

Rapport de Projet : ImagesTaches

Nawfoel Ardjoune & Chaolei CAI

December 15, 2019

Contents

1	Introduction	1
2	Dependences	1
3	Compilation du projet	1
4	Objectives et consigne du Projet	2
5	Structures de données de base	2
6	Explications sur les fonctions utilisées	3
6.1	image.c	3
6.2	propager.c	4
6.3	traitement.c	5

1 Introduction

Ce document constitue notre rapport sur le projet de fin de semestre L2 pour le cours d'impérative 2 de M.Bourdin.

Le projet est consultable depuis ce lien github: <https://github.com/bk211/ImagesTaches>

2 Dependences

Ce projet nécessite le support de l'API GL4Dummies de M.Belhadj, par extension, vous devez avoir sur votre machine les dépendances de GL4Dummies (SDL2 et OpenGL pour ne citer que les plus importants).

3 Compilation du projet

Un makefile est présent dans le répertoire de racine, pour compiler le programme, une simple commande "make" suffit à obtenir l'exécutable "exec". Enfin, lancer le programme exec pour faire apparaître la fenêtre d'affichage. Par défaut l'image affiché est l'image avant le traitement, pour afficher le l'image après application du traitement, pressez la touche "r" de votre clavier.

Pour terminer le programme, vous pouvez presser la touche "ECHAP", "q" ou la croix situé dans le coin supérieur droite.

4 Objectives et consigne du Projet

Il faut sur, une image, trouver automatiquement toutes les "taches de couleur". L'image est donnée comme un grand tableau avec un octet par couleur par "pixel" (trois couleurs, R, G et B).

C'est donc une vaste matrice de (XMAX x YMAX x 3) octets. Trouver les taches consiste à trouver et numéroter les zones connexes ayant la même couleur.

Une bonne méthode consisterait à commencer par trouver tous les pixels connexes ayant une certaine couleur, ils forment une tache.

Puis à réitérer ce processus pour tous les pixels. On a alors toutes les taches, si elles ont été listées, il est maintenant facile de leur appliquer un traitement.

Un traitement pourrait être de délimiter en noir les taches de couleur. Enfin, une tache peut être une zone dont les pixels ont à peu près la même couleur. On définira cette approximation et on fera la délimitation de ces zones comme précédemment.

Le rapport contiendra des explications sur les procédures mises en oeuvre, le mode d'emploi, le listing du programme et des traces d'utilisation.

Le programme devra tourner sur au moins une des machines en libre-service du Bocal et sera écrit en C, avec, si nécessaire, utilisation des bibliothèques OpenGL, Glu, Glut et GL4D, seules bibliothèques non standard autorisées.

5 Structures de données de base

```
typedef unsigned char color;

typedef struct pixel pixel;
struct pixel{
    color R;
    color G;
    color B;
};
```

```
typedef struct image image;
struct image{
    color * tab;
    int w;
    int h;
};
```

Tout d'abord, "color" est un alias que j'ai donné au type unsigned char (il occupe précisément 1 octet dans la mémoire), il a été mise en place pour quantifier la couleur dans le système RGB, en effet, dans ce dernier, les valeurs que peuvent prendre une couleur est comprise dans l'intervalle [0;255].

Par extension, on arrive à notre structure pixel, qui est constitué de 3 composants "color" RGB.

Enfin, une image sera pour nous une structure composée de 3 éléments:

Le premier, "tab" est le tableau de couleur qu'il faudra allouer dynamiquement selon le besoin.

"w" et "h" sont les variables qui indiquent les dimensions de l'image.

J'ai choisi de prendre un tableau à simple entrée comme nous avons déjà les dimensions de l'image, la manipulation des indices n'est pas fondamentalement plus dur d'un tableau à 2 entrées, ce n'est qu'une histoire de conversion entre les coordonnées x et y.

A vrai dire, dans les premières versions, il s'agit plutôt d'un pointeur vers une structure "pixel" que nous avons utilisée. Mais comme dans la consigne, il est dit qu'il faut utiliser une matrice de (XMAX x YMAX x 3) octets, nous sommes revenue vers le pointeur vers "color".

6 Explications sur les fonctions utilisées

6.1 image.c

pixel build_pixel(color tab, int pos)

retourne une structure pixel à partir des couleurs qui se situe en position "pos" du tableau de couleur donné en paramètre.

void modify_pixel(color tab, int pos, color R, color G, color B)*

Comme son nom l'indique, il modifie les couleurs situées à la position "pos" du tableau avec les paramètres donnés.

*void affiche_pixel(color * tab, int pos)*

Affiche le trio de couleur situé à la position pos.

image create_image(int w, int h)

C'est la fonction d'initialisation d'image, il retourne une structure image avec la bonne dimension tout en allouant la mémoire qu'il faut pour le tableau. Cependant, il ne remplit pas le tableau avec une valeur de défaut.

image cpy_image(image src)

Crée une copie de l'image src et la renvoie.

image create_test_image(int i)

La fonction retourne différents images de test selon l'argument donnée en paramètre.

*int compare_pixel(pixel pi, color * tab, int pos, int option)*

Retourne le résultat de la comparaison entre le pixel et le trio de couleurs à la position pos.

"option" correspond en réalité à un macro qui se situe dans "image.h"

S'il est nul, alors la fonction fait un test strict entre les 2 couleurs.

Sinon on fait alors une comparaison "soft", c'est à dire qu'on considère identique 2 pixel s'ils ont des intensité de couleur proche. Il y a aussi 2 macro dédiés "DIFF_TOT" et "DIFF_MONO": "DIFF_MONO" est la différence maximale qu'on accepte pour un seul couleur. "DIFF_TOT" est la différence maximale qu'on accepte pour la totalité des couleurs,

void affiche_image(image img)

Permet d'afficher l'image sur l'écran du terminale, le résultat est bien sûr très moche, et devient rapidement illisible pour de grande dimension, cependant, il a été très utile pour visualiser rapidement l'image pendant la phase de développement du projet, l'affiche via OpenGL n'intervient que très tardivement dans notre projet.

6.2 propager.c

int propager(image g, int tab[], int i, int num_tache, pixel p)

Dans ce fichier, il n'y a qu'une seule fonction, c'est la fonction de test pour identifier les différentes taches de couleurs.

Néanmoins, elle est très facile à comprendre.

"tab" est un tableau constitué de "0" à l'état initial, un peu comme ce que nous avons appris dans votre cours pour les arbres, les cases non traitées contiennent des "0", à l'inverse, si une case n'est pas nul, il contient forcément le numéro de tache qu'on lui a attribué. (il est instantié par la fonction supérieur qui appellera propager).

"i" est le numéro de case que la fonction est en train de traiter.

Enfin "p" est la couleur de référence pour faire les tests de comparaison.

Au départ, il faut vérifier si la case "i" est déjà traité ou non, dans le cas nul, il faut lancer le test de couleur pour voir si les 2 pixels(trio de couleurs et pixel par abus) sont identique. Si la comparaison s'avère bonne, la case "i" est marqué avec le numéro de tache donné en argument. Puis la fonction se propage récursivement dans les 4 cases en proximité (gauche, haut, droite et bas).

6.3 traitement.c

```
void affiche_tab(image g,int tab[])
```

Cette fonction a pour but d'afficher le tableau "tab" cité dans la section précédente, c'est pour des besoin de visualisation lors du developpement qu'elle a été crée.

```
void init_tab(image g,int tab[])
```

Comme son nom l'indique, la fonction initialise toutes les cases du tableau "tab" avec "0", à vrai dire, il n'est pas nécessaire de donner image g en argument, nous avons juste besoin de connaître les dimensions du tableau voir même juste sa taille dans ce cas présent. Mais bon, c'est pas une fonction que nous utilisons dans toutes les fonctions, elle a été crée car nous voulions segmenter au maximum les taches.

Et en effet, la fonction a été utilisée au moins 2 fois, ce qui justifie amplement son utilité et d'éviter de copier-coller 2 boucle for identique.

