REPUBLIQUE DU CAMEROUN
UNIVERSITE DE DOUALA
FACULTE DES SCIENCES
DEPARTEMENT MATHEMATIQUE INFORMATIQUE

REPUBLIC OF COMEROON
THE UNIVERSITY OF DOUALA
FACULTY OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCES & MATHEMATICS

REALISATION D'UN PROGRAMME C QUI OPERE SUR LES IMAGES POUR ISOLER LA COMPOSANTE ROUGE, VERTE ET BLEU D'UNE IMAGE DIGITAL D'UN FORMAT DEFINI A

NKWANHOU SITCHI René Blaise
17S88123

Table des matières

IMAGE DIGITALE ET FORMAT D'IMAGE

Une mage digitale est un ensemble de points colorés appelés pixels. Les pixels peuvent avoir chacun une couleur différente. Certains perçoivent en générale une image digitale comme une grille rectangulaire.
Une image rectangulaire peut être représentée mathématiquement par une matrice. Il existe plus manières de représenter cette matrice dans un ordinateur (ou dans la puce mémoire d'une caméra numérique) ce qui a pour résultat la présence de plusieurs formats d'image ; entre autres on peut citer : le format BMP, GIF, JPG, TIFF, …
On ce qui nous concerne pour mener à bien notre tâche nous allons nous intéresser à la famille d'image ***Netpbm*** *introduite par* ***Jeffrey Poskanzer*** *en 1980.*

Les formats de la famille Netpbm

PBM (portable bitmap)

Les images de ce format ne peuvent contenir que des pixels colorés en noir ou en blanc comme son nom l'indique (bitmap ; 0 correspond à la couleur noire ; 1 correspond à la couleur blanche).

PGM (portable graymap)

Les pixels sont colorés avec des nuances de gris. La valeur d'un pixel en général un entier oscillant dans [0,M] pour un M donné (en général M = 255).

PPM (portable pixmap)

Les pixels s'étendent sur un plus large spectre de couleurs. Un pixel ici est un triplet (r,g,b) de trois entiers avec :

r*: l'intensité de rouge*
g*: l'intensité de vert* $0 \leq r, g, b \leq M$
b*: l'intensité de bleu* *M = 255 la plupart du temps*

Chacun de ses 3 formats une variante 'plain' et une variante 'raw'.

La variante ***'plain'*** *est stockée sous forme textuelle, elle est triviale à inspecter et à éditer il suffit juste de disposer d'un simple éditeur de texte. Malheureusement elle est vorace en espace mémoire.*

La variante ***'raw'*** *pour sa part est plus économe en mémoire car les données stockée sous forme binaire.*

De ces 3 formats d'images, celui qui nous semble idéal pour réaliser notre tâche est le format ppm car il permet de représenter les images en couleur.

Structure d'une image PPM

Une image ppm comporte 2 parties l'entête et corps

```
P3
# a sample PPM file
4 4
255
10 20 30     30 30 30     11 11 40     50 100 0
10 20 30     30 30 30     11 11 40     50 100 0
10 20 30     30 30 30     11 11 40     50 100 0
10 20 30     30 30 30     11 11 40     50 100 0
```

Contenu d'un fichier ppm sous forme plain.

L'entête

Un certain nombre de lignes parmi les premières de ce fichier constituent son entête. Elles fournissent résumé du contenu de l'image.
L'entête d'un fichier ppm est formée de 4 entrées, lesquelles seront définies à l'aide de celle du fichier représenté plus haut :

```
P3
# a sample PPM file
4 4
255
```

- *P3 défini le format de l'image, généralement appelé nombre magique.*
  Pour un fichier ppm stockée en mode plain le nombre magique est P3.
  Pour le mode raw le nombre magique est P6.
  Ce sont les deux seules valeurs autorisées pour un fichier ppm; il est aussi important de noter que le nombre magique est case sensive.

- *Ensuite viennent le nombre de colonnes et de lignes dans l'image. Dans l'exemple l'image est formée de 4 lignes de 4 pixels chacune.*
  Le nombre de colonne correspond à la largeur de l'image.
  Le nombre de ligne correspond à la hauteur de l'image.

- *Enfin nous avons la valeur maximale que peut prendre un pixel (le M évoqué plus haut).*

On peut remarquer dans l'exemple, la présence d'une ligne débutant par '#' : il s'agit d'un commentaire ; il prend fin avec le début d'une nouvelle ligne. Il peut avoir plusieurs commentaires dans l'entête mais ils ne sont pas permis au sein du corps de l'image (la matrice des pixels).

Le corps

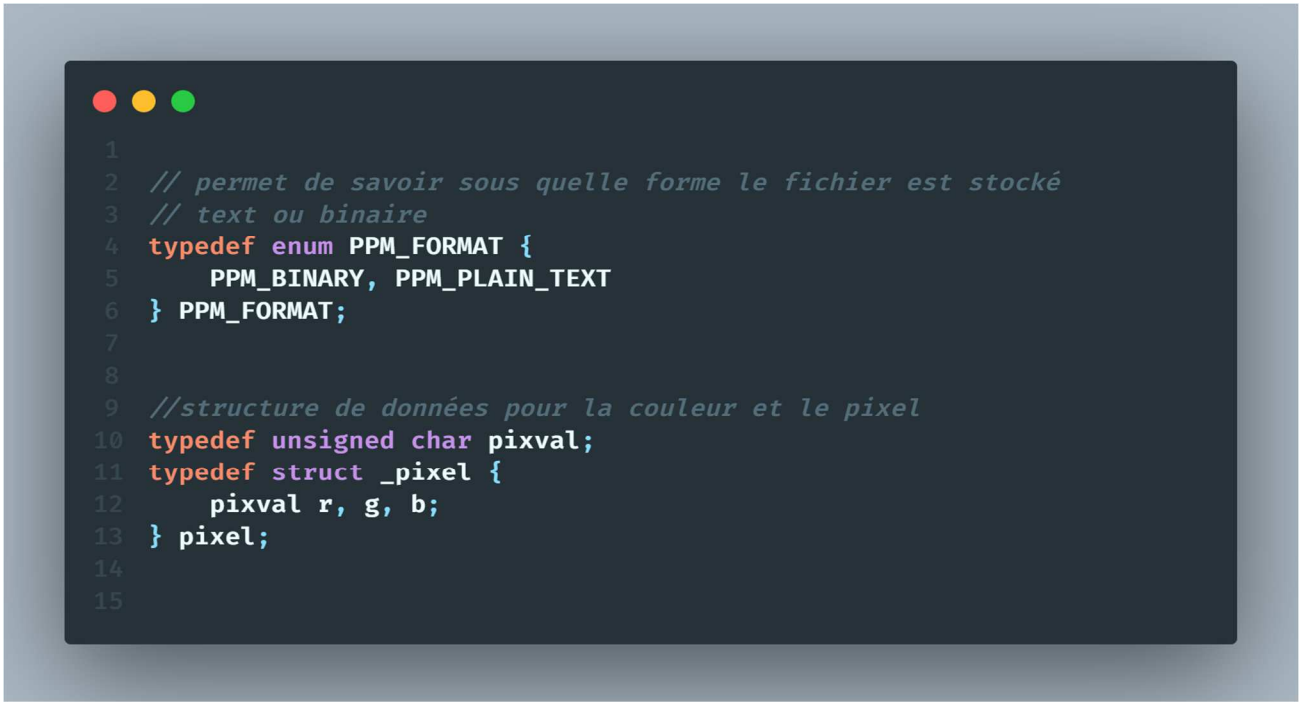
Il est formé de la matrice des pixels de l'image. A titre de rappel un pixel dans un fichier ppm est formé de 3 entiers (r,g,b).
Il est important de noter que les pixels sont séparés par un espace blanc (' '), mais dans l'exemple plus d'un espace blanc a été utilisé pour faciliter la compréhension à un être humain, mais les machines n'ont pas ce problème, tout cela peut même être représenté sur une seule ligne.
Il est important de savoir que formats d'images moderne utilisent une forme de compression pour réduire le nombre d'informations stockés (et par conséquence leur taille). Le format ppm est souvent utilisé comme intermédiaire pour convertir un format d'image en un autre.
Une image ppm peut être visualisée à l'aide de logiciel tel que GIMP.

Mise en place de la solution.

Avant de commencer il est important de savoir que la composant x d'une image est cette même image à l'exception que tous les pixels sont à 0 sauf celui représentant l'intensité de la couleur x (x = rouge ou x = vert ou x = bleu).
Ceci compris il ne nous reste plus qu'à mettre sur pied le programme C qui va accomplir cette tâche pour nous.

Etapes

Créons des types permettant de représenter notre image



```c

// permet de savoir sous quelle forme le fichier est stocké
// text ou binaire
typedef enum PPM_FORMAT {
    PPM_BINARY, PPM_PLAIN_TEXT
} PPM_FORMAT;


//structure de données pour la couleur et le pixel
typedef unsigned char pixval;
typedef struct _pixel {
    pixval r, g, b;
} pixel;

```

Puisse que l'entête d'un fichier ppm peut contenir des commentaires, il faut trouver un moyen des ignorer et les caractères valides.
Aussi tôt dit aussi tôt fait :

```c

char ppm_getc(FILE *fp) {
    char ch = 0;

    // ignorer les commentaires (débute avec #)
    while ((ch = getc(fp)) == '#') {
        while ((ch = getc(fp)) ≠ '\n');
    }

    return ch;
}
```

Faisons une pierre deux coups et créons une méthode pour lire un entier de notre matrice de pixels.

```c
int ppm_getint(FILE *fp) {
    char ch = 0;
    int i = 0;

    // continuer à lire un caracter tant que c'est un espace blanc
    // la boucle do while est idélae pour controler la valeur de après chaque appel de ppm_getc()
    do {
        ch = ppm_getc(fp);
    }while (ch == ' ' || ch == '\t' || ch == '\n');

    // EOF check and number check
    if (ch == EOF) {
        fprintf(stderr, "Erreur: nous avons rencontré le caractère EOF
        qui indique la fin du fichier.\n");
        exit(1);
    }
    if (ch < '0' || ch > '9') {
        fprintf(stderr, "Error: caractère invalid
        détecté lors de la lecture d'un pixel.\n");
        exit(1);
    }

    // conversion de la chaine en entier
    // le test plus haut controle si ch est un chiffre décimal
    do {
        i = i * 10 + ch - '0';
        ch = ppm_getc(fp);
    } while (ch ≥ '0' && ch ≤ '9');

    return i;
}
```

Et voilà une autre pour récupérer le nombre magique du fichier (P3 ou P6) :

```c
PPM_FORMAT ppm_readmagicnumber(FILE *fp) {

    if ((ppm_getc(fp)) ≠ 'P') {
        fprintf(stderr, "Erreur: Incorrect format for PPM, the first line should be P3 or P6.\n");
        exit(1);
    }

    //plain ou raw
    int n = ppm_getint(fp);
    if (n == 3) {
        return PPM_PLAIN_TEXT;
    }
    if (n == 6) {
        return PPM_BINARY;
    }

    fprintf(stderr, "Erreur: format incorrect la première ligne doit être P3 or P6.\n");
    exit(1);
}
```

Ça serait aussi bien d'avoir une autre fonction pour allouer et libérer l'espace mémoire nécessaire au stockage de la matrice de pixels

```c
pixel *ppm_allocarray(int cols, int rows) {
    pixel *pixels = NULL;

    pixels = (pixel *)malloc(rows * cols * sizeof(pixel));
    if (pixels == NULL) {
        fprintf(stderr, "Erreur: Echec d'allocation mémoire.\n");
        exit(1);
    }

    return pixels;
}


void ppm_freearray(pixel *pixels) {
    free(pixels);
}

```

Il ne reste plus qu'à pouvoir lire les données de notre fichier et de pouvoir écrire nos 3 images

```c

void ppm_writeppm(FILE *fred,FILE *fgreen,FILE *fblue, pixel *pixels, int cols, int rows, pixval maxvalP)
{

    fprintf(fred, "P3\n%d\n%d\n%d", cols, rows, maxvalP);
    fprintf(fgreen, "P3\n%d\n%d\n%d", cols, rows, maxvalP);
    fprintf(fblue, "P3\n%d\n%d\n%d", cols, rows, maxvalP);

    int i;
    for (i = 0; i < rows; ++i) {
        int index = i * cols;
        fprintf(fred, "\n");
        fprintf(fgreen, "\n");
        fprintf(fblue, "\n");
        int j ;
        for (j = 0; j < cols; ++j) {
            fprintf(fred, "%d %d %d\t", pixels[index].r, 0, 0);
            fprintf(fgreen, "%d %d %d\t", 0, pixels[index].g, 0);
            fprintf(fblue, "%d %d %d\t", 0, 0, pixels[index].b);
            ++index;
        }
    }
}

```

```c
void ppm_readppminit(FILE *fp, int *colsP, int *rowsP, pixval *maxvalP, PPM_FORMAT *formatP) {
    *formatP = ppm_readmagicnumber(fp);
    if ((*colsP = ppm_getint(fp)) ⩽ 0) {
        fprintf(stderr, "Erreur: largeur de l'image incorrecte pour le format ppm.\n");
        exit(1);
    }
    if ((*rowsP = ppm_getint(fp)) ⩽ 0) {
        fprintf(stderr, "Erreur: largeur de l'image incorrecte pour le format ppm.\n");
        exit(1);
    }
    // read the max color value, must be 255 here
    if ((*maxvalP = (pixval)ppm_getint(fp)) ≠ 255) {
        fprintf(stderr, "Erreur: la couleur maximale de ce fichier est  %d.
        juste 255 (8-bits par pixel) est supporté.\n", *maxvalP);
        exit(1);
    }
}


pixel *ppm_readppm(FILE *fp, int *colsP, int *rowsP, pixval *maxvalP) {
    PPM_FORMAT format;
    pixel *pixels;

    ppm_readppminit(fp, colsP, rowsP, maxvalP, &format);
    pixels = ppm_allocarray(*colsP, *rowsP);

    int n_pixels = (*rowsP) * (*colsP);

    if (format == PPM_BINARY) {
        fread(pixels, sizeof(pixel), n_pixels, fp);
        return pixels;
    }
    if (format == PPM_PLAIN_TEXT) {
        int i;
        for ( i = 0; i < n_pixels; ++i) {
            pixels[i].r = ppm_getint(fp);
            pixels[i].g = ppm_getint(fp);
            pixels[i].b = ppm_getint(fp);
        }
        return pixels;
    }

    fprintf(stderr, "Erreur: le nombre magique doit être soit P3 ou P6 pour le format ppm.\n");
    exit(1);
}
```

Il ne reste plus qu'à mélanger tout cela au sein de notre fichier main

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "prototype.h"

int main(int argc, char *argv[])
{
     FILE* fichier = NULL;
     fichier = fopen(argv[1],"r");
     if(fichier){
        printf("%s ouverture okay",argv[1]);
        int cols, rows;
        pixval maxvalP;
        pixel *pixels = ppm_readppm(fichier, &cols, &rows, &maxvalP);

        FILE* fichier1 =NULL;
        FILE* fichier2 =NULL;
        FILE* fichier3=NULL;
        FILE* fichier4=NULL;
         fichier1= fopen("red.ppm","w");
         fichier2 = fopen("green.ppm","w");
         fichier3 = fopen("blue.ppm","w");
        if(fichier1 && fichier2 && fichier3 ){
            ppm_writeppm(fichier1,fichier2,fichier3, pixels, cols, rows, maxvalP);
            fclose(fichier1);
            fclose(fichier2);
            fclose(fichier2);
        }else{
            fprintf(stderr, "Erreur: echec d'allocation mémoire.\n");
        }


        printf("\nfin");

        fclose(fichier);
     }else{
         printf("Problème lors de l'ouverture du fichier");
        fclose(fichier);
     }
     return 0;
}
```

Références

- *D. Phillips; Image Processing in C: Analyzing and Enhancing Digital Images, RandD Publications, 1994*
- *Rouben Rostamian - Programming Projects in C for Students of Engineering, Science, and Mathematics (2014, SIAM)*
- *https://fr.wikipedia.org/wiki/Portable_pixmap*

code source: ***https://github.com/cityBlaise/CProgrammingImageProcessing.git***