en valeur entière entre 0 et 255 représentant ainsi la nuance de noir :

```

1 int conv256(char A, char *C[6][7]) {
2
3 int couleur = 500;
4 for(int i = 0; i < 6; i++){
5     for(int j = 0; j < 7; j++){
6
7         if(A == *C[i][j]){
8             couleur = (j*36 + i*6);
9         }
10    }
11 }
12 return(couleur);
13 }
```

il y a un total de 42 caractères différents, nous avons donc 42 nuances de noir différentes dont le pas est donc d'environ $\frac{255}{42} \simeq 6$. Une deuxième fonction pour remplir le buffer à partir de l'indice du caractère lu dans le fichier txt.

```

1 unsigned char* remplircases(int numero_c, int width, int height, unsigned char
    *rgbbuffer, int jwidth, int jheight, int NB, char *C[6][7], int longe, int
    large){
2 //voir code si besoin
3 return(rgbbuffer);
4 }
```

Ensuite une fonction qui ouvre le fichier txt converti et qui remplit les cases pour chaque caractère lu dans le txt.

```

1 unsigned char* lectxt_conv(unsigned char* srcbuffer, int jwidth, int
    jheight, int width, int height){
2 //ouverture du txt
3 FILE * fp;
4 fp = fopen("texteimage.txt", "r");
5
6 if (fp == NULL) {
7     printf("erreur \n");
8 }
9 int numero_c = 0;
10 int NB;
11 // premiere occurrence
12 char c = fgetc(fp);
13 NB = conv256(c);
14 srcbuffer = remplircases(numero_c, width, height, srcbuffer, jwidth, jheight,
    NB);
15 }
```

```
16 while((c = fgetc(fp)) != EOF){
17     // je rempli mon tableau des caracteres pour ensuite convertir le
    srcbuffer
18     //printf("%d",numero_c);
19     //printf("%c\n", c);
20     NB = conv256(c);
21     if(NB== 500){
22         numero_c = numero_c-1;
23     }
24
25     numero_c = numero_c + 1;
26     srcbuffer = remplircases(numero_c,width,height,srcbuffer,jwidth,
    jheight,NB,longe, large);
27 }
28 return(srcbuffer);
29
30 }
```

Enfin pour créer notre image, nous avons utilisé la fonction **void Image_NB** qui va déterminer la largeur et la hauteur de l'image, copie dans **srcbuffer** le resultat de **lectxt_conv**, compresse **srcbuffer** dans **jbuffer** et enfin l'écrit dans une image **JPEG** :

```
1 void Image_NB(unsigned char* srcbuffer,unsigned char *jbuffer, int jwidth
    , int jheight, int tjsamp, int tjstatus, tjhandle handle, long unsigned
    int jsize,char *C[6][7],int longe, int large ){
2
3     \\si besoin voir le code
4
5 }
```



Le seul probleme que nous avons recontré est que dans la lecture du fichier txt il y a

un caractère vide qui est lu à chaque ligne. C'est pourquoi il faut bien vérifier que les caractères sont dans le tableau de caractères à chaque occurrence. Pour cela nous avons donné une valeur de 500 si le caractère n'est pas dans le tableau pour pouvoir reconnaître ce cas.

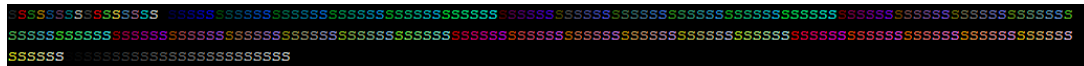
1.6 Pixel art

Le but de cette section est d'afficher dans les terminal les caractères de l'image **ASCII** dans la couleur correspondante. Pour cela nous avons utilisé le **Code d'échappement ANSI**.

Les **séquences d'échappement ANSI** sont une norme pour la signalisation intrabande afin de contrôler l'emplacement du curseur, la couleur, le style de police et d'autres options sur les terminaux de texte vidéo et les émulateurs de terminal. Certaines séquences d'octets, la plupart commençant par un caractère d'échappement ASCII et un caractère de crochet, sont incorporées dans le texte. Le terminal interprète ces séquences comme des commandes, plutôt que comme du texte à afficher textuellement.

Ainsi la commande suivante nous affiche des le caractère **s** dans différente couleur (0..255).

```
1 printf("\x1b[38;5;%dm%s",i,"s");
```



Afin de faire correspondre les couleurs, nous avons modifier le tableau correspondant à leur emplacement pour avoir un résultat se rapprochant de la réalité.

Pour cela, nous avons construit la fonction suivante :

```
1 // B représente le tableau de couleur
2 void affichage(char *C[6][7], Tableau T, char *B[256]) {
3     for(int i = 0; i < longT; i++) {
4         for(int j = 0; j < larT; j++) {
5             printf("%s%s", B[(int)(T.T[i][j].h*256)], C[(int)(T.T[i][j].h*6)][(
6                 int)(T.T[i][j].l*7)]);
7             printf("\n");
8         }
9     }
```

Avant modification



Après modification



1.7 Programmation Modulaire et Finitions

Lors de la conception d'un programme plus conséquent (projet), il est bon de découper le code du programme en plusieurs parties. On aura alors plusieurs fichiers source. Les objectifs :

- Plus de lisibilité du code.
- Travail collaboratif (chacun peut faire une partie distincte).

- Compilation séparée :
- Compilation de gros fichiers coûteuse ($> 10^6$ lignes de programme) : seules les parties modifiées seront compilées.
- Compilation par parties : les programmes objets sont regroupés par l'éditeur de liens pour obtenir le programme exécutable final.
- Interdépendance entre les parties du programme.

Nous avons divisé et partagé le projet en 3 parties indépendantes les unes des autres et rajouter le fichier contenant la fonction **main()**. Afin de ne définir qu'une seule fois les structures, nous avons créé un fichier **functions.h** où les structures (**hsl**, **size** et **Tableau**) et les prototypes des différentes fonctions sont définis.

• Dans cette Partie 1, nous avons mis dans le fichier **projet_pt1.c** les fonctions nécessaires à la **mise en œuvre de la transformation en ASCII** et à l'**écriture dans un fichier texte**.

• Dans cette Partie 2, nous avons mis dans le fichier **projet_pt2.c** les fonctions nécessaires au passage, à partir d'un fichier texte, d'une image en caractère ASCII à une image en noir et blanc

• Dans cette Partie 3, nous avons mis dans le fichier **projet_pt3.c** la fonction nécessaire à l'affichage en **Pixel ART**.

• Enfin dans la fonction **projet_final.c** nous avons mis les données, une fonction permettant de personnaliser les caractères d'échappement, c'est-à-dire la longueur et la largeur dans le terminal. Ce fichier contient également le programme principal (**main**) où a été construit un menu des fonctionnalités de notre programme. Nous avons aussi rajouté des variables qui nous donnerons le temps d'exécution de chaque partie.

Construction d'une commande simplifiant la compilation

Nous avons créé un fichier **Makefile** contenant le processus de compilation :

```

1 CC=gcc
2
3 all:executable
4
5 executable: projet_final.o projet_pt1.o projet_pt2.o projet_pt3.o
6 $(CC) -o executable projet_final.o projet_pt1.o projet_pt2.o projet_pt3.o
7         -lturbojpeg -lm
8 projet_pt1.o:projet_pt1.c functions.h
9 $(CC) -c projet_pt1.c -lturbojpeg -lm
10 projet_pt2.o:projet_pt2.c functions.h
11 $(CC) -c projet_pt2.c -lturbojpeg -lm
12 projet_pt3.o:projet_pt3.c functions.h
13 $(CC) -c projet_pt3.c -lturbojpeg -lm

```

```
13 clean :  
14  rm projet_final.o projet_pt1.o projet_pt2.o projet_pt3.o executable core
```

Ensuite, il suffit de faire la commande **make** en ligne de commande dans le répertoire contenant le fichier **Makefile** pour compiler le programme.

CHAPITRE

2

ORGANISATION DU TRAVAIL ET DIFFICULTÉES RENCONTRÉE

2.1 Organisation du travail

Au début de notre projet il était primordial de s'organiser pour ne pas se perdre dans les tâches du projet. Pour cela l'équipe enseignante nous a conseillé de se partager les tâches cependant ce que nous trouvions dommage c'est le fait que l'on ne voit pas la globalité de l'ensemble du travail de ce projet. Donc pour remédier à cela nous avons décidé de voir en commun chaque partie demandé en discutant des idées de tout le monde, une fois cela fait on gardait les meilleures idées puis pour la partie code nous nous sommes partagées les sections.

Pour la partie 1 : le code a été réalisé par Ayman Fakhouri et Gabriel Blanchot. Pour la partie 2 : le code a été réalisé par Gabriel Blanchot.

Pour la partie 3 : le code a été réalisé par Cleque Mboulou Moutoubi.

Pour le menu : le code a été réalisé par Ayman Fakhouri et Rami Mechi.

Pour la compilation et organisation des fichiers : le travail a été réalisée par Cleque Moutoubi et Ayman Fakhouri En ce qui concerne le compte le rendu celui-ci a été essentiellement rédigé par Cleque Mboulou Moutoubi et Ayman Fakhouri et pofiné et détaillé par Blanchot Gabriel et Mechi Rami.

Veuillez noter que Rami Mechi n'a pas pu réellement programmer par soucis de santé en effet comme vous savez déjà il a subi une intervention aux niveau des yeux qui l'empêche de passer du temps devant les écrans.

2.2 Difficultés rencontrées

Comme vous vous doutez d'avance nous n'avons rencontré aucune difficulté!.

Trêve de plaisanterie nous avons rencontré un tas de difficultés...

Pour commencer la plus grande difficulté que nous avons rencontré était la fameuse erreur que vous connaissez sans l'ombre d'un doute : "core dumped", cette erreur nous a fait faire des cauchemars. En effet tout d'abord il faut noter que cette erreur survient lors de l'exécution d'un fichier déjà compilée et donc notamment lorsque l'on a fini de coder une partie avec toutes les fonctions nécessaires à l'exécution du programme. Lorsque cette erreur survient cela signifie que nous avons mal géré un pointeur quelque part, or on se sert des pointeurs dans énormément de fonctions donc pour retrouver dans quelle fonction cela posait soucis c'était un véritable défi, de plus une fois trouvée la fonction il fallait trouver dans quel endroit de la fonction on avait un problème ce qui n'était vraiment pas évident.

Ensuite une autre grande difficulté que nous avons rencontrée était le manque du matériel de la faculté, en effet avec les vacances nous avons été obligé de travailler chez nous avec nos ordinateurs sans l'accès à GUACAMOLE, or nos ordinateurs utilisent comme système d'exploitation Windows, nous avons donc dû télécharger un compilateur le paramétrer... le tout en suivant des tutos youtube pas très clairs... Une fois installé et paramétré le compilateur nous avons dû télécharger la librairie `libturbojpeg` et là on a eu des difficultés en effet nous n'avons pas réussi à trouver des tutos clairs pour l'installation de librairies pour le langage C. Ayant bataillé un peu nous avons pu installer la librairie mais l'installation ne s'était pas bien déroulée et donc lors des compilations nous avions énormément d'erreurs qui venaient du fait que le compilateur n'arrivait pas à reconnaître certaines fonctions comme `malloc` par exemple.

A ce moment là nous avons dû improviser, Cleque a eu l'idée donc de télécharger un émulateur qui permet d'émuler le système d'exploitation linux sur nos pc. Le téléchargement de cet émulateur et de tous les fichiers nécessaires à l'exécution de l'émulateur et à la compilation de nos programmes nous a pris une après-midi entière.

Par la suite nous avons donc tous opéré avec cet émulateur (qui était très lent chez ceux dont l'ordinateur était modeste

Une autre difficulté rencontrée était l'organisation des fichiers pour la compilation, en effet n'ayant pas accordée au début d'importance à la programmation modulaire, nous avons procédé à la programmation de chaque partie sans nous soucier de comment on doit compiler nos fichiers ce qui nous a bien entendu posé soucis au moment de la compilation simultanée, par la suite nous avons dû passer pas mal de temps à réarranger nos fichiers pour mener à bien notre compilation.

Enfin bien évidemment le temps imparti pour réaliser ce projet a été sans doute notre plus grande difficulté, en effet le projet nous a été donné la semaine juste avant nos examens, bien évidemment nous n'avons pas pu le travailler à ce moment et encore moins la semaine d'après. Lors des vacances les premiers jours nous n'avons pas pu le travailler non plus car nous n'étions pas disponibles (ce que j'espère est compréhensible, vacances en famille...). Cependant comme vous pouvez le constater le travail a été fait dans le moindre des détails demandé et avec soin, en espérant que vous notiez l'effort fourni par notre groupe.

BIBLIOGRAPHIE

- [1] [RVB vers HSL \[en ligne\]](#)
- [2] [ANSI Color Codes](#)
- [3] [Code d'échappement ANSI](#)
- [4] [PROGRAMMATION EN LANGAGE C, UNIVERSITE DE LIMOGES](#)