

Les fichiers et les images sont disponibles disponibles sur le gitHub :

<https://github.com/Manaelle/Traitement-Image/tree/master/TP3>

EXERCICE 1

Code des trois fonctions demandées :

```
ImageGrisF filtre_derivee_methode2(ImageGrisF &I)
{
    UINT L = I.largeur();
    UINT H = I.hauteur();

    ImageGrisF I1, I2, I3, I4, I5, I6, I7, I8;
    FiltreLineaire K1, K2, K3, K4, K5, K6, K7, K8;

    // appliquer le dégradé N
    K1 = FiltreLineaire(FiltreLineaire::DeriveeN);
    I1 = convolution(I, K1);

    // appliquer le dégradé S
    K2 = FiltreLineaire(FiltreLineaire::DeriveeS);
    I2 = convolution(I, K2);

    // appliquer le dégradé E
    K3 = FiltreLineaire(FiltreLineaire::DeriveeE);
    I3 = convolution(I, K3);

    // appliquer le dégradé W
    K4 = FiltreLineaire(FiltreLineaire::DeriveeW);
    I4 = convolution(I, K4);

    // appliquer le dégradé NE
    K5 = FiltreLineaire(FiltreLineaire::DeriveeNE);
    I5 = convolution(I, K5);

    // appliquer le dégradé SE
```

```
K6 = FiltreLineaire(FiltreLineaire::DeriveeSE);  
  
I6 = convolution(I,K6);  
  
// appliquer le dégradé NW  
  
K7 = FiltreLineaire(FiltreLineaire::DeriveeNW);  
  
I7 = convolution(I,K7);  
  
// appliquer le dégradé SW  
  
K8 = FiltreLineaire(FiltreLineaire::DeriveeSW);  
  
I8 = convolution(I,K8);  
  
// calcul de l'image finale  
// methode 2 : IR = max(|I1|, |I2| ... |Ik|)  
// calcul de la somme  
  
ImageGrisF IR(L,H,0.0);  
  
I1 = abs(I1); IR += I1;  
I2 = abs(I2); IR += I2;  
I3 = abs(I3); IR += I3;  
I4 = abs(I4); IR += I4;  
I5 = abs(I5); IR += I5;  
I6 = abs(I6); IR += I6;  
I7 = abs(I7); IR += I7;  
I8 = abs(I8); IR += I8;  
  
IR = max(I1,I2);  
  
IR = max(IR,I3);  
  
IR = max(IR,I4);  
  
IR = max(IR,I5);  
  
IR = max(IR,I6);  
  
IR = max(IR,I7);  
  
IR = max(IR,I8);  
  
  
return IR;}
```

```
ImageGrisF filtre_Sobel_Prewitt(ImageGrisF &I)
{
    UINT L = I.largeur();
    UINT H = I.hauteur();

    ImageGrisF I1, I2, I3;
    FiltreLineaire K1, K2, K3;

    // appliquer le dégradé N
    K1 = FiltreLineaire(FiltreLineaire::SobelN_S);
    I1 = convolution(I,K1);

    // appliquer le dégradé N
    K2 = FiltreLineaire(FiltreLineaire::SobelNE_SW);
    I2 = convolution(I,K2);

    // appliquer le dégradé N
    K3 = FiltreLineaire(FiltreLineaire::SobelNW_SE);
    I3 = convolution(I,K3);

    // calcul de la somme
    ImageGrisF IR(L,H,0.0);
    ImageGrisF IS(L,H,0.0);
    I1 = abs(I1);
    I2 = abs(I2);
    I3 = abs(I3);
    IS= max(I1,I2);
    IS= max(IS,I3);

    // appliquer le dégradé N
    K1 = FiltreLineaire(FiltreLineaire::PrewittE_W);
    I1 = convolution(I,K1);
```

```
// appliquer le dégradé N
```

```
K2 = FiltreLineaire(FiltreLineaire::PrewittNE_SW);
```

```
I2 = convolution(I,K2);
```

```
// appliquer le dégradé N
```

```
K3 = FiltreLineaire(FiltreLineaire::PrewittNW_SE);
```

```
I3 = convolution(I,K3);
```

```
ImageGrisF IP(L,H,0.0);
```

```
I1 = abs(I1);
```

```
I2 = abs(I2);
```

```
I3 = abs(I3);
```

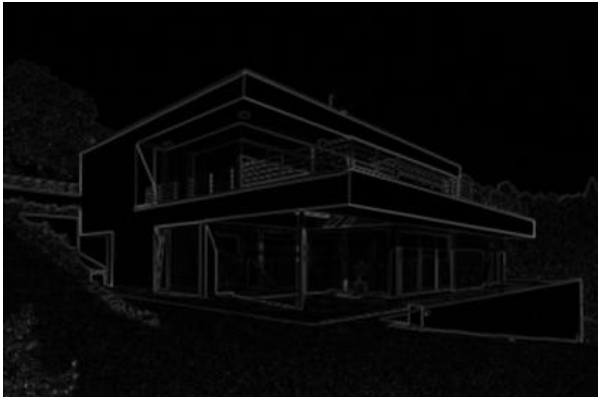
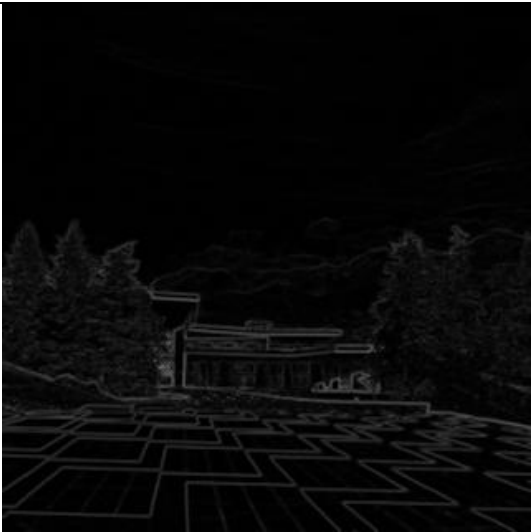
```
IP= max(I1,I2);
```

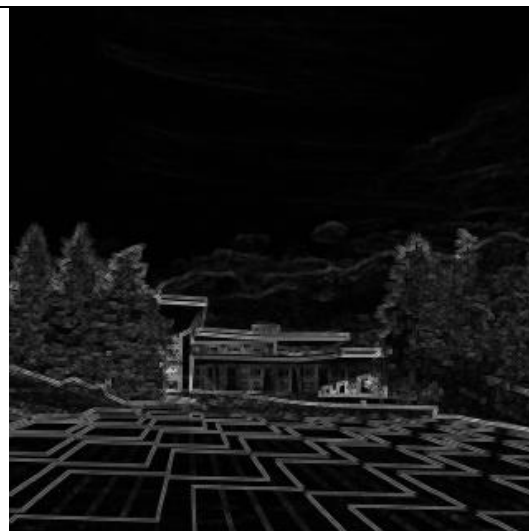
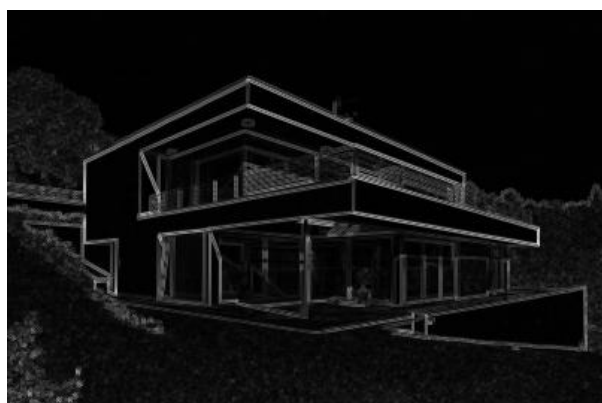
```
IP= max(IP,I3);
```

```
IR = max(IS,IP);
```

```
return IR;
```

On obtient les résultats suivants :

IMAGE 1	IMAGE 2
	
Filtre dérivée : Méthode 1	



Filtre dérivée : Méthode 2



Seuillage automatique : Méthode 1



Seuillage automatique : Méthode 2



Seuillage automatique : Sobel&Prewitt



Filtre Sobel&Prewitt

EXERCICE 2

Avec la fonction suivante... :

```
ImageGrisF transformee_hough(ImageGrisF &I, UINT M, UINT N)
{
    // dimensions de l'image I
    UINT L = I.largeur();
    UINT H = I.hauteur();

    int x,y;
    UINT z;
    float t,a;

    // D : longueur de la diagonale de l'image I
    float D = sqrt((float)L*(float)L+(float)H*(float)H);
```

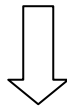
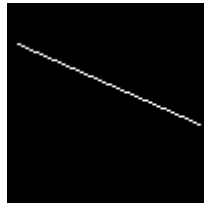
```
// création de l'image de la transformée de Hough
// de dimensions N x N et tous les pixels avec la valeur 0.0
ImageGrisF TH(M,N,0.0);

// parcours de l'image I et traitement des pixels blancs (égaux à 1)
for (int X=0;X<L/2;X++){
    for (int Y=0; Y<H/2;Y++){
        if (I(X,Y)==1){
            x=X-L/2;
            y=Y-H/2;
            for(UINT k=0;k<M-1;k++){
                t=k*3.142/M;
                a=-x*sin(t)+y*cos(t);
                z=(N*(a+D/2))/D;
                //printf("%d  ",z);
                TH.pixel(k,z)+=1;
            }
        }
    }
}

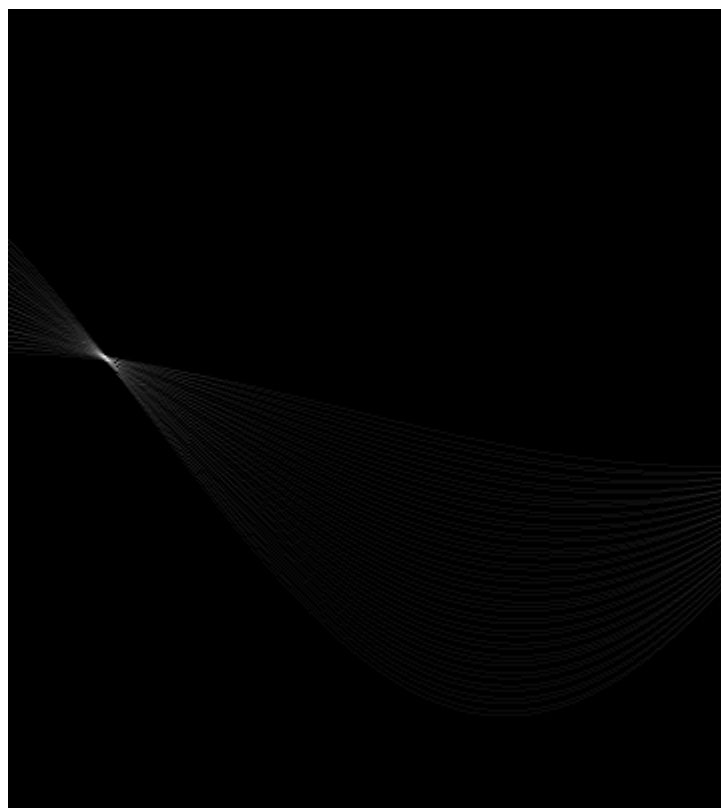
printf("ok");
return TH;
}
```

On obtient :

Image d'origine



Transformée de Hough associée



EXERCICE 3

On ajoute les lignes suivantes au main() :

```
if (partieB){  
    // -----  
    // . détermination des maxima locaux de TH  
    // . affichage des maxima locaux de TH  
    // . affichage des droites correspondantes sur l'image I  
  
    // calculer la sequence de maxima locaux  
    float seuil = 1.0/3.0;
```



```
vector<pixel_valeur> V = sequence_maxima_locaux(TH, seuil, 1);

printf("Séquence des maxima local (taille = %ld) :\n", V.size());

print(V);

// représenter les maxima locaux sur l'image TH comme des points
rouges

// faire une copie de TH --> image couleur THC

ImageCouleurF THC;

THC = gris2rgb(TH);

RGBF Rouge(1.0,0.0,0.0);

UINT nb_max_admis = 10;

for (UINT i=0; i<V.size() && i<nb_max_admis; i++)
{
    pixel_valeur pv = V[i];
    dessiner_pixel(THC, pv.x, pv.y, Rouge, 3);
}

THC.afficher();

// représenter les droites correspondant aux maxima locaux sur
l'image I

// faire une copie de I --> image couleur IC

ImageCouleurF IC;

IC = gris2rgb(I);

for (UINT i=0; i<V.size() && i<nb_max_admis; i++)
{
    pixel_valeur pv = V[i];

    // trouver les valeurs (theta,alpha) correspondant à
    (pv.x,pv.y)

    float theta = M_PI*pv.x/M;
    float alpha = -0.5*D+pv.y*D/N;

    tracer_droite(IC, theta, alpha, Rouge, 2);
}

IC.afficher(); }
```

En reprenant l'image d'origine de l'exercice 2, on obtient :

Transformée de Hough avec les maxima

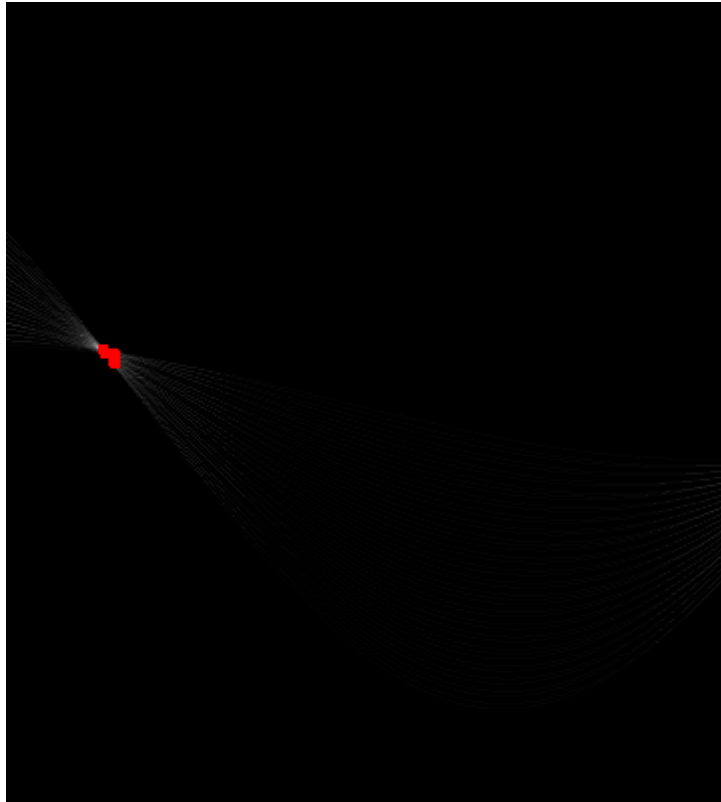


Image d'origine avec tracée de lignes (grâce aux maxima)

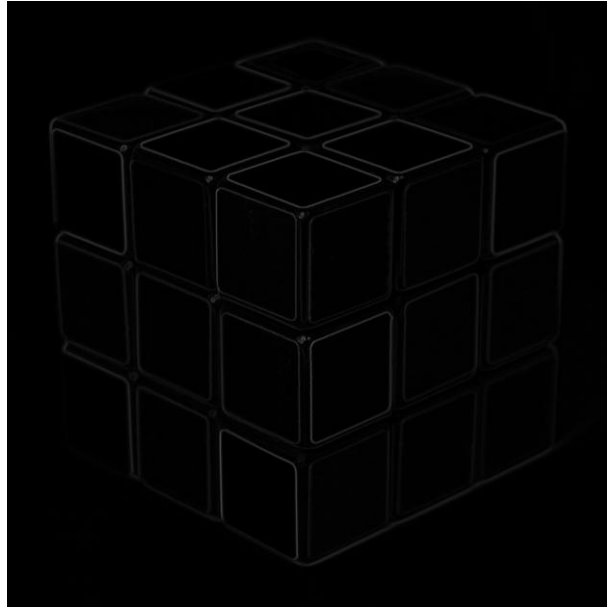


Le programme a été testé sur d'autres images, mais on a oublié de les sauvegarder, on les ajoutera sur le Github lundi (impossible de le faire chez nous...)

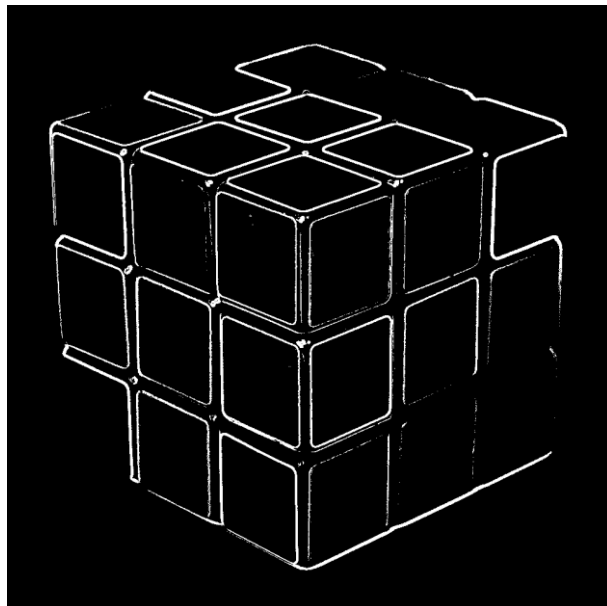
EXERCICE 4

La combinaison du filtrage + Transformée se fait dans **exo4.cpp**. On obtient, pour l'image rubic-cube.bmp :

Filtre Sobel&Prewitt



Seuillage automatique

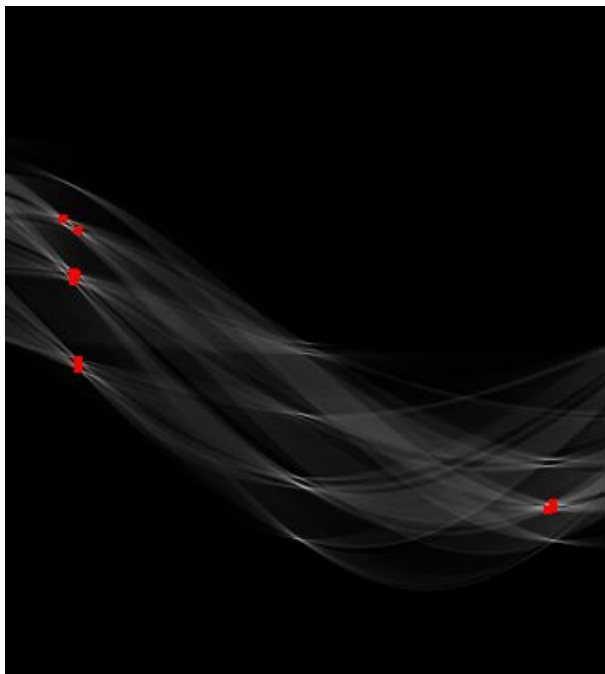


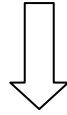


Transformée de Hough

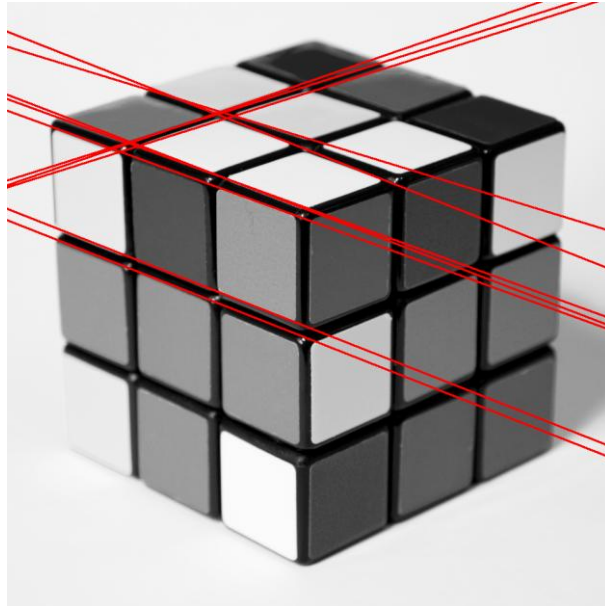


Détection des maxima





Rendu final



CONCLUSION :

La Transformée de Hough permet la détection de figures géométriques simples (droites, cercles...), et fonctionne particulièrement bien lorsque l'image fournie contient des lignes/figures géométriques évidentes (rubicube). Elle ne détecte cependant pas toutes les lignes, étant donné que cette méthode procède par sélection de maxima.

Quant aux filtrages, elles nécessitent toutes de passer par un seuillage (élimination du bruit, renforcement des contours). La méthode passant par la dérivation « brute » donne un résultat précis, mais demande un (long) calcul pour toutes les directions possibles (8 dans ce TP). L'exécution est donc plus lente. Le filtre de Sobel et Prewitt demandent chacun d'eux 3 filtrages : ils sont plus robustes et le résultat est tout aussi satisfaisant. C'est pourquoi on le préférera par la suite dans les autres exercices.