Traitement d'images
TP1

Julien BORDAS, Antoine LADRECH

Mars 2017

1 Lissage linéaire

1.1 Filtrage gaussien par FFT

La FFT de la gaussienne bidimensionnelle s'écrit :

$$TF(G(x,y)) = \tilde{G}(u,v) = e^{-2\pi^2\sigma^2(u^2+v^2)}$$

Les fréquences étant normalisées, les appels à cette fonction se font de la manière suivante :

```
double FTGauss2D(double sigma, double u, double v) {
    return exp(-2*pow(M_PI*sigma,2)*(pow(u,2)+pow(v,2)));
}

[ ... ]

im4[i][j] = im2_shift[i][j] * FTGauss2D(sigma, (i-nl/2.0)/nl, (j-nc/2.0)/nc);
im5[i][j] = im3_shift[i][j] * FTGauss2D(sigma, (i-nl/2.0)/nl, (j-nc/2.0)/nc);
```

Quel est l'effet du paramètre σ sur les images ?
Comment cela se traduit-il sur le facteur PSNR (Peak Signal Noise Ratio) ?

Sur toutes les compositions d'images, les valeurs de σ sont réparties comme suit :

$$\sigma = 0.5 \quad \sigma = 1$$
$$\sigma = 5 \quad \sigma = 10$$

1.1.1 Bruitage "Poivre et Sel"

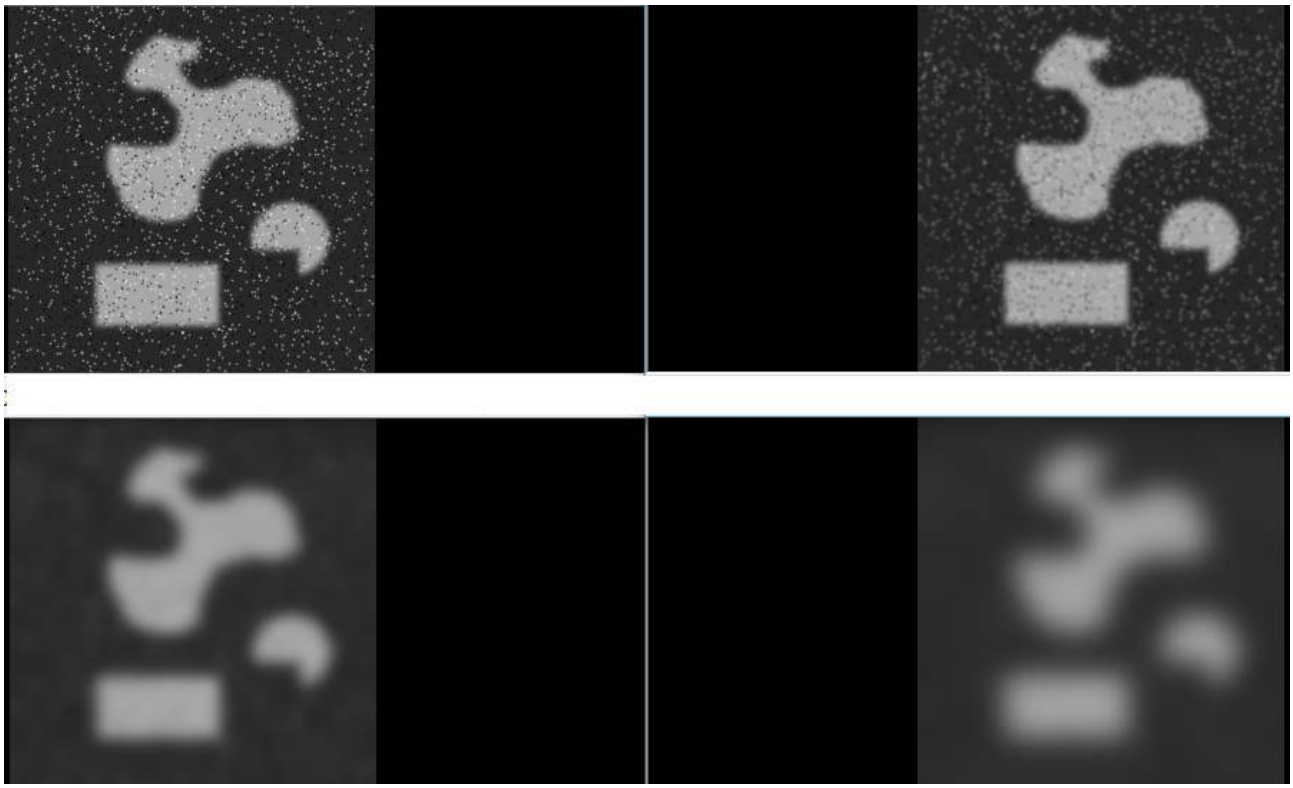

FIGURE 1 – Bruitage de niveau 1

```
PSNR entre formes1.pgm et :
# formes1pets1(s=0.5).pgm       --> 21.520569
# formes1pets1(s=1).pgm         --> 24.495013
# formes1pets1(s=5).pgm         --> 22.822663
# formes1pets1(s=10).pgm        --> 20.105781
```

On remarque que pour un bruitage poivre et sel de niveau 1, $\sigma = 1$ semble être la valeur la plus efficace du filtre.

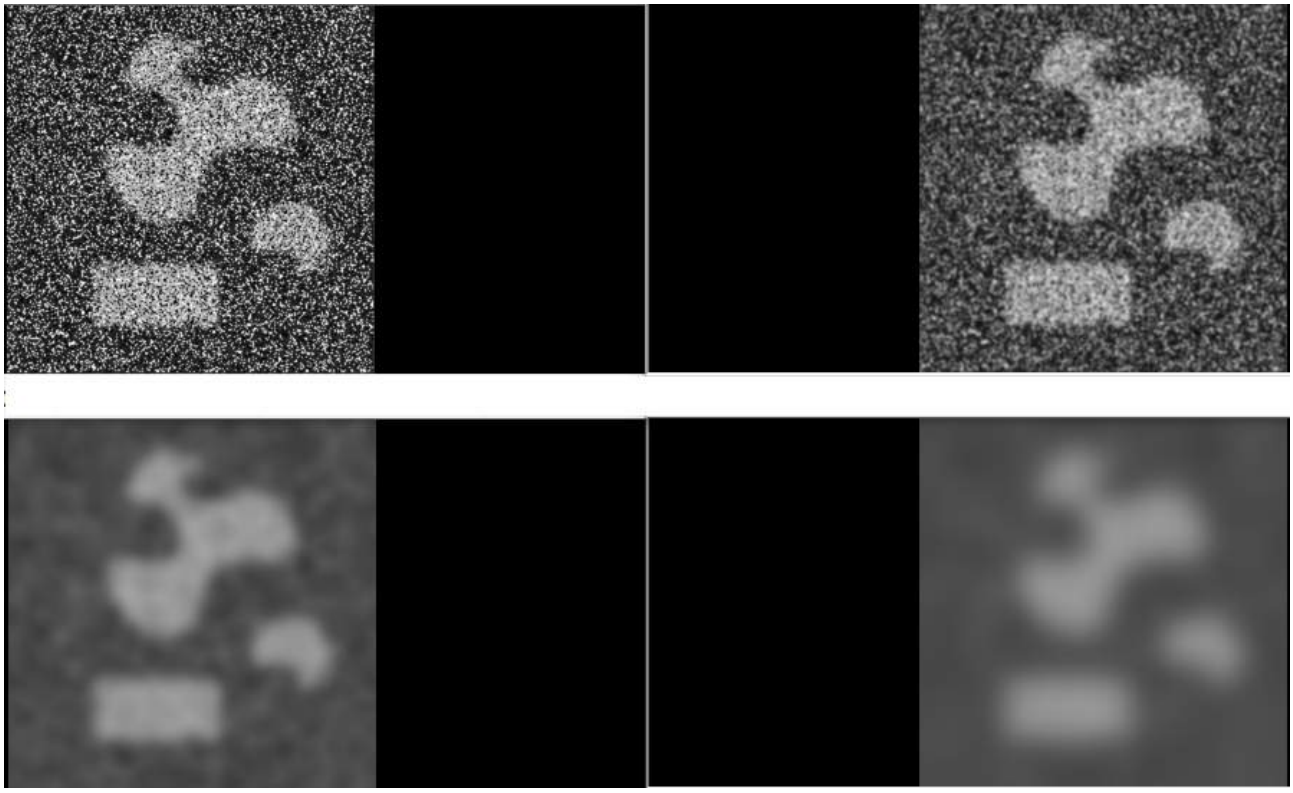

FIGURE 2 – Bruitage de niveau 10

```
PSNR entre formes1.pgm et :
# formes1pets10(s=0.5).pgm      --> 12.690174
# formes1pets10(s=1).pgm        --> 15.683970
# formes1pets10(s=5).pgm        --> 16.851957
# formes1pets10(s=10).pgm       --> 16.005382
```

On remarque que pour un bruitage poivre et sel de niveau 10, $\sigma = 5$ semble être la valeur la plus efficace du filtre.

1.1.2 Bruitage "Speeckle"

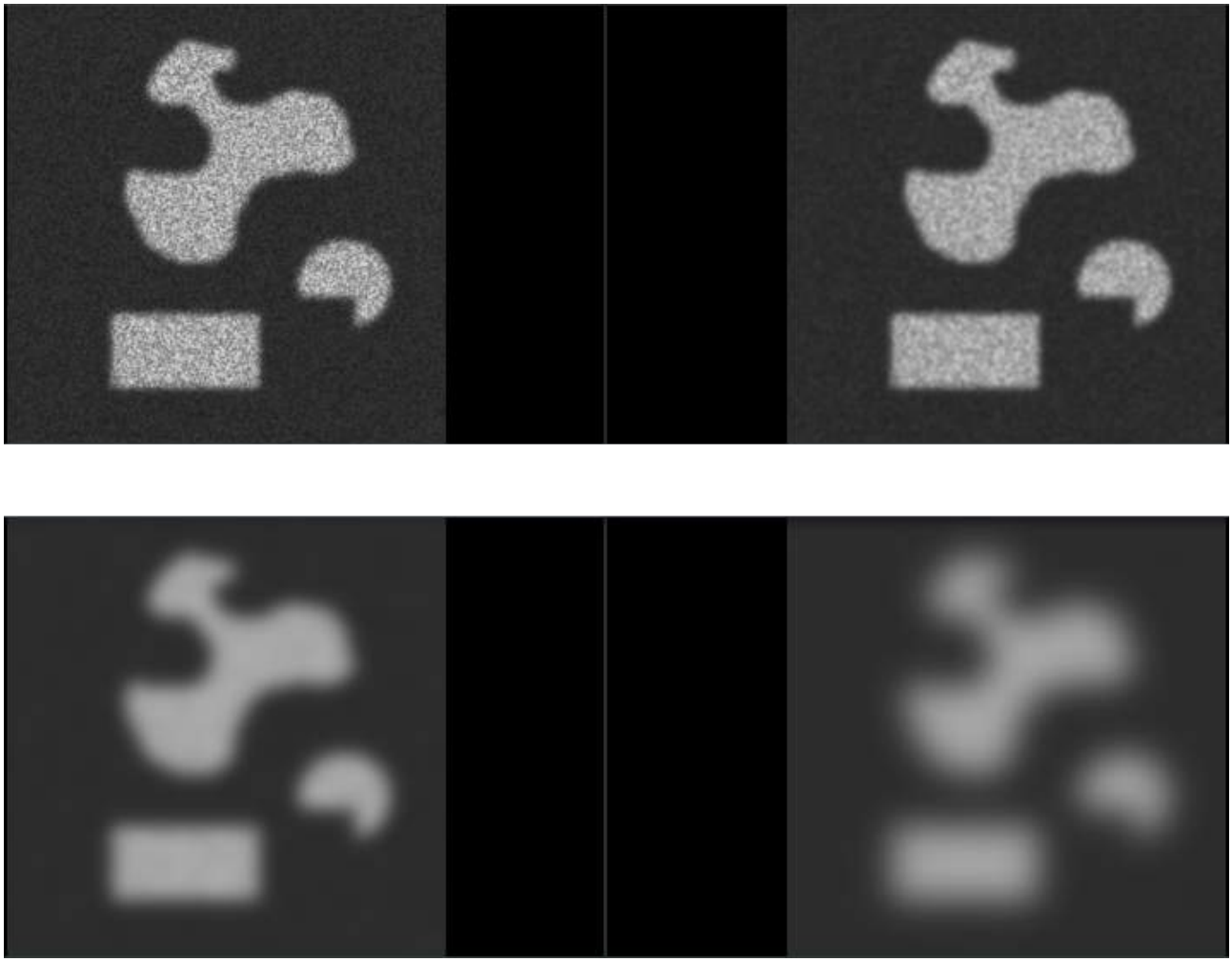

FIGURE 3 – Bruitage de niveau 1

```
PSNR entre formes1.pgm et :
# formes1sp(s=0.5).pgm  --> 24.765725
# formes1sp(s=1).pgm    --> 26.340396
# formes1sp(s=5).pgm    --> 23.285268
# formes1sp(s=10).pgm   --> 20.429612
```

On remarque que pour un bruitage Speeckle de niveau 1, $\sigma = 1$ semble être la valeur la plus efficace du filtre.

1.1.3 Bruitage "Gaussien"

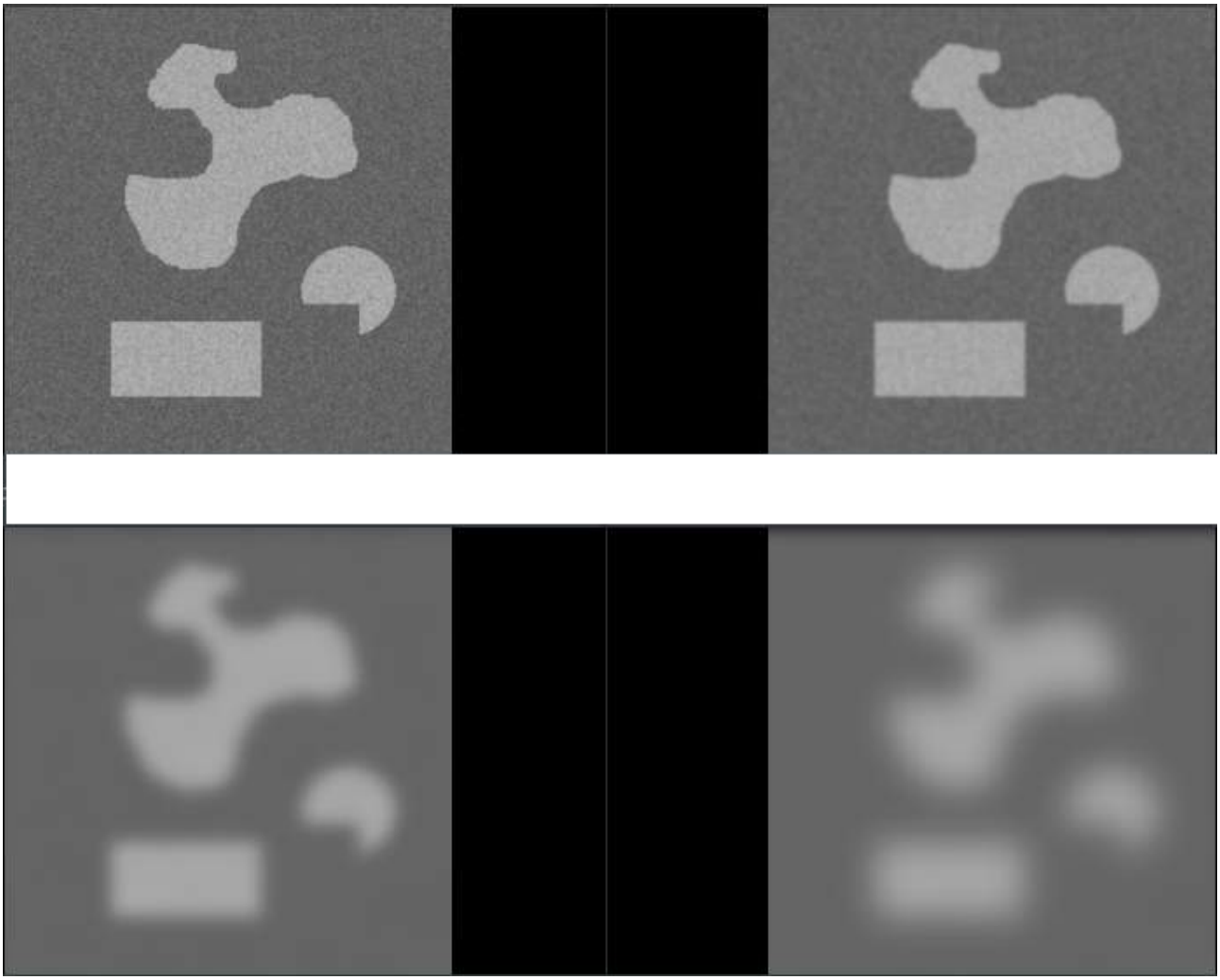

FIGURE 4 – Bruitage de niveau 10

```
PSNR entre formes2.pgm et :
# formes2bb10(s=0.5).pgm        --> 32.737622
# formes2bb10(s=1).pgm  --> 34.654342
# formes2bb10(s=5).pgm  --> 29.048736
# formes2bb10(s=10).pgm         --> 25.987008
```

On remarque que pour un bruitage gaussien de niveau 10, $\sigma = 1$ semble être la valeur la plus efficace du filtre.

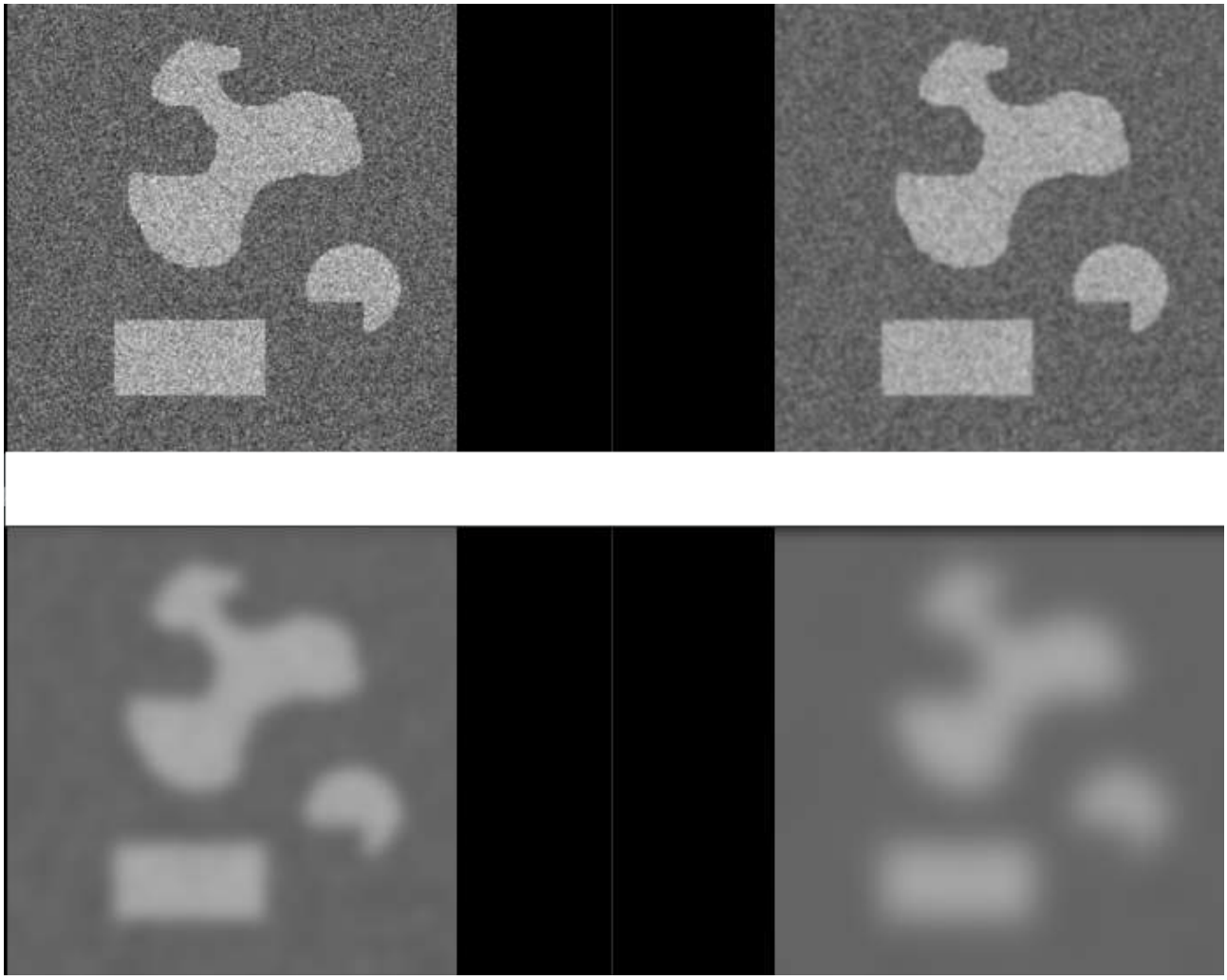

FIGURE 5 – Bruitage de niveau 30

```
PSNR entre formes2.pgm et :
# formes2bb30(s=0.5).pgm        --> 23.738022
# formes2bb30(s=1).pgm  --> 28.806597
# formes2bb30(s=5).pgm  --> 28.972213
# formes2bb30(s=10).pgm         --> 25.993184
```

On remarque que pour un bruitage gaussien de niveau 30, $\sigma = 1$ ou $\sigma = 5$ semblent être des valeurs efficaces pour le filtre.

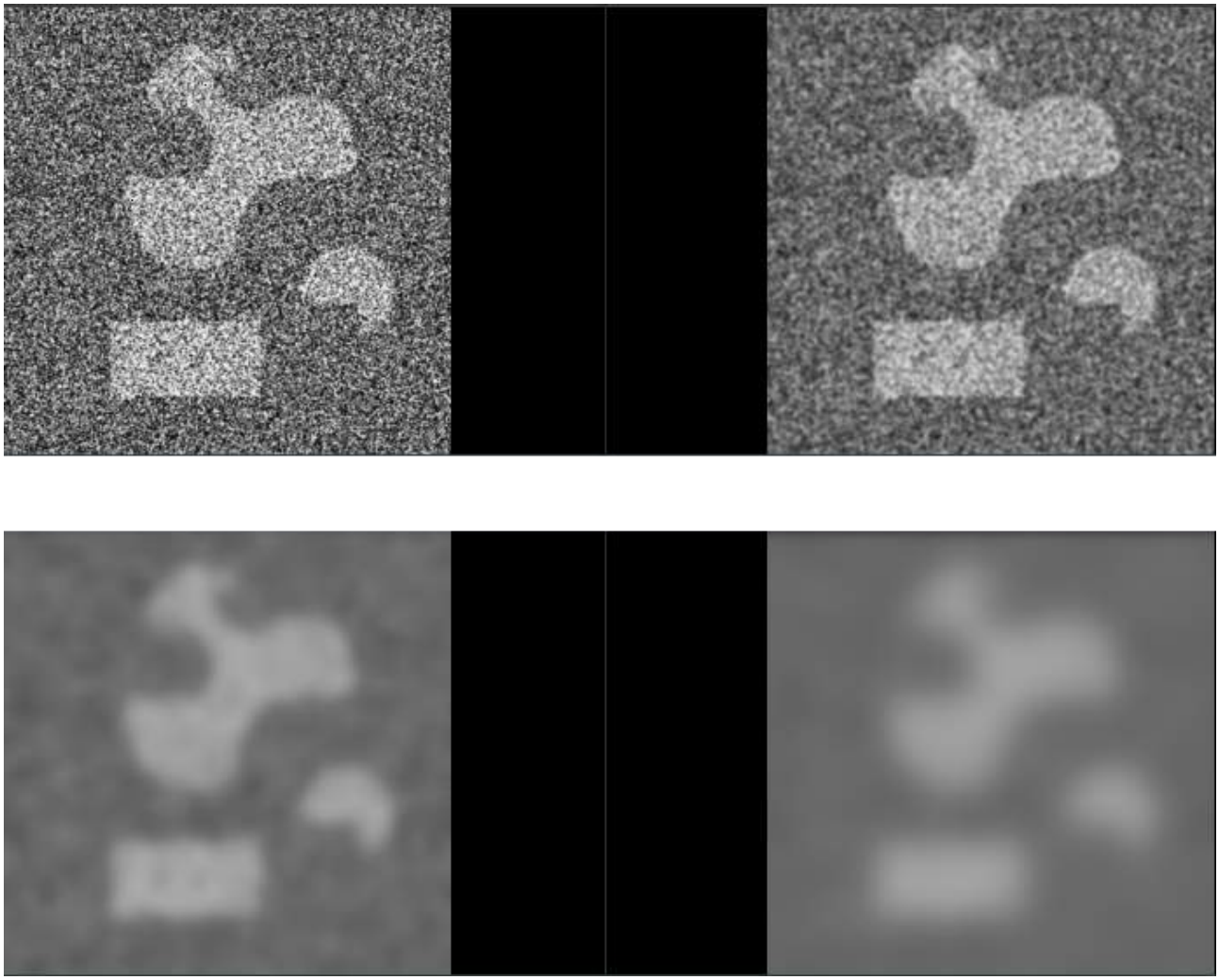

FIGURE 6 – Bruitage de niveau 67

```
PSNR entre formes2.pgm et :
# formes2bb67(s=0.5).pgm        --> 16.052760
# formes2bb67(s=1).pgm  --> 21.600187
# formes2bb67(s=5).pgm  --> 27.344552
# formes2bb67(s=10).pgm _       --> 25.176931
```

On remarque que pour un bruitage gaussien de niveau 67, $\sigma = 5$ semble être la valeur la plus efficace du filtre.

On peut donc en déduire, après avoir comparé les différentes réponses du filtre gaussien aux différents bruits, que celui-ci est plus efficace sur un bruitage gaussien.
En effet, sur les images soumises à un bruit gaussien et filtrées par filtrage gaussien, les valeurs du PSNR sont les plus élevées.

1.2 Convolution spatiale

Implanter la convolution gaussienne d'une image par un masque séparable.

```
//Calcul du masque gaussien
//Choix d'implémentation : Longueur du masque = 2W+1

double* masque_gaussien(int n, double sigma, int W) {
        double* R = calloc(2*W+1, sizeof(double));
        double denom = 0;
        for (int i = -n ; i <= n ; ++i) {
                denom += Gauss1D(sigma, i);
        }
        for (int k = -W ; k <= W ; ++k) {
                R[k+W] = Gauss1D(sigma, k)/denom;
        }
        return R;
}

double** convolution1DX(double** I, int nl, int nc, int W, double* filtre_x) {
        double **R = alloue_image_double(nl,nc);
        for (int x = 0; x < nl; ++x) {
                for (int y = 0; y < nc; ++y) {
                        R[x][y] = 0;
                        for (int i = -W; i <= W ; i++) {
                                R[x][y] += I[(x+i+nl)%nl][y] * filtre_x[i+W];
                        }
                }
        }
        return R;
}

double** convolution1DY(double** I, int nl, int nc, int W, double* filtre_y) {
        double **R = alloue_image_double(nl,nc);
        for (int x = 0; x < nl; ++x) {
                for (int y = 0; y < nc; ++y) {
                        R[x][y] = 0;
                        for (int j = -W; j <= W ; j++)
                                R[x][y] += I[x][(y+j+nc)%nc] * filtre_y[j+W];
                }
        }
        return R;
}
```

Après avoir réalisé le filtre à variables séparables demandé, nous avons calculé le PSNR d'une image filtrée avec ce masque et avec la méthode par FFT, pour différentes valeurs de σ et $L = 2W + 1$.

Pour l'image **formes1pets10.pgm** les résultats ont été affichés dans le tableau ci-dessous :

L=2W+1/Sigma	0.5	1	5	10
1	15.452003	9.787636	8.684666	8.743531
3	26.134466	21.462004	9.131256	8.870975
5	**26.144681**	42.874951	10.022167	9.094248
7	26.144681	58.429508	11.347553	9.428947
9	26.144681	**58.873852**	13.099211	9.874691
11	26.144681	58.869923	15.26563	10.430244
13	26.144681	58.869923	17.835008	11.094284
15	26.144681	58.869923	20.793269	11.865571
17	26.144681	58.869923	24.125622	12.74272
19	26.144681	58.869923	27.81883	13.724232
21	26.144681	58.869923	31.857289	14.808439
23	26.144681	58.869923	36.227209	15.993391
25	26.144681	58.869923	40.874971	17.278226
27	26.144681	58.869923	45.776903	18.660265
29	26.144681	58.869923	49.872442	20.138128
31	26.144681	58.869923	52.790022	21.711338
33	26.144681	58.869923	55.750238	23.374528
35	26.144681	58.869923	58.970861	25.129547
37	26.144681	58.869923	62.323157	26.970307
39	26.144681	58.869923	65.745644	28.901429
41	26.144681	58.869923	69.367134	30.914748
43	26.144681	58.869923	73.032244	33.007796
45	26.144681	58.869923	77.374657	35.183452
47	26.144681	58.869923	82.49349	37.439684
49	26.144681	58.869923	88.51409	39.767794
51	26.144681	58.869923	93.285303	42.102937
53	26.144681	58.869923	96.295603	44.46581
55	26.144681	58.869923	96.295603	47.011747
57	26.144681	58.869923	**inf**	49.1179
59	26.144681	58.869923	inf	50.57165
61	26.144681	58.869923	inf	52.001149
107	26.144681	58.869923	inf	91.52439
109	26.144681	58.869923	inf	93.285303
111	26.144681	58.869923	inf	93.285303
113	26.144681	58.869923	inf	**inf**
115	26.144681	58.869923	inf	inf

Les valeurs en gras correspondent aux valeurs maximales du PSNR de L à variance fixe, elles correspondent donc à la valeur de L optimale pour un filtre par masque gaussien, à σ donné.

Les résultats varient d'une image sur l'autre et en fonction des différents bruits qui ont été appliqués. Ils restent cependant cohérents en moyenne et nous permettent d'affirmer que la taille du masque optimal est une fonction linéaire de σ.
Pour la suite nous garderons la relation suivante : $\boxed{L(\sigma) = 10\sigma}$ qui semble être la plus pertinente

1.3 Complexité et comparaison des 2 méthodes

Mesurer les temps associés aux de méthodes de filtrage pour différentes valeurs de σ.

```
clock_t debut, fin;
double cpu_time_used;
for (int sigma = 0; sigma < 4 ; ++sigma) {
    int W = (10 * tab_sigma[sigma] - 1)/2; //formule retenue en question 1.2, adaptée à W = (L-1)/2

    printf("Sigma = %f\n", tab_sigma[sigma]);
    printf("W = %d\n", W);

    //Calcul temps masque
    debut = clock();
    double* filtre_x = masque_gaussien(nl, tab_sigma[sigma], W);
    double* filtre_y = masque_gaussien(nc, tab_sigma[sigma], W);
    i2 = convolution1DX(i1, nl, nc, W, filtre_x);
    i3 = convolution1DY(i2, nl, nc, W, filtre_y);
    out = imdouble2uchar(i3, nl, nc);
    fin = clock();
    printf("Duree pour masque : W = %d, temps = %f s\n", W, ((double)fin-debut)/
CLOCKS_PER_SEC);

    //Calcul temps fft
    debut = clock();
    unsigned char** fft =  filter (in, nl, nc, tab_sigma[sigma]);
    fin = clock();
    printf("Duree pour fft : %f s\n\n", ((double)fin-debut)/CLOCKS_PER_SEC);
}
```

Voici l'output de cet algorithme sur 3 exemples représentant des images et bruits différents :

— $formes1pets10.pgm$

```
Sigma = 0.500000
W = 2
Duree pour masque : W = 2, temps = 0.010000 s
Duree pour fft : 0.040000 s

Sigma = 1.000000
W = 4
Duree pour masque : W = 4, temps = 0.020000 s
Duree pour fft : 0.030000 s

Sigma = 5.000000
W = 24
Duree pour masque : W = 24, temps = 0.050000 s
Duree pour fft : 0.030000 s

Sigma = 10.000000
W = 49
Duree pour masque : W = 49, temps = 0.090000 s
Duree pour fft : 0.030000 s
```

— $formes2bb10.pgm$

```
Sigma = 0.500000
W = 2
Duree pour masque : W = 2, temps = 0.010000 s
Duree pour fft : 0.040000 s

Sigma = 1.000000
W = 4
Duree pour masque : W = 4, temps = 0.010000 s
Duree pour fft : 0.030000 s

Sigma = 5.000000
W = 24
Duree pour masque : W = 24, temps = 0.060000 s
Duree pour fft : 0.030000 s

Sigma = 10.000000
W = 49
Duree pour masque : W = 49, temps = 0.090000 s
Duree pour fft : 0.030000 s
```

— *globulesbb25.pgm*

```
Sigma = 0.500000
W = 2
Duree pour masque : W = 2, temps = 0.040000 s
Duree pour fft : 0.150000 s

Sigma = 1.000000
W = 4
Duree pour masque : W = 4, temps = 0.040000 s
Duree pour fft : 0.110000 s

Sigma = 5.000000
W = 24
Duree pour masque : W = 24, temps = 0.180000 s
Duree pour fft : 0.120000 s

Sigma = 10.000000
W = 49
Duree pour masque : W = 49, temps = 0.400000 s
Duree pour fft : 0.120000 s
```

Encore un fois les résultats restent cohérents entre eux : pour chaque image, le temps d'exécution est plus élevé avec un filtre par FFT pour $\sigma \leq 1$, puis s'inverse et devient plus élevé pour le filtrage par masque gaussien (quand $\sigma \in \{5, 10\}$).

Ainsi, on peut en déduire que la valeur critique de σ se situe entre 1 et 5 pour toutes les images. Une légère modification de l'algorithme précédent nous a ensuite permis d'affiner ce résultat avec une précision de 0.1 :

```
float  temps_masque;
float  temps_fft;
 double sigma = 1.0;
while (1) {
    int  W = (10 * sigma - 1)/2; //formule retenue en question 1.2, adaptée à W = (L-1)/2

    //Calcul temps masque
    debut = clock();
    double* filtre_x  = masque_gaussien(nl, sigma, W);
    double* filtre_y  = masque_gaussien(nc, sigma, W);
    i2 = convolution1DX(i1, nl, nc, W, filtre_x);
    i3 = convolution1DY(i2, nl, nc, W, filtre_y);
    out = imdouble2uchar(i3, nl, nc);
    fin  = clock();
    temps_masque = ((double)fin-debut)/CLOCKS_PER_SEC;

    //Calcul temps fft
```

```
    debut = clock();
    unsigned char** fft =  filter (in, nl, nc, sigma);
    fin = clock();
    temps_fft = ((double)fin-debut)/CLOCKS_PER_SEC;

        if (temps_fft < temps_masque)
        break;

    sigma += 0.1;
}
printf ("Sigma critique = %f\n", sigma);
```

En exécutant ce nouvel algorithme successivement sur les trois images précédentes, on obtient le résultat suivant :

```
[ladrecha@ensipc331 src]$ ./part1_3 ../in/formes1pets10.pgm
Sigma critique = 2.500000
[ladrecha@ensipc331 src]$
[ladrecha@ensipc331 src]$ ./part1_3 ../in/formes2bb10.pgm
Sigma critique = 3.100000
[ladrecha@ensipc331 src]$
[ladrecha@ensipc331 src]$ ./part1_3 ../in/globulesbb25.pgm
Sigma critique = 3.500000
```

Si les valeur obtenues varient en fonction des images, des bruits ou même de l'exécution elle-même, on observe quand même qu'elles restent très proches d'une valeur σ critique : $\boxed{\sigma_{crit} = 3.0}$. Donc si la variance souhaitée est **inférieure** à σ_{crit}, il vaut mieux utiliser un filtre par masque gaussien, alors que si elle est **supérieure**, la FFT sera plus efficace.

2 Détection de contours

2.1 Opérateurs différentiels du premier ordre

Implanter une méthode de calcul du gradient.

```
double** computeGradx(unsigned char** in, int nl, int nc, double** M1) {
        double** Gx = alloue_image_double(nl,nc);
        double** in_double = imuchar2double(in,nl,nc);

        for (int i = 0; i < nl; ++i) {
                for (int j = 0; j < nc; ++j) {
                        Gx[i][j] = M1[0][0]*in_double[(i-1+nl)%nl][(j-1+nc)%nc] +
                                        M1[0][1]*in_double[(i-1+nl)%nl][j] +
                                        M1[0][2]*in_double[(i-1+nl)%nl][(j+1+nc)%nc] +
                                        M1[1][0]*in_double[i][(j-1+nc)%nc] +
                                        M1[1][1]*in_double[i][j] +
                                        M1[1][2]*in_double[i][(j+1+nc)%nc] +
                                        M1[2][0]*in_double[(i+1+nl)%nl][(j-1+nc)%nc] +
                                        M1[2][1]*in_double[(i+1+nl)%nl][j] +
                                        M1[2][2]*in_double[(i+1+nl)%nl][(j+1+nc)%nc];
                }
        }
        return Gx;
}

double** computeGrady(unsigned char** in, int nl, int nc, double** M1) {
        double** Gy = alloue_image_double(nl,nc);
        double** M2 = alloue_image_double(3,3);
        double** in_double = imuchar2double(in, nl, nc);
        M2 = transpose(M1);

        for (int i = 0; i < nl; ++i) {
                for (int j = 0; j < nc; ++j) {
                        Gy[i][j] = M2[0][0]*in_double[(i-1+nl)%nl][(j-1+nc)%nc] +
                                        M2[0][1]*in_double[(i-1+nl)%nl][j] +
                                        M2[0][2]*in_double[(i-1+nl)%nl][(j+1+nc)%nc] +
                                        M2[1][0]*in_double[i][(j-1+nc)%nc] +
                                        M2[1][1]*in_double[i][j] +
                                        M2[1][2]*in_double[i][(j+1+nc)%nc] +
                                        M2[2][0]*in_double[(i+1+nl)%nl][(j-1+nc)%nc] +
                                        M2[2][1]*in_double[(i+1+nl)%nl][j] +
                                        M2[2][2]*in_double[(i+1+nl)%nl][(j+1+nc)%nc];
                }
        }
        return Gy;
}
```

```
double** computeModG(unsigned char** in, int nl, int nc, double** M1) {
        double** modG = alloue_image_double(nl, nc);
        double** Gx = computeGradx(in, nl, nc, M1);
        double** Gy = computeGrady(in, nl, nc, M1);
        for (int i = 0; i < nl; ++i) {
                for (int j = 0; j < nc; ++j) {
                        modG[i][j] = sqrt(pow(Gx[i][j],2)+pow(Gy[i][j],2));
                }
        }
        return modG;
}
```

Implanter une méthode de détection des contours par seuillage.

```
double** M1 = alloue_image_double(3,3);

// PREWITT GRADIENT
M1[0][0] = -1;
M1[0][1] = 0;
M1[0][2] = 1;
M1[1][0] = -1;
M1[1][1] = 0;
M1[1][2] = 1;
M1[2][0] = -1;
M1[2][1] = 0;
M1[2][2] = 1;

// SOBEL GRADIENT
M1[0][0] = -1;
M1[0][1] = 0;
M1[0][2] = 1;
M1[1][0] = -2;
M1[1][1] = 0;
M1[1][2] = 2;
M1[2][0] = -1;
M1[2][1] = 0;
M1[2][2] = 1;

double** modGrad = computeModG(in, nl, nc, M1);

double seuil = 0;

for (int i = 0; i < nl; ++i) {
        for (int j = 0; j < nc; ++j) {
                if (modGrad[i][j] > seuil-1000)
                        seuil = modGrad[i][j];
                else
                        modGrad[i][j] = 0;
        }
}

out = imdouble2uchar(modGrad, nl, nc);
ecritureimagepgm(av[2],out,nl,nc);
```

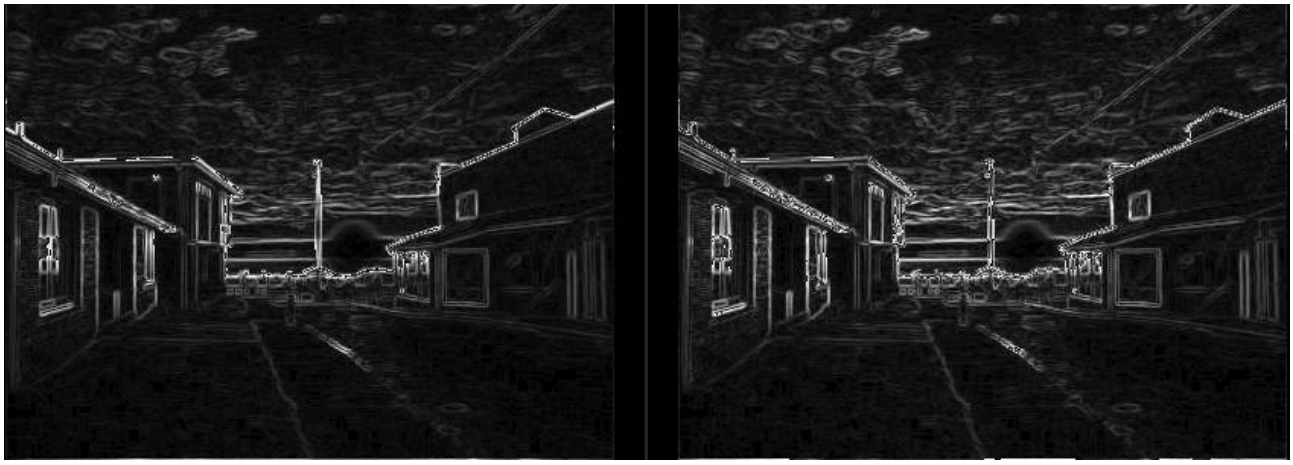

FIGURE 7 – Masque de Prewitt (à gauche) et de Sobel (à droite)

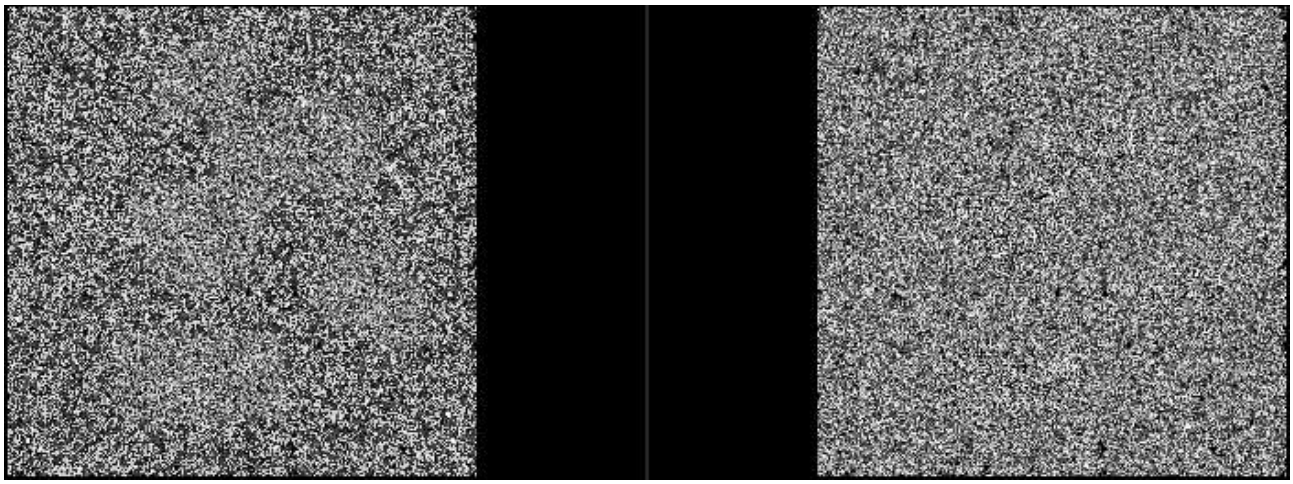

FIGURE 8 – Masque de Prewitt (à gauche) et de Sobel (à droite)

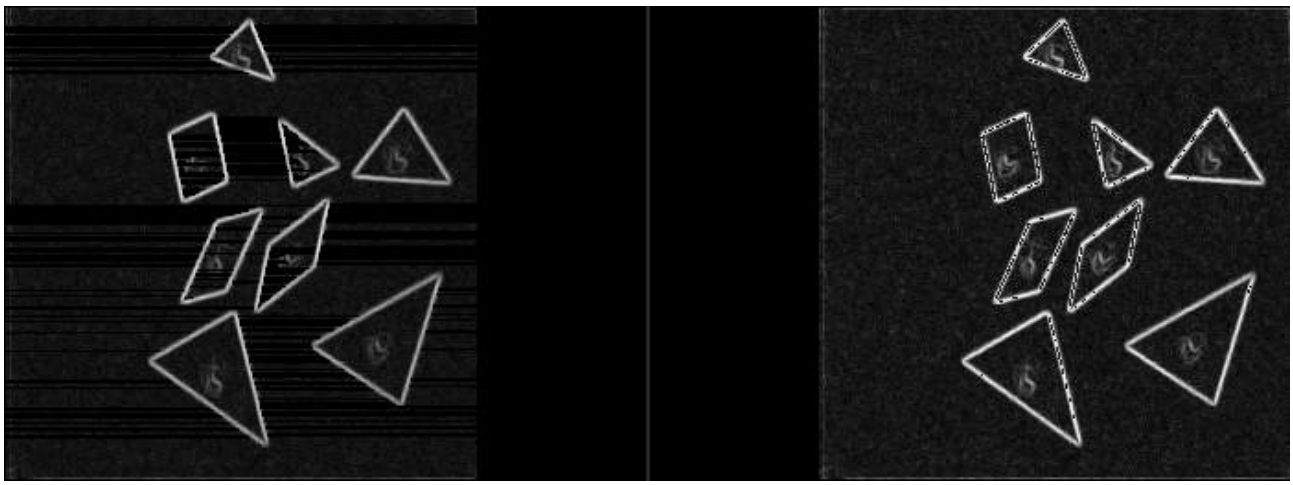

FIGURE 9 – Masque de Prewitt (à gauche) et de Sobel (à droite)

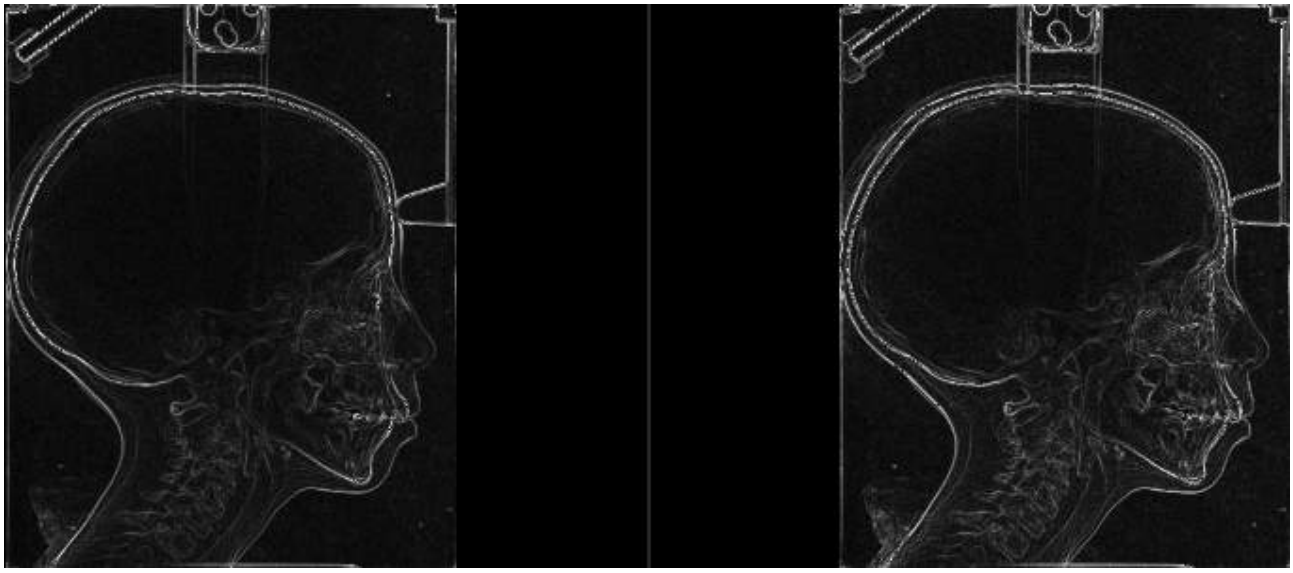

FIGURE 10 – Masque de Prewitt (à gauche) et de Sobel (à droite)

Les deux filtres de Prewitt et de Sobel sont tous les deux directionnels.
On peut remarquer que la détection des contours par masque de Sobel est plus efficace car la série des valeurs 1 2 1 est approximativement celle d'un masque de gaussienne.

2.2 Opérateurs différentiels du deuxième ordre

Implanter une méthode de détection des contours du deuxième ordre.

Nous avons choisi d'implanter la méthode de détection **LoG** (Laplacian of Gaussian), qui se construit par superposition d'un masque spécifique sur une entrée filtrée.

```
// Filtrage de l'entrée
filtered_in = filter(in, nl, nc, 0.5);

// Application du filtrage LoG
in = lectureimagepgm(tmp_name, &nl, &nc);
double** LOG = computeLOG(in, nl, nc, M1);

double seuil = 0;

for (int i = 0; i < nl; ++i) {
        for (int j = 0; j < nc; ++j) {
                if (LOG[i][j] > seuil-1000)
                        seuil = LOG[i][j];
                else
                        LOG[i][j] = 0;
        }
}
```

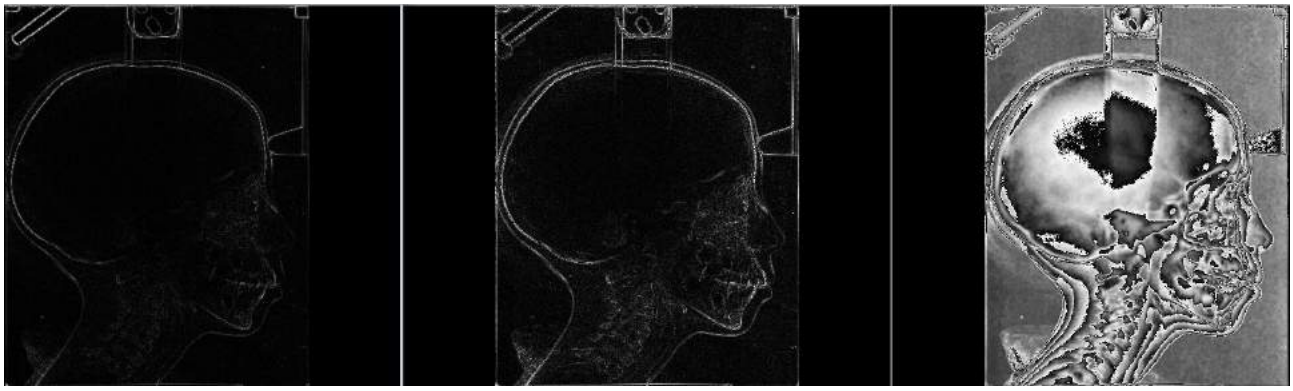

FIGURE 11 – Contours par masques sur entrée lissée

On peut remarquer qu'en fonction des différents filtres, le résultat est plus ou moins proche de l'image réelle.
Le dernier résultat montre qu'en fonction du masque, des exagérations peuvent survenir, dues au bruit qui n'a pas été assez filtré par la gaussienne.

2.3 Comparaison détection par Gradient / Laplacien

Détection par Gradient :

- Orientation des contours
- Bonne localisation des contours malgrès une épaisseur plus conséquente
- Peu sensible au bruit

Détection par Laplacien :

- Contours fermés
- Invariance en rotation