PROGRAMMATION IMPÉRATIVE II (2018)

projet 3 : Images et formes

LIES AMAROUCHE

ETIENNE PENAULT

03 DÉCEMBRE 2018

1 Introduction :

Notre projet porte sur les images ainsi que les formes. Son principe est simple: A travers plusieurs étapes comme la récupération ou le traitement des pixels, pouvoir récupérer les formes que possède une image et les afficher. A travers ce rapport nous verrons comment la réalisation de ce projet à pu être possible, son fonctionnement, les objectifs fixés, nos difficultés rencontrées lors de sa conception, etc.

2 Utilisation

Pour utiliser notre programme, il faudra au préalable avoir une image à disposition, avec de préférence, des pixels à fort contrastes! Pour ce faire, l'utilisation des images est exclusivement réservée aux images PPM (P6), un format d'image qui est facile à lire du point de vue du programme.

Pour exécuter le programme, il faut passer le nom de l'image à analyser en argument, sinon, le programme lira l'image par défaut indiquée en dur dans le code (par exemple "./Exe toto.ppm" avec une image nommée "toto.ppm").

Une fois le programme lancé, nous avons à faire à une fenêtre. Cette dernière affiche l'image en question. Après un clique gauche sur l'image, un menu déroulant apparaît avec une liste d'options:

1. Quitter
2. Image originale
3. Pixels a fort contraste
4. Pixels a fort contraste connexes
5. Alignements (Affiche identique à "Pixels à fort contraste connexes")
6. Formes

Une fois une options sélectionnée (hormis "Quitter"), l'image modifiée apparaît. Nous pouvons choisir n'importe quelle proposition de cette liste dans n'importe quel ordre, revenir à la proposition d'avant, etc.

Pour quitter le programme nous pouvons soit:

- Taper sur "q"
- Taper sur "esc"
- Clique gauche sur l'image –> Quitter

3 Fonctionnement du programme

En premier lieu, nous nous nous sommes imprégnés du sujet, par une documentation sur les images PPM sur internet (le cours n'étant pas encore disponible à ce moment là) et par des questions au professeur pour éviter de trop dévier du sujet.

Lors du commencement du projet, nous étions parti sur une utilisation des images PPM, mais P3, c'est a dire que contrairement aux P6, les données stockées dans l'images sont en ASCII et non en binaire. Nous avons vu avec le professeur à travers des questions et finalement nous avons utilisé les PPM P6 car la méthode de lecture était déjà faites sur le site, nous avions plus qu'à réutiliser la fonction pour lire les données de l'image.

Une fois les données de l'images récupérées, on les stocks dans un tableau de pixels pour faciliter le traitement.
Il fallait donc extraire les pixels à fort contraste de ce tableau. Pour cela, en préambule, il fallait calculer la distance des couleurs entres deux pixels a l'aide de la fonction "calculContraste" dans le fichier "contraste.c". Cette fonction est appelée dans la fonction "remplirVectorPixel" du fichier "pixel.c" qui retourne un nouveau tableau de pixels, mais avec uniquement des pixels à fort contraste (cette fonction va parcourir le tableau et ainsi comparer les pixels avec ses voisins. Grâce à la fonction "testContraste" du fichier "constraste.c", on va juger si cette distance est suffisante, si oui, on ajoute le pixel à notre nouveau vecteur, sinon on ne fait rien). Une fois notre vecteur créé, on le transforme en donnée image pour ainsi l'afficher par la suite.

Avec notre nouveau tableau de pixels fortement contrastés, il faut que nous récupérions uniquement les pixels fortement contrastés qui se touchent, c'est à dire connexes! Pour se faire, nous créons un nouveau vecteur de pixels par le biais de la fonction "vectorPixelConnexes" du fichier "pixel.c". Cette dernière va comparer la position du pixel courant et vérifier si d'après ses coordonnées, il possède un voisin direct, si c'est le cas, on rajoute le pixel à notre tableau. Une fois le vecteur récupéré, on le retranscrit en image pour pouvoir actualiser l'affichage. On sauvegarde ce vecteur dans une nouvelle variable image.

A partir de notre vecteur de pixels, nous allons créer un vecteur d'alignement, c'est à dire un vecteur, de vecteur de pixels, qui forment un alignement ou une courbe continue. Pour ce faire, nous allons utiliser la fonction "vectorAlign" du fichier "alignement.c". Le rendu image de ce vecteur est identique au précédent, mais contrairement au précédent, le vecteur d'alignement nous sera utile pour gérer la détection de formes contrairement à celui des forts contrastes connexes. On créer une nouvelle image à afficher, pour voir que nous ne nous sommes pas trompés, même si elle est identique à celles des pixels à fort contraste connexes. On stock ces données dans un variable image.

Grâce à "vectorForme" du fichier "forms.c", on va pouvoir retourner un vecteur d'alignement, mais uniquement ceux qui créaient les formes, c'est à dire que le vecteur retourné, ne possède pas tous les nuages de points, contrairement à l'image des alignements.Pour ce faire, dans cette fonction, on va analyser, pixel par pixel, les alentours de ce dernier, sur une distance dépendant de la taille de l'image en question.
Si cette zone autour de ce pixel possède un nombre de pixel supérieur au seuil qui est définit proportionnellement à la taille de l'image, dans le code (SEUIL), alors on ne fait rien, sinon on l'ajoute. On converti ces données en une variable image.

REMARQUE: Lors de l'exécution du programme, on créé donc différentes variables de types images pour chaque cas possible.

Pour finir, le programme possède un interface graphique OpenGL. Cette dernière à été implémentée dans le fichier "affichage.c" à partir de l'exemple ici même : http://www.ai.univ-paris8.fr/ jj/Cours/Algo/Images/main.c.
Le cœur de l'affichage à lieu dans le switch de la fonction "menu", qui va, selon le choix de l'utilisateur, actualiser la variable "img" avec une des variables de type "image", précédemment définis dans le "main", par rapport choix de l'utilisateur.

4 Problèmes/Difficultés rencontrées

Au cours de notre projet, nous avons rencontrés plusieurs soucis. Tout d'abord, l'analyse d'image était un sujet assez abstrait pour nous car nous n'en n'avions jamais fait auparavant. Nous avons eu du mal à nous familiariser à cette nouvelle approche qu'est la décomposition d'une image pixel par pixel. De plus, nous avons commencé par utiliser les PPM P3, ce qui a engendré un changement de structure, et d'approche pour la continuation de notre projet.

Au cours de notre projet, nous n'avions pas saisi la compréhension de formes et pensions qu'une forme était à l'échelle des pixels (par exemple 4 pixels côte à côte forment un carré), et non à l'échelle visuelle, nous avions donc pris du retard par rapport à ce qui était demandé.

Pour finir, le rendu final des formes ne donne pas un résultat probant sur certaines images, par exemple celui de l'image du requin ("requinleopard.ppm") ne prend pas toute les formes (Cela doit être dû à "SEUIL" ou à "DISTANCE" dans le fichier "forms.c", qui ne doivent pas être bon pour cette image), ou encore celui de la peinture ("chatou.ppm") n'est pas du tout pertinent (Contrairement à l'image du requin, le soucis provient dès le début, la récupération de pixels à fort contraste n'est déjà pas pertinente).Cela vient surement de l'universalité de SEUIL et DISTANCE, or il est toujours possible de tester avec d'autres valeurs dans le code !

Depuis l'implémentation de l'analyse des formes, le programme met un temps considérable à se lancer car il doit effectuer beaucoup de calcul pour cette partie. Nous ne voyons pas comment faire mieux, à part baisser la distance, mais le rendu serait vraiment moins correct.

REMARQUE: Il est toujours possible de désactiver le chargement des formes en mettant les lignes 48 et 49 de "main.c" en commentaire:
//f = vectorForme(z, l);
//b = afficheImageAlignement(img, f);

5 Choix

Nous avons fait le choix de séquencer au maximum notre travail, c'est à dire, faire un fichier par registre de fonction, question de clarté, organiser notre header selon les fonctions de chaque fichier etc.

Dans notre projet, nous avons préféré utiliser les vecteurs au dépend des listes pour une raison de facilité, nous sommes amenés à parcourir plusieurs fois nos vecteurs ou encore comparer leurs valeurs. Cela aurait été, selon nous, plus complexes pour un résultat identique.

6 Conclusion

Pour conclure, ce projet nous aura permis d'avoir un avant goût du domaine de l'image en informatique, plus particulièrement en programmation. Même si le sujet avait l'air assez distant de ce qu'on a l'habitude de faire en programmation, nous avons réussi à avoir un résultat visuel assez concret ce qui est un minimum gratifiant. Malheureusement, notre programme est encore perfectible, le rendu des formes n'est pas parfait sur toute les images.

7 Code Source

7.1 ppm.h

```
#include <GL/gl.h>

/*DEFINITION DES STRUCTURES*/

struct Pixel{
GLubyte r,g,b; //unsigned char car un pixel comporte 255
               couleurs, comme le nombre de bytes d'un unsigned char.
};
typedef struct Pixel pixel;

struct coord
{
int x,y;
};
typedef struct coord coord;


struct Image{
int hauteur, largeur;
pixel * pix;
};
typedef struct Image image_t;
typedef struct Image * image; //Evite de passer image_t* pour l'utilisation


struct VectorPixel{
coord * tab;
int n;
int cap;
};
typedef struct VectorPixel vectorPixel_t;
typedef struct VectorPixel * vectorPixel;

struct VectorAlignement{
vectorPixel * tab;
int n;
int cap;
};
typedef struct VectorAlignement vectorAlignement_t;
typedef struct VectorAlignement * vectorAlignement;

/*********************************************************************/
/*********************************************************************/

/*DEFINITION DES PROTOTYPES*/
```

```
/*Image*/
image charger(const char *); //FAIT//// Prend le nom de l'image et retourne
l'image en question (Enfait ses données).
image imageVierge(int,int); //FAIT//// Hauteur,Largeur.
void supprimer(image); //FAIT////
image afficheimagemodifier(image, vectorPixel);//FAIT////
image afficheImageAlignement(image , vectorAlignement);
image CopieImage(image);//FAIT////

/*Pixel*/
pixel getPixel(image, int, int); //FAIT//// Image, coordonnées x et y.
void SetPixel(image,int,int,pixel);//FAIT////
int pixelConnexes(coord,coord); //FAIT//// Test si deux pixels
 se touches (Diagonale comprise).
void remplirVectorPixel(image , vectorPixel);//FAIT////
int inVectorPixel(vectorPixel, coord);//FAIT////
void afficheVectorPixel(vectorPixel);//FAIT////
vectorPixel newVectorPixel(); //FAIT////
void pushBackPixel(vectorPixel, coord); //FAIT////
void pop_back(vectorPixel v);//FAIT////
void deleteVectorPixel(vectorPixel); //FAIT////
void reserveCoordonnes(vectorPixel, int); //FAIT////
int size(vectorPixel);//FAIT////
vectorPixel vectorPixelConnexes(vectorPixel);//FAIT////
vectorPixel copy(vectorPixel);//FAIT////
int equalcoor(coord , coord);//FAIT////
int empty(vectorPixel);//FAIT////
void clear(vectorPixel);//FAIT///

/*Contraste*/
double calculContraste(pixel, pixel); //FAIT////
int testContraste(double); //FAIT//// Test si le contraste est suffisament grand
pour le garder.


/*Vector d'Alignements*/
void afficheVectorAlignement(vectorAlignement); //FAIT////
vectorAlignement newVectorAlignement(); //FAIT////
vectorAlignement trouveAlignement(vectorPixel); //Renvoie un tableau,
contenant les Alignements.
void pushBackAlignement(vectorAlignement, vectorPixel); //FAIT////
void deleteVectorAlignement(vectorAlignement); //FAIT////
void reserveAlignement(vectorAlignement, int); //FAIT////
int Nbalignes (vectorPixel );//FAIT////
vectorAlignement vectorAlign(vectorPixel);//FAIT////
int coleinaire(coord,coord,coord);//FAIT////
int ligne (vectorPixel);//FAIT////
vectorAlignement vectorligne(vectorAlignement);//FAIT////
vectorAlignement decoupe(vectorPixel);//FAIT////
```

```
/*Mémoire*/
void* my_malloc(unsigned int); //FAIT//// unsigned int == size_t.
void* my_realloc(void *, unsigned int);//FAIT////

/*Affichage*/
void init();//FAIT////
void Reshape(int, int);//FAIT////
void Draw();//FAIT////
void menu(int);//FAIT////
void Keyboard(unsigned char, int, int);//FAIT////
void Mouse(int, int, int, int);//FAIT////

/*Forme*/
vectorAlignement vectorForme(image, vectorAlignement);//FAIT////
int voisin(image I, int largeur, int hauteur, vectorPixel v, int taille);//FAIT////
int isole(image,vectorPixel);//FAIT////
```

7.2 contraste.c

```
#include<stdio.h>
#include <math.h> //On ajoute -lm à la compilation.
#include "ppm.h"

#define SEUIL_CONTRASTE 7 //10

/*****************CONTRASTE*******************/

/*PREDICAT QUI TEST SI LE PIXEL EST CONTRASTÉ*/

int testContraste(double x){
return x > SEUIL_CONTRASTE; //SEUIL_CONTRASTE à déclarer dans le define
 en début de fichier.
}

/*CALCUL LE CONTRASTE ENTRE DEUX PIXELS*/

double calculContraste(pixel x, pixel y){
return sqrt(((y.r - x.r)*(y.r - x.r)) +
((y.g - x.g)*(y.g - x.g)) +
((y.b - x.b)*(y.b - x.b)));
}
```

7.3 image.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include "ppm.h"
#define CREATOR "RPFELGUEIRAS"
```

```
/*********************IMAGE*********************/

/*CREER UN NOUVELLE IMAGE*/

image imageVierge(int l, int h){
/* allocation memoire */
image img = (image) malloc(sizeof(image_t));
assert(img);
img->pix = (pixel *) malloc(sizeof(pixel)*l*h);
    assert(img->pix);
img->hauteur = h;
    img->largeur = l;
    return img;
}

/*CHARGE L'IMAGE, GROSSE PARTIE DU CODE PRISE SUR http://www.ai.univ-paris8.fr
/~jj/Cours/Algo/Images/ppm.c*/
image charger(const char* filename){
    char buff[16];
    image img;
    FILE *fp;
    int c,rgb_comp_color, size ,i , j;
    pixel a;
    //open PPM file for reading
    fp = fopen(filename, "rb");
    if (!fp) {
        fprintf(stderr, "Impossible d'ouvrir le fichier: '%s'\n", filename);
        exit(1);
    }

    //read image format
    if (!fgets(buff, sizeof(buff), fp)) {
        perror(filename);
        exit(1);
    }

    //check the image format
    if (buff[0] != 'P' || buff[1] != '6') {
         fprintf(stderr, "Format invalide (uniquement 'P6')\n");
         exit(1);
    }

    //alloc memory form image
    img = (image)malloc(sizeof(image_t));
    if (!img) {
         fprintf(stderr, "Erreur mémoire\n");
         exit(1);
    }
```

```
//check for comments
c = getc(fp);


ungetc(c, fp);
//read image size information
if (fscanf(fp, "%d %d", &img->largeur, &img->hauteur) != 2) {
     fprintf(stderr, "Erreur du chargement de la taille de l'image\n");
     exit(1);
}

//read rgb component
if (fscanf(fp, "%d", &rgb_comp_color) != 1) {
     fprintf(stderr, "Invalid rgb component (error loading '%s')\n", filename);
     exit(1);
}

//check rgb component depth
if (rgb_comp_color!= 255) {
     fprintf(stderr, "'%s' does not have 8-bits components\n", filename);
     exit(1);
}

while (fgetc(fp) != '\n') ;
//memory allocation for pixel data
img->pix = (pixel*)malloc(img->largeur * img->hauteur * sizeof(pixel));

if (!img) {
     fprintf(stderr, "Erreur mémoire\n");
     exit(1);
}

if (fread(img->pix, 3 * img->largeur, img->hauteur, fp)
!= (unsigned int)img->hauteur) {
     fprintf(stderr, "Erreur la du chargement de l'image '%s'\n", filename);
     exit(1);
}

/*RETOURNE L'IMAGE*/

for (i = 0;i <img->largeur; ++i) {
    for(size = img ->hauteur -1,j=0;j<size;j++,--size){
     a = img->pix[img->largeur*j+i];
     img->pix[img->largeur*j+i]= img->pix[img->largeur*size+i];
     img->pix[img->largeur *size+i]=a;

    }
}
```

```
    fclose(fp);
    return img;
}

/*LIBÈRE LA MÉMOIRE DE L'IMAGE*/

void supprimer(image img){
    if(img){
        free(img->pix);
        free(img);
    }
    else
        return;
}

/*RETOURNE L'IMAGE IDENTIQUE A ELLE EN ARGUMENT*/

image CopieImage(image I){
    image res;
    if (!I)
        return NULL;
    res = imageVierge(I->largeur,I->hauteur);
    memcpy(res->pix,I->pix,I->largeur*I->hauteur*sizeof(pixel));
    return res;
}

/*RETOURNE L'IMAGE AVEC SEULEMENT LES PIXELS CONTENUE DANS LE VECTEUR DE PIXEL*/

image afficheimagemodifier(image I , vectorPixel v){
    int i,j;
    image src;
    src =CopieImage(I);
    for(i=0;i<I->largeur;++i){
        for(j=0;j<I->hauteur;++j){
            pixel p;
            p.r = 255;
            p.g = 255;
            p.b = 255;
            SetPixel(src,i,j,p);
        }
    }
    for(i=0;i<v->n;++i){
        pixel p;
        p.r = getPixel(I,v->tab[i].x,v->tab[i].y).r;
        p.g = getPixel(I,v->tab[i].x,v->tab[i].y).g;
        p.b = getPixel(I,v->tab[i].x,v->tab[i].y).b;
        SetPixel(src,v->tab[i].x,v->tab[i].y,p);
    }
    return src;
}
```

```
/*RETOURNE L'IMAGE AVEC SEULEMENT LES PIXELS CONTENUE DANS LE VECTEUR D'ALIGNEMENT*/

image afficheImageAlignement(image I, vectorAlignement v){
int i,j;
    image src;
    src =CopieImage(I);
    for(i=0;i<I->largeur;++i){
        for(j=0;j<I->hauteur;++j){
            pixel p;
            p.r = 255;
            p.g = 255;
            p.b = 255;
            SetPixel(src,i,j,p);
        }
    }
    for(i = 0; i<v->n; i++){
for(j = 0; j<v->tab[i]->n; j++){
         pixel p;
         p.r = getPixel(I,v->tab[i]->tab[j].x,v->tab[i]->tab[j].y).r;
         p.g = getPixel(I,v->tab[i]->tab[j].x,v->tab[i]->tab[j].y).g;
         p.b = getPixel(I,v->tab[i]->tab[j].x,v->tab[i]->tab[j].y).b;
         SetPixel(src,v->tab[i]->tab[j].x,v->tab[i]->tab[j].y,p);
     }
    }
    return src;
}
```

7.4 memoire.c

```
#include <stdio.h>
#include <stdlib.h>
#include "ppm.h"

/* alloue de la memoire avec malloc et teste si malloc a bien fonctionner */

void *my_malloc(unsigned int size) {
    void *p;
    p = malloc(size);
    if(p == NULL) {
       fprintf(stderr , "Erreur : impossible d'allouer la memoire\n");
       exit(-1);
    }
    return p;
}
/* alloue de la memoire avec realloc et teste si realloc s'est bien passé */

void *my_realloc(void *p, unsigned int size) {
     p = realloc(p, size);
```

```
    if(p == NULL) {
        fprintf(stderr , "Erreur : impossible de reallouer la memoire\n");
        exit(-1);
    }
    return p;
}
```

7.5 pixel.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>

#include "ppm.h"


/************VECTEUR DE pixel*************/


vectorPixel newVectorPixel(){ //Qui permet de déclarer un vectorPixel vide
de taille 0 et capacite 0.
vectorPixel v = my_malloc(sizeof(*v));
    v->tab = my_malloc(sizeof(coord));
    v->n = 0;
    v->cap = 0;
    return v;
}

void reservePixel(vectorPixel v , int newcap){ //Permet de réserver de la mémoire pour
accueillir un nouveau élément du vectorPixel
v->tab = my_realloc(v->tab, newcap * sizeof(coord));
        v->cap = newcap;
     if(v->n > v->cap)
              v->n = v->cap;
}

void pushBackPixel(vectorPixel v, coord data){  //Qui permet de rajouter
un élément en dernière position du vectoPixel.

reservePixel(v, v->cap + 1);
v->tab[v->n] = data;
v->n++;
}
void pop_back(vectorPixel v) { //retire le dernier element du vectorPixel.
    if(v->n == 0) {
    fprintf(stderr , "Erreur : suppression du dernier element d'unvector vide\n");
    exit(-1);
    }
    v->n--;
}
vectorPixel copy(vectorPixel v) { //fais une copie du vectorPixel
     vectorPixel vcop = my_malloc(sizeof(*vcop));
     vcop->tab = my_malloc(v->n * sizeof(coord));
     memcpy(vcop->tab, v->tab, v->n * sizeof(coord));
     vcop->n = v->n;
     vcop->cap = v->n;
```

```
     return vcop;
}
int empty(vectorPixel v) { //teste si le vector Pixel est vide
    return v->n == 0;
}
int size(vectorPixel v) { //renvoie la taille du vectorPixel
     return v->n;
}
void clear(vectorPixel v) { //vide le vectorPixel
  v->tab = realloc(v->tab,sizeof(coord));
  v->n = 0;
  v->cap = 2;
}
void deleteVectorPixel(vectorPixel v){ //supprime le vectorPixel de la mémoire
free(v->tab);
    free(v);
}

void afficheVectorPixel(vectorPixel v){ //affiche le vectorPixel
int n = 0;
for(int i = 0; i<v->n; i++){
printf("\t v[%d] = x: %d\ty: %d\n", i,v->tab[i].x, v->tab[i].y);
n++;
}
printf("Il y a %d élement dans le tableau\n", n);
}

/*Dans cette fonction on parcours l'image grâce a deux boucles "for" on la parcours de
 gauche a droite de haut en bas on récupère les pixels grâce a getPixel.
On calcule la distance avec tout ses voisins et on regarde si cette distance avec
chacun de ses voisins et supérieur a notre seuil,
si c'est le cas on récupère les coordonnées de ce pixels
et les mets dans notre vectorPixel
qu'on a declarer vide et qu'on remplie de tout les pixels de fort contraste */


void remplirVectorPixel(image I, vectorPixel v){
for(int i=0;i<I->largeur;i++){
      for(int j=0;j<I->hauteur;j++){

          pixel p  = getPixel(I,i,j);


             if ( i ==0 && j==0){
                 pixel p1 = getPixel(I,i+1,j);
              pixel p2 = getPixel(I,i,j+1);
              pixel p3 = getPixel(I,i+1,j+1);
              double d1 = calculContraste(p,p1);
             double d2 = calculContraste(p,p2);
             double d3 = calculContraste(p,p3);
```

```
 if (testContraste(d1)&&testContraste(d2)&&testContraste(d3)){
        coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);


    }
   }

if ((i==0)&& (j==I->hauteur -1)){

    pixel p1 = getPixel(I,i+1,j);
    pixel p2 = getPixel(I,i,j-1);
 pixel p3 = getPixel(I,i+1,j-1);
 double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);


 if (testContraste(d1)&&testContraste(d2)&&testContraste(d3)){
        coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);


    }
   }
if (i == 0 && j!=0 && j!= I->hauteur-1){
    pixel p1 = getPixel(I,i+1,j);
 pixel p2 = getPixel(I,i+1,j-1);
 pixel p3 = getPixel(I,i+1,j+1);
 pixel p4 = getPixel(I,i,j-1);
 pixel p5 = getPixel(I,i,j+1);
    double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);
double d4 = calculContraste(p,p4);
double d5 = calculContraste(p,p5);
    if (testContraste(d1)&&testContraste(d2)&&testContraste(d3)
    &&testContraste(d4)&&testContraste(d5)){

        coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);
```

```
    }
   }

if ( i == I->largeur-1 && j==0){
 pixel p1 = getPixel(I,i-1,j);
 pixel p2 = getPixel(I,i,j+1);
 pixel p3 = getPixel(I,i-1,j+1);
 double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);
 if (testContraste(d1)&&testContraste(d2)&&testContraste(d3)){
 coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);


   }
   }
if ((i==I->largeur-1)&& (j==I->hauteur -1)){

    pixel p1 = getPixel(I,i-1,j);
 pixel p2 = getPixel(I,i,j-1);
 pixel p3 = getPixel(I,i-1,j-1);
 double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);
 if (testContraste(d1)&&testContraste(d2)&&testContraste(d3)){
 coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);


    }
   }

if (i == I->largeur-1 && j != 0 && j != I->hauteur-1){
    pixel p1 = getPixel(I,i-1,j);
 pixel p2 = getPixel(I,i-1,j-1);
    pixel p3 = getPixel(I,i-1,j+1);
 pixel p4 = getPixel(I,i,j-1);
 pixel p5 = getPixel(I,i,j+1);
    double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);
double d4 = calculContraste(p,p4);
double d5 = calculContraste(p,p5);
    if (testContraste(d1)&&testContraste(d2)&&testContraste(d3)
    &&testContraste(d4)&&testContraste(d5)){
```

```
 coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);


    }
   }

 if (j==0 && i!= 0 && i!= I->largeur-1){
 pixel p1 = getPixel(I,i+1,j);
 pixel p2 = getPixel(I,i-1,j);
 pixel p3 = getPixel(I,i-1,j+1);
 pixel p4 = getPixel(I,i,j+1);
 pixel p5 = getPixel(I,i+1,j+1);
 double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);
double d4 = calculContraste(p,p4);
double d5 = calculContraste(p,p5);
    if (testContraste(d1)&&testContraste(d2)
    &&testContraste(d3)&&testContraste(d4)
    &&testContraste(d5)){
 coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);


    }
   }
if (j == I->hauteur-1 && i!= 0 && i!= I->largeur-1){
    pixel p1 = getPixel(I,i-1,j);
 pixel p2 = getPixel(I,i+1,j);
 pixel p3 = getPixel(I,i-1,j-1);
 pixel p4 = getPixel(I,i,j-1);
 pixel p5 = getPixel(I,i+1,j-1);
 double d1 = calculContraste(p,p1);
double d2 = calculContraste(p,p2);
double d3 = calculContraste(p,p3);
double d4 = calculContraste(p,p4);
double d5 = calculContraste(p,p5);
    if (testContraste(d1)&&testContraste(d2)
    &&testContraste(d3)&&testContraste(d4)
    &&testContraste(d5)){
     coord t;
        t.x = i;
    t.y = j;
    pushBackPixel(v,t);
```

```
            }
        }

        if(i>0 && j > 0 && i<I->largeur-1 && j < I->hauteur - 1){

        pixel p1 = getPixel(I,i-1,j-1);
        pixel p2 = getPixel(I,i-1,j);
        pixel p3 = getPixel(I,i-1,j+1);
        pixel p4 = getPixel(I,i,j-1);
        pixel p5 = getPixel(I,i,j+1);
        pixel p6 = getPixel(I,i+1,j-1);
        pixel p7 = getPixel(I,i+1,j);
        pixel p8 = getPixel(I,i+1,j+1);
        double d1 = calculContraste(p,p1);
        double d2 = calculContraste(p,p2);
        double d3 = calculContraste(p,p3);
        double d4 = calculContraste(p,p4);
        double d5 = calculContraste(p,p5);
        double d6 = calculContraste(p,p6);
        double d7 = calculContraste(p,p7);
        double d8 = calculContraste(p,p8);

        if (testContraste(d1)&&testContraste(d2)
        &&testContraste(d3)&&testContraste(d4)
        &&testContraste(d5)&&testContraste(d6)
        &&testContraste(d7)&&testContraste(d8)){

               coord t;
                t.x = i;
            t.y = j;
            pushBackPixel(v,t);


           }
           }

    }
  }
}

int equalcoor(coord a , coord b){ //teste si deux
pixels ont la même coordonnées.
     return ((a.x == b.x) && (a.y == b.y)) ;
}
vectorPixel vectorPixelConnexes(vectorPixel v){
vectorPixel v1 = newVectorPixel();
int i=0;
while(i<v->n-1){
if(pixelConnexes(v->tab[i],v->tab[i+1])){
```

```
if(!equalcoor(v1->tab[v1->n-1],v->tab[i]))
pushBackPixel(v1,v->tab[i]);
pushBackPixel(v1,v->tab[i+1]);
}
i=i+1;
}
return v1;
}

/******************PIXEL*********************/


pixel getPixel(image img, int largeur, int hauteur){ //permet de renvoyer un pixel
du tableau a partir de ses coordonnées
if(img){
return img->pix[img->largeur*hauteur+largeur];
}
printf("Erreur lors de la récuperation du Pixel");
return img->pix[0];
}
void SetPixel(image I,int largeur,int hauteur,pixel p) //permet de sauvegarder
les changements qu'on opérer sur un pixel a partir de ses coordonnées.
{
assert(I && largeur>=0 && largeur<I->largeur && hauteur>=0 && hauteur<I->hauteur);
I->pix[I->largeur*hauteur+largeur] = p;
}

int pixelConnexes(coord a,coord b){ //teste di deux pixels sont connexes
return(((a.x == b.x) && (a.y == b.y+1)) ||
   ((a.x == b.x) && (a.y == b.y-1)) ||
   ((a.x == b.x-1) && (a.y == b.y)) ||
   ((a.x == b.x+1) && (a.y == b.y)) ||
   ((a.x == b.x+1) && (a.y == b.y+1)) ||
   ((a.x == b.x-1) && (a.y == b.y+1)) ||
   ((a.x == b.x+1) && (a.y == b.y-1)) ||
   ((a.x == b.x-1) && (a.y == b.y-1)));
}
```

7.6 alignement.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ppm.h"

int coleinaire(coord a ,coord b,coord c){ //Teste si trois pixels sont sur la même ligne.
   coord ab ,ac ;
   int t;
```

```
   ab.x= b.x - a.x;
   ab.y= b.y - a.y;
   ac.x= c.x - a.x;
   ac.y= c.y - a.y;
   t = ab.x * ac.y - ac.x * ab.y;

   return (!t);
}

int ligne (vectorPixel v){ //prend en argument un vectorPixel et
 teste si les pixels qui le composent forment une ligne.
  int i;
  if (size(v) < 2)
    return 0;
  if (size(v)== 2)
    return 1;
  if(size(v) > 2){
    for(i=0;i<v->n-2;i++){
      if(!(coleinaire(v->tab[i],v->tab[i+1],v->tab[i+2])))
        return 0;
    }
    return 1;
  }
  return 0;
}

int linair(vectorPixel v , coord p){ /* rends en argument un vectorPixel et des coordonnes
d'un pixel,teste si les coordonnes
des pixels qui composent ce vectorPixel
forme une ligne
avec les coordonnes du pixel qu'on a mis en argument.*/
  pushBackPixel(v,p);
  if(ligne(v)){
    pop_back(v);
    return 1;
  }
  pop_back(v);
  return 0;
}

vectorAlignement decoupe(vectorPixel v){  /*Prend un vectorPixel en argument et renvoie
un vectorAlignement.Cette fonction
parcours le vectorPixel et le découpe en ligne
a chaque fois qu'on trouve
une ligne on la met dans le vectorAlignement
a la fin on le renvoie.*/
  int i;
  vectorAlignement a = newVectorAlignement();
  vectorPixel v1=newVectorPixel();
  vectorPixel v2=newVectorPixel();
```

```
  pushBackPixel(v1,v->tab[0]);
  pushBackPixel(v1,v->tab[1]);
  for(i=2;i<v->n;i++){
    if(linair(v1,v->tab[i]))
      pushBackPixel(v1,v->tab[i]);
    else{
      v2 = copy(v1);
      pushBackAlignement(a,v2);
      clear(v1);
      pushBackPixel(v1,v->tab[i-1]);
      pushBackPixel(v1,v->tab[i]);
    }
  }
  pushBackAlignement(a,v1);
  return a;
}


void cleara(vectorAlignement v) {//vide le vectorAlignement
  v->tab = realloc(v->tab,sizeof(vectorPixel));
  v->n = 0;
  v->cap = 2;
}

vectorAlignement copyy(vectorAlignement v) { //fais une copie du vectorAlignement
     vectorAlignement vcop = my_malloc(sizeof(*vcop));
     vcop->tab = my_malloc(v->n * sizeof(vectorPixel));
     memcpy(vcop->tab, v->tab, v->n * sizeof(vectorPixel));
     vcop->n = v->n;
     vcop->cap = v->n;
     return vcop;
}


vectorAlignement newVectorAlignement(){  //permet de déclarer un vectoralignement vide
de taille 0 et capacite 0.
vectorAlignement v = my_malloc(sizeof(*v));
    v->tab = my_malloc(sizeof(vectorPixel));
    v->n = 0;
    v->cap = 0;
    return v;
}

void reserveAlignement(vectorAlignement v, int newcap){//Permet de réserver
de la mémoire pour accueillir
un nouvel élément du vectorAlignement
v->tab = my_realloc(v->tab, newcap * sizeof(vectorPixel));
        v->cap = newcap;
     if(v->n > v->cap)
             v->n = v->cap;
```

```
}


void pushBackAlignement(vectorAlignement v, vectorPixel data){ //Qui permet de rajouter un
vectorPixel en dernière position du vectoAlignement.
reserveAlignement(v, v->cap + 1);
v->tab[v->n] = data;
v->n++;
}

void deleteVectorAlignement(vectorAlignement v){ //supprime le vectorPixel de la mémoire
free(v->tab);
    free(v);
}


void afficheVectorAlignement(vectorAlignement v){ //affiche le vectorAlignement
for(int i = 0; i<v->n; i++){
  printf("va[%d]=",i);
   afficheVectorPixel(v->tab[i]);
}
}
/* Cette fonction prend le vectorpixel qui est renvoyer par vectorpixelconnexes
le parcours tant que les pixels du vectorPixel forme une ligne ou une courbe continue
et met chaque lignes ou courbe dans un vectoralignement et a la fin de la boucle
on renvoie le vectoralignemet.*/
vectorAlignement vectorAlign(vectorPixel v){
  vectorAlignement a = newVectorAlignement();
  vectorPixel v1 = newVectorPixel();
  vectorPixel v2 = newVectorPixel();
  int i;
  for( i = 0;i<v->n;++i){
    if ((pixelConnexes(v->tab[i],v->tab[i+1]))){
        if(!equalcoor(v1->tab[v1->n-1],v->tab[i]))
         pushBackPixel(v1,v->tab[i]);
      pushBackPixel(v1,v->tab[i+1]);

    }
    else{
       if(!(empty(v1))){
        v2 = copy(v1);
        pushBackAlignement(a,v2);
        clear(v1);
       }
    }
  }
  return a;
}
/*Cette fonction prend le vectorAlignement renvoyer par
la fonctions vectoralign et découpe chacun de ses éléments en lignes
```

```
ce qui nous fais un vectoralignement de lignes et a la fin de la boucle on le renvoie.*/

vectorAlignement vectorligne(vectorAlignement a){
  vectorAlignement v = newVectorAlignement();
  vectorAlignement b = newVectorAlignement();
  int i,j;
  for(i=0;i<a->n;i++){
    if(ligne(a->tab[i]))
      pushBackAlignement(v,a->tab[i]);
    else{
      b = decoupe(a->tab[i]);
      for(j=0;j<b->n;++j)
        pushBackAlignement(v,b->tab[j]);

    }
  }
  return v;
}
```

7.7 forms.c

```
#include <stdio.h>
#include "ppm.h"

extern image img;

/*Les deux valeurs sont proportionnelles pour l'universalité du programme.
cette méthode est universelle par rapport a la taille de l'image, pas a la
densité de points que nous pouvons y trouver.

Plus la valeur de DISTANCE est grande, plus il y aura de calcul, donc plus
de temps (mais par contre, cela sera plus précis, par contre, il faudrait
changer la formule de SEUIL car il ne dépend pas de DISTANCE).
Donc il y a des consessions à faire entre:
-la rapidité du programme.
-la qualité des formes affichés.

Il suffit de changer la valeur avec laquelle on divise pour etre plus
ou moins exigeant sur les formes que l'on laisse.

En effet si on a une DISTANCE minuscule pour un grand seuil, le programme va
garder que très peu de points. Si l'ont veut modifier cette valeur, on ajoute/soustrait
maximum 2 a cette valeur, sinon cela prendra, soit trop de points, soit pas assez.

Le plus dur est de trouver le bon "ratio" entre distance et seuil, cela depend
en fonction de l'image, sa taille etc.

-Pépins pour l'image du requin
-Ne marche pas du tout pour l'image chatou.*/


#define SEUIL ((img->hauteur * img->largeur) / (img->hauteur+ img->largeur))
#define DISTANCE SEUIL / 19

/*****************FORMES*******************/


/*CREER UN VECTEUR NE GARDANT QUE LES CONTOUR DES FORMES*/

vectorAlignement vectorForme(image img, vectorAlignement v){
int i,j;
vectorAlignement a = newVectorAlignement();
for(i = 0; i<v->n; i++){
for(j = 0; j<v->tab[i]->n; j++){
if(isole(img, v->tab[i])) //on vérifie si l'alignement en question ne possède pas trop de
pushBackAlignement(a,v->tab[i]);//si c'est le cas on ajoute l'alignemnt a notre vecteur.
}
}
return a;
```

```
}


/*Excepté les pixels du vecteur, on vérifie les alentours du vecteur
délimité par DISTANCE (on peut imaginer que le point analysé est le milieu
d'un carré. Le pixels est donc a "DISTANCE" pixels de chaque coté. On va
parcourir tout le carré pour compter le nombre de pixels qu'il y a dedans.).

Combien de pixels il y a en dessous de SEUIL ?

Si le nombre est supérieur à ce qu'on a définie, on ne fais rien,
sinon on l'ajoute au nouveau vecteur d'alignement.
Predicat pour savoir si le vecteur de pixel est isolé d'autres pixels.*/


int voisin(image I, int largeur, int hauteur, vectorPixel v, int taille){
int nb = 0;
for(int i = largeur-DISTANCE; i<largeur+DISTANCE; ++i){
for(int j = hauteur-DISTANCE; j<hauteur+DISTANCE; ++j){
pixel p;
        p.r = getPixel(I,i,j).r;
         p.g = getPixel(I,i,j).g;
         p.b = getPixel(I,i,j).b;
if(p.r != 255 || p.g != 255 || p.b != 255 ){ //Si le pixel de l'image n'es pas blanc.
nb++;
}
}
}
if(nb < SEUIL - taille)//On enlève "taille", c'est a dire le nombre de pixels
qu'il y a dans le vecteur, car on ne veut pas les compter.
return 1;
//Peut ête utile de décommenter pour voir les pixels qui prend.
//printf("%d\n",nb);
return 0;
}

/*PERMET DE SAVOIR SI L'ALIGNEMENT NE POSSEDE PAS TROP DE POINT AUX ALENTOURS*/

int isole(image img, vectorPixel v){
int i, taille;

/*Sert a définir la taille de la variable taille*/

for(i = 0; i<v->n; i++){//on parcour la taille de l'alignement
continue;
}
if(i>DISTANCE*2) //inutile de garder la taille totale si la taille est plus grande que
2*DISTANCE, cad la largeur ou la hauteur du carré analysé.
i=DISTANCE*2;
taille = i;
```

```
for(i = 0; i<v->n; i++){
if(!voisin(img, v->tab[i].x, v->tab[i].y, v, taille)) //On appel !voisin a
chaque pixels de l'alignement.
return 0;
}
return 1;
}
```

7.8 main.c

```
#include <stdio.h>
#include "ppm.h"

#define NOM_DEFAULT "Cordiliere2_V3.ppm" //On définie le nom du fichier par defaut.

#include <GL/glut.h>
#include <GL/glu.h>

//Definit en global, car appelé en avec extern dans "affichage.c"
image img,x,y,z,a,b;

int main(int argc, char *argv[]){

    if(argc  == 2){ //Si on met un argument apres l'executable.
        img = charger(argv[1]); //Alors l'argument apres l'executable est le nom du fichie
        à charger.
    } else if(argc == 1){
        img = charger(NOM_DEFAULT); // Sinon on charge le fichier par défaut définе
        en tête de fichier.
    } else{
        printf("Nombre d'arguments invalides.\n(Appel: ./Exe OU ./Exe image.ppm)\n");
        exit(1);
    }

    vectorPixel v,v1;
vectorAlignement l,f;

    x=img; //On stock l'image originale dans x.

    /*On remplit un vecteur de pixels à fort contraste
    et on créer une nouvelle image a partir de ce vecteur*/

    v = newVectorPixel();
    remplirVectorPixel(img,v);
    y=afficheimagemodifier(img,v);

    /*On remplit un vecteur de pixels à fort contraste CONNEXES
    et on créer une nouvelle image a partir de ce vecteur*/

    v1 = vectorPixelConnexes(v);
    z=afficheimagemodifier(img,v1);

    /*On remplit un nouveau vecteur d'alignements, c'est a dire que ce vecteur
    contiens plusieur tableaux de pixels qui forment des lignes*/

    //MEME RENDU QUE LES PIXELS A FORT CONTRASTE
    CONNEXES MAIS PAS LE MEME TYPE DE VECTEUR.
    l = vectorAlign(v1);
```

```
    a = afficheImageAlignement(img, l);

    /*On remplit un nouveau vecteur de formes (qui est en fait un vecteur
    d'alignement sans les nuages de points a l'interieur des formes),
    puis on créer une nouvelle image.*/

    f = vectorForme(z, l);//On reprend l'image qu'avec les pixels fort contraste connexes
    b = afficheImageAlignement(img, f);

    /*AFFICHAGE*/

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize(0,0);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Projet 3: Contrastes et Alignements");

    init();
    glutCreateMenu(menu);

    /*LES 5 OPTIONS DE NOTRE MENU*/

    glutAddMenuEntry("Quitter", 0);
    glutAddMenuEntry("Image originale", 1);
    glutAddMenuEntry("Pixels a fort contraste", 2);
    glutAddMenuEntry("Pixels a fort contraste connexes", 3);
    glutAddMenuEntry("Alignements",4);
    glutAddMenuEntry("Forme",5);


    glutAttachMenu(GLUT_LEFT_BUTTON);


    glutDisplayFunc(Draw);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);

    glutMouseFunc(Mouse);
    glutMainLoop();

    return 0;
}
```

7.9 affichage.c

```
#include <stdio.h>

#include "ppm.h"

#include <GL/glut.h>
#include <GL/glu.h>

extern image img,x,y,z,a,b; //On passe les images du main en extern ici.

/*ON DEFINIT LE CODE ASCII DES TOUCHES EN QUESTION*/
#define ESCAPE 27
#define Q 113

void Keyboard(unsigned char key, int x, int y)  {
  switch(key){
  case ESCAPE:
    supprimer(img);
    exit(1);
    break;
  case Q:
    supprimer(img);
    exit(1);
    break;
  default:
    fprintf(stderr, "Touche inutile\n");
  }
}

void Mouse(int button, int state, int x, int y) {

  switch(button){
  case GLUT_LEFT_BUTTON:
    break;
  case GLUT_MIDDLE_BUTTON:
    break;
  case GLUT_RIGHT_BUTTON:
    break;
  }
  glutPostRedisplay();
}

void init(){
if(img == NULL) {
     printf("L'image ne s'est pas chargé correctement.\n");
     return;
   }else{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_FLAT);
```

```
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
      glutReshapeWindow(img->largeur, img->hauteur);
   }
}

void Reshape(int w, int h) {
   glViewport(0, 0, (GLsizei)w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, (GLdouble) w, 0.0, (GLdouble)h);
    glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}

void Draw(){
   glClear(GL_COLOR_BUFFER_BIT);
    glDrawPixels(img->largeur, img->hauteur, GL_RGB, GL_UNSIGNED_BYTE, img->pix);

    glFlush();
}

/*FONCTION REPRENNANT TOUT LES CHOIX DE NOTRE MENU DANS UN SWITCH*/

/*On voit ici qu'on ne charge pas l'image quand on veut l'afficher,
mais plutôt, on attribut juste la valeur de l'image à afficher à la
variable "img". La valeur de l'image à afficher est calculé au début
du programme (dans le main), cela évite de recalculer la valeur de
la nouvelle image si on veut alterner entres deux pour les comparer
(on pense ici à la detections de formes qui met un temps considérable.*/

void menu(int etat){

  switch(etat){
    case 0:
      supprimer(img);
      exit(1);
      break;
    case 1:
        img = x;
        printf("AFFICHE L'IMAGE ORIGINALE'\n");
      break;
    case 2:
        img = y;
        printf("AFFICHE LES PIXELS A FORT CONTRASTE\n");
      break;
    case 3:
        img = z;
        printf("AFFICHE LES PIXELS A FORT CONTRASTE CONNEXES\n");
      break;
    case 4:
```

```
        printf("AFFICHE LES ALIGNEMENTS\n");
        img = a;
      break;
    case 5:
     printf("AFFICHE LES FORMES\n");
     img = b;
      break;
    default:
      break;
  }
}
```

7.10 makefile

```
#Makefile Générique
#Projet 3 / PI2
#Etienne PENAULT / Lies AMAROUCHE

OBJS = main.o memoire.o contraste.o pixel.o image.o alignement.o forms.o affichage.o
OUT = Exe
CC  = gcc
FLAGS  = -O3 -c -Wall

all: $(OBJS)
$(CC) $(OBJS) -o $(OUT) -lm -lglut -lGL -lGLU
main.o: main.c
$(CC) $(FLAGS) main.c
image.o: image.c
$(CC) $(FLAGS) image.c
memoire.o: memoire.c
$(CC) $(FLAGS) memoire.c
contraste.o: contraste.c
$(CC) $(FLAGS) contraste.c
pixel.o: pixel.c
$(CC) $(FLAGS) pixel.c
alignement.o: alignement.c
$(CC) $(FLAGS) alignement.c
forms.o: forms.c
$(CC) $(FLAGS) forms.c
affichage.o: affichage.c
$(CC) $(FLAGS) affichage.c
clean:
rm -f $(OBJS) $(OUT)
```