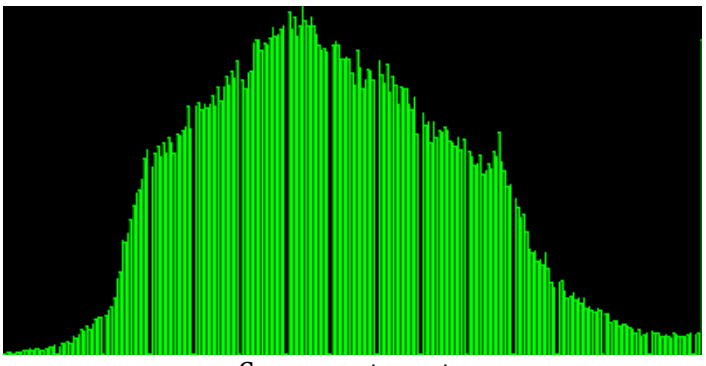
Manal Benaissa – Sylvain Salibas

EXERCICE 1

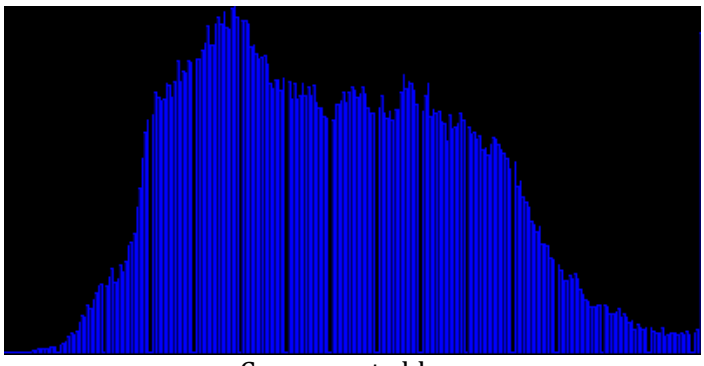
```
	// définition de quelques couleurs
RGB noir(0,0,0);
RGB blanc(255,255,255);
RGB rouge(255,0,0);
RGB vert(0,255,0);
RGB bleu(0,0,255);
RGB cyan(255,255,255);
RGB magenta(255,0,255);
RGB jaune(255,255,0);

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

	// chargement de l'image
	if (argc<2)
		I = ImageCouleurF("bastille.png");
	else
		I = ImageCouleurF(argv[1]);
	I.afficher();

	// nombre de classes de l'histogramme
	UINT nb_classes;
	if (argc<3)
		nb_classes = 256;
	else
		nb_classes = atoi(argv[2]);

	// ramener le nombre de classes entre 2 et 256
	if (nb_classes<2  ) nb_classes = 2  ;
	if (nb_classes>256) nb_classes = 256;
```

```
        // calculer l'histogramme de l'image I avec nb_classes classes
        Histogramme HR(I,nb_classes,true,false,false);
        Histogramme HG(I,nb_classes,false,true,false);
        Histogramme HB(I,nb_classes,false,false,true);
        Histogramme H(I,nb_classes,false,false,false);

        HR.afficher(noir, rouge, 400, 200);
        HG.afficher(noir, vert, 400, 200);
        HB.afficher(noir, bleu, 400, 200);
        H.afficher(blanc, noir, 400, 200);

        // affichage avec axe des x entre 0 et 1
        // avec la couleur noire pour le fond et
        // la couleur blanc pour l'histogramme
        //H.afficher_0_1(noir,blanc);

        // en attente de taper une touche
        printf("Taper un caractere au clavier pour continuer");
        getchar();
}
```

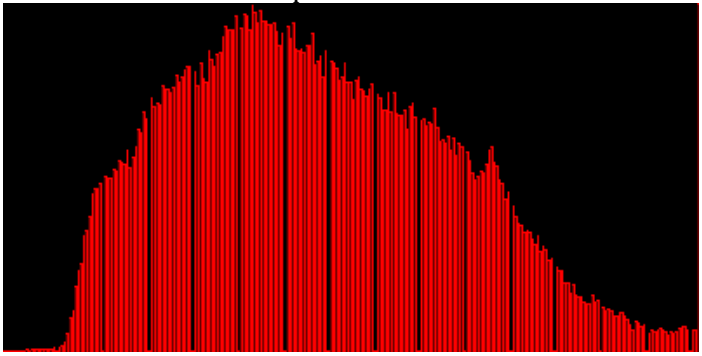
On obtient, pour l'image « Bastille.png », les histogrammes suivants :



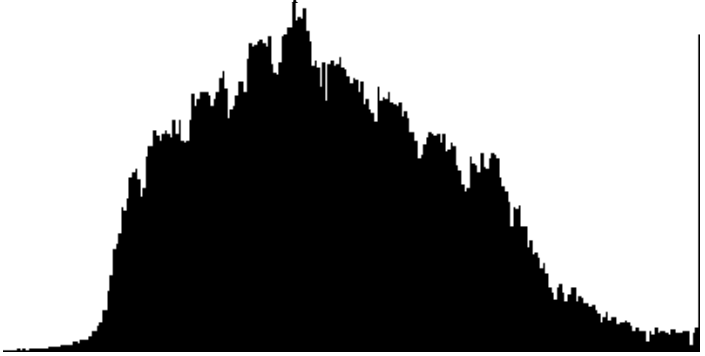
Composante verte



Composante bleue



Composante rouge

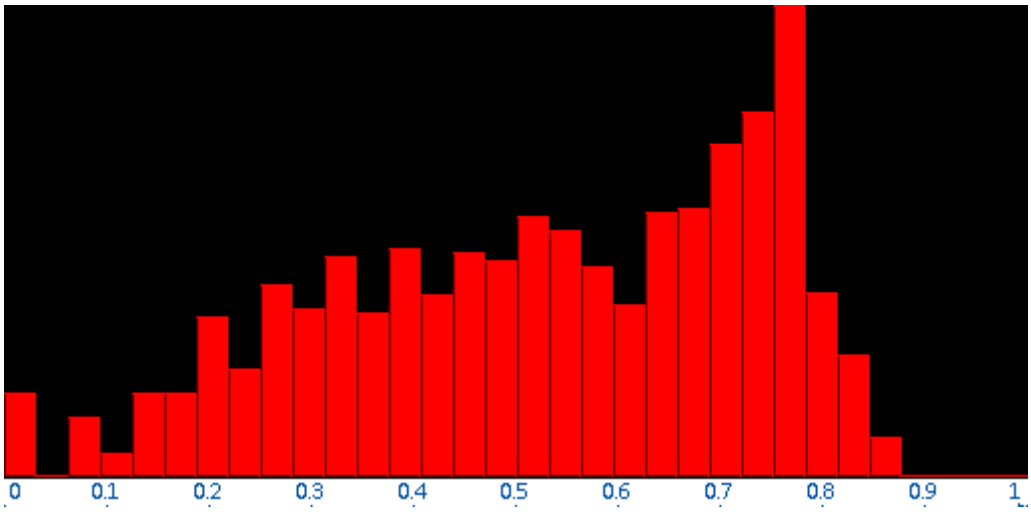


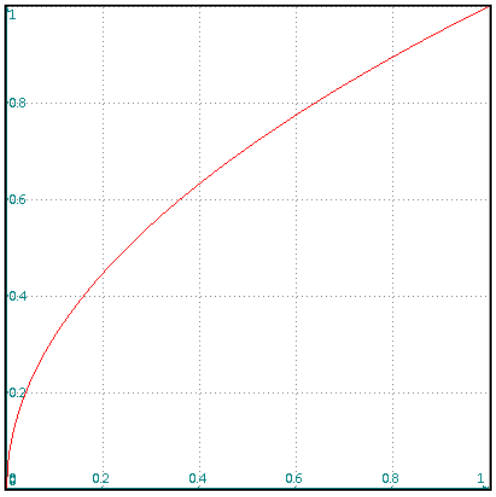
Conversion en gris

EXERCICE 2

```
float f1(float x)
{
	return pow(x,0.5); // correction gamma avec alpha=0.5
}

void exemple1()
{
	ImageCouleurF I("oiseau.png");
	traitement_image_gris(I, Fonction(f1));
}

float f4(float x)
{
	return pow(x,1.6); // correction gamma avec alpha=1.6
}

void exemple4()
{
	ImageGrisF I("arbre-brume.png");
	traitement_image_gris(I, Fonction(f4));
}

float f2(float x)
{
	return 1.5*x; // éclaircissement x 1.5
}

float f3(float x)
{
	return 0.9*x; // assombrissement x0.9
}
```

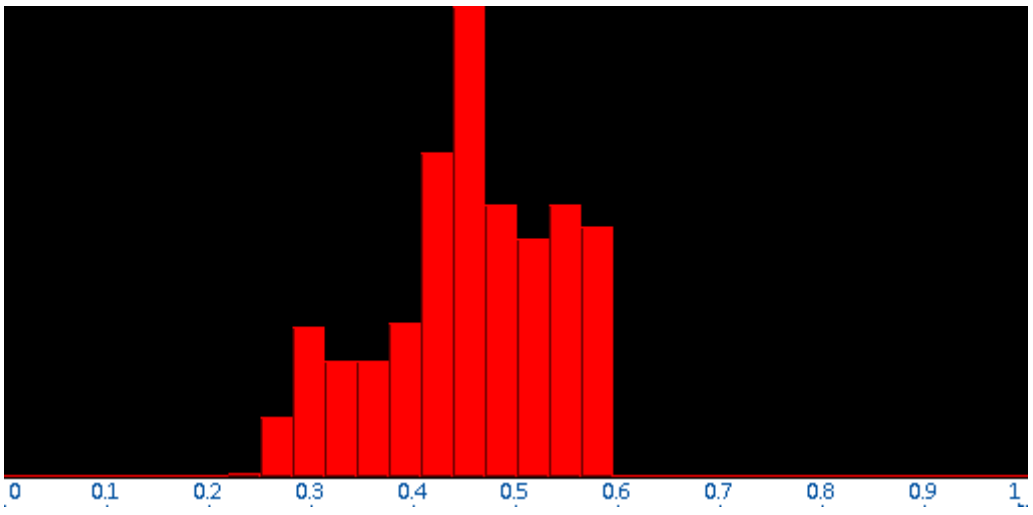
```
float f5(float x)
{
	if (x <=0.15) return 0.0;
	else if (x <= 0.3) return 0.15;
	return x;
}

void exemple2()
{
	ImageCouleurF I("plante.png");
	traitement_image_gris(I, Fonction(f2));
}

void exemple5()
{
	ImageCouleurF I("belledonne.pgm");
	traitement_image_gris(I, Fonction(f5));
}

void exemple3()
{
	ImageCouleurF I("amphi-weil.png");
	traitement_image_gris(I, Fonction(f3));
}
```

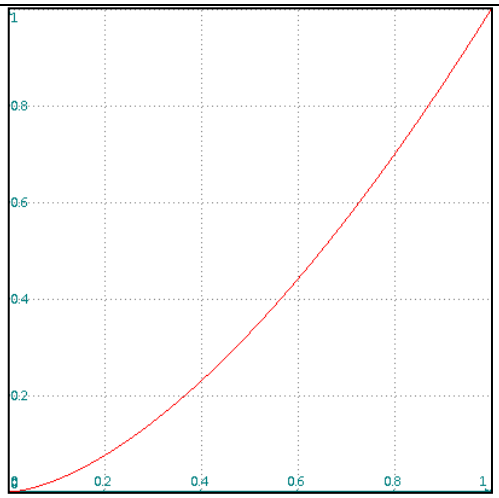




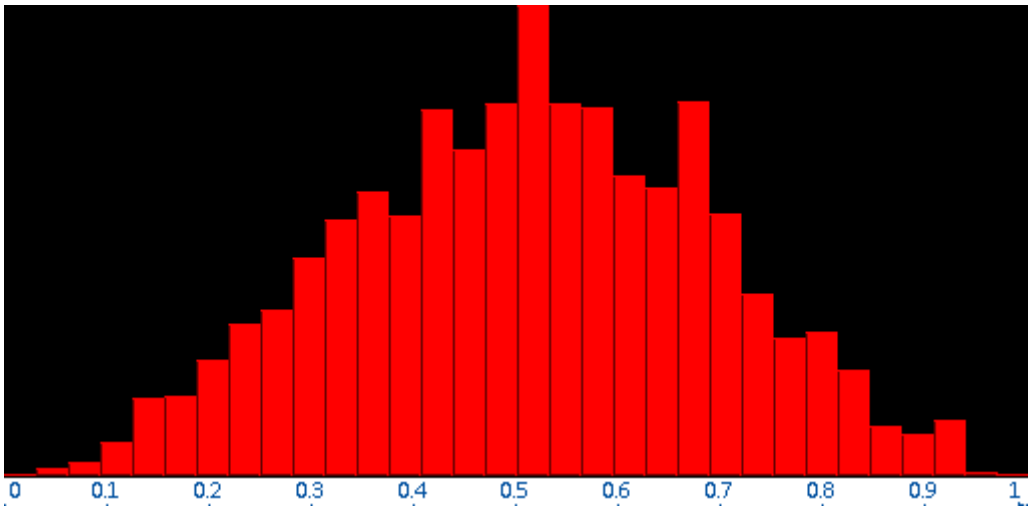
FONCTIONS/HISTO DE OISEAU.PNG

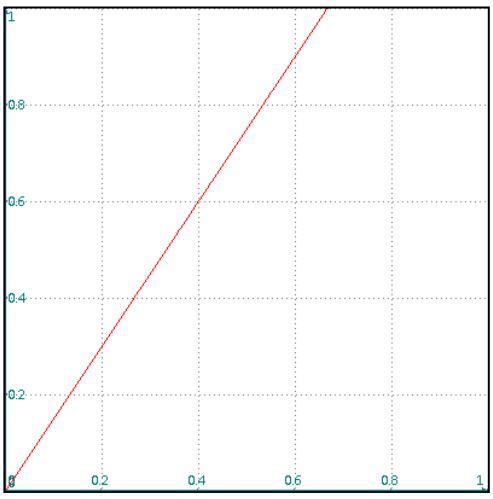
correction gamma avec alpha=0.5



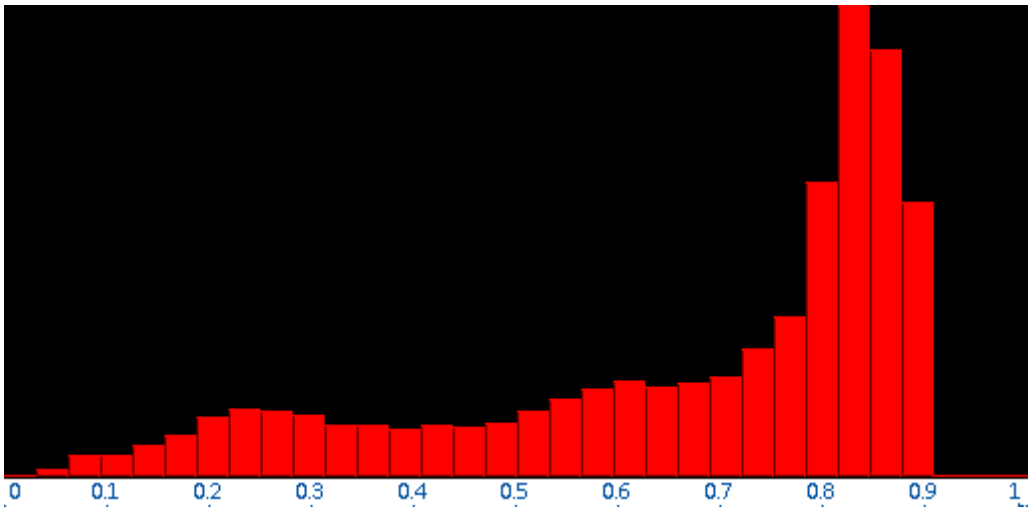


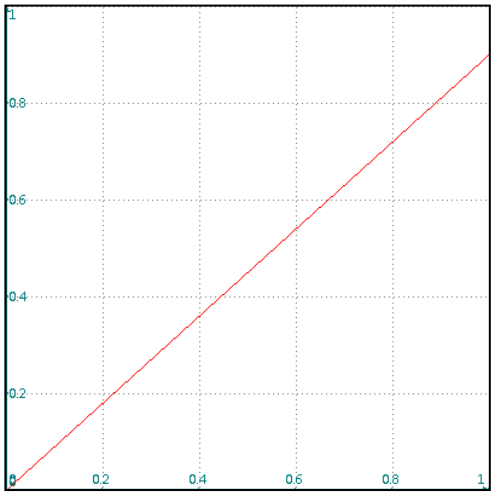
FONCTIONS/HISTO DE ARBRE-BRUME.PNG
correction gamma avec alpha=1.6

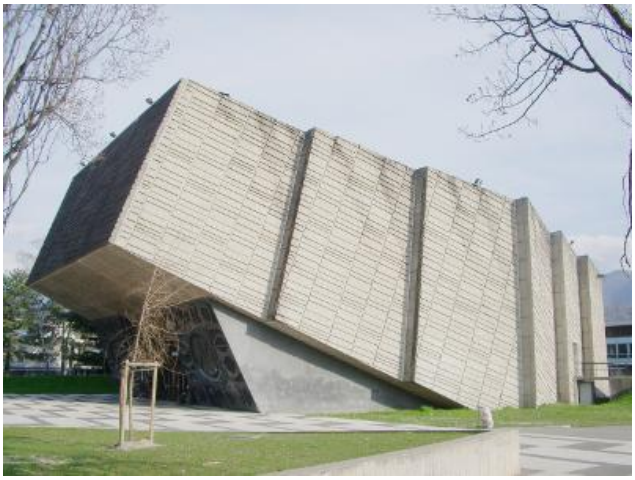




FONCTIONS/HISTO DE PLANTE.PNG
éclaircissement x 1.5



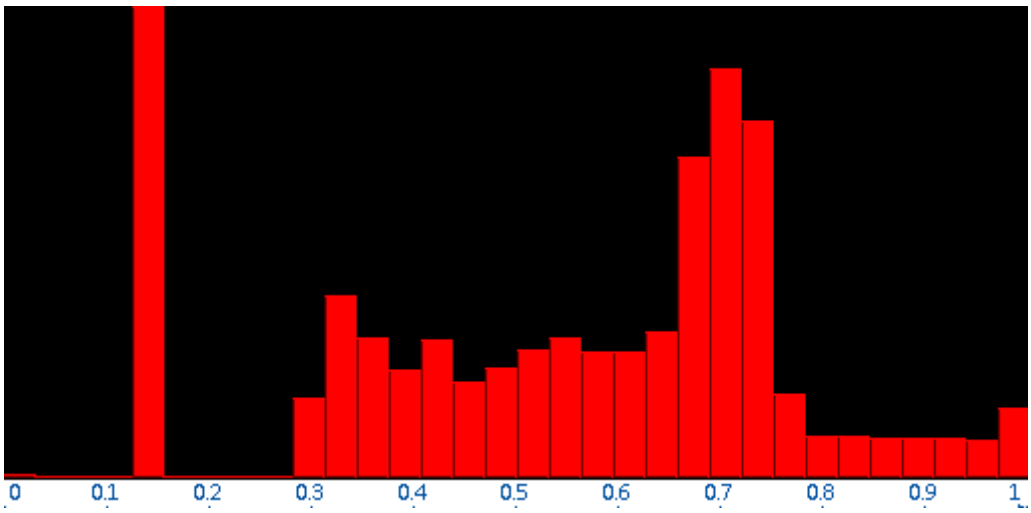


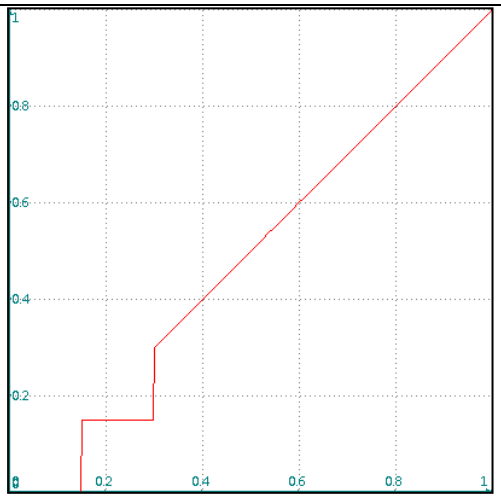


FONCTIONS/HISTO DE AMPHI.PNG
assombrissement x0.9

EXERCICE 3

Le code est dans l'exercice 2.





FONCTIONS/HISTO DE BELLEDONNE.PNG

EXERCICE 4

```
ImageCouleurF agrandissement_bilineaire(ImageCouleurF& I1, float a)

{

	UINT L1 = I1.largeur();

	UINT H1 = I1.hauteur();

	UINT L2 = (UINT)round(L1*a);

	UINT H2 = (UINT)round(H1*a);

	ImageCouleurF I2(L2,H2);


	for (UINT x2=0; x2<L2; x2++)

	for (UINT y2=0; y2<H2; y2++)

	{


		UINT xr = (UINT)(x2*(L1-1.0)/(L2-1.0));
```

```
            UINT yr = (UINT)(y2*(H1-1.0)/(H2-1.0));

            // Partie entière inférieure (avec floor() ):

            UINT x1 = (UINT)floor(xr);

            UINT y1 = (UINT)floor(yr);

            //partie décimale :

            UINT dx = (UINT)(xr - x1);

            UINT dy = (UINT)(yr - y1);


            I2(x2,y2) = (1-dx)*(1-dy)*I1(xr,yr)+ dx*(1-dy)*I1(x1+1,y1) +
(1-dx)*dy*I1(x1,y1+1) + dx*dy*I1(x1+1,y1+1);

      return I2;

}
```

On obtient les images suivantes, avec, à gauche, celles qui ont subies un agrandissement par interpolation au plus proche. A droite, celles dont l'agrandissement est bilinéaire (par soucis de lisibilité, seule une partie de l'image est affichée lors des agrandissements 2 et 3.) En théorie, un agrandissement bilinéaire donne un rendu plus « propre » que celui par interpolation au plus proche, qui peut donner un rendu plus pixélisé. Ceci-dit (sans doute car l'agrandissement n'est pas assez grand), il y a peu, voir pas de différence entre les deux méthodes. Il n'y a pas non plus de réelle différence en matière de temps d'exécution…

Agrandissement 1,5

Agrandissement 1,5

Agrandissement 2

Agrandissement 2

Agrandissement 3

Agrandissement 3

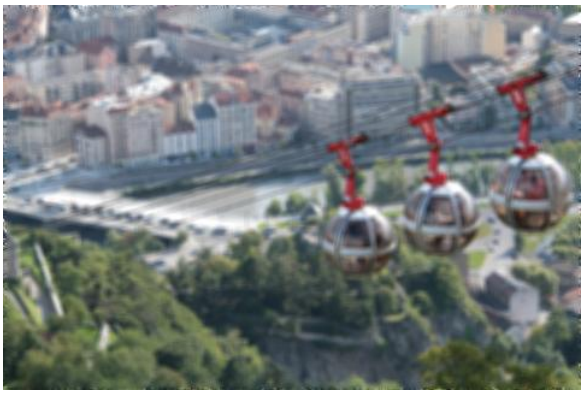
Un dernier exemple est ajouté uniquement pour l'agrandissement 3 :

EXERCICE 5

```
ImageCouleurF convolution(ImageCouleurF& I, FiltreLineaire K)

{

	UINT L = I.largeur();

	UINT H = I.hauteur();

	ImageCouleurF R(L,H);


	int n = K.taille();


	for (int i=0; i<L; i++)

	for (int j=0; j<H; j++)

	{

		R(i,j)=RGBF(0,0,0);

		if (i < L-n && j < H-n && j > n && i > n){

			for (int k=-n; k<=n; k++)

			{

				for (int l=-n; l<=n; l++)

				{

					R(i,j) = R(i,j) + K(k,l)*I(i-k, j-l);

				}

			}

		}else{

			R(i,j)=I(i,j);

		}

	}

	return R;

}
```

Ce code ne gère pas les contours : on repose les pixels tels qu'ils sont, s'ils appartiennent au contour. Nous avons tenté de modifier la matrice de convolution selon le pixel, mais nous nous sommes rendu compte que cela dénaturait totalement l'effet voulu. La procédure qui nous a paru la plus correct était d'agrandir assez l'image pour lui appliquer la matrice de convolution sur la bonne zone, mais nous n'avions pas accès non plus à la fonction crée dans l'exo 4. La méthode la plus naïve aurait été de simplement dupliquer les pixels de bord sur le contour, mais cette méthode est peu esthétique, pour peu que le contour soit épais…

Image d'origine



Application de la matrice de convolution

EXERCICE 6

```
ImageCouleurF filtre_median(ImageCouleurF& I, int n)

{

	UINT L = I.largeur();

	UINT H = I.hauteur();

	RGBF V[(2*n+1)*(2*n+1)];

	ImageCouleurF R(L-2*n,H-2*n);


	for (int i = n; i< L-n;i++){

		for (int j = n; j< H-n;j++){

			int x=0;

			R(i-n,j-n)=RGBF(0,0,0);


			for (int k=-n; k<=n; k++)

			{

				for (int l=-n; l<=n; l++)

				{

					V[x]=I(i+k,j+l);

					x++;

				}

			}

			tri(V,(2*n+1));

			R(i-n,j-n)=V[n+1];

		}

	}

	return R;
```

On remarquera sur les images ci-dessous que le filtre median est efficace sur un bruitage minime ou discret, mais qu'il devient complètement inutile sur une image trop bruitée. On perd cependant en qualité d'image, puisque le filtre va « flouter » (il ne s'agit pas non plus d'un floutage classique, qui étendrait les pixels morts.)

Image d'origine 1

Application du filtre median

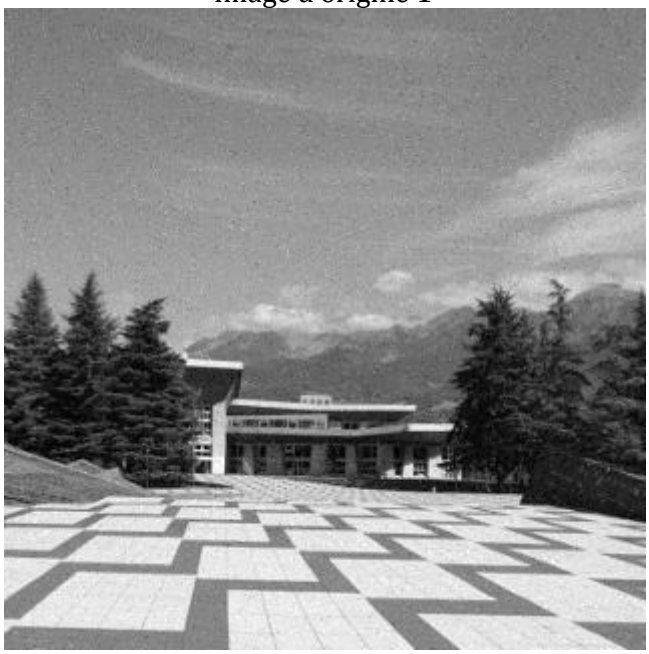


Image d'origine 2

Application du filtre median

Image d'origine 3

Application du filtre median