Transformation visuelle d'une image avec la morphologie mathématique.

TIPE 2025 : Transition, transformation, conversion.

Étienne Debacq, n°16370.

CPGE MPI.

Novembre 2024 - Mai 2025.

Introduction historique.

Georges Matheron.

École des Mines de Paris.

Problématique.

Dans quelle mesure la morphologie mathématique permet-elle d'obtenir des transformations efficaces pour les images grayscale ?

Objectifs.

- Une étude de la morphologie mathématique.
- Une implémentation des outils étudiés dans le langage C.
- Une recherche d'algorithmes optimisés.

Sommaire.

Présentation de la théorie.

Deux points de vue possibles.

- Point de vue ensembliste dans $\mathcal{P}(\mathbb{R}^n)$.
- Point de vue fonctionnelle dans $\mathcal{F}(\mathbb{R}^n, \mathbb{R})$.

Restriction.

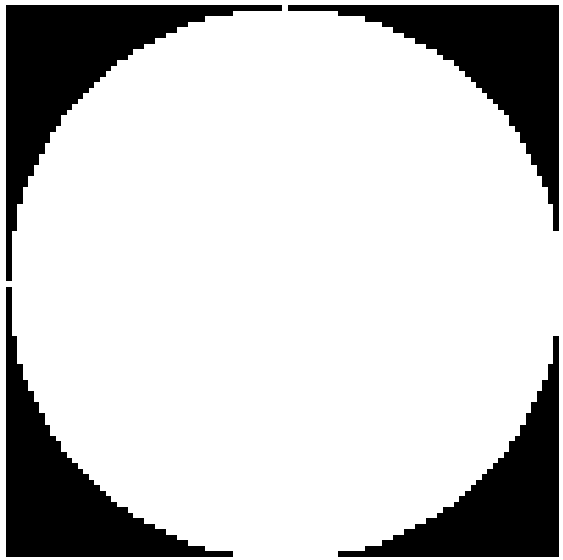
- De $\mathbb{R}^n$ à $\mathbb{N}^n$.
- Étude pour $n = 2$.

Définition : élément structurant.

Ensemble $\mathcal{B}$ dont la forme et la taille sont choisies.
On appelle origine de $\mathcal{B}$ un point de l'espace arbitraire.

Rôle :

Permet de modifier l'image à notre guise.



Disque structurant de rayon 50 et d'origine le centre du disque.

Définition : dilatation fonctionnelle.

On note $\mathcal{D}(f, \mathcal{B})$ la fonction suivante :

$$\mathcal{D}(f, \mathcal{B}) : x \mapsto \sup\{f(y), y \in \tau(\mathcal{B}, x)\}$$

où $\tau(\mathcal{B}, x)$ est la translation de $\mathcal{B}$ dont l'origine est x.

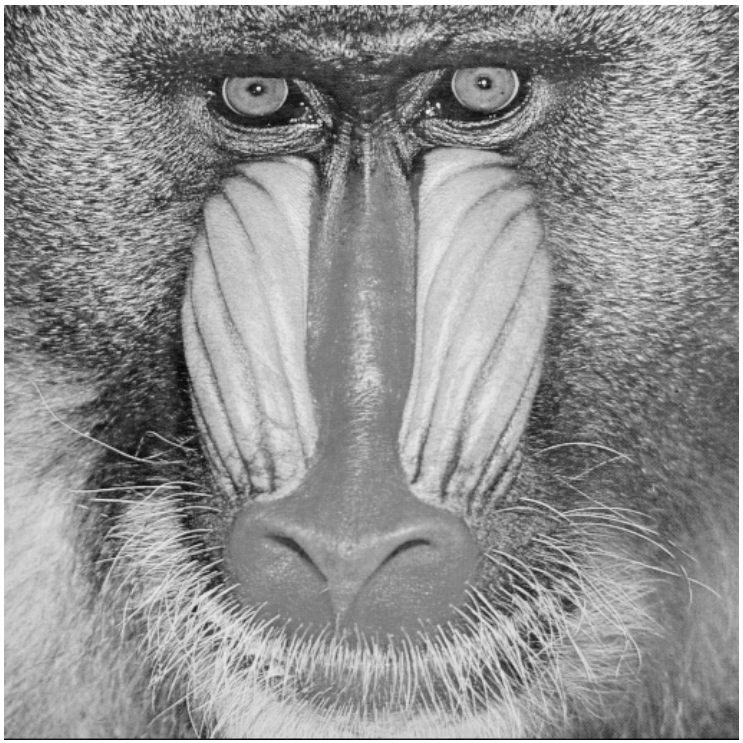
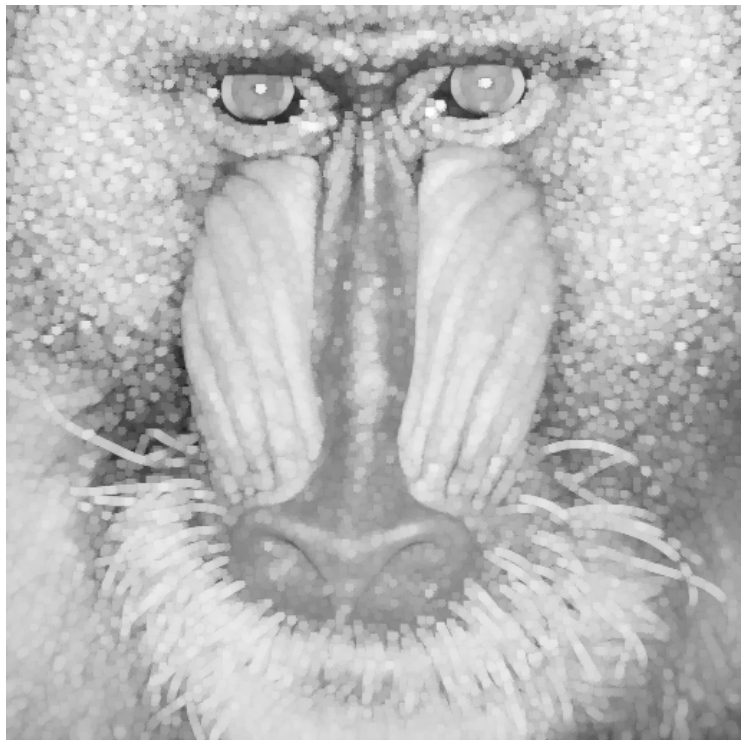


Image d'origine.



Dilatation par un disque de rayon 3.

Définition : érosion fonctionnelle.

On note $\mathcal{E}(f, \mathcal{B})$ la fonction suivante :

$$\mathcal{E}(f, \mathcal{B}) : x \mapsto \inf\{f(y), y \in \tau(\mathcal{B}, x)\}$$

où $\tau(\mathcal{B}, x)$ est la translation de $\mathcal{B}$ dont l'origine est x.

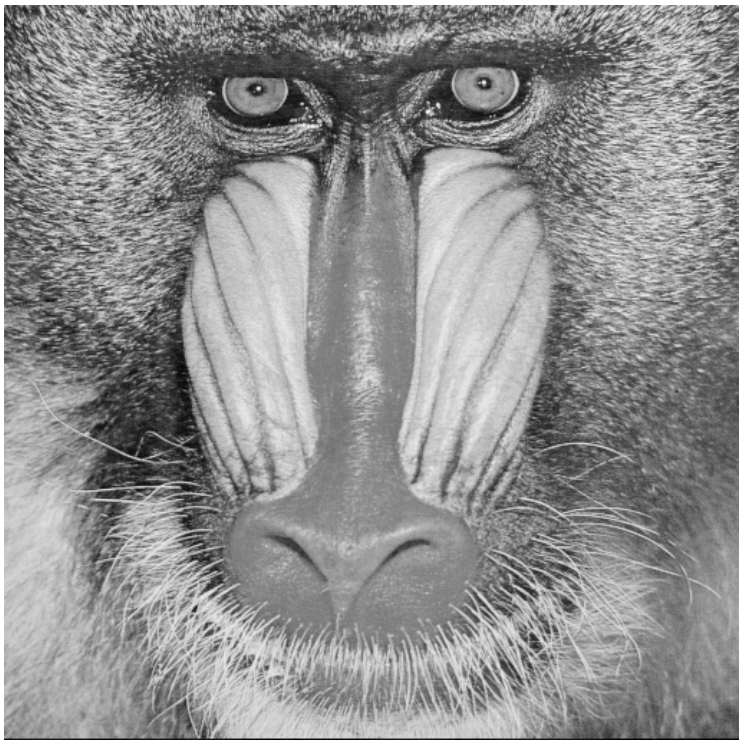
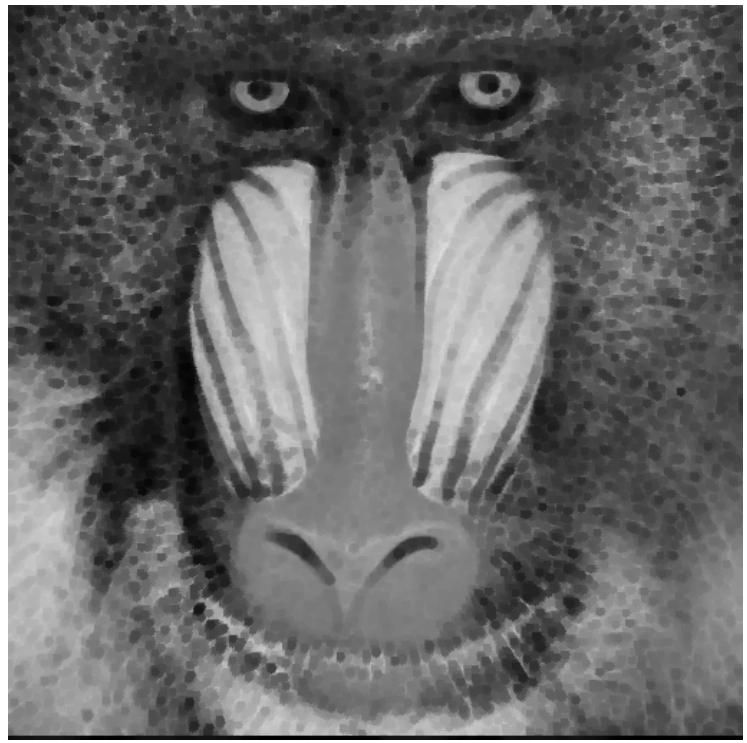


Image d'origine.



Érosion par un disque de rayon 3.

Définition : ouverture fonctionnelle.

L'ouverture fonctionnelle de f par $\mathcal{B}$ est :

$$f_{\mathcal{B}} := \mathcal{D}(\mathcal{E}(f, \mathcal{B}), \mathcal{S}_{\mathcal{B}})$$

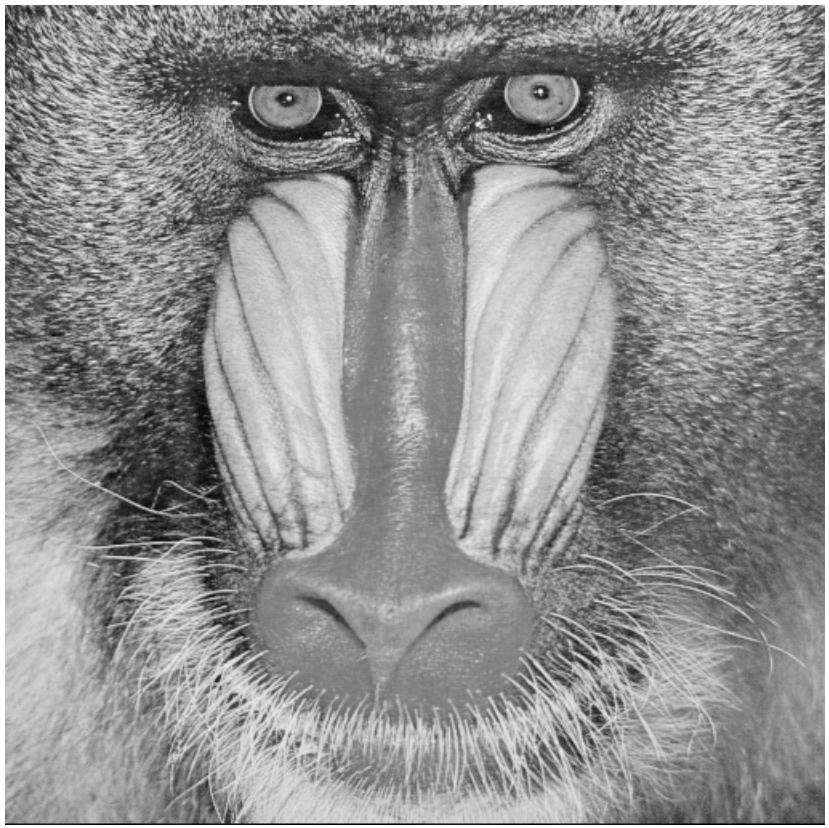
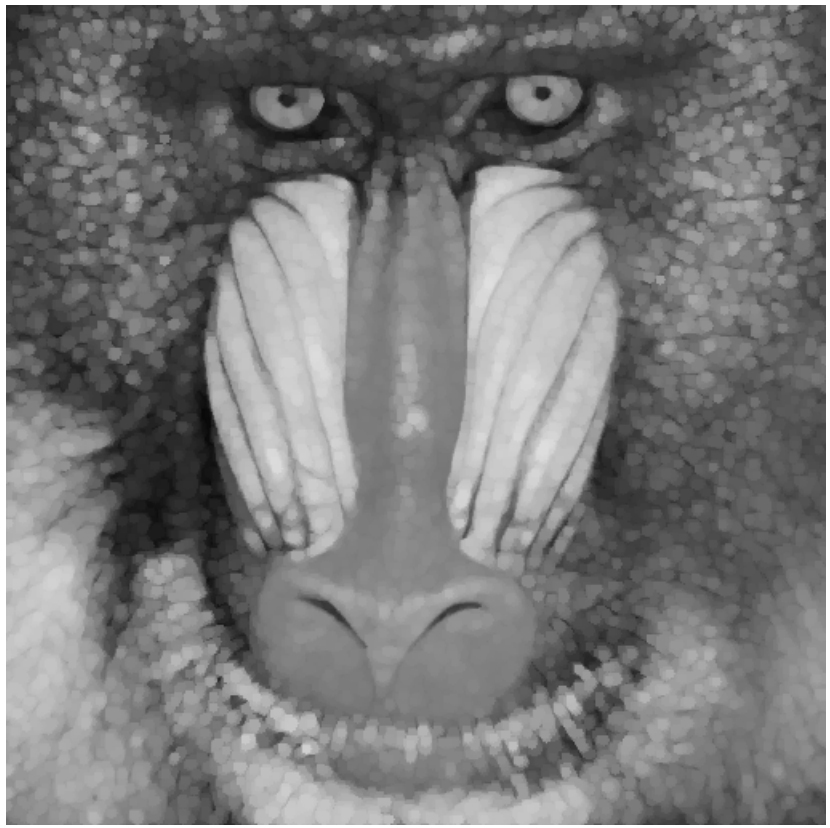


Image d'origine.



Ouverture par un disque de rayon 3.

Définition : fermeture fonctionnelle.

La fermeture fonctionnelle de f par $\mathcal{B}$ est :

$$f^{\mathcal{B}} := \mathcal{E}(\mathcal{D}(f, \mathcal{B}), \mathcal{S}_{\mathcal{B}})$$

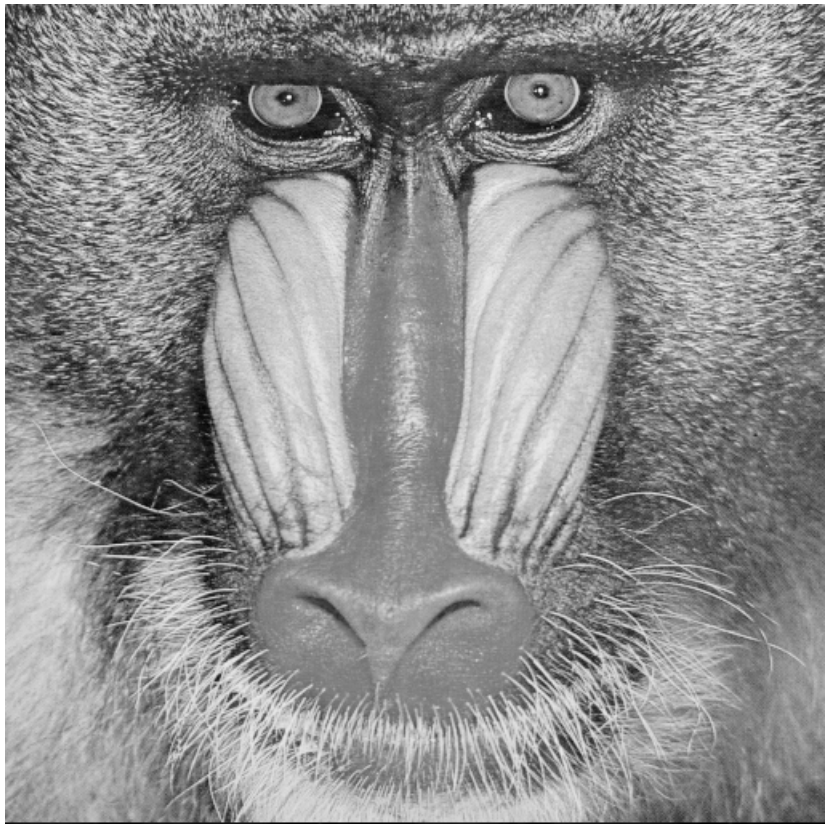
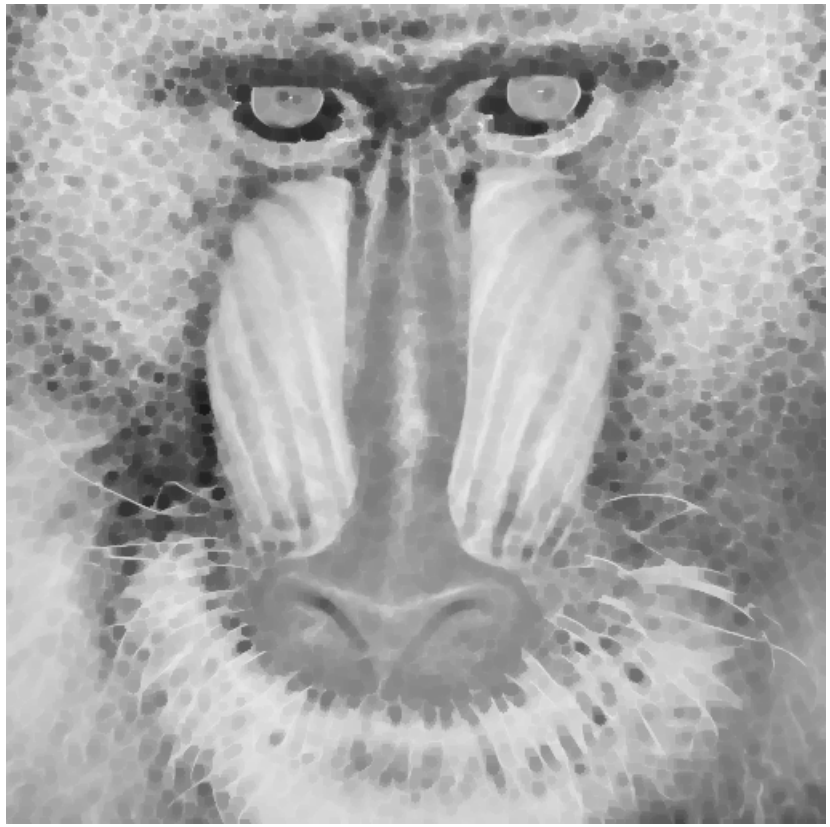


Image d'origine.



Fermeture par un disque de rayon 3.

Restriction.

$\mathcal{B}$ est un disque centré, de rayon variable et petit.

Conséquence :

Dans le cas d'un disque centré : $\mathcal{S}_{\mathcal{B}} = \mathcal{B}$.

Choix du langage.

Justifications du choix du C.

- Exécution plus rapide que d'autres langages.
- Pas de fuite de mémoire et contrôle de celle-ci.
- Au programme de MPI.

Implémentation des opérations fondamentales.

Définition : types structurés.

Trois types structurés :

- Image : pour tracer les figures au format .pgm ;
- Fonction : $\mathcal{M}_{L_0,L_1}(\llbracket 0;255 \rrbracket)$;
- Structurant : $\mathcal{M}_{2r}(\mathbb{Z}/2\mathbb{Z})$ munie d'une origine dans $\mathbb{N}^2$.

Complexité.

Remarque :

$r \ll L_0$ et $r \ll L_1$

Conséquence :

Négligeabilité des opérations sur les éléments structurants.

Théorème : complexité des opérations.

Toutes les opérations sont en $\mathcal{O}(L_0 \times L_1)$ pour cette implémentation.

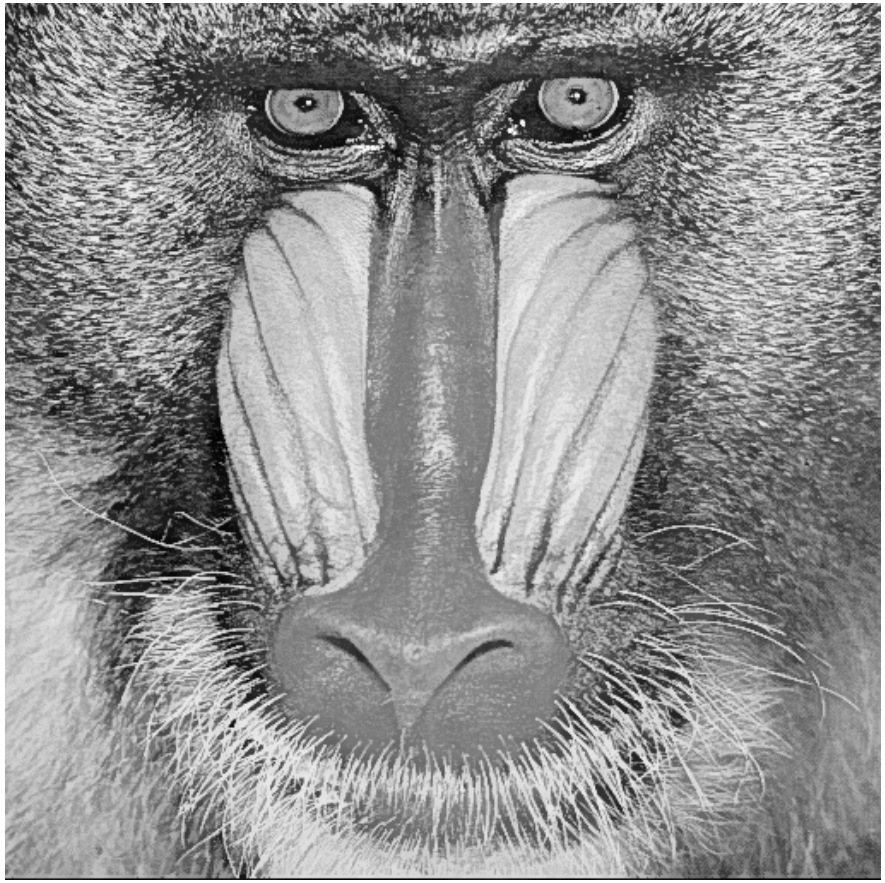
Création d'opérations complexes.

Définition : rehaussement de contraste morphologique.

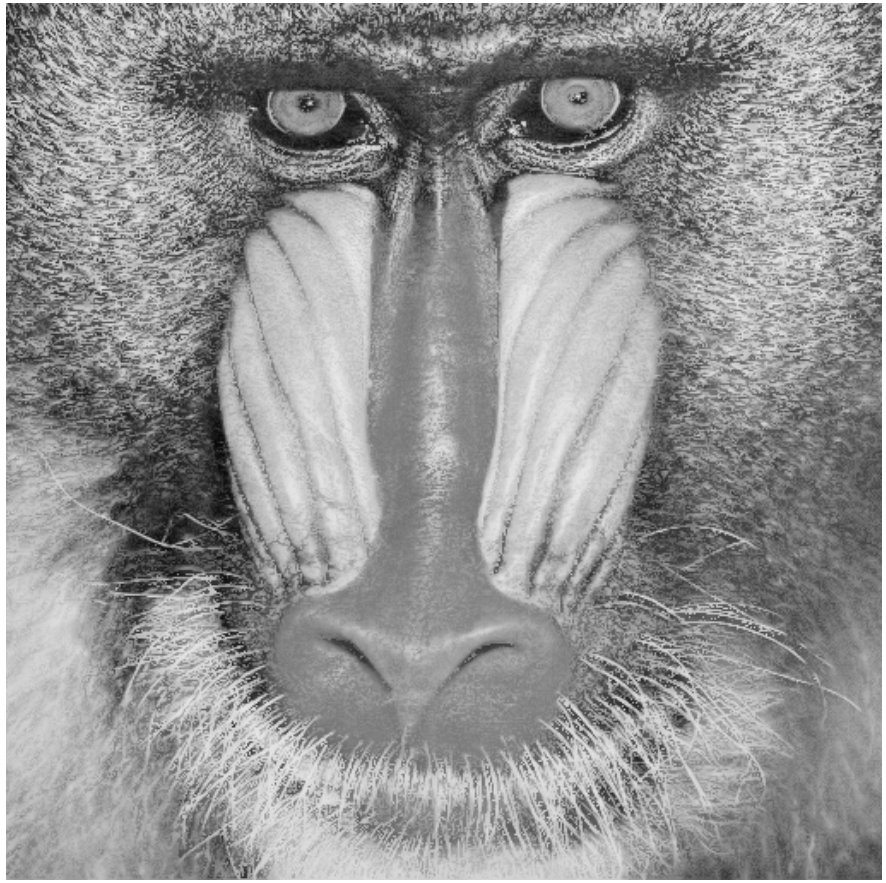
Pour une fonction f, $\underline{f}$ minorant f et $\overline{f}$ majorant f, $\alpha \geq 0$ et $\beta \geq 0$ avec $\alpha + \beta < 1$. On a :

$$\mathcal{C}_f(x) = \begin{cases} \underline{f}(x) & \text{si } \underline{f}(x) \leq f(x) \leq \underline{f}(x) + \alpha \Delta f(x) \\ f(x) & \text{si } \underline{f}(x) + \alpha \Delta f(x) \leq f(x) \leq \overline{f}(x) - \beta \Delta f(x) \\ \overline{f}(x) & \text{si } \overline{f}(x) - \beta \Delta f(x) \leq f(x) \leq \overline{f}(x) \end{cases} \quad (1)$$

où $\Delta f = \overline{f} - \underline{f}$.



$\alpha = 0,33; \beta = 0,33; \mathcal{E}$ et $\mathcal{D}$.



$\alpha = 0,33; \beta = 0,33; f_{\mathcal{B}}$ et $f^{\mathcal{B}}$.

Définition : gradient morphologique.

Le gradient morphologique de f est défini par :

$$\mathcal{G}r_f = \mathcal{D}(f, \mathcal{B}) - \mathcal{E}(f, \mathcal{B})$$

où $\mathcal{B}$ est le disque unité.

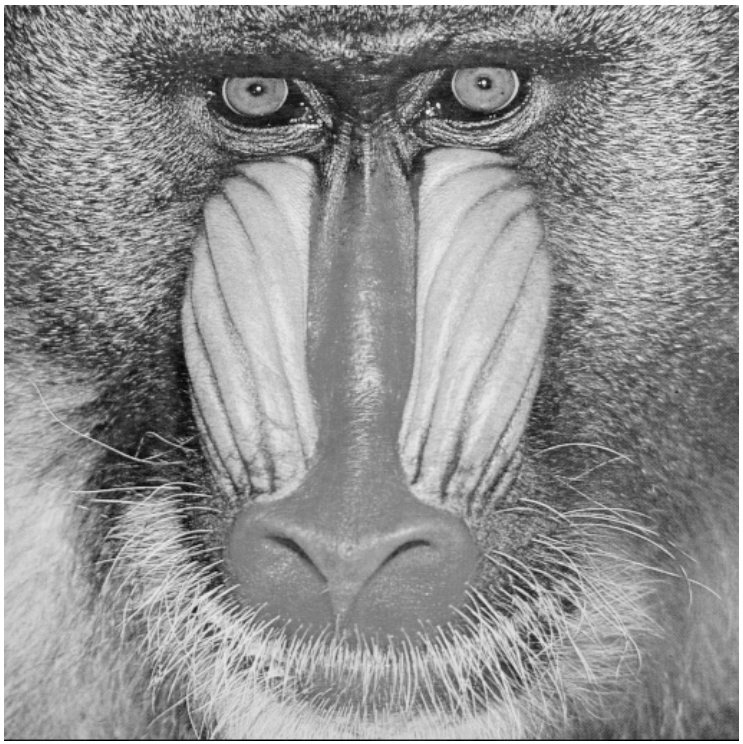


Image d'origine.

Gradient morphologique.

Définition : chapeau haut-de-forme.

On définit le chapeau haut-de-forme de f par $\mathcal{B}$ par :

$$\mathcal{CHF}_f = f - f_{\mathcal{B}}$$

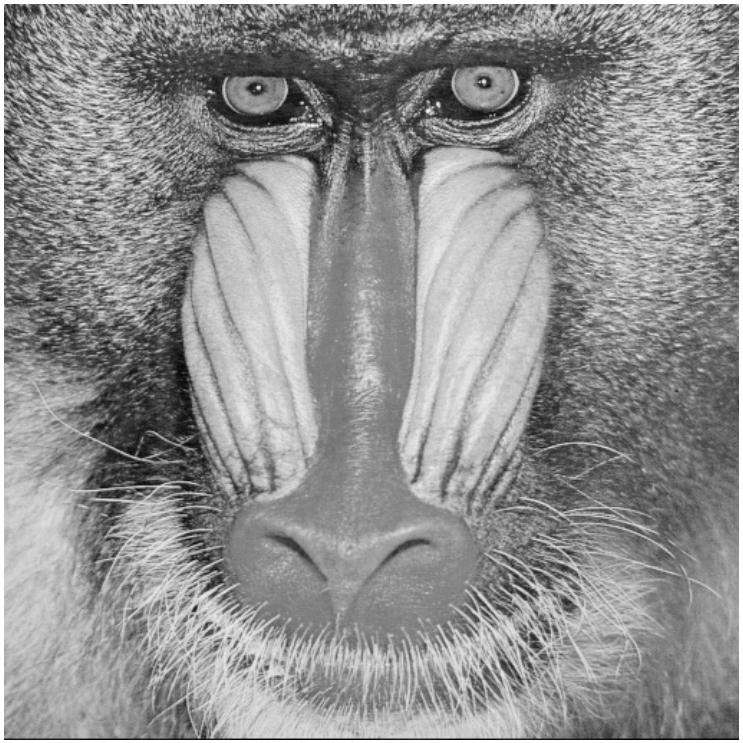
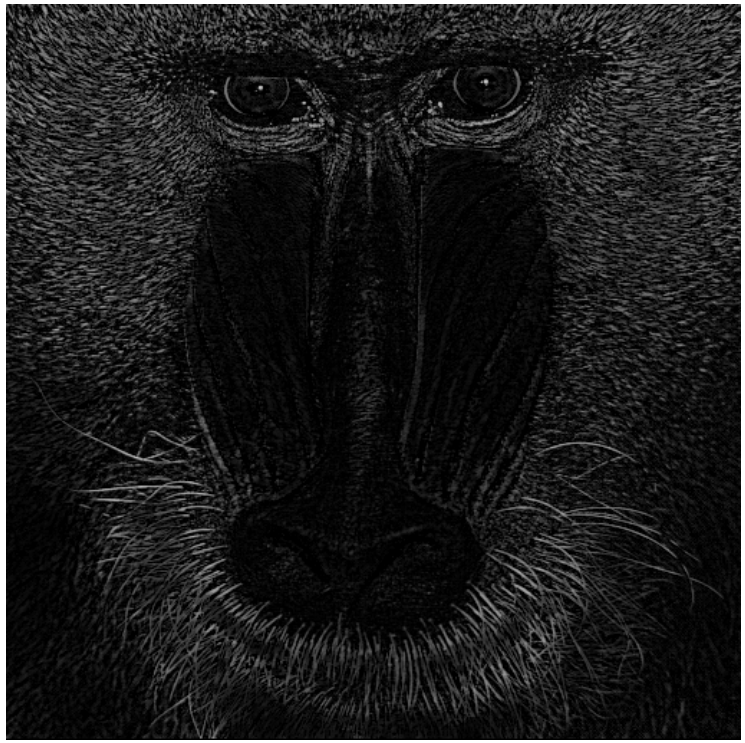


Image d'origine.



Chapeau haut-de-forme par un disque de rayon 3.

Conclusion : adaptation aux couleurs.

Adaptation sur chaque partie des images en s'appuyant sur les images grayscale :
Couleurs $\rightsquigarrow$ trois composantes $\rightsquigarrow$ traitement grayscale sur ces composantes $\rightsquigarrow$ recomposition de l'image.

Conclusion : quelques applications.

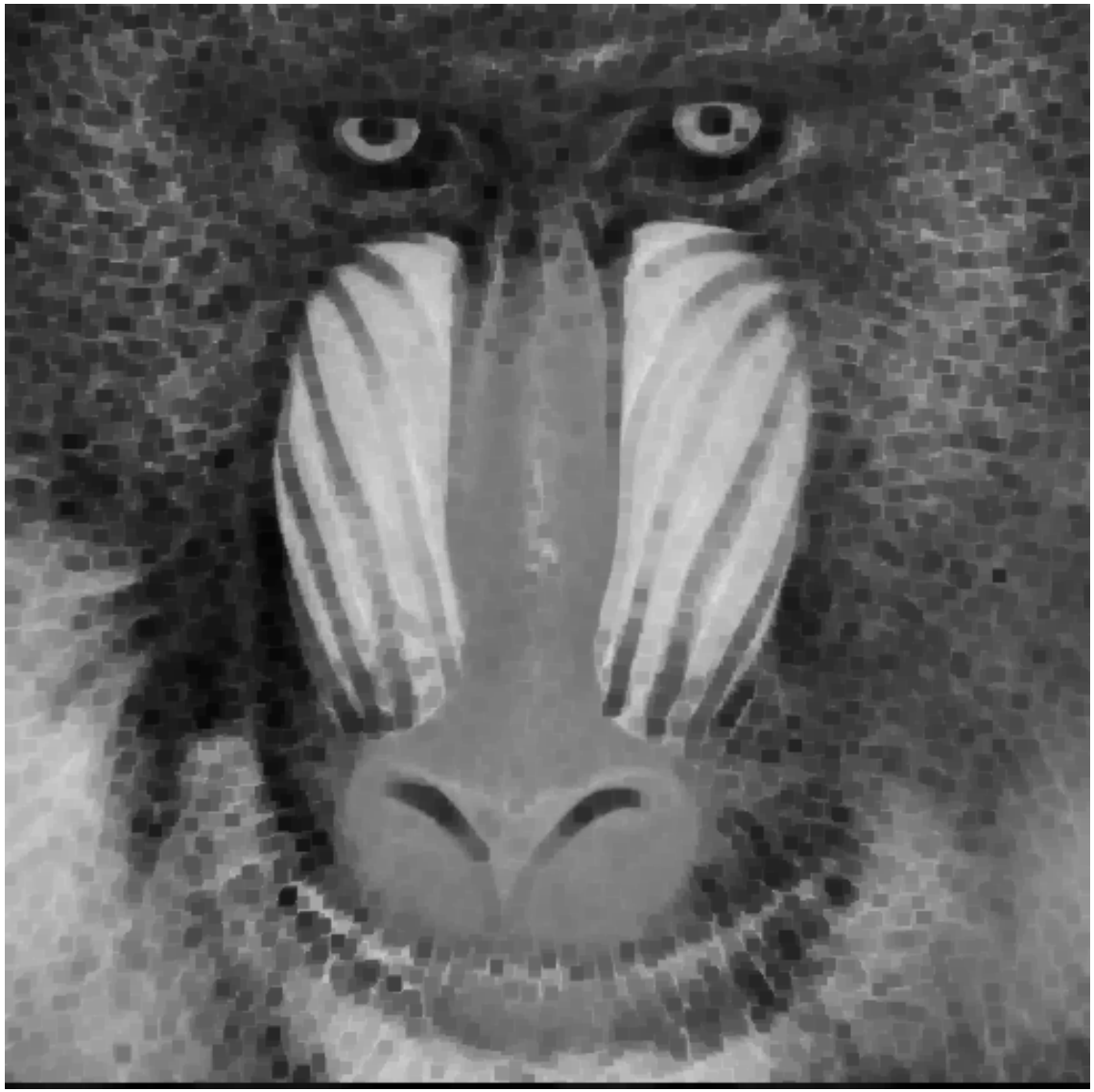
$\hookrightarrow$ Reconnaissance de formes.

$\hookrightarrow$ Tamisage pour une étude des sols.

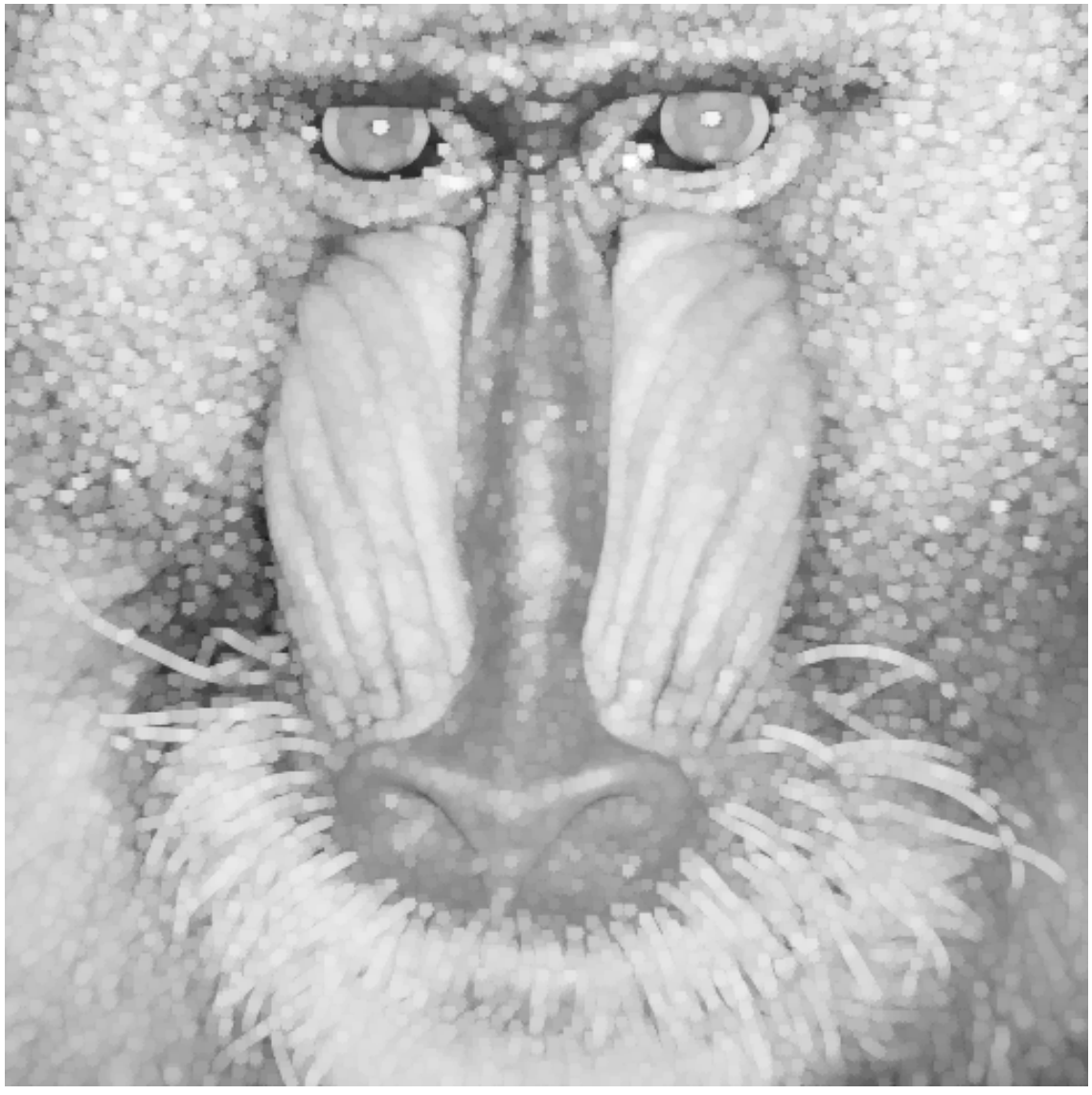
$\hookrightarrow$ Ligne de partage des eaux.

FIN.
Merci de votre attention.

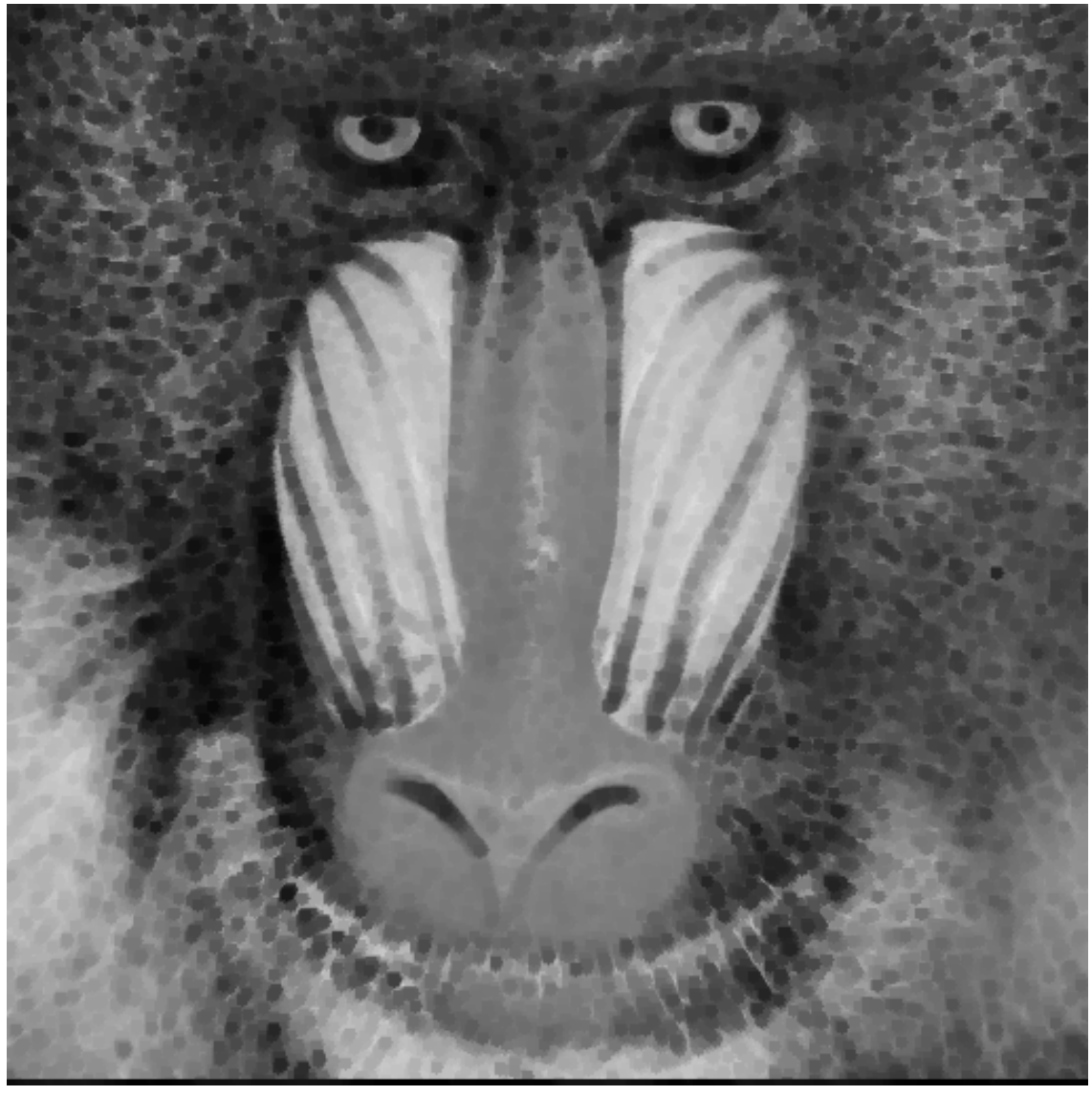
Annexe : illustrations.



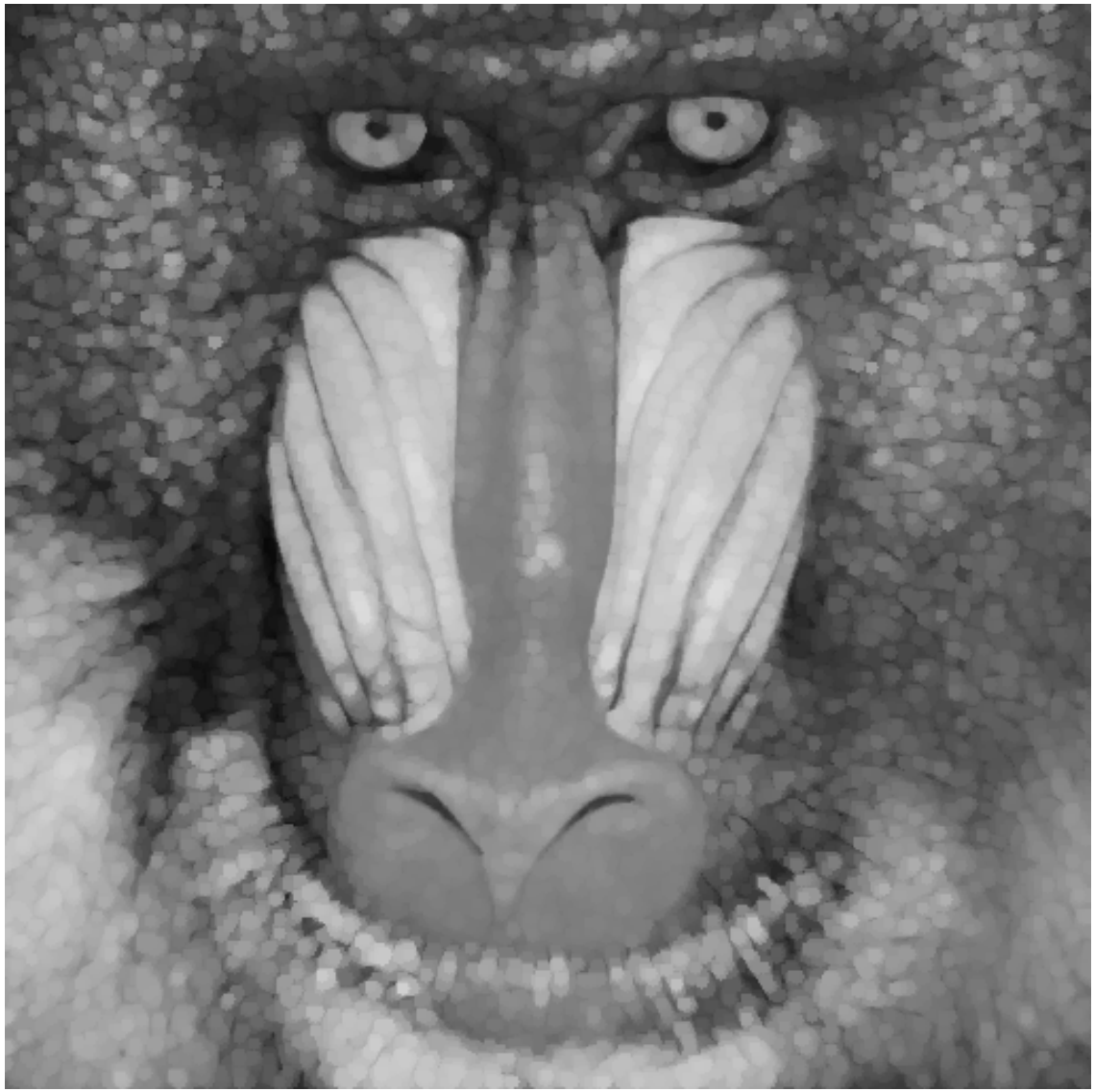
Érosion par un carré de côté 3.



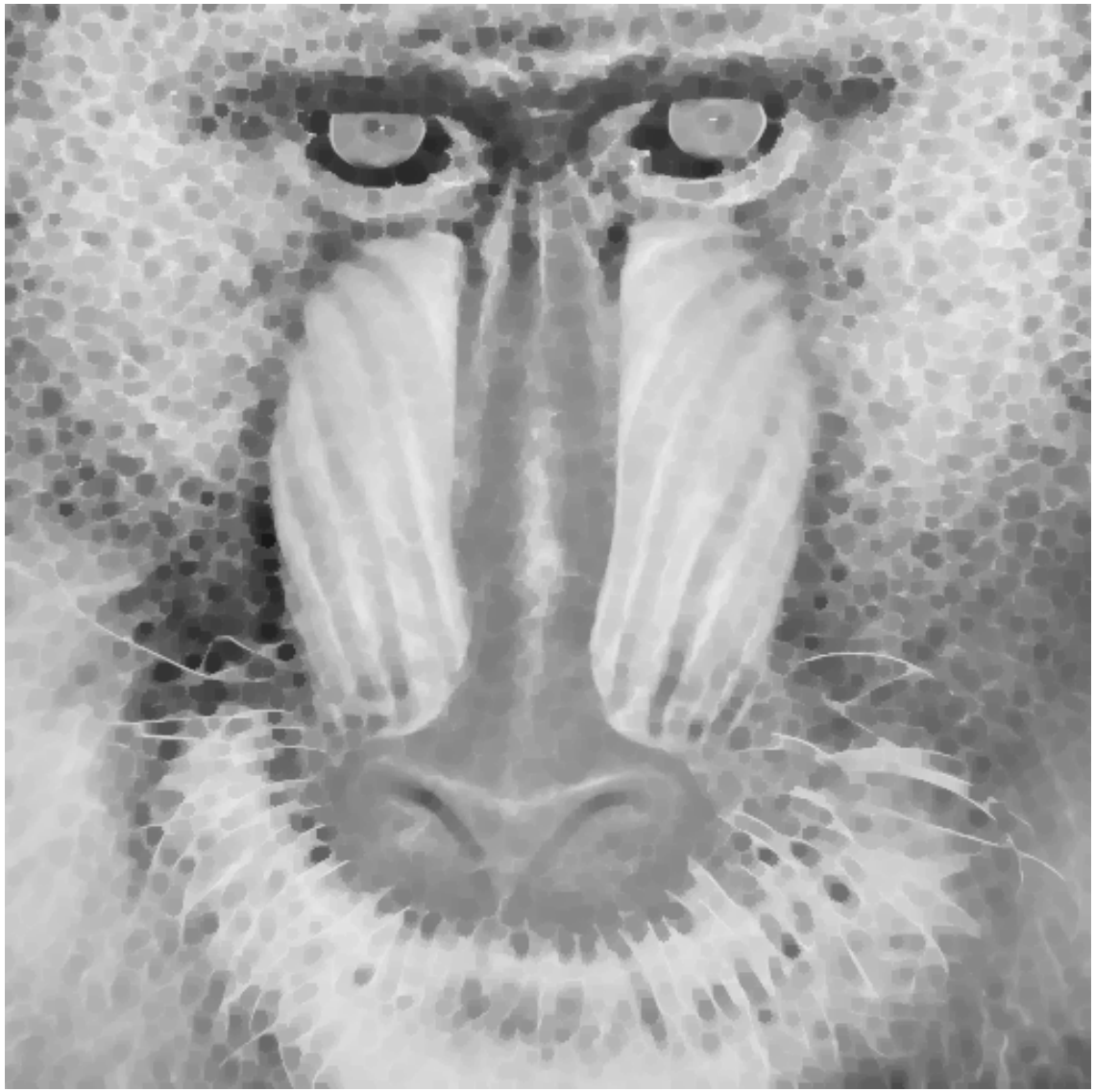
Dilatation par un disque de rayon 3.



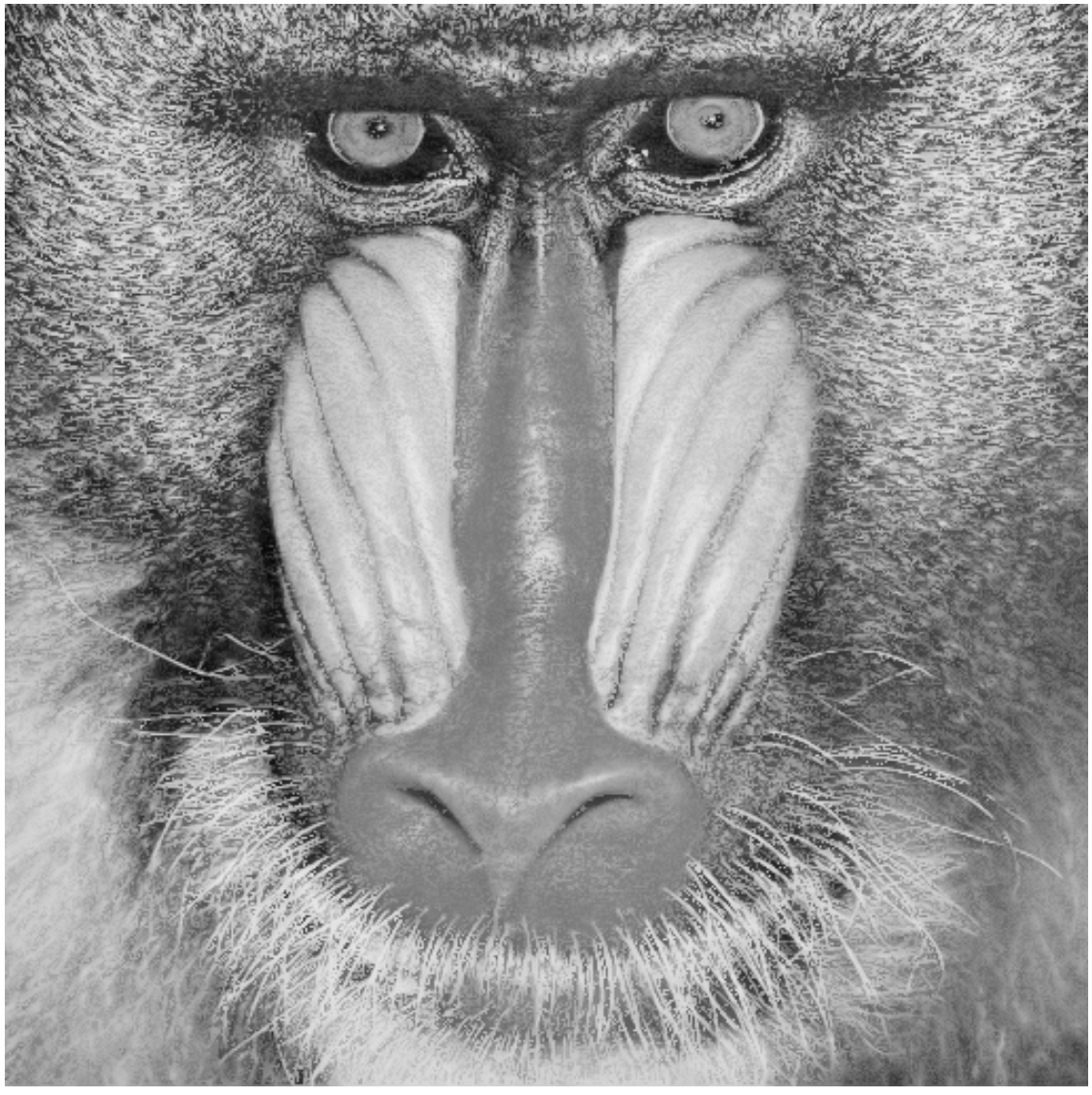
Érosion par un disque de rayon 3.



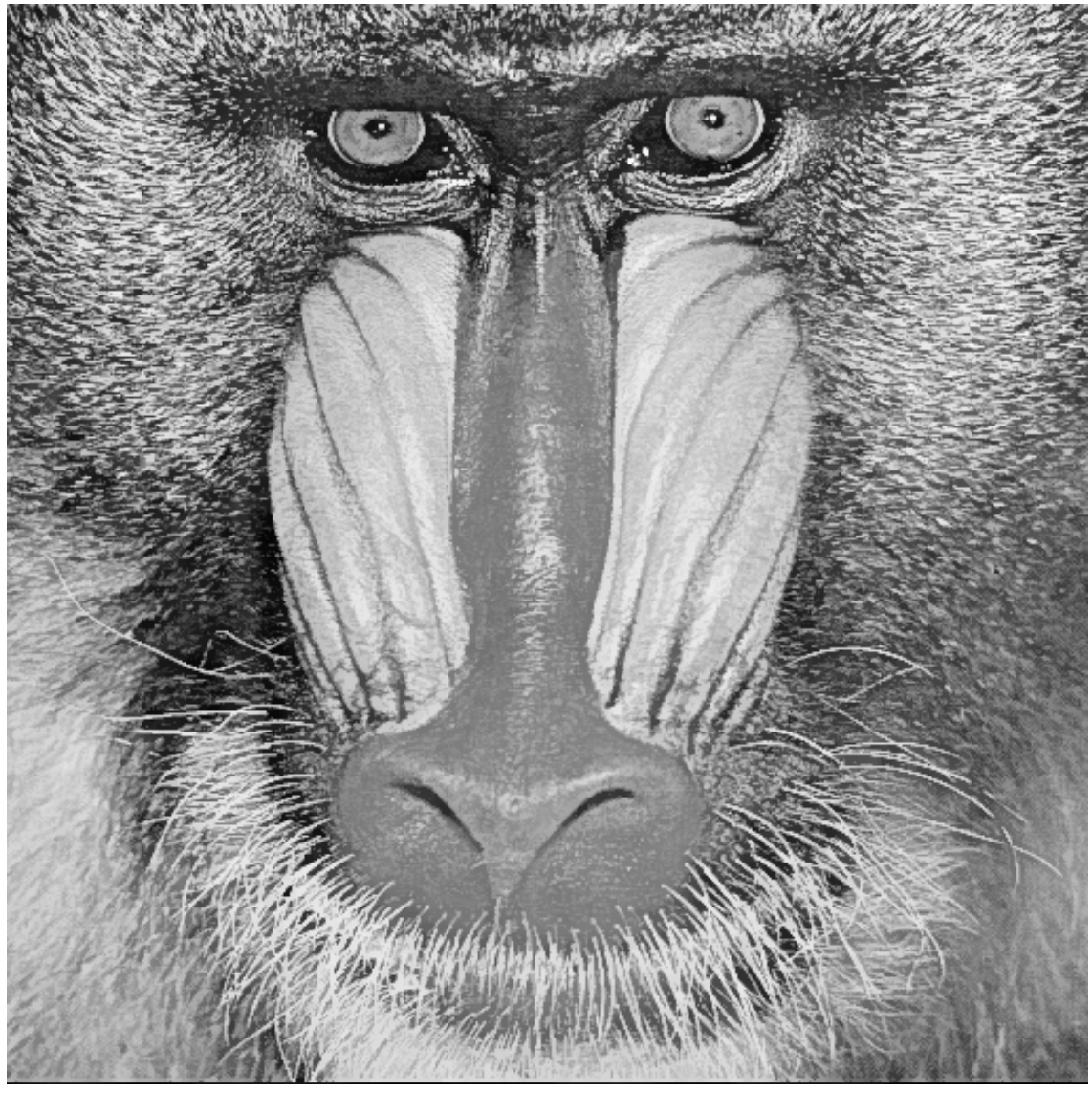
Ouverture par un disque de rayon 3.



Fermeture par un disque de rayon 3.

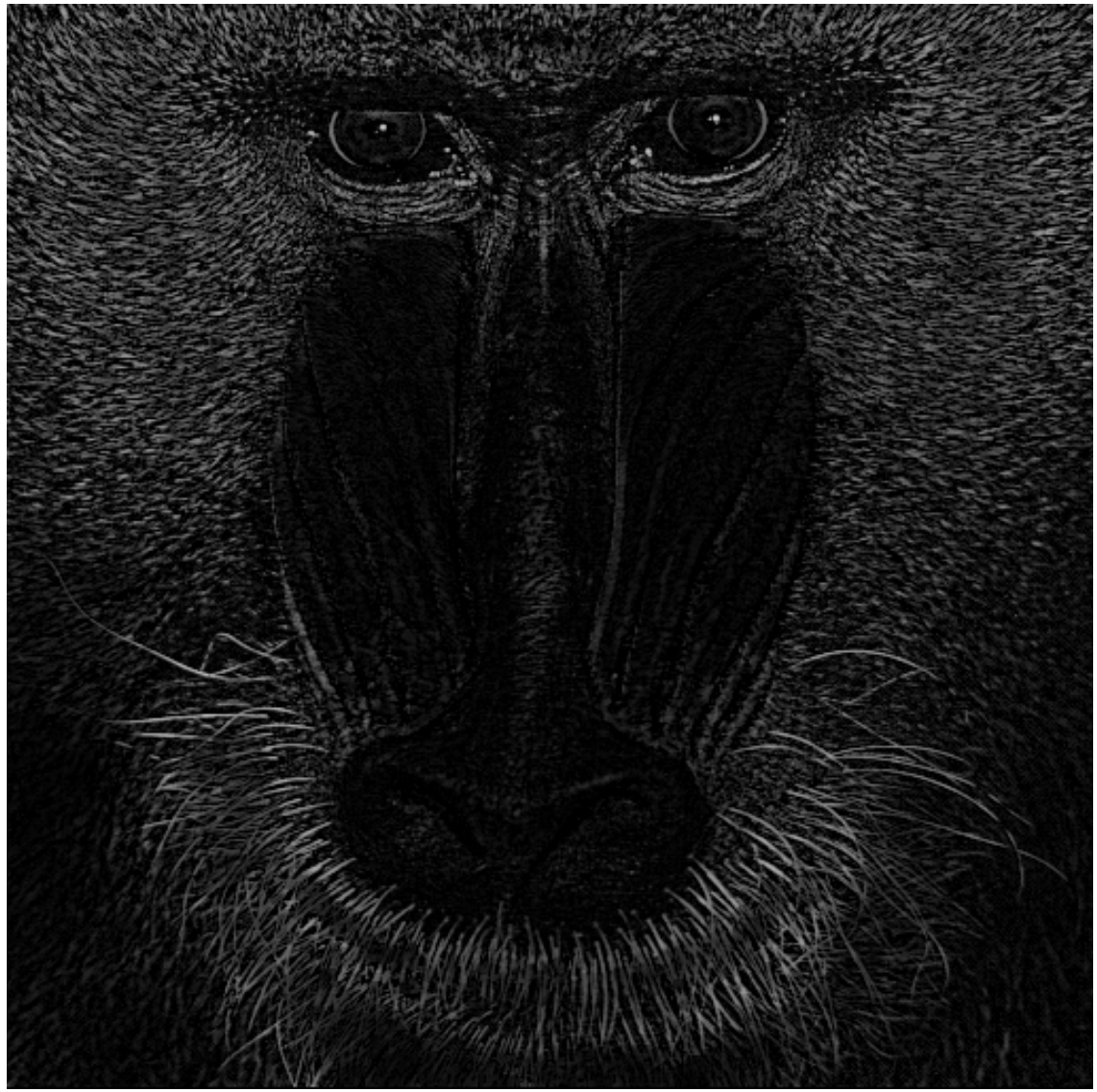


$\alpha = 0,4; \beta = 0,5; f_{\mathcal{B}}$ et $f^{\mathcal{B}}$.



$\alpha = 0,4; \beta = 0,5; \mathcal{E}$ et $\mathcal{D}$.

Gradient morphologique.



Chapeau haut-de-forme par un disque de rayon 3.

Annexe : algorithmes.

Algorithm Dilatation fonctionelle.

Require: $f \in \mathcal{F}(\mathbb{N}^2, [\![0;255]\!])$, $\mathcal{B} \in \mathcal{P}(E)$ un élément structurant.
Ensure: : $\mathcal{D}(f, \mathcal{B})$.

1: $g \leftarrow 0_{\mathcal{F}(\mathbb{N}^2,[\![0;255]\!])}$
2: $x \leftarrow 0_{\mathbb{N}^2}$
3: **for** $i \leftarrow 0$ jusqu'à $L_0 - 1$ **do**
4: **for** $j \leftarrow 0$ jusqu'à $L_1 - 1$ **do**
5: $x \leftarrow (i, j)$
6: $g(x) \leftarrow \sup_{y \in \tau(\mathcal{B}, x)} \{f(y)\}$
7: **end for**
8: **end for**
9: **return** g

Annexe : code.

```
1  typedef struct fonction{
2      int *L;
3      int **valeurs;
4  } fonction;
5
6  typedef struct structurant{
7      bool **forme;
8      int *origine;
9      int rayon;
10 } structurant;
11
12 typedef struct image{
13     int hauteur;
14     int largeur;
15     int **valeurs;
16 } image;
```

```c
void ecrire_image(char *nom_fichier, image *img){
    FILE *fichier = fopen(nom_fichier, "w");
    if(fichier == NULL){
        printf("Erreur ! Fichier non ouvert.");
        return;
    }
    fprintf(fichier, "P2\n%d %d\n", img->largeur, img->
    hauteur);
    fprintf(fichier, "%d\n", 255);
    for(int i = 0; i < img->hauteur; i++){
        for(int j = 0; j < img->largeur; j++){
            fprintf(fichier, "%d ", img->valeurs[i][j]);
        }
        fprintf(fichier, "\n");
    }
    fclose(fichier);
}
```

```c
image *recupere_image(char *nom_fichier){
    FILE *fichier = fopen(nom_fichier, "r");
    if(fichier == NULL){
        printf("Erreur ! Fichier non ouvert.\n");
        return NULL;
    }
    char *mode = malloc(3 * sizeof(char));
    if (mode == NULL) {
        printf("Erreur d'allocation pour le mode.\n");
        fclose(fichier);
        return NULL;
    }

    int hauteur, largeur, max;
    fscanf(fichier, "%s %d %d %d", mode, &hauteur, &largeur,
     &max);
    printf("%s -> Mode: %s, Largeur: %d, Hauteur: %d, Max: %
    d\n", nom_fichier, mode, largeur, hauteur, max);
```

```
1     image *img = malloc(sizeof(image));
2     if (img == NULL) {
3         printf("Erreur d'allocation pour l'image.\n");
4         free(mode);
5         fclose(fichier);
6         return NULL;
7     }
8
9     img->hauteur = hauteur;
10    img->largeur = largeur;
11    img->valeurs = malloc(hauteur * sizeof(int*));
12    if (img->valeurs == NULL) {
13        printf("Erreur d'allocation pour les lignes.\n");
14        free(mode);
15        free(img);
16        fclose(fichier);
17        return NULL;
18    }
```

```
1     for(int i = 0; i < hauteur; i++){
2         img->valeurs[i] = malloc(largeur * sizeof(int));
3         if (img->valeurs[i] == NULL) {
4             printf("Erreur d'allocation pour la colonne %d.\
    n", i);
5             for (int k = 0; k < i; k++) free(img->valeurs[k
    ]);
6             free(img->valeurs);
7             free(img);
8             free(mode);
9             fclose(fichier);
10            return NULL;
11        }
12    }
13
14    for(int i = 0; i < img->hauteur; i++){
15        for(int j = 0; j < img->largeur; j++){
16            fscanf(fichier, "%d", &(img->valeurs[i][j]));
17        }
18    }
```

```
    free(mode);
    fclose(fichier);
    return img;
}

image *image_depuis_fonction(fonction *f){
    image *img = malloc(sizeof(image));
    img->hauteur = f->L[0];
    img->largeur = f->L[1];
    img->valeurs = malloc(img->hauteur * sizeof(int *));
    for(int i = 0; i < img->hauteur; i++){
        img->valeurs[i] = malloc(img->largeur * sizeof(int))
    ;
        for(int j = 0; j < img->largeur; j++){
            img->valeurs[i][j] = f->valeurs[i][j];
        }
    }
    return img;
}
```

```
fonction *fonction_nulle(int *L){
    fonction *f_0 = malloc(sizeof(fonction));
    f_0->L = L;
    f_0->valeurs = malloc(L[0]*sizeof(int*));
    for(int i = 0; i < L[0]; i++){
        f_0->valeurs[i] = malloc(L[1]*sizeof(int));
        for(int j = 0; j < L[1]; j++){
            f_0->valeurs[i][j] = 0;
        }
    }
    return f_0;
}
```

```
structurant *structurant_vide(int *origine, int rayon){
    structurant *ret = malloc(sizeof(structurant));
    ret->rayon = rayon;
    ret->forme = malloc(2*rayon*sizeof(bool*));
    for(int i = 0; i < 2*rayon; i++){
        ret->forme[i] = calloc(2*rayon, sizeof(bool));
    }
    ret->origine = origine;
    return ret;
}
```

```c
void dessine_fonction(fonction *f, fonction *g){
    for(int i = 0; i < f->L[0]; i++){
        for(int j = 0; j < f->L[1]; j++){
            if(g->valeurs[i][j] > 0){
                f->valeurs[i][j] = g->valeurs[i][j];
            }
        }
    }
    return;
}

void libere_fonction(fonction *f){
    for(int i = 0; i < f->L[0]; i++){
        free(f->valeurs[i]);
    }
    free(f->valeurs);
    free(f);
    return;
}
```

```c
void libere_structurant(structurant *B){
    for(int i = 0; i < 2*B->rayon; i++){
        free(B->forme[i]);
    }
    free(B->forme);
    free(B);
    return;
}

void libere_image(image *img){
    for(int i = 0; i < img->hauteur; i++){
        free(img->valeurs[i]);
    }
    free(img->valeurs);
    free(img);
    return;
}
```

```
fonction *fonction_of_image(image *img){
    int *L = malloc(2*sizeof(int));
    L[0] = img->hauteur; L[1] = img->largeur;
    fonction *C = fonction_nulle(L);
    for(int i = 0; i < L[0]; i++){
        for(int j = 0; j < L[1]; j++){
            C->valeurs[i][j] = img->valeurs[i][j];
        }
    }
    return C;
}
```

```
1  fonction *fonction_of_structurant(structurant *B){
2      int *L = malloc(2*sizeof(int));
3      L[0] = 2*B->rayon;
4      L[1] = 2*B->rayon;
5      fonction *f = fonction_nulle(L);
6      f->L = L;
7      for(int i = 0; i < 2*B->rayon; i++){
8          for(int j = 0; j < 2*B->rayon; j++){
9              f->valeurs[i][j] = 255*B->forme[i][j];
10         }
11     }
12     return f;
13 }
14
15 bool est_dedans(int i, int j, int *L){
16     return 0 <= i && i < L[0] && 0 <= j && j < L[1];
17 }
```

```
1  int sup(fonction *f, structurant *B){
2      int s = 0;
3      int o[2] = {B->origine[0], B->origine[1]};
4      int r = B->rayon;
5      for(int i = o[0] - r; i < o[0] + r; i++){
6          for(int j = o[1] - r; j < o[1] + r; j++){
7              if(est_dedans(i, j, f->L) && B->forme[i + r - o
       [0]][j + r - o[1]] && s <= f->valeurs[i][j]){
8                  s = f->valeurs[i][j];
9              }
10         }
11     }
12     return s;
13 }
```

```
1  int inf(fonction *f, structurant *B){
2      int s = 255;
3      int o[2] = {B->origine[0], B->origine[1]};
4      int r = B->rayon;
5      for(int i = o[0] - r; i < o[0] + r; i++){
6          for(int j = o[1] - r; j < o[1] + r; j++){
7              if(est_dedans(i, j, f->L) && B->forme[i + r - o
       [0]][j + r - o[1]] && s >= f->valeurs[i][j]){
8                  s = f->valeurs[i][j];
9              }
10         }
11     }
12     return s;
13 }
```

```
1  void translate_B_vers_x(structurant *B, int *x){
2      B->origine = x;
3      return;
4  }
5
6  structurant *Disque(int *L, int r, int *o){
7      structurant *B = structurant_vide(o, r);
8      for(int i = o[0] - r; i < o[0] + r; i++){
9          for(int j = o[1] - r; j< o[1] + r; j++){
10             int dist_2 = (i-o[0]) * (i-o[0]) + (j-o[1]) * (j
       -o[1]);
11             if(est_dedans(i, j, L)){
12                 if(dist_2 <= r*r){
13                     B->forme[i + r - o[0]][j + r - o[1]] =
       true;
14                 }
15             }
16         }
17     }
18     return B;
19 }
```

```
fonction *dilatation_fonctionnelle(fonction *f, structurant *B){
    fonction *f_d = fonction_nulle(f->L);
    int *x = malloc(2*sizeof(int));
    for(int i = 0; i < f->L[0]; i++){
        x[0] = i;
        for(int j = 0; j < f->L[1]; j++){
            x[1] = j;
            translate_B_vers_x(B, x);
            f_d->valeurs[i][j] = sup(f, B);
        }
    }
    free(x);
    return f_d;
}
```

```
1  fonction *erosion_fonctionnelle(fonction *f, structurant *B)
       {
2      fonction *f_e = fonction_nulle(f->L);
3      int *x = malloc(2*sizeof(int));
4      for(int i = 0; i < f->L[0]; i++){
5          x[0] = i;
6          for(int j = 0; j < f->L[1]; j++){
7              x[1] = j;
8              translate_B_vers_x(B, x);
9              f_e->valeurs[i][j] = inf(f, B);
10         }
11     }
12     free(x);
13     return f_e;
14 }
```

```
1 fonction *ouverture_numerique(fonction *f, structurant *B){
2     return dilatation_fonctionnelle(erosion_fonctionnelle(f,
      B), B);
3 }
4
5 fonction *fermeture_numerique(fonction *f, structurant *B){
6     return erosion_fonctionnelle(dilatation_fonctionnelle(f,
      B), B);
7 }
```

```
fonction *rehaussement_contraste(fonction *f, double alpha,
    double beta, fonction *f_inf, fonction *f_sup){
     fonction *g = fonction_nulle(f->L);
     for(int i = 0; i < f->L[0]; i++){
          for(int j = 0; j < f->L[1]; j++){
               int Delta_f_i_j = f_sup->valeurs[i][j] - f_inf->
    valeurs[i][j];
               if(f_inf->valeurs[i][j] <= f->valeurs[i][j] && f
    ->valeurs[i][j] <= f_inf->valeurs[i][j] + alpha*
    Delta_f_i_j){
                    g->valeurs[i][j] = f_inf->valeurs[i][j];
               }else if(f_inf->valeurs[i][j] + alpha*
    Delta_f_i_j <= f->valeurs[i][j] && f->valeurs[i][j] <=
    f_sup->valeurs[i][j] - beta*Delta_f_i_j){
                    g->valeurs[i][j] = f->valeurs[i][j];
               }else{
                    g->valeurs[i][j] = f_sup->valeurs[i][j];
               }
          }
     }
     return g;
}
```

```
1  fonction *gradient_morpho(fonction *f, int *origine){
2      fonction *g = fonction_nulle(f->L);
3      structurant *B = Disque(f->L, 1, origine);
4      fonction *D = dilatation_fonctionnelle(f, B);
5      fonction *E = erosion_fonctionnelle(f, B);
6      for(int i = 0; i < f->L[0]; i++){
7          for(int j = 0; j < f->L[1]; j++){
8              g->valeurs[i][j] = D->valeurs[i][j] - E->valeurs
       [i][j];
9          }
10     }
11     return g;
12 }
```

```
1  fonction *chapeau_haut_de_forme(fonction *f, structurant *B)
       {
2      fonction *g = fonction_nulle(f->L);
3      fonction *ouv = ouverture_numerique(f, B);
4      for(int i = 0; i < f->L[0]; i++){
5          for(int j = 0; j < f->L[1]; j++){
6              if(f->valeurs[i][j] - ouv->valeurs[i][j] <= 0){
7                  g->valeurs[i][j] = 0;
8              }
9              else{
10                 g->valeurs[i][j] = f->valeurs[i][j] - ouv->
       valeurs[i][j];
11             }
12         }
13     }
14     return g;
15 }
```

```c
int main(void){

    double alpha = 0.4;
    double beta = 0.5;

    int r_dilatation = 2;
    int r_erosion = 2;
    int r_ouverture = 2;
    int r_fermeture = 2;
    int r_chapeau = 2;

    image *img = recupere_image("Femme_P2.pgm");
    int L[2] = {img->hauteur, img->largeur};
    int origine[2] = {img->hauteur/2, img->largeur/2};
    fonction *f = fonction_of_image(img);
    structurant *D = Disque(L, 2, origine);

    fonction *f_inf = ouverture_numerique(f, D);
    fonction *f_sup = fermeture_numerique(f, D);
```

```
1      fonction *f_dilatation = dilatation_fonctionnelle(f, D);
2      fonction *f_erosion = erosion_fonctionnelle(f, D);
3      fonction *f_ouverture = ouverture_numerique(f, D);
4      fonction *f_fermeture = fermeture_numerique(f, D);
5      fonction *f_rehaussement = rehaussement_contraste(f,
     alpha, beta, f_inf, f_sup);
6      fonction *f_gradient = gradient_morpho(f, origine);
7      fonction *f_chapeau = chapeau_haut_de_forme(f, D);
```

```
1  ecrire_image("Femme_dilatee_r=2.pgm",
   image_depuis_fonction(f_dilatation));
2  ecrire_image("Femme_erodee_r=2.pgm",
   image_depuis_fonction(f_erosion));
3  ecrire_image("Femme_ouverte_r=2.pgm",
   image_depuis_fonction(f_ouverture));
4  ecrire_image("Femme_fermee_r=2.pgm",
   image_depuis_fonction(f_fermeture));
5  ecrire_image("Femme_rehaussee_ouv_fer_alpha=04_beta=05.
   pgm", image_depuis_fonction(f_rehaussement));
6  ecrire_image("Femme_gradient.pgm", image_depuis_fonction
   (f_gradient));
7  ecrire_image("Femme_chapeau_hdf_r=2.pgm",
   image_depuis_fonction(f_chapeau));
```

```
    libere_structurant(D);
    libere_fonction(f);
    libere_fonction(f_inf);
    libere_fonction(f_sup);
    libere_fonction(f_dilatation);
    libere_fonction(f_erosion);
    libere_fonction(f_ouverture);
    libere_fonction(f_fermeture);
    libere_fonction(f_rehaussement);
    libere_fonction(f_gradient);
    libere_fonction(f_chapeau);
    libere_image(img);

    return 0;
}
```