Structure de données et programmation avancée :

Ligne de partage des eaux

Enseignant :
CHAUSSARD John

Nom du binôme :
EL AMRANI Ismail 11928769
BEN YAHIA Ilyes 11707486

2019/2020

Table des matières :

Introduction **4**

Structure d’image **5**

Calculer l’image de gradient **7**

Calculer la ligne de partage des eaux **9**

Aller plus loin **12**

Conclusion **14**

I. Introduction

Dans le cadre de la matiere structure de données et programmation avancée , on nous a demandé de faire un projet qui porte sur la ligne des partages des eaux .Dans ce rapport , on va essayer de vous parler de ce projet , de son objectif , et de vous montrer les résultats de notre travail.

Le sujet porte sur la ligne des partage des eaux qui est un algorithme permettant de segmenter un objet dans une image, c'est à dire trouver avec précision les contours de cet objet , et c'est ce qu'on va essayer de le montrer sur ce rapport à travers des résultats de notre programme sur des différentes images .

Le rapport contiendra aussi les difficultés qu'on a trouvé en traitant ce projet , les fautes qu'on avait commis et leur solution , des explications des différentes fonctions utilisées sur le code et finalement la réponse des questions posées sur l'énoncé du projet .

II. Structure d'image

Pour stocker les données relatives à l'image ,on a proposé une structure de données myimage où on va stocker sa largeur et son hauteur , ainsi qu'un tableau de 2 dimensions qui va representer le canal rouge de l'image .

```
typedef struct myimage{
        uint32_t hauteur;
        uint32_t largeur;
        uint8_t** tab;
 }myimage;
```

La fonction:
myimage *lireimage(**char*** nom_fichier)

- En s'inspirant du corrigé du TP3 d'informatique de base on a proposé une fonction Lireimage qui prend en argument le nom du fichier PNG et lie l'image et stocke ses données dans la structure result : On utilise la fonction lodepng_decode32_file qui donne un tableau de unsigned char de type RVBa qu'on a appelé image "**unsigned char*** image" , où la première , la deuxième , la troisième , et la quatrième case du tableau indique respectivement , la quantité du rouge , la quantité du vert , la quantité du bleu , et la transparence du premier pixel de l'image , et puis on passe au deuxième pixel .

- Or la fonction a pour but , stocker les données de l'image dans la structure result , donc on a alloué la structure result :

```
myimage *result=Allouerimage(hauteur,largeur); avec hauteur est la hauteur de l'image et largeur est la largeur de l'image
```

- Le tableau de la structure result reste toujours vide , donc on doit le remplir par les quantités du rouge des pixels de l'image , mais le tableau obtenu par lodepng_decode32_file ne contient pas que la quantité du rouge , c'est pour cette raison qu'on a fait la boucle suivante :

```
for(i=0;i<hauteur;i++){
                for(j=0;j<largeur;j++)
                        result->tab[i][j]=image[(i*largeur+j)*4];
        }
```

La fonction:
void Ecrireimage(myimage* im,**char*** nom_fichier)

- De même , on a proposé la fonction Ecrireimage qui prend en argument l'image im lue par la fonction Lireimage et un nom du fichier. Cette fois , on remplit le tableau de unsigned char de type RVBa qu'on a appelé image_array de la manière suivante :

```
for(i=0;i<im->hauteur;i++){
        for(j=0;j<im->largeur;j++){
            image_array[(i*im->largeur+j)*4]=im->tab[i][j];
            image_array[((i*im->largeur+j)*4)+1]=im->tab[i][j];
            image_array[((i*im->largeur+j)*4)+2]=im->tab[i][j];
            image_array[((i*im->largeur+j)*4)+3]=255;
        }
```

- Les 3 premiers cases sont égales car L'image étant en noir et blanc, la valeur de rouge de chaque pixel est égale à sa valeur de bleu ainsi qu'à sa valeur de vert , et la transparence est égale à 255 .

La fonction:
void LibererImage(myimage* im)

- C'est une fonction qui sert à libérer de la mémoire de la structure de données im qui représentant l'image

La fonction:
myimage *Allouerimage(uint32_t hauteur,uint32_t largeur)

- Cette fonction prend en argument la largeur , et la hauteur de l'image qu'on veut créer et renvoie une structure de données représentant une image vide de hauteur hauteur et largeur largeur .

III. Calculer l'image de gradient

La fonction:
myimage *CalculerGradient(myimage *im, uint32_t r)

- En s'inspirant l'**Algorithme 1** , on a réussi à proposer la fonction CalculerGradient qui sert à calculer le gradient de l'image

```
for(i=0;i<(im->hauteur);i++){
        for(j=0;j<(im->largeur);j++){
            //Pour chaque couple (i,j) on initialise max à 0 et min à 255
            max=0;
            min=255;
            for(k=i-r;k<i+r;k++){
                for(l=j-r;l<j+r;l++){
                    //Pour k de i-r à i+r et l de j-r à j+r On vérifie que le pixel(k,l) ne sort pas de l'image im , si oui on l'ignore
                    if(k>=0 && k<im->hauteur && l>=0 && l<im->largeur){
                        // On cherche le maximum et le minimum de im->tab pour pour k de i-r à i+r et l de j-r à j+r
                        if(im->tab[k][l]>max)
                            max=im->tab[k][l];
                        if(im->tab[k][l]<min)
                            min=im->tab[k][l];
                    }
                }
            }
            gradient->tab[i][j]=max-min;
        }
    }
```

- La condition if(k>=0 && k<im->hauteur && l>=0 && l<im->largeur) : Elle sert à ignorer le pixel (k,l) qui "sort" de l'image , et on prend compte que les pixels qui vérifient la condition.

On a pris cette image pour faire des tests :

Après l’application de l’algorithme, on aura cette image:

IV. Calculer la ligne de partage des eaux

Pour calculer l'image de la ligne de partage des eaux on a utilisé un algorithme fournie avec les énoncés du projet qui prend deux images gradient et marquer et modifier l'image marqueur pour donner la même valeur à tous les pixels à l'intérieur de l'objet et une autre valeur pour les pixels à l'extérieur (les valeurs sont définie dans l'image marqueur dans le projet pour la partie 4 il nous a été demandé de représenter les pixels hors de l'objet par 100 et les pixels dans l'objet par 200)

Pour implémenter cet algorithme nous avons utilisé une structure de données pour stocker les couples (i,j) qui représente les coordonnés des pixels qu'ont une valeur différente de 0 (pas noire). Mais il faut les stocker dans l'ordre croissant pour la suite de l'algorithme.
Et donc pour implémenter la structure on aura pu utiliser une liste triée par les valeur de Gradient. Mais en utilisant cette structure notre algorithme prend beaucoup de temps lors de l'exécution car la fonction ajouter_pixel qui insère un couple (i,j) dans une liste elle doit la parcourir pour le mettre dans le bon endroit.
Pour cela on a pensé à utiliser une structure de table de hachage que insère les couple(i,j) au bon endroit et avec une complexité en temps qui égale O(1).

```
typedef struct maillon{
        uint32_t i;
        uint32_t j;
        struct maillon* suivant;
}maillon;
```

- Une structure d'un maillon pour stocker les coordonnés d'un pixel dans la liste

```
typedef struct clpe{
        uint32_t taille;
        maillon* debut;
}clpe;
```

- Une structure qui stocke une liste de maillon

```
typedef struct tab_clpe{
        clpe* tab[256];
}tab_clpe;
```

- Une structure de table de hachage qui contient un tableau de 256 case qui pointe sur une liste de maillon et chaque case tab[k] contient la liste de toutes les coordonnées (i,j) qui on k comme valeur de leur gradient.

tab_clpe* newtabclpe();

- Une fonction qui crée un nouveau table de hachage vide et initialise toutes les cases à une liste vide (taille = 0 et debut pointe vers null).

maillon* newmaillon(uint32_t i,uint32_t j);

- Une fonction qui crée une nouvelle maillon en initialisant c'est valeur à i et j qui son entrée en paramètre.

void ajouter_pixel(tab_clpe* L,uint32_t i,uint32_t j,uint8_t gradient);

- Une fonction qui prend les coordonne d'un pixel et son gradient et crée un maillon avec les coordonnées (i,j) et l'insère dans la bonne liste du table de hachage grâce au gradient.

_Bool estvide(tab_clpe* L);

- Une fonction pour vérifier si un table de hachage est vide ou pas : elle renvoie 1 si le table est vide sinon 0.

maillon* retirer_pexel(tab_clpe* L);

- Une fonction qui renvoi le premier maillon dans le table de hachage.
  Cette fonction commence par L->tab[0] si elle n'est pas vide elle retire le premier maillon le renvoie sinon elle passe à L->tab[1] sachant que la liste n'est pas vide car la fonction **CalculerLPE** verifie ca avant d'appeler cette fonction.

void CalculerLPE(myimage* Gradient, myimage* M);

- Une fonction qui suit l'algorithme de calcule de la ligne de partage des eaux.

Après l’application de l’algorithme, on aura cette image:

V. Aller plus loin

- Comment mettre en place plusieurs objets à détecter dans l'image ?

Pour répondre à cette question on a pensé à prendre une image marqueur avec avec des pixels à une valeur 240 qui représente des pixels dans l'objet 1, des points à une valeur 160 qui représente des pixels dans l'objet 2 et des points à une valeur 80 qui représente des pixels en dehors des deux objets.
mais cette réponse reste une hypothèse parce que malheureusement on a pas pu faire l'image marqueur.

- Comment représenter les résultats de l'algorithme non pas en niveaux de gris, mais en couleur (avec des couleurs sélectionnées aléatoirement)

Pour répondre à cette question on a créé la fonction
```
void EcrireimageclpeColeur(myimage* im,char* nom_fichier)
```

qui est identique à la fonction
```
void EcrireimageclpeColeur(myimage* im,char* nom_fichier)
```

mais elle crée deux couleur différente et elle change tous les pixels qui ont la même valeur(100 et 200) avec une des deux couleur sélectionnées aléatoirement avant de créer l'image.

Après l'application de la fonction, on aura cette image:

- Pour calculer le temps d'exécution on a rajouter deux variable pour calculer le temps de d'exécution de l'algorithme de la ligne de partage des eaux et un deuxième pour calculer le temps totale d'exécution de programme.

- Comment représenter les résultats en faisant ressortir le contour des objets en rouge sur l'image résultat ?

Pour répondre à cette question on a créé la fonction
void EcrireimageContourRouge(myimage* im, myimage* M,**char*** nom_fichier)

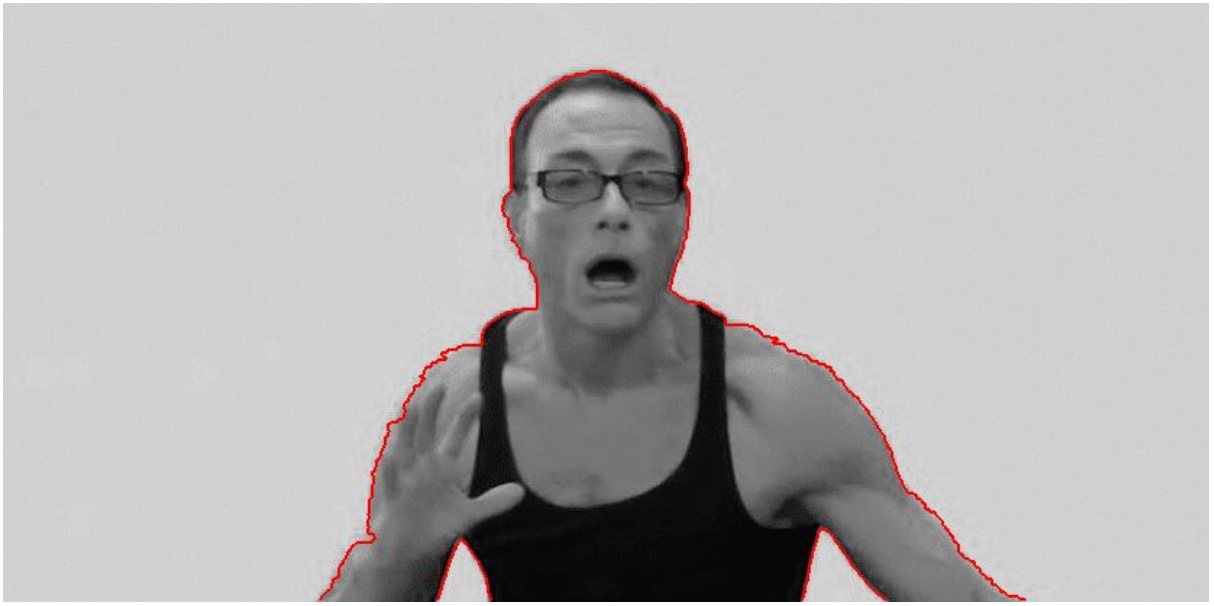
cette fonction prend deux structure d'image et un nom de fichier dans lequel elle va sauvegarder l'image.
La première structure "im" contient l'image initial sur laquelle on va faire la modification et la créé à la fin de la fonction.
la deuxième structure "M" contient l'image renvoyée par l'algorithme de ligne de partage des eaux.

le principe de l'algorithme suivi est :

```
pour chaque pixel(i, j) de l'image M faire
   si M(i, j)!= M(i-1, j) et M(i, j)!= M(i+1, j) et M(i, j)!= M(i, j-1) et M(i, j)!= M(i, j+1) alors
      mettre ce pixel en rouge
   fin
fin
```

lors du test si le pixel est différent de ses voisins ou pas, tous les pixels qui sortent de l'image on l'ignore.

Après l'application de la fonction, on aura cette image:



VI. Conclusion

Lors de ce projet , on a réussi à faire marcher l'algorithme de la ligne de partage des eaux qui a pour objectif de segmenter un objet dans une image , mais malheureusement , on n'a pas réussi à créer des marqueurs des autres images , pour tester notre algorithme sur nos propres images et pour essayer notre hypothèse à détecter deux objets, Mais nous avons réussi à détecter un unique objet dans une image et à ressortir ses contour en rouge .

Cependant , ce projet était intéressant parce qu'il nous a donné la chance de connaître une nouvelle librairie LodePNG qu'on a utilisé sur notre travail pour lire des images , pour les créer , et aussi pour calculer la ligne des partages des eaux .