Rapport Traitement d'image

Mouvement - Couleur

Maxime FABRE - Florian LEPOT

Table des matières

Introduction

Pour ce travail, il nous a été demandé d'implémenter un programme de détection de mouvements sur une suite d'images enregistré à partir d'une caméra. Nous avions 2 algorithmes à traiter :

- Détection de mouvement simple : On compare toutes les images à l'image de base
- Détection de mouvement par un gabarit : On établit un gabarit à partir des 50 premières images et on comparera chaque image suivante avec les valeurs établies dans le gabarit.

D'autre part, il nous était demandé de réaliser une conversion d'images en RGB vers YUV et YUV vers RGB, puis d'appliquer sur une image en couleur l'algorithme de Kirsh suivi d'un affinage.

I. Détection de mouvement

Le but de ces TP était la Réalisation d'opérateurs de détection de mouvement, dans un cadre applicatif : La vidéo surveillance : caméra fixe, détection des objets mobiles grâce au mouvement à partir de deux techniques simples : l'une de différence d'images, l'autre d'ajustement par rapport à un gabarit.

Comme dit précédemment, nous allons utiliser deux algorithmes différents pour parvenir à une détection de mouvement. Ces deux algorithmes seront plus ou moins efficaces, nous verrons cela plus tard.

1) Lecture et écriture de fichier

a. Lecture

Pour effectuer la lecture des différentes images prises par la caméra, nous séparons le nom de la 1ere image en plusieurs parties afin de pouvoir changer le numéro de l'image à lire et pouvoir ainsi enregistrer les images de résultat plus facilement.

```
// génère le nom de l'image actuelle
memset(nom, 0, sizeof (nom));
sprintf(chaine, "%04d", 0);
strcat(nom, "NImCote");
strcat(nom, chaine);
strcat(nom,".ppm");
```

Premièrement, nous initialisons à 0 le nom, ensuite nous séparons le nom de l'image en 3, avec d'une part le début de son nom, ensuite le numéro de l'image courante et enfin l'extension de cette image pour pouvoir l'ouvrir correctement.

Cette action s'effectue dans la boucle qui va parcourir toutes les images à traiter.

b. Ecriture

```
// enregsitrement de l'image résultat
memset(nomres, 0, sizeof (nomres));
strcat(nomres, "Res");
strcat(nomres, nom);
if(!(fichres = fopen(nomres,"wb"))) {
   fprintf(stderr,"ouverture du fichier Image Resultat %s impossible\n", nomres);
   break;
}
fprintf(fichres,"P6\n%d %d\n255\n",(int)ncol, (int)nlig);
EnregistreImage(fichres, imres);
printf("Résultat de l'image n: %d termine \n", num);
```

Pour l'écriture, le principe est le même que la lecture, celle-ci se fera d'autant plus facilement que le nom de l'image courante est déjà stockée. L'écriture fait partie évidemment de la boucle permettant de traiter toutes les images.

2) Différence simple du premier ordre

a. Principe

La différence d'image s'effectue pixel à pixel, dans les 3 plans RVB. Premièrement nous allons définir un seuil. Pour chaque pixel nous calculerons alors la différence entre le pixel de l'image courante (l'image que l'on traite parmi toutes celle fournit par la caméra) et l'image de référence (qui n'est pas nécessairement la première image de la séquence).

Si la différence entre les 2 pixels est inférieure au seuil dans les 3 plans RVB, alors celui-ci sera considéré comme immobile. On assignera alors pour ce pixel dans l'image final la couleur noir, pour ainsi indiquer qu'il n'y a pas eu de mouvement sur ce pixel entre l'image de référence et l'image courante. Nous enregistrerons donc autant d'image de résultats que d'image prise par la caméra.

b. Algorithme

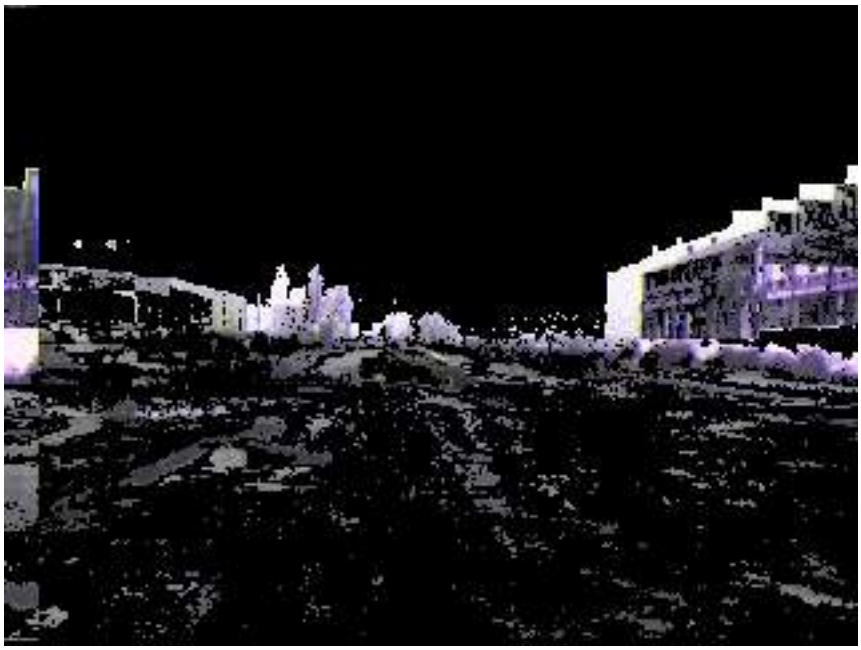
On dispose de la première image de la séquence, et pour chaque image ensuite, nous allons comparer la première image ainsi que l'image courante, et s'il y a des différentes, nous les affichons, sinon nous mettons simplement du noir.

```
// img_base : Image de base
Pour img_courante de 1 à N // N : Nombre d'images de la séquence
   Pour point_Y(point) de 0 à NbLignes
   Pour point_X(point) de 0 à NbColonnes

      diff_R = | I_R(img_courante, point) - I_R(img_base, point) |
      diff_V = | I_V(img_courante, point) - I_V(img_base, point |
      diff_B = | I_B(img_courante, point) - I_B(img_base, point |

      Si diff_R <= seuil et diff_V <= seuil et diff_B <= seuil
         I_R(img_resultat, point) = 0
         I_V(img_resultat, point) = 0
         I_B(img_resultat, point) = 0
      Sinon
         I_R(img_resultat, point) = I_R(img_courante, point)
         I_V(img_resultat, point) = I_V(img_courante, point)
         I_B(img_resultat, point) = I_B(img_courante, point)
```

c. Code Source

```
for(POINT_Y(point) = 0; POINT_Y(point) < NLIG(image); POINT_Y(point)++)
     for(POINT_X(point) = 0; POINT_X(point) < NCOL(image); POINT_X(point)++)
  {
    int diffR, diffV, diffB;
    diffR = (int) PIXEL_R(imcur, point) - (int) PIXEL_R(image, point);
    diffV = (int) PIXEL_V(imcur, point) - (int) PIXEL_V(image, point);
    diffB = (int) PIXEL_B(imcur, point) - (int) PIXEL_B(image, point);
    diffR = (diffR >= 0) ? diffR : -diffR;
    diffV = (diffV >= 0) ? diffV : -diffV;
    diffB = (diffB >= 0) ? diffB : -diffB;

    if(diffR <= seuil && diffB <= seuil && diffV <= seuil) {
        PIXEL_R(imres, point) = 0;
        PIXEL_V(imres, point) = 0;
        PIXEL_B(imres, point) = 0;
    }
    else {
        printf("diff : %d  %d \n", POINT_X(point), POINT_Y(point));
        PIXEL_R(imres, point) = PIXEL_R(imcur, point);
        PIXEL_V(imres, point) = PIXEL_V(imcur, point);
        PIXEL_B(imres, point) = PIXEL_B(imcur, point);
    }
  }
```

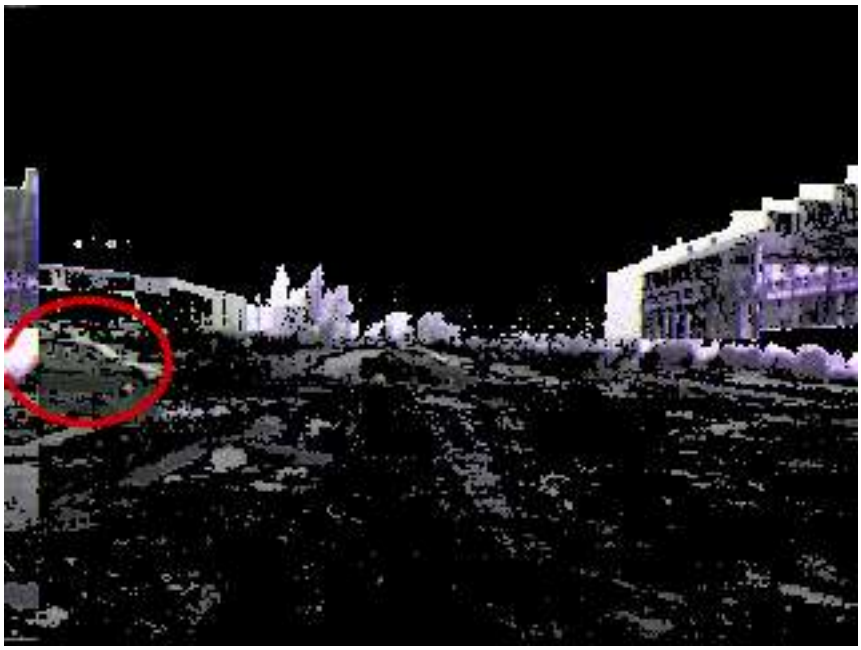
d. Résultats



1. Image résultat #1

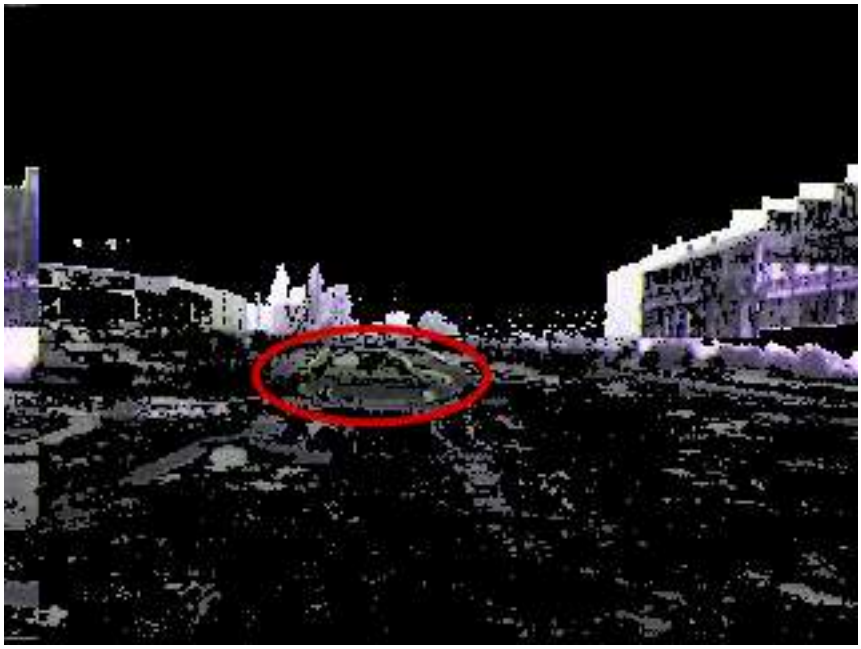
Comme on peut le voir sur cette image qui est l'image résultat #1 (Donc la comparaison entre l'image 0 et l'image 1), on a de nombreuses variations qui peuvent venir de micro mouvements du capteur ou du paysage (Avec le vent, l'herbe bouge, les arbres aussi, il peut y avoir de petites variations de lumières qui feront apparaitre du mouvement là où il n'y en a pas vraiment).

Ces différences détectées peuvent être atténuées avec la mise en place du seuil et sa variation. Plus le seuil sera élevé, moins il y aura de détections « parasites » sur l'image de résultat.



2. Image résultat #91

Vers l'image résultat #91, on commence à voir arriver des mouvements comme ceux que l'on aimerait détecter. Ici, une voiture entre dans le champ de la caméra et on la voit bien distinctement.



3. Image résultat #138

Sur cette image, la voiture apparue précédemment continue son chemin et passe ici devant la voiture stationnée, on la voit toujours distinctement malgré les nombreuses détections « parasites ».

Nous allons voir que l'ajustement par rapport à un gabarit permet notamment de diminuer drastiquement ces petites détections indésirables.

3) Ajustement par rapport à un gabarit

Cette fois-ci nous utiliserons un autre type d'algorithme pour arriver toujours à détecter un mouvement. Nous utiliserons un gabarit. Cette méthode est bien plus efficace que la précédente dans le sens où elle permet d'éliminer le bruit, et les faux mouvements (tel qu'une branche d'arbre qui bougera à cause du vent). Les images de résultats seront alors beaucoup plus précises.

a. Principe

Le principe consiste à construire un gabarit à partir des n premières images de la séquence, qui ne contiennent pas la cible en mouvement. Ces images permettent d'apprendre les effets combinés « du bruit » de la caméra, des variations d'illumination dues aux nuages par exemple dans la séquence fournie, et au mouvement des branches arbres, du au vent, dans le fond de la scène.

Un gabarit simple peut être réalisé à partir de deux images « minimale » et « maximale » dans laquelle l'image courante doit se trouver en absence de mouvement. Cependant pour optimiser l'algorithme et utiliser moins de mémoire, nous pouvons remplacer les 2 images « minimale » et « maximale » par des tableaux, on évite ainsi de créer 2 nouvelles images temporaires, ce qui est une opération lourde, et on travaille simplement sur des tableaux qui contiendront les valeurs minimales et les valeurs maximales de chaque couleur pour chaque point.

b. Algorithme

```
// On initialise les tableaux qui contiendront les valeurs min et
max de chaque pixel, pour créer le gabaris
// Les tableaux sont : min_R, min_V, min_B, max_R, max_V, max_B

Pour les 50 premières images
   Pour chaque point
       si I_R(img_courante, point) < min_R(point)
           min_R(point) = I_R(img_courante, point)
       si I_V(img_courante, point) < min_V(point)
           min_V(point) = I_V(img_courante, point)
       si I_B(img_courante, point) < min_B(point)
           min_B(point) = I_B(img_courante, point)
       si I_R(img_courante, point) > max_R(point)
           max_R(point) = I_R(img_courante, point)
       si I_V(img_courante, point) > max_V(point)
           max_V(point) = I_V(img_courante, point)
       si I_B(img_courante, point) > max_B(point)
           max_B(point) = I_B(img_courante, point)

Pour toutes les autres images
   Pour chaque point
```

```
        si (min_R(point) <= I_R(img_courante, point)
            ET min_V(point) <= I_V(img_courante, point)
            ET min_B(point) <= I_B(img_courante, point)
            ET max_R(point) >= I_R(img_courante, point)
            ET max_V(point) >= I_V(img_courante, point)
            ET max_B(point) >= I_B(img_courante, point))
           I_R(img_resultat, point) = 0
           I_V(img_resultat, point) = 0
           I_B(img_resultat, point) = 0
        sinon
           I_R(img_resultat, point) = I_R(img_courante, point)
           I_V(img_resultat, point) = I_V(img_courante, point)
           I_B(img_resultat, point) = I_B(img_courante, point)
```

c. Code Source

1. *Initialisation des tableaux*

```
int** min_R = (int **)malloc(nlig * sizeof(int*));
int** min_V = (int **)malloc(nlig * sizeof(int*));
int** min_B = (int **)malloc(nlig * sizeof(int*));
int** max_R = (int **)malloc(nlig * sizeof(int*));
int** max_V = (int **)malloc(nlig * sizeof(int*));
int** max_B = (int **)malloc(nlig * sizeof(int*));
int i =0;
int j =0;

for(i =0; i < nlig; i++)
{
    min_R[i] = (int *)malloc(ncol * sizeof(int));
    min_V[i] = (int *)malloc(ncol * sizeof(int));
    min_B[i] = (int *)malloc(ncol * sizeof(int));
    max_R[i] = (int *)malloc(ncol * sizeof(int));
    max_V[i] = (int *)malloc(ncol * sizeof(int));
    max_B[i] = (int *)malloc(ncol * sizeof(int));

    for(j =0; j < ncol; j++)
    {
        min_R[i][j] = 255;
        min_V[i][j] = 255;
        min_B[i][j] = 255;
        max_R[i][j] = 0;
        max_V[i][j] = 0;
        max_B[i][j] = 0;
    }
}
```

On initialise donc 6 tableaux de même dimension que l'image. Chaque tableau correspondra aux valeurs de la couleur (minimale ou maximale) qui constituera notre gabarit. Les valeurs minimales sont initialisées à 255 et les valeurs maximales à 0.

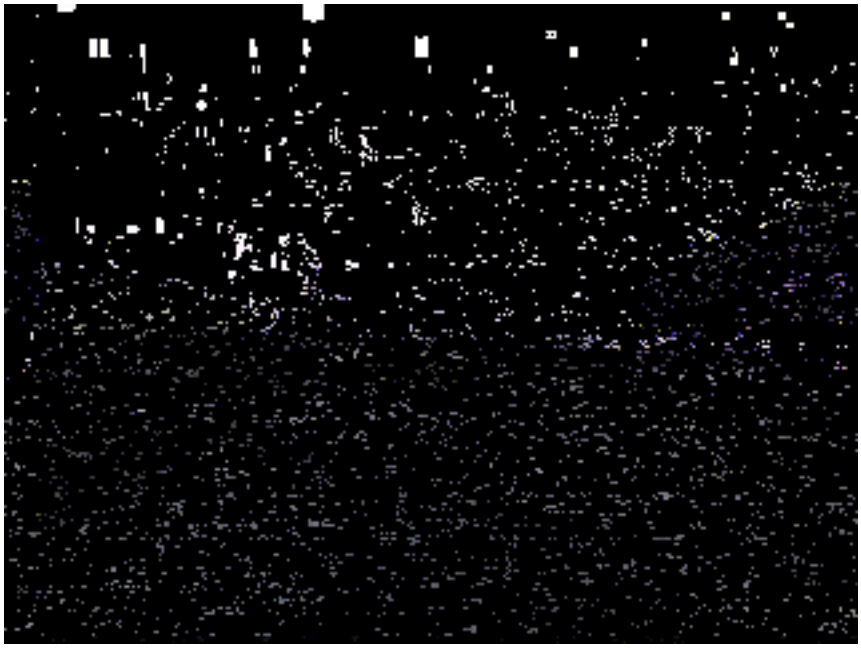
La création de ces 6 tableaux permet de ne pas créer 2 images qui contiendront certes ces informations, mais aussi d'autres informations non nécessaires que nous évitons pour des soucis de performance.

2. *Détection de mouvement*

```
if (num  < 50) {
    for(POINT_Y(point) = 0; POINT_Y(point) < NLIG(image); POINT_Y(point)++)
    for(POINT_X(point) = 0; POINT_X(point) < NCOL(image); POINT_X(point)++)
    {
        if ((int) PIXEL_R(imcur, point) < min_R[POINT_Y(point)][POINT_X(point)]) {
            min_R[POINT_Y(point)][POINT_X(point)] = PIXEL_R(imcur, point);
        }
        if ((int) PIXEL_V(imcur, point) < min_V[POINT_Y(point)][POINT_X(point)]) {
            min_V[POINT_Y(point)][POINT_X(point)] = PIXEL_V(imcur, point);
        }
        if ((int) PIXEL_B(imcur, point) < min_B[POINT_Y(point)][POINT_X(point)]) {
            min_B[POINT_Y(point)][POINT_X(point)] = PIXEL_B(imcur, point);
        }
        if ((int) PIXEL_R(imcur, point) > max_R[POINT_Y(point)][POINT_X(point)]) {
            max_R[POINT_Y(point)][POINT_X(point)] = PIXEL_R(imcur, point);
        }
        if ((int) PIXEL_V(imcur, point) > max_V[POINT_Y(point)][POINT_X(point)]) {
            max_V[POINT_Y(point)][POINT_X(point)] = PIXEL_V(imcur, point);
        }
        if ((int) PIXEL_B(imcur, point) > max_B[POINT_Y(point)][POINT_X(point)]) {
            max_B[POINT_Y(point)][POINT_X(point)] = PIXEL_B(imcur, point);
        }
    }
}

for(POINT_Y(point) = 0; POINT_Y(point) < NLIG(image); POINT_Y(point)++)
for(POINT_X(point) = 0; POINT_X(point) < NCOL(image); POINT_X(point)++)
{
    if(min_R[POINT_Y(point)][POINT_X(point)] <= PIXEL_R(imcur, point)
        && min_V[POINT_Y(point)][POINT_X(point)] <= PIXEL_V(imcur, point)
        && min_B[POINT_Y(point)][POINT_X(point)] <= PIXEL_B(imcur, point)
        && max_R[POINT_Y(point)][POINT_X(point)] >= PIXEL_R(imcur, point)
        && max_V[POINT_Y(point)][POINT_X(point)] >= PIXEL_V(imcur, point)
        && max_B[POINT_Y(point)][POINT_X(point)] >= PIXEL_B(imcur, point))
    {
        PIXEL_R(imres, point) = 0;
        PIXEL_V(imres, point) = 0;
        PIXEL_B(imres, point) = 0;
    }
    else {
        PIXEL_R(imres, point) = PIXEL_R(imcur, point);
        PIXEL_V(imres, point) = PIXEL_V(imcur, point);
        PIXEL_B(imres, point) = PIXEL_B(imcur, point);
    }
}
```

La détection de mouvement est compris dans une boucle qui ne s'arrêtera que lorsque toutes les images prises par la caméra auront été traitées.

d. Résultats



4. Image résultat #57

Après la mise en place du gabarit, nous comparons chaque image avec les bornes établies dans ce gabarit. Pour une image sans mouvement, nous obtenons une image comme celle-ci. Pour avoir un résultat plus net il est tout à fait possible d'effectuer une ouverture.

5. Image résultat #71

Sur cette image, on commence à voir apparaitre la même voiture que dans la détection précédente. On peut voir que la détection a l'air d'être plus précise et plus nette que précédemment, on voit distinctement l'avant de la voiture, les roues et les jantes.

6. Image résultat #99

Sur cette image de résultat, on voit clairement la voiture qui est en plein déplacement. On la voit beaucoup plus distinctement que lors de la détection sans gabarit car seuls les changements importants sont visibles. Comme dit précédemment, pour enlever les petits points partout sur l'écran, nous pouvons effectuer une ouverture qui retirera la grande majorité de ces points indésirables.

En conclusion pour cette partie du TP, nous pouvons voir que la détection via l'utilisation d'un gabarit est nettement plus efficace que sans. Le gabarit permet en effet d'avoir plus de précision lors de la détection et moins de mouvements « parasites ».

II. Couleurs

Le but de ces Travaux Pratiques est la Réalisation d'Opérateurs en Imagerie Couleur. Nous traiterons deux points différents :

- L'Espace Colorimétrique YUV : les transformations entre les espaces colorimétriques RVB et YUV.
- L'Extension à la Couleur RVB des opérateurs N&B : le lissage moyen, et la détection de points de contours (incluant : calcul du gradient couleur, seuillage sur la norme du gradient et affinage), en utilisant l'opérateur de Kirsh à 4 directions.

1) Espace Colorimétrique YUV – Conversion RGB – YUV

a. Principe

Dans cette partie du TP, le but était de développer un programme permettant de convertir une image utilisant l'espace colorimétrique RGV vers une image utilisant l'espace colorimétrique YUV.

Pour cela, nous disposions des formules de transformation qu'il fallait appliquer dans un cas concret :

Y= 0.3R + 0.59G + 0.11B
U = R - Y = 0.7R - 0.59G - 0.11B (Différence des Rouges)
V = B - Y = -0.3R - 0.59G + 0.89B (Différence des Bleus)

R = Y + U
G = Y - (0.3/0.59)U - (0.11/0.59)V
B = Y + V

Dans le cas de notre programme, nous avons rajouté une vérification sur les valeurs finales obtenues après transformation pour rester dans la plage [0 ; 255]. Toutes les valeurs supérieures à 255 sont donc ramenées à 255 et toutes les valeurs inférieures à 0 sont ramenées à 0. Cela évite des irrégularités lors d'une transformation inverse pour récupérer l'image de base.

b. Algorithme

Conversion RGB -> YUV

```
Pour tous les points
    si 0,3 * I_R(img, point) + 0,59 * I_G(img, point) + 0,11 *
I_B(img, point)) > 255)
        I_Y(imgres, point) = 255
    sinon si 0,3 * I_R(img, point) + 0,59 * I_G(img, point) + 0,11 *
I_B(img, point)) < 0)
        I_Y(imgres, point) = 0
    sinon
        I_Y(imgres, point) = 0,3 * I_R(img, point) + 0,59 * I_G(img,
point) + 0,11 * I_B(img, point)

    si 128,0 * (0,7 * I_R(img, point) - 0,59 * I_G(img, point) -
0,11 * I_B(img, point)) / 255,0 + 128,0) > 255)
        I_U(imgres, point) = 255
    sinon 128,0 * (0,7 * I_R(img, point) - 0,59 * I_G(img, point) -
0,11 * I_B(img, point)) / 255,0 + 128,0) < 0)
        I_U(imgres, point) = 0
    sinon
        I_U(imgres, point) = 128,0 * (0,7 * I_R(img, point) - 0,59 *
I_G(img, point) - 0,11 * I_B(img, point)) / 255,0 + 128,0)

    si 128,0 * (- 0,3 * I_R(img, point) - 0,59 * I_G(img, point) +
0,89 * I_B(img, point)) / 255,0 + 128,0) > 255)
        I_V(imgres, point) = 255
    sinon 128,0 * (- 0,3 * I_R(img, point) - 0,59 * I_G(img, point)
+ 0,89 * I_B(img, point)) / 255,0 + 128,0)< 0)
        I_V(imgres, point) = 0
```

```
    sinon
        I_V(imgres, point) = 128,0 * (- 0,3 * I_R(img, point) - 0,59
* I_G(img, point) + 0,89 * I_B(img, point)) / 255,0 + 128,0)
```

c. Code Source

Conversion RGB -> YUV

```
if((0.3 * PIXEL_R(image, point) + 0.59 * PIXEL_G(image, point) + 0.11 * PIXEL_B(image, point)) > 255)
    PIXEL_Y(imres, point) = 255;
else if((0.3 * PIXEL_R(image, point) + 0.59 * PIXEL_G(image, point) + 0.11 * PIXEL_B(image, point)) < 0)
    PIXEL_Y(imres, point) = 0;
else
    PIXEL_Y(imres, point) = (0.3 * PIXEL_R(image, point) + 0.59 * PIXEL_G(image, point) + 0.11 * PIXEL_B(image, point));

if((128.0 * (0.7 * PIXEL_R(image, point) - 0.59 * PIXEL_G(image, point) - 0.11 * PIXEL_B(image, point) )/255.0 + 128.0) > 255)
    PIXEL_U(imres, point) = 255;
else if((128.0 * (0.7 * PIXEL_R(image, point) - 0.59 * PIXEL_G(image, point) - 0.11 * PIXEL_B(image, point) )/255.0 + 128.0) < 0)
    PIXEL_U(imres, point) = 0;
else
    PIXEL_U(imres, point) = 128.0 * (0.7 * PIXEL_R(image, point) - 0.59 * PIXEL_G(image, point) - 0.11 * PIXEL_B(image, point) )/255.0 + 128.0;

if((128.0 * (-0.3 * PIXEL_R(image, point) - 0.59 * PIXEL_G(image, point) + 0.89 * PIXEL_B(image, point) )/255.0 + 128.0) > 255)
    PIXEL_V(imres, point) = 255;
else if((128.0 * (-0.3 * PIXEL_R(image, point) - 0.59 * PIXEL_G(image, point) + 0.89 * PIXEL_B(image, point) )/255.0 + 128.0) < 0)
    PIXEL_V(imres, point) = 0;
else
    PIXEL_V(imres, point) = 128.0 * (-0.3 * PIXEL_R(image, point) - 0.59 * PIXEL_G(image, point) + 0.89 * PIXEL_B(image, point) )/255.0 + 128.0;
```

Conversion YUV -> RGB

```
if((PIXEL_Y(imres, point) + 255.0 * PIXEL_U(imres, point) / 128.0 - 255.0) > 255)
    PIXEL_R(imres2, point) = 255;
else if((PIXEL_Y(imres, point) + 255.0 * PIXEL_U(imres, point) /128.0 - 255.0) < 0)
    PIXEL_R(imres2, point) = 0;
else
    PIXEL_R(imres2, point) = PIXEL_Y(imres, point) + 255.0 * PIXEL_U(imres, point) /128.0 - 255.0;

if((PIXEL_Y(imres, point) + 255.0 * PIXEL_V(imres, point) /128.0 - 255.0) > 255)
    PIXEL_B(imres2, point) = 255;
else if((PIXEL_Y(imres, point) + 255.0 * PIXEL_V(imres, point) /128.0 - 255.0) < 0)
    PIXEL_B(imres2, point) = 0;
else
    PIXEL_B(imres2, point) = PIXEL_Y(imres, point) + 255.0 * PIXEL_V(imres, point) /128.0 - 255.0;

if((PIXEL_Y(imres, point) - (0.3*255.0/(0.59*128.0)) * PIXEL_U(imres, point) -
  (0.11*255.0/(0.59*128.0)) * PIXEL_V(imres, point) + 0.41*255.0/0.59) > 255)
    PIXEL_G(imres2, point) = 255;
else if(PIXEL_Y(imres, point) - (0.3*255.0/(0.59*128.0)) * PIXEL_U(imres, point) -
  (0.11*255.0/(0.59*128.0)) * PIXEL_V(imres, point) + 0.41*255.0/0.59 < 0)
    PIXEL_G(imres2, point) = 0;
else
    PIXEL_G(imres2, point) = PIXEL_Y(imres, point) - (0.3*255.0/(0.59*128.0)) *
      PIXEL_U(imres, point) - (0.11*255.0/(0.59*128.0)) * PIXEL_V(imres, point) + 0.41*255.0/0.59;
```

d. Résultats

7. Image de base en RVB

8. Image convertie en YUV

9. Image à nouveau reconvertie en RVB

On peut voir que la conversion de RGB à YUV se fait correctement avec l'obtention d'une image qui parait « aplatie ». Le système tente d'ouvrir l'image en RGB malgré le fait qu'elle soir codé en YUV, d'où cette impression d'aplatie.

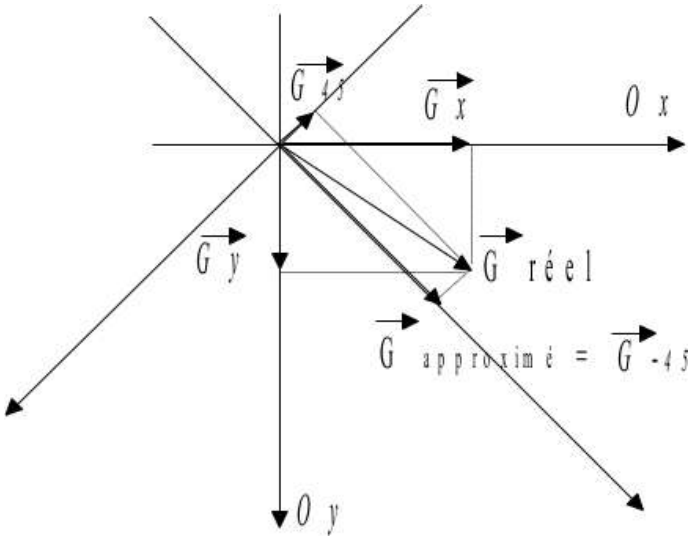
Lorsque nous reconvertissons cette image en YUV en RGB, nous obtenons à nouveau notre image de départ sans différences. Cela, notamment grâce au maintien des bornes [0 ; 255] lors des différentes conversions. Passez du RGB au YUV et retourner au RGB n'a donc aucune n'influence.

2) Extension de l'opérateur Kirsh N&B à l'imagerie couleur RVB

a. Principe

1. Kirsh N&B

Le principe de l'opérateur Kirsch est d'approximer le gradient selon 4 directions représentes par la figure 1 ($\vec{G}_x$, $\vec{G}_y$, $\vec{G}_{45}$, $\vec{G}_{-45}$). Cela a pour effet de mettre en valeur les forts changements de valeur (gradient) sur ces axes-ci.



10. Approximation du gradient avec Kirsh

La particularité de cet opérateur est quatre images sont calculées, l'une relative à chacun des axes. L'algorithme va générer deux images, l'une basée sur la valeur de l'argument max, l'autre basée sur la valeur de la norme maximale. L'utilisation de l'image de l'argument sera utilisée ultérieurement lors de l'étape d'affinage (seconde partie).

On balaye chaque pixel auquel on calcule la valeur des gradients, en fonctions des masques de convolution suivant.

-1/3	0	1/3
-1/3	0	1/3
-1/3	0	1/3

$\overrightarrow{G_x}$

-1/3	-1/3	-1/3
0	0	0
1/3	1/3	1/3

$\overrightarrow{G_y}$

-1/3	-1/3	0
-1/3	0	1/3
0	1/3	1/3

$\overrightarrow{G_{-45}}$

0	1/3	1/3
-1/3	0	1/3
-1/3	-1/3	0

$\overrightarrow{G_{45}}$

Comme l'illustre l'image ci-dessous, deux approches sont ensuite possibles, l'une (a) consiste à effectuer séparément l'approximation la norme du gradient et de son argument pour chacune des quatre directions, puis de choisir la meilleure approximation à la fin. L'autre approche (b), consiste à regrouper ces étapes de calcul permettant ainsi d'optimiser le code. Dans le cadre de ce travail la première approche a été mise en place. La valeur du gradient récupérée (Gmax) est la valeur la meilleure approximation, soit la valeur maximale de ses quatre filtres pour pixel donnés. Afin de trouver l'argument correspondant, nous utilisons la table si dessous.

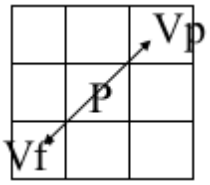
Si Gmax	Valeur argument (argG)
$\overrightarrow{G_x}$	0
$\overrightarrow{G_{45}}$	1
$\overrightarrow{G_y}$	6
$\overrightarrow{G_{-45}}$	7

2. Affinage N&B

Le but de l'affinage est de rendre les contours plus fins (1 pixel d'épaisseurs). Pour ce faire on définis un point appartenant a un contour, pour tout pixel respectant la règle :

« Un Point est de Contour si et seulement si la Norme de son Gradient est Maximale Locale (dans le voisinage 3x3) Directionnelle, dans la Direction du Gradient »

Cela revient à comparer la norme du pixel courant en fonction de ses deux voisins dans l'axe du gradient. Si le pixel ne valide pas la règle sa valeur n'est pas prise en compte dans l'image résultat.



Dans l'exemple ci-contre, si la norme de P est supérieure à Vf et Vp alors le pixel est conservé et sa valeur prise en compte. Les pixels Vf et Vp ne seront quant à eux pas pris en compte car ils ne constituent pas le maximal local

3. *Kirsh et affinage couleur*

Le principe reste le même sauf que nous l'adaptons à un plan RVB. Le résultat sera une image en noir et blanc mais le traitement de l'algorithme se fera sur une image couleur RGB.

b. Algorithme

1. *Kirsh couleur*

```
// On crée tabPixelVoisin : tableau(3*3) des valeurs des pixels autour du
pixel courant

Pour chaque pixel de l'image (sauf les bords)
   K = 0 //initialisation du compteur de tabPixelVoisin
   Pour chaque pixel voisin
      tabPixelVoisin[K] = I(img, pointvoisin)
      K++


   Pour chaque couleur
      //Calcul de la norme
      calcul Gx
      calcul Gy
      calcul G(45)
      calcul G(-45)

      //Conversion en valeur absolue
      Gx = | (Gx) |
      Gy = | (Gy) |
      G(45) = | G(45) |
      G(-45) =| G(-45)|
      G = max { Gx ; Gy ; G(45) ; G(-45) }

   //On affecte les arguments correspondant
   Si (G = G_R)
         Si (G est egal à GxR)
```

```
            argG = 0
         Si (G est egal à GyR)
            argG = 6
         Si (G est egal à G(45)R)
            argG = 1
         Si (G est egal à G(-45)R)
            argG = 7

   Si (G = G_V)
         Si (G est egal à GxV)
            argG = 0
         Si (G est egal à GyV)
            argG = 6
         Si (G est egal à G(45)V)
            argG = 1
         Si (G est egal à G(-45)V)
            argG = 7

   Si (G = G_B)
         Si (G est egal à GxB)
            argG = 0
         Si (G est egal à GyB)
            argG = 6
         Si (G est egal à G(45)B)
            argG = 1
         Si (G est egal à G(-45)B)
            argG = 7

   //On remplis les images en sortie
   Image résultat normée (pixel courant) = G
   Image résultat argument (pixel courant) = argG
```

2. *Affinage*

```
//On parcours chaque pixel de l'image (sauf les bords)
Pour y = 1 à y = Nlig -2
Pour x = 1 à x = Ncol -2

   Si P(x, y) est un point de contour

      Recherche voisin V'(x',y')
      Recherche voisin V''(x'',y'')

      Si (norme(V') > norme (P) ) ou (norme(V'') > norme (P) )
         Image résultat affinage (P(x,y)) = 0 //P(x,y) n'est pas un point
de contour

      Sinon
         Image résultat affinage (P(x,y)) = norme (P)
```

c. Source

1. Kirsh

```
for(POINT_Y(point) = 1; POINT_Y(point) < NLIG(image) - 1; POINT_Y(point)++)
    for(POINT_X(point) = 1; POINT_X(point) < NCOL(image) - 1; POINT_X(point)++)
    {
        /*--Recuperation des valeurs du pixel courant dans une tableaux--*/
        int K = 0; //compteur dans tabVoisin
        /* --- Balayage Video du voisinage 3x3 --- */
        for(j = 0; j < 3; j++)
        for(i = 0; i < 3; i++)
        {
            /* calcul des coordonnees absolues du point voisin */
            POINT_X(pointv) = POINT_X(point) + i - 1;
            POINT_Y(pointv) = POINT_Y(point) + j - 1;

            tabVoisinR[K] = (int) PIXEL_R(image, pointv);
            tabVoisinV[K] = (int) PIXEL_V(image, pointv);
            tabVoisinB[K] = (int) PIXEL_B(image, pointv);

            K++;
        } /* --- fin du balayage du voisinage --- */

        GxR, GyR, GdR, GiR, GxV, GyV, GdV, GiV, GxB, GyB, GdB, GiB = 0; /* initialisation des variables temporaires */

        //Calcul des normes//
        GxR = (-tabVoisinR[0] + tabVoisinR[2] - tabVoisinR[3] + tabVoisinR[5] - tabVoisinR[6] + tabVoisinR[8])/3;
        GyR = (-tabVoisinR[0] - tabVoisinR[1] - tabVoisinR[2] + tabVoisinR[6] + tabVoisinR[7] + tabVoisinR[8])/3;
        GdR = (-tabVoisinR[0] - tabVoisinR[1] - tabVoisinR[3] + tabVoisinR[5] + tabVoisinR[7] + tabVoisinR[8])/3;
        GiR = (+tabVoisinR[1] + tabVoisinR[2] - tabVoisinR[3] + tabVoisinR[5] - tabVoisinR[6] - tabVoisinR[7])/3;

        GxV = (-tabVoisinV[0] + tabVoisinV[2] - tabVoisinV[3] + tabVoisinV[5] - tabVoisinV[6] + tabVoisinV[8])/3;
        GyV = (-tabVoisinV[0] - tabVoisinV[1] - tabVoisinV[2] + tabVoisinV[6] + tabVoisinV[7] + tabVoisinV[8])/3;
        GdV = (-tabVoisinV[0] - tabVoisinV[1] - tabVoisinV[3] + tabVoisinV[5] + tabVoisinV[7] + tabVoisinV[8])/3;
        GiV = (+tabVoisinV[1] + tabVoisinV[2] - tabVoisinV[3] + tabVoisinV[5] - tabVoisinV[6] - tabVoisinV[7])/3;

        GxB = (-tabVoisinB[0] + tabVoisinB[2] - tabVoisinB[3] + tabVoisinB[5] - tabVoisinB[6] + tabVoisinB[8])/3;
        GyB = (-tabVoisinB[0] - tabVoisinB[1] - tabVoisinB[2] + tabVoisinB[6] + tabVoisinB[7] + tabVoisinB[8])/3;
        GdB = (-tabVoisinB[0] - tabVoisinB[1] - tabVoisinB[3] + tabVoisinB[5] + tabVoisinB[7] + tabVoisinB[8])/3;
        GiB = (+tabVoisinB[1] + tabVoisinB[2] - tabVoisinB[3] + tabVoisinB[5] - tabVoisinB[6] - tabVoisinB[7])/3;
```

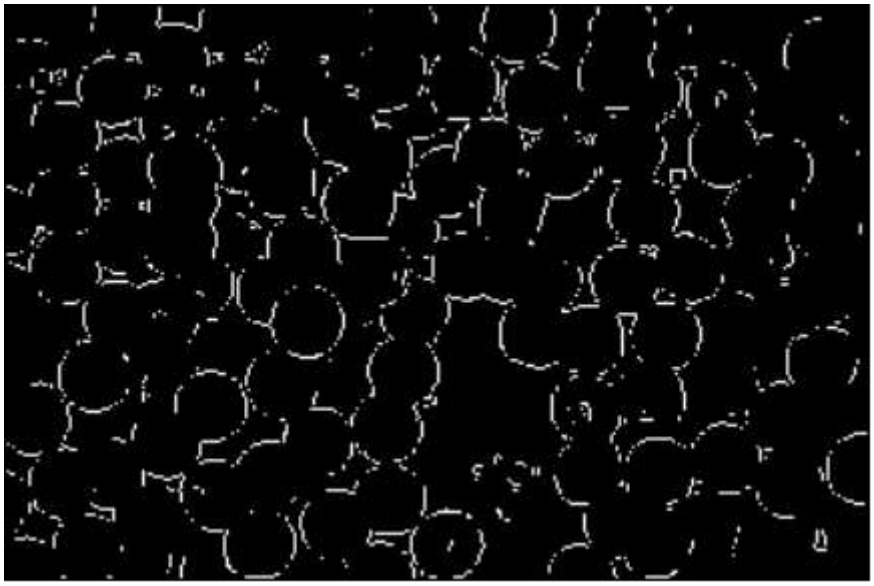
```
//Recuperation de la valeur absolue
GxR = (GxR >= 0) ? GxR : -GxR;
GyR = (GyR >= 0) ? GyR : -GyR;
GdR = (GdR >= 0) ? GdR : -GdR;
GiR = (GiR >= 0) ? GiR : -GiR;

GxV = (GxV >= 0) ? GxV : -GxV;
GyV = (GyV >= 0) ? GyV : -GyV;
GdV = (GdV >= 0) ? GdV : -GdV;
GiV = (GiV >= 0) ? GiV : -GiV;

GxB = (GxB >= 0) ? GxB : -GxB;
GyB = (GyB >= 0) ? GyB : -GyB;
GdB = (GdB >= 0) ? GdB : -GdB;
GiB = (GiB >= 0) ? GiB : -GiB;

//Estimation de la norme correspondante (plus grande valeur)
GmaxR = GxR;
GmaxR = (GmaxR > GyR) ? GmaxR : GyR;
GmaxR = (GmaxR > GdR) ? GmaxR : GdR;
GmaxR = (GmaxR > GiR) ? GmaxR : GiR;

GmaxV = GxV;
GmaxV = (GmaxV > GyV) ? GmaxV : GyV;
GmaxV = (GmaxV > GdV) ? GmaxV : GdV;
GmaxV = (GmaxV > GiV) ? GmaxV : GiV;

GmaxB = GxB;
GmaxB = (GmaxB > GyB) ? GmaxB : GyB;
GmaxB = (GmaxB > GdB) ? GmaxB : GdB;
GmaxB = (GmaxB > GiB) ? GmaxB : GiB;

if(GmaxR > GmaxV && GmaxR > GmaxB)
   Gmax = GmaxR;
if(GmaxV > GmaxR && GmaxV > GmaxB)
   Gmax = GmaxV;
if(GmaxB > GmaxR && GmaxB > GmaxV)
   Gmax = GmaxB;
  //affectation de la valeur du gradient max au pixel courant dans l'image norme
  PIXEL(imnorm, point) = (unsigned char)Gmax;
```

```
if(Gmax == GmaxR) {
      Arg = 0; //reinitiallisation de l'argument

      if(Gmax == GxR) {
         Arg = 0;
      }
      else if(Gmax == GyR) {
         Arg = 6;
      }
      else if(Gmax == GdR) {
         Arg = 1;
      }
      else  {
         Arg = 7;
      }
}
else if(Gmax == GmaxV) {
      Arg = 0; //reinitiallisation de l'argument

      if(Gmax == GxV) {
         Arg = 0;
      }
      else if(Gmax == GyV) {
         Arg = 6;
      }
      else if(Gmax == GdV) {
         Arg = 1;
      }
      else  {
         Arg = 7;
      }
}
else {
    Arg = 0; //reinitiallisation de l'argument

      if(Gmax == GxB) {
         Arg = 0;
      }
      else if(Gmax == GyB) {
         Arg = 6;
      }
      else if(Gmax == GdB) {
         Arg = 1;
      }
      else  {
         Arg = 7;
      }
}

PIXEL(imarg, point) = (unsigned char) Arg;
```

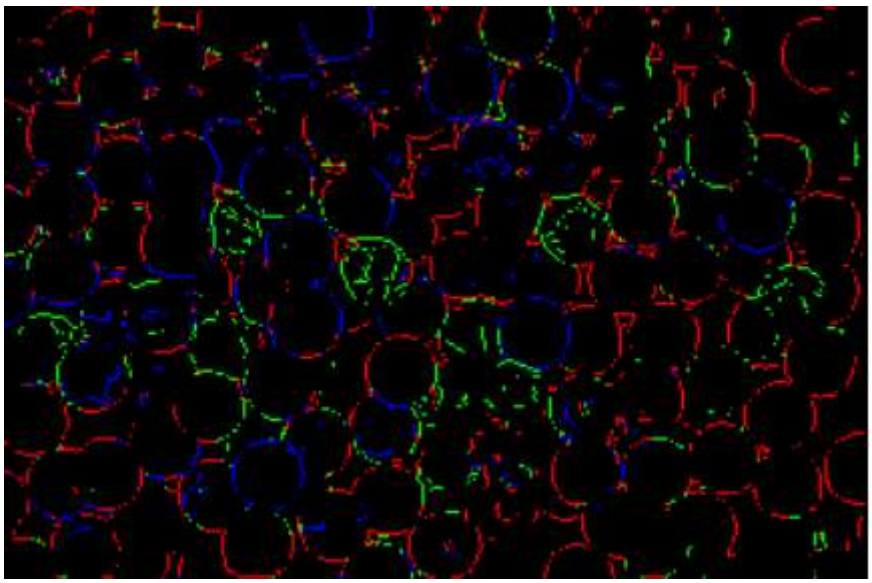
2. *Affinage*

```
for(POINT_Y(point) = 1; POINT_Y(point) < NLIG(imnorm) - 1; POINT_Y(point)++)
    for(POINT_X(point) = 1; POINT_X(point) < NCOL(imnorm) - 1; POINT_X(point)++)
    {
        if(PIXEL(imnorm, point) > 20) //Valeur du pixel sur l'image seuillée
        {
            /* Recherche des points voisins */
            switch(PIXEL(imarg, point)) //Valeur du de l'image argument
            {
                case 0:
                    POINT_X(pointv1) = POINT_X(point) - 1; // Point voisin 1
                    POINT_Y(pointv1) = POINT_Y(point);
                    POINT_X(pointv2) = POINT_X(point) + 1; // Point voisin 2
                    POINT_Y(pointv2) = POINT_Y(point);
                break;
                case 6:
                    POINT_X(pointv1) = POINT_X(point);
                    POINT_Y(pointv1) = POINT_Y(point) - 1;
                    POINT_X(pointv2) = POINT_X(point);
                    POINT_Y(pointv2) = POINT_Y(point) + 1;
                break;
                case 1:
                    POINT_X(pointv1) = POINT_X(point) + 1;
                    POINT_Y(pointv1) = POINT_Y(point) - 1;
                    POINT_X(pointv2) = POINT_X(point) - 1;
                    POINT_Y(pointv2) = POINT_Y(point) + 1;
                break;
                case 7:
                    POINT_X(pointv1) = POINT_X(point) - 1;
                    POINT_Y(pointv1) = POINT_Y(point) - 1;
                    POINT_X(pointv2) = POINT_X(point) + 1;
                    POINT_Y(pointv2) = POINT_Y(point) + 1;
                break;
                default:
                    fprintf(stderr,"Erreur de code Argument\n");
            }
            /* fin de la recherche du point voisin */
            /*Comparaison de la valeur des normes du gradient*/
            if(PIXEL(imnorm,pointv1) > PIXEL(imnorm,point) || PIXEL(imnorm,pointv2) > PIXEL(imnorm,point))
            {
                PIXEL(imres,point)=0;
            }
            else
            {
                PIXEL(imres,point)=255;
            }
        }
    }
```

d. Résultats



11. Gradient N&B



12. Gradient Couleur

Sur l'image de résultat de l'affinage, on voit clairement que les contours sont nets et précis, ils ne font en effet pas plus qu'un pixel de large, nous avons effectivement effectué l'affinage sur l'image du gradient.

Nous avons également effectué une sortie où on affiche la couleur du gradient maximal.

Afin de réduire le bruit et les petites particules présentes dans les images, il pourra être intéressant d'effectuer soit une ouverte soit une fermeture. Pour avoir une image avec des contours bien marqués, il pourra aussi être intéressant d'utiliser un algorithme de prolongement de contour, mais ceci n'est pas demandé dans le cadre de ce travail.

Conclusion

Pour finir sur ce projet, nous avons essayé de respecter au maximum le cahier des charges. En plus d'avoir réalisé nos objectifs, de plus nous avons fait notre maximum pour essayer d'optimiser le plus possible le code fournit.

Nous avons aussi beaucoup appris quant à la détection d'un mouvement sur une suite d'images, ainsi que l'algorithme de Kirsh et de la conversion sur des images couleurs. Ce projet nous a permis d'apprendre et de concrétiser les connaissances vues en cours, et de l'appliquer sur des images couleurs et non en noir et blanc contrairement aux TP précédents. Ce projet nous a également appris à travailler en binôme de manière efficace.