Tableaux 2D par l'exemple

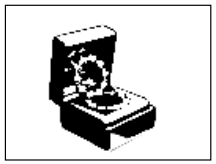
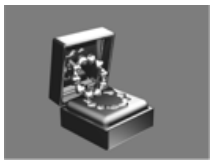
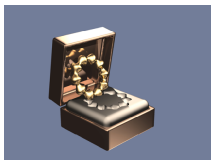
images numériques

Image = représentation 2D

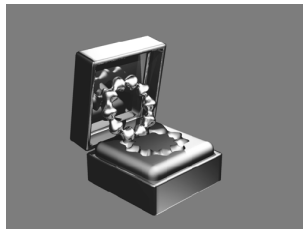
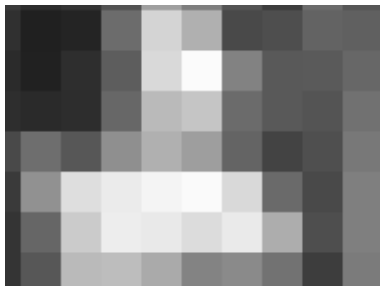
- monde réel, synthèse, dessin, visualisation de données, reconstruction, ...
- Unité ? Niveaux (de gris)
- Fichier, mémoire, organisation, affichage, création, amélioration, compression, ...

Niveaux de gris

- Noir et blanc, niveau de gris, couleur, ...

Pixels

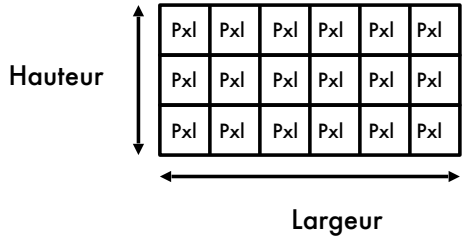



Le type pixel

- Images binaires. 1 bit/pixel. En pratique 1 octet 0 ou 255
- Images de niveaux de gris. En général un octet, valeur entre 0 et 255.
- `typedef unsigned char Pixel;`
- Images couleurs. index ou triplet d'octets (RVB) ou de plus en plus fréquemment de mots (2 octets : HDR).
- `typedef unsigned char Pixel[3];`
-
```
typedef struct {
     unsigned char R, V, B;
} Pixel ;
```

Image = tableau 2D

- *Matrice* (cf TP MAN) de pixels Pxl_{ij}



- ou en couleur, 1 tableau par plan (rare)

Images numériques

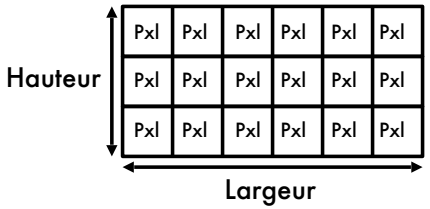
Quelle structure de données ?

Choix de la structure

- Affichage (matériel)
- type de traitement (local, voisinage,)
- stockage (mémoire, fichiers)

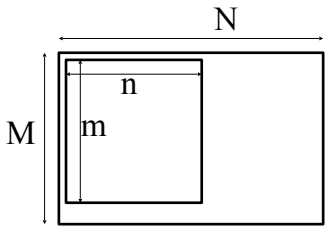
Matrices statiques

Taille d'image fixe : Hauteur et Largeur constantes




```
typedef Pixel Image[Hauteur][Largeur];
Image bitmap;
```

Matrices statiques



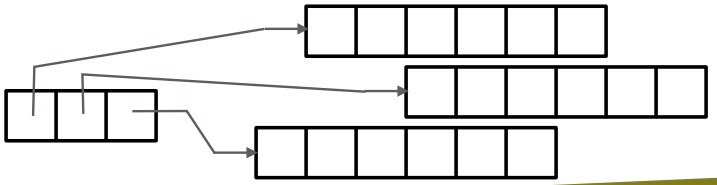

```
/* M et N constantes */
typedef Pixel Image[M][N];
Image bitmap;
int m, n;
/* dimensions réelles <= dimensions max */
```

Tableaux 2D dynamiques

M lignes de N colonnes de pixels :

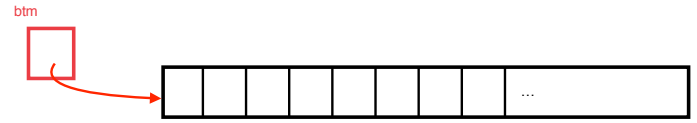
- « Vrai » tableau 2D `btm[i][j]`

```
Pixel **btm = NULL;
btm=(Pixel **)calloc(M,sizeof(Pixel*));
for (m=0;m<M;m++)
  btm[m]=(Pixel*)calloc(N,sizeof(Pixel);
```

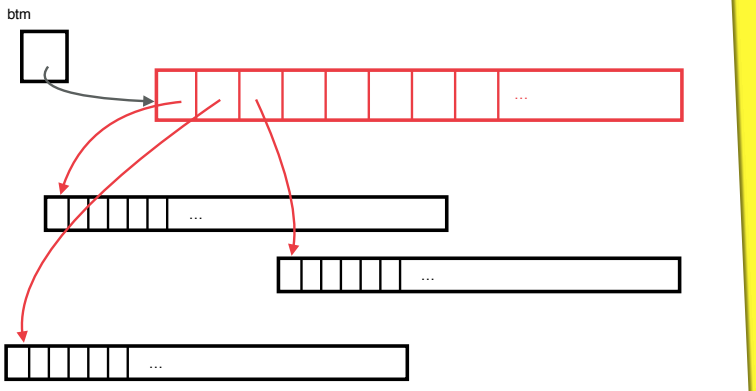


```
Pixel **btm = NULL;
btm=(Pixel **)calloc(M,sizeof(Pixel*));
for (m=0;m<M;m++)
  btm[m]=(Pixel*)calloc(N,sizeof(Pixel);
```

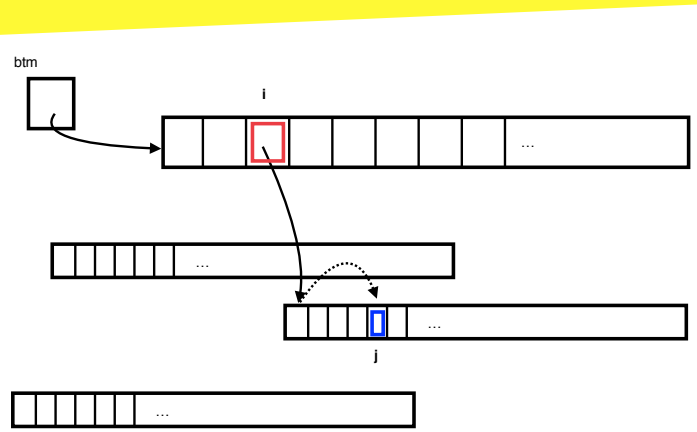
```
Pixel **btm = NULL;
btm=(Pixel **)calloc(M,sizeof(Pixel*));
for (m=0;m<M;m++)
  btm[m]=(Pixel*)calloc(N,sizeof(Pixel);
```




```
Pixel **btm = NULL;
btm=(Pixel **)calloc(M,sizeof(Pixel*));
for (m=0;m<M;m++)
  btm[m]=(Pixel*)calloc(N,sizeof(Pixel);
```




```
btm
btm[i] // ligne d'indice i : @ du pixel i,0
btm[i][j] // pixel i,j
```



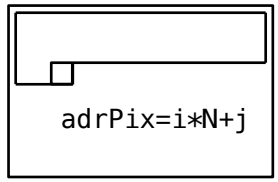
Tableaux 2D dynamiques

M lignes de N colonnes de pixels :

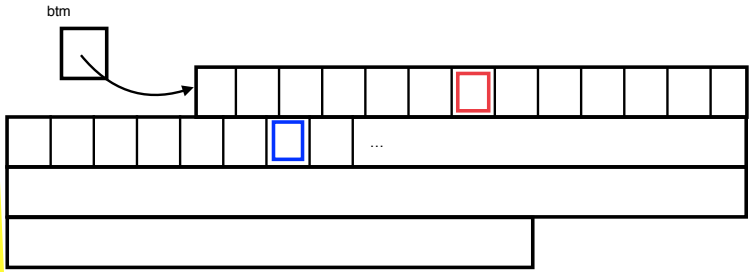
- tableau 1D dynamique de M*N pixels

```
Pixel *btm = NULL;
btm=(Pixel *)malloc(M*N*sizeof(Pixel));
```

MxN



```
btm
&btm[N*i] (ou btm+N*i)// @ pixel i,0 (ligne indice i)
btm[i*N + j] // pixel i,j
```



`Pixel **btm;` vs `Pixel *btm;`

- plusieurs blocs mémoire
- copie => itération
- expression du voisinage
- boucles imbriquées
- double déréférencement

- un bloc mémoire
- copie simple
- perte notion voisins
- parcours linéaire
- calcul d'adresse

`Pixel **btm;` vs `Pixel *btm;`

- plusieurs blocs mémoire
- copie => itération
- expression du voisinage
- boucles imbriquées
- double déréférencement

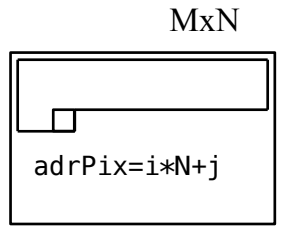
- un bloc mémoire
- copie simple
- perte notion voisins
- parcours linéaire
- calcul d'adresse

En pratique

M lignes de N colonnes de pixels :

- tableau 1D dynamique de M*N pixels

```
typedef unsigned char Pixel;
Pixel *btm = NULL;
btm=(Pixel *)
 malloc(M*N*sizeof(Pixel));
```



Utile ...

```
typedef unsigned char Pixel;
```

```
typedef struct {
     unsigned char R, V, B;
} Pixel;
```

```
Pixel getPixel(Pixel *img, int  i, int j, int L) {
   return img[(i*L +j)*sizeof(Pixel)];
}
```

Utile ...

```
typedef unsigned char Pixel;
```

```
typedef struct {
     unsigned char R, V, B;
} Pixel;
```

```
setPixel(Pixel *img,int  i,int j,int L,Pixel p) {
   img[(i*L +j)*sizeof(Pixel)] = p;
}
```

Mais ...

Exemple : inversion vidéo

```
int i,j;
Pixel v;
for(i=0; i<h; i+=1)
    for(j=0;j<l;j+=1) {
       v=getPixel(img,i,j,l);
       setPixel(res,i,j,l,255-v);
    }
```

```
int i,j,adr=0;
for(i=0; i<h; i+=1)
    for(j=0;j<l;j+=1) {
      res[adr]=255-img[adr];
      adr += 1;
    }
```

Exemple : inversion vidéo

```
int i,j;
Pixel v;
for(i=0; i<h; i+=1)
    for(j=0;j<l;j+=1) {
       v=getPixel(img,i,j,l);
       setPixel(res,i,j,l,255-v);
    }
```

```
int adr;
for(adr=0; adr<l*h; adr+=1)
      res[adr]=255-img[adr];
```

```
int adr;
for(adr=0; adr<l*h; adr+=1)
      img[adr]=255-img[adr];
```

Exemple : inversion vidéo

```
int i,j;
Pixel v;
for(i=0; i<h; i+=1)
    for(j=0;j<l;j+=1) {
       v=getPixel(img,i,j,l);
       setPixel(res,i,j,l,255-v);
    }
```

```
int adr; // unsigned char Pixel[3]
for(adr=0; adr<l*h*3; adr+=1)
      res[adr]=255-img[adr];
```

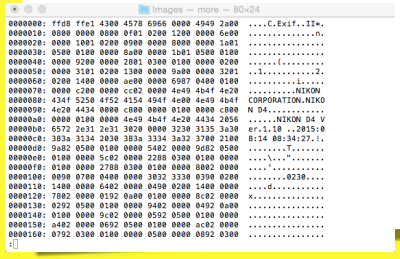
Exemple : inversion vidéo

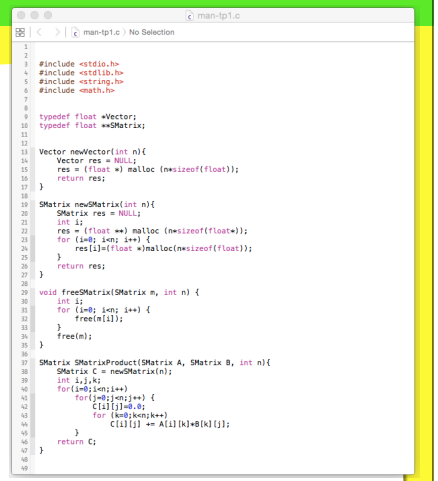
```
int i,j;
Pixel v;
for(i=0; i<h; i+=1)
    for(j=0;j<l;j+=1) {
       v=getPixel(img,i,j,l);
       setPixel(res,i,j,l,255-v);
    }
```

```
//struct {unsigned char R,V,B;} Pixel;
int adr;
for(adr=0; adr<l*h; adr+=1) {
      res[adr].R=255-img[adr].R;
      res[adr].V=255-img[adr].V;
      res[adr].B=255-img[adr].B;
}
```

Fichiers

- mémoire « permanente »
- texte, binaires
- séquentiels, indexés, …
- structurés …





Accès aux fichiers

- niveau 3 : accès « tamponné »
- Descripteur : `FILE *f;`
- Ouverture `f = fopen(filename, mode);`
- avec mode = `"r|w|a[b][+]"` — exemple `"rb"`
- fermeture `fclose(f);`

Accès aux fichiers «texte»

- entrées/sorties formatées :
  - lecture `fscanf(f, "<format>" <,adresse>*);`
  - écriture : `fprintf(f, "<format>" <,expr>*);`
- mode ligne :
  - lecture : `fgets(buffer, size, f);`
  - écriture : `fputs(buffer, f);`
- autres fonctions
  - `ftell, fseek`

Détection des erreurs

- lecture ligne : `fgets(buffer, size, f);`
- décodage ligne
  - `sscanf(buffer, "<format>" <,adresse>*);`
  - retourne le nombre de conversions réussies
- Exemple : permet de savoir si lecture d'un nombre réel

```
fgets(buffer, size, f); // ou stdin
nb = sscanf(buffer, "%f", &v);
```

Accès aux fichiers binaires

- écriture : `fwrite(buffer, n, size, f);`
- lecture : `fread(buffer, n, size, f);`
- autres fonctions
  - `ftell, fseek`
- + : rapide (pas de conversions)
- **- : portable** (représentation des nombres)

Fichiers mixtes …

- Combinent avantages des 2 types d'information
- Souvent entête au format texte
- Contient des informations sur le format des informations binaires : nombre, taille, boutisme, ...
- Courant en imagerie

Images PNM

Entête d'un fichier PGM :

```
P5
# L=nbc H=nbl
500 500
# commentaires optionnels
255
# commentaires optionnels
```

puis nblignes * nbcolonnes pixels

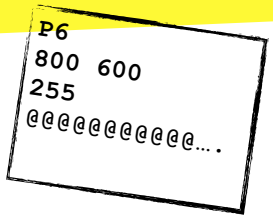
Images PNM

- Format reconnu par
  - gimp
  - atril
  - convert
  - …

Exemple : lire le « pgm »


```
char buffer[128];
int width, height;
unsigned char *bitmap=NULL;
fgets(buffer,128,f);
if (strcmp(buffer, "P5\n")) error("not binary pgm file\n");
fgets(buffer,128,f);
sscanf(buffer, "%d %d", &width, &height);
fgets(buffer,128,f);
bitmap=malloc(width*height*sizeof(unsigned char));
fread(bitmap, width*height, sizeof(unsigned char));
```

Exemple : lire le « ppm »

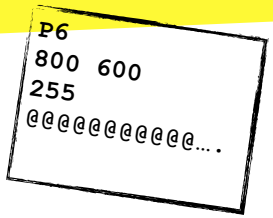



```
char buffer[128];
int width, height;
unsigned char *bitmap=NULL;
fgets(buffer,128,f);
if (strcmp(buffer, "P6\n")) error("not binary ppm file\n");
fgets(buffer,128,f);
sscanf(buffer, "%d %d", &width, &height);
fgets(buffer,128,f);
bitmap=malloc(3*width*height*sizeof(unsigned char));
fread(bitmap, 3*width*height, sizeof(unsigned char));
```

E/S : écrire le « pgm »


```
f = fopen(nomFichier, "wb");
fprintf(f, "P5\n%d %d\n255\n",largeur, hauteur);
fwrite(bitmap, width*height, sizeof(unsigned char), f);
fclose(f);
```

E/S : écrire le « ppm »




```
f = fopen(nomFichier, "wb");
fprintf(f, "P6\n%d %d\n255\n",largeur, hauteur);
fwrite(bitmap, width*height, 3*sizeof(unsigned char), f);
fclose(f);
```

TP « image »

- Fonctions de lecture et écriture fournies
- dans pnm.c
- #include<pnm.h>

```
fich.name=argv[1];
lirefich(&fich); xs=fich.xsize; ys=fich.ysize;
res = inversion_videoNB(fich.bitmap, xs, ys, &xs2, &ys2);
fich2.bitmap=res;
fich2.xsize=xs2;
fich2.ysize=ys2;
fich2.maxvalue=255;
fich2.name=argv[2];
fich2.type=5;
sauverfich(&fich2);
```

Exemple : inversion vidéo

```
unsigned char *inversionVideoNB(unsigned char *img,int xs,int ys,int *nc,int
*nl)
{
    unsigned char *res=NULL;
    int i;

    res = calloc(xs*ys,1); // tester
    for(i=0;i<xs*ys;i=i+1) res[i]=255-img[i];

    *nc=xs;
    *nl=ys;
    return res;
}
```

Modification ou création ?

- Attention aux risques d'effets de bord indésirables
- Solution : duplication
- Obligatoire si fonction sur voisinage
- Possibilité d'utiliser une zone tampon …
- Dépend des applications
- Cas des images « agrandies »

```
void negatif(Pixel *img, int l, int h) {
      int i;
      for(i=0; i<width*height; i++)
            res[i] = 255 - img[i];
}
Pixel * negatifNB(Pixel *img, int l, int h) {
      Pixel *res=malloc(h*l);
      int i;
      for(i=0; i<width*height; i++)
            res[i] = 255 - img[i];
      return res;
}
```

Programmation modulaire

- Structurer applications de taille plus importante
- facilite le travail à plusieurs
- répartir les fonctions dans différents fichiers, qui regroupent des fonctions qui traitent le même type de données, concernent une partie cohérente d'un problème, etc.

Programmation modulaire

```
// pnm.c
PnmFile readPnmFile(char *filename) {
  // . . .
}

void writePnmFile(char *filename, PnmFile image) {
  // . . .
}
```

Programmation modulaire

```
// effets.c
void negatifNB(Pixel *img, int l, int h) {
  // . . .
}
void plusClairNB(Pixel *img, int l, int h) {
  // . . .
}
// . . .
```

Programmation modulaire

```
// main.c
typedef . . .
extern PnmFile readPnmFile(char *filename);
// . . .
int main(int argc, char **argv) {
  PnmFile img;
  img1 = readPnmFile(argv[1]);
  negatifNB(img.bitmap, img.l, img.h);
  writePnmFile(argv[2], image) {
  return 0;
}
```

headers

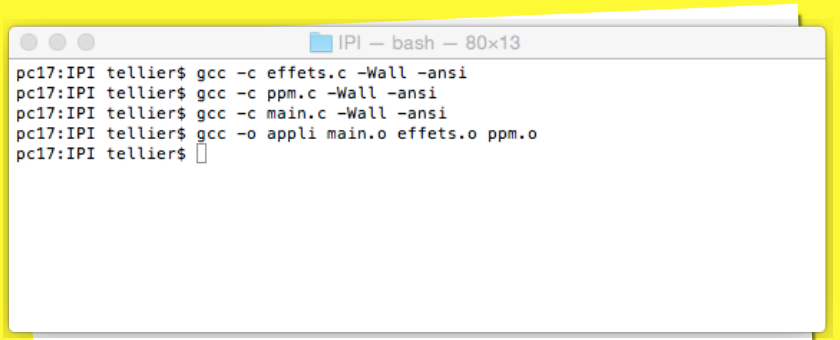
```
//pnm.h
typedef struct {
  // . . .
} PnmFile;
extern PnmFile readPnmFile(char *filename);
```

```
// effets.h
extern void negatifNB(Pixel *img, int l, int h);
extern void plusClairNB(Pixel *img, int l, int h);
// . . .
```

Programmation modulaire

```c
// main.c
#include "pnm.h"
#include "effets.h"
int main(int argc, char **argv) {
  PnmFile img;
  img1 = readPnmFile(argv[1]);
  negatifNB(img.bitmap, img.l, img.h);
  writePnmFile(argv[2], img) {
  return 0;
}
```

Compilation séparée + EDL



- gcc -o appli main.c effets.c pnm.c -Wall -Wextra

Makefile (basique)

```
effets.o :  effets.c effets.h
            gcc -c effets.c -Wall -ansi
pnm.o :     pnm.c pnm.h
            gcc -c pnm.c -Wall -ansi
main.o :    main.c pnm.h effets.h
            gcc -c main.c -Wall -ansi
# ou *.o
appli:      main.o effets.o pnm.o
            gcc -o appli main.o effets.o pnm.o
clean:
            rm *.o
```

Makefile (mieux)

```
PROG=appli
CSRC=main.c pnm.c effets.c
COBJ=$(CSRC:.c=.o)
CFLAGS=-Wall -ansi
.c.o:
   gcc -c $*.c $(CFLAGS)
$(PROG):$(COBJ)
      gcc -o $(PROG) $(COBJ) $(LIBS)
clean:
      rm *.o *.bak $(PROG)
depend:
   makedepend $(CSRC)
# DO NOT DELETE
effets.o: effets.h
ppm.c:    ppm.h
```

Effets photométriques

- Transformation Couleur vers Niveaux de Gris
  - Calcul de la luminance
  - 0.707*R+0.202*V+0.071*B

```
Pixel* imgNB=malloc(H*L);

for(i=0;i<H*L;i++) {
  imgNB[i]= 0.707*img[3*i]
            +0.202*img[3*i+1]
            +0.071*img[3*i+2];
}
```

Eclaircir

```
int i, v;
for (i=0;i<sx*sy;i++) {
    v=img[i]*1.05;
    if (v > 255) v=255;
    res[i]=v;
}
```

Effets géométriques

- Balayer l'image résultat : pas de pixels non initialisés
- Eventuellement pratiquer des interpolations

Fonte

Principe

- Sélectionner des pixels au hasard
- Si ils sont plus sombres que ceux sur la ligne inférieure, le faire « glisser » vers le bas

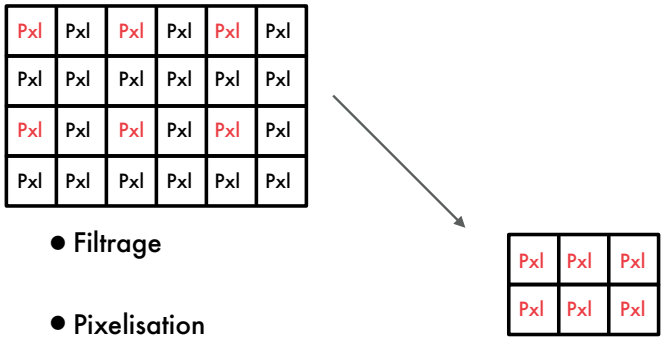
Fonte (NB)

```
/* initialiser res : copie de image source */
for(i=0;i<N;i++) {
  l=random()*sy/(float)RAND_MAX;
  c=random()*sx/(float)RAND_MAX;
  adr = (l*sx+c);
  if (l<sy-1)
    if (res[adr]<res[adr+sx]))
        res[adr+sx]=res[adr];
}
```

Fonte (RVB)

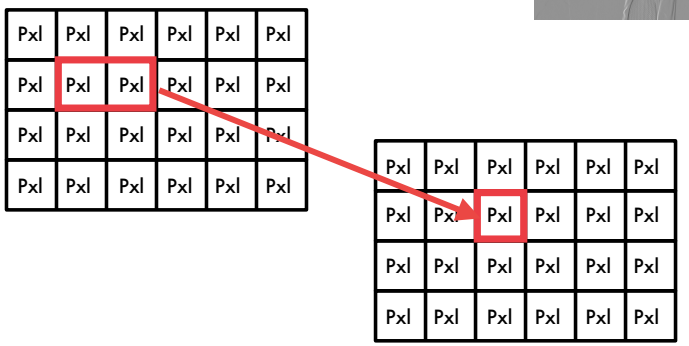
```
/* initialiser res : copie de image source */
for(i=0;i<N;i++) {
  l=random()*sy/(float)RAND_MAX;
  c=random()*sx/(float)RAND_MAX;
  adr = (l*sx+c)*np;
  if (l<sy-1)
    if (gris(res,adr)<gris(res,adr+sx*np))
      for(k=0;k<np;k++) {
        res[adr+sx*np+k]=res[adr+k];
      }
}
```

Sous-échantillonnage



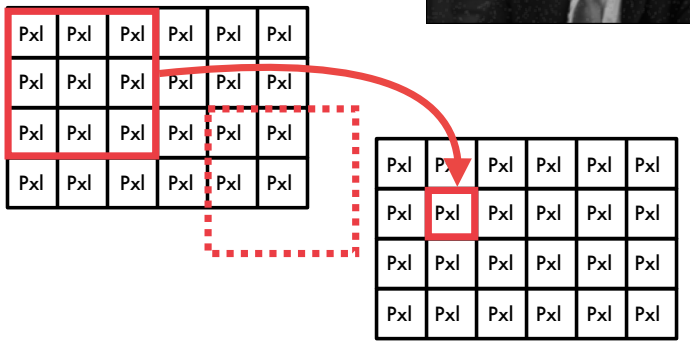
- Filtrage
- Pixelisation

Relief

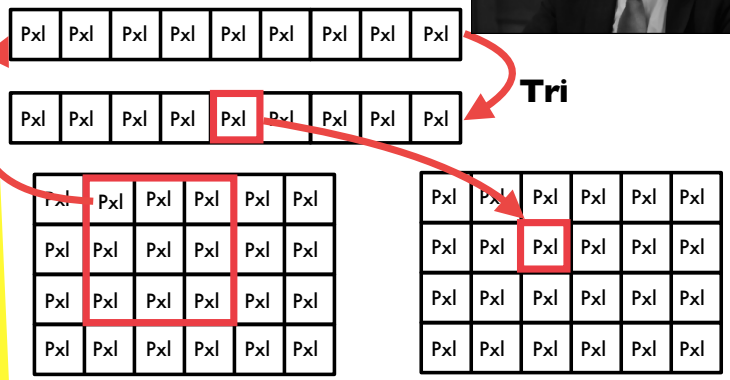


Filtrage

Floutage



Filtrage médian



Librairies

```
LIB=libeffets.a
CSRC=effets.c effetsgeom.c effetsphotom.c
COBJ=$(CSRC:.c=.o)
CFLAGS=-Wall -ansi
.c.o:
   gcc -c $*.c $(CFLAGS)
$(LIB):  $(COBJ)
      ar rsv $(LIBS) $(COBJ)
      mv $(LIB) $(LIBDIR)
clean:
      rm *.o *.bak $(PROG)
depend:
   makedepend $(CSRC)
# DO NOT DELETE
```

$ gcc myprog.c -o myprog -leffets

Le TP

Images pnm (P5)

- `plusClairNB`
- `deriv1xNB`
- `effetFonteNB`
- `quartImageNB`
- `filtrerMedianImageNB`
- `filtrerImageNB`

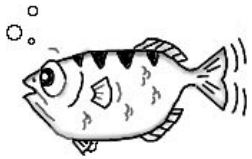
Doxygen

- Dans les commentaires
  - `\file [<name>]`
  - `\brief {brief description}`
  - `\author { list of authors }`
  - `...`
  - `\struct`
  - `...`
  - `\fn`
  - `\param`
  - `\return`

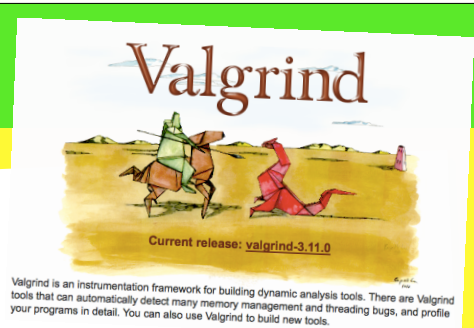
```
/*!
 \file effets-photom.c
 \author
 \date
 \brief module des fonctions de correction de la couleur et
de la luminosité des images
 */
```

```
/*!
 \fn unsigned char *inversion_video(unsigned char *img, int xs, int ys,
int *nc, int *nl)
 \brief transforme l'image en "négatif" (complément à 255)
 \param img : adresse du bitmap
 \param xs : largeur de l'image
 \param ys : hauteur de l'image
 \param nc : pointeur sur le nombre de colonnes (modifiable) de l'image
résultat
 \param nl : pointeur sur le nombre de lignes (modifiable) de l'image
résultat
 \return l'image obtenue
 */
```

GDB the GNU Project Debugger



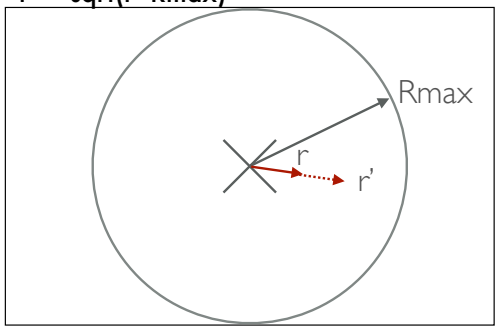
```
$ gcc myprog.c -o myprog -g
$ gdb myprog arg1 arg2
(gdb) run
(gdb) break
(gdb) print
(gdb) next
(gdb) step
(gdb) watch
(gdb) where
(gdb) list
(gdb) up
(gdb) down
(gdb) delete
(gdb) cont
(gdb) ...
```




```
$ gcc myprog.c -o myprog -g -O0

$ myprog arg1 arg2

devient

$ valgrind –leak-check=yes myprog arg1 arg
```

Fish Eye

- r' = sqrt(r*Rmax)



Fish Eye

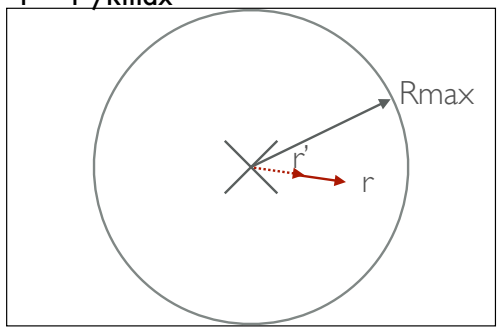
```
R=sqrt((sy/2)*(sy/2));
for(i=0;i<sy;i++)
  for(j=0;j<sx;j++){
    //r = sqrt((i-sy/2)*(i-sy/2)+(j-sx/2)*(j-sx/2));
    r = (i-sy/2)*(i-sy/2)+(j-sx/2)*(j-sx/2);
    a = atan2((double)(i-sy/2),(double)(j-sx/2));
    //rr = r*r/R;
    rr = r/R;
    x = (double)sx*0.5 + rr*cos(a);
    y = (double)sy*0.5 + rr*sin(a);
    if (y<0) y=0; else if (y>sy-1) y=sy-1;
    if (x<0) x=0; else if (x>sx-1) x=sx-1;
    if (r<R) {
      setPixel(res,i,j,sx,getPixel(y,x,sx));
      //res[i*sx+j]=img[y*sx+x];
 }
```

Fish Eye

```
R=sqrt((sy/2)*(sy/2));
for(i=0;i<sy;i++)
  for(j=0;j<sx;j++){
    //r = sqrt((i-sy/2)*(i-sy/2)+(j-sx/2)*(j-sx/2));
    r = (i-sy/2)*(i-sy/2)+(j-sx/2)*(j-sx/2);
    a = atan2((double)(i-sy/2),(double)(j-sx/2));
    //rr = r*r/R;
    rr = r/R;
    x = (double)sx*0.5 + rr*cos(a);
    y = (double)sy*0.5 + rr*sin(a);
    if (y<0) y=0; else if (y>sy-1) y=sy-1;
    if (x<0) x=0; else if (x>sx-1) x=sx-1;
    if (r<R)
      for(k=0;k<np;k++)
        res[i*sx*np+j*np+k]=img[y*sx*np+x*np+k];
  }
```

Caricature

- $r' = r^2/Rmax$



Caricature

```
R=sqrt((sy/2)*(sy/2));
for(i=0;i<sy;i++)
  for(j=0;j<sx;j++) {
    r = sqrt((i-sy/2)*(i-sy/2) +(j-sx/2)*(j-sx/2));
    a = atan2((double)(i-sy/2),(double)(j-sx/2));
    rr = sqrt(r*R);
    x = sx*0.5 + rr*cos(a);
    y = sy*0.5 + rr*sin(a);
    if (y<0) y=0; else if (y>sy-1) y=sy-1;
    if (x<0) x=0; else if (x>sx-1) x=sx-1;
    if (r<R)
      for(k=0;k<np;k++)
        res[i*sx*np+j*np+k]=img[y*sx*np+x*np+k];
  }
```