

Tableaux 2D par l'exemple

# images numériques

Niveaux de gris

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

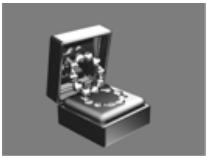
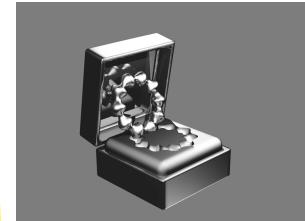


Image = représentation 2D

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

Pixels

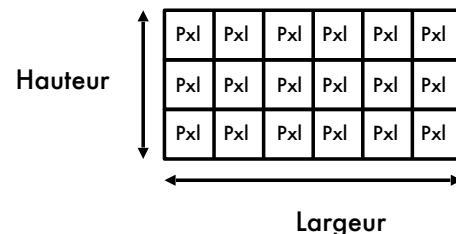


## Le type pixel

- 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. Indice 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 de pixels  $P_{ij}$



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

## Images numériques

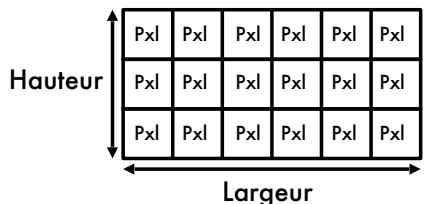
**Quelle  
structure de  
données ?**

## Choix de la structure

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

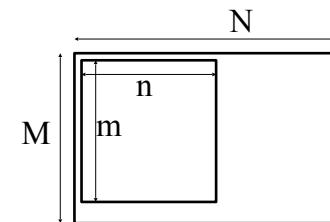
# Matrices statiques

Taille d'image fixe : Hauteur et Largeur constantes



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

# Matrices statiques



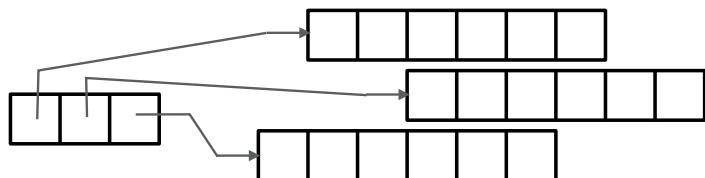
```
/* M et N constantes */
typedef Pixel Image[M][N];
Image bitmap;
int m, n;
/* dimensions réelles <= dimensions max */
```

# 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));
```



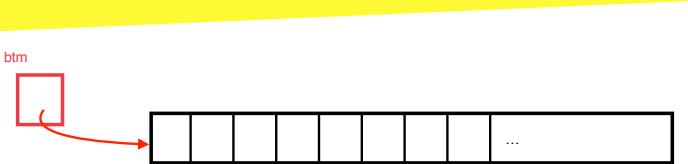
```
Pixel **btm = NULL;
btm=(Pixel **)calloc(M,sizeof(Pixel *));
for (m=0;m<M;m++)
    btm[m]=(Pixel *)calloc(N,sizeof(Pixel));
```

btm  
NULL

```

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

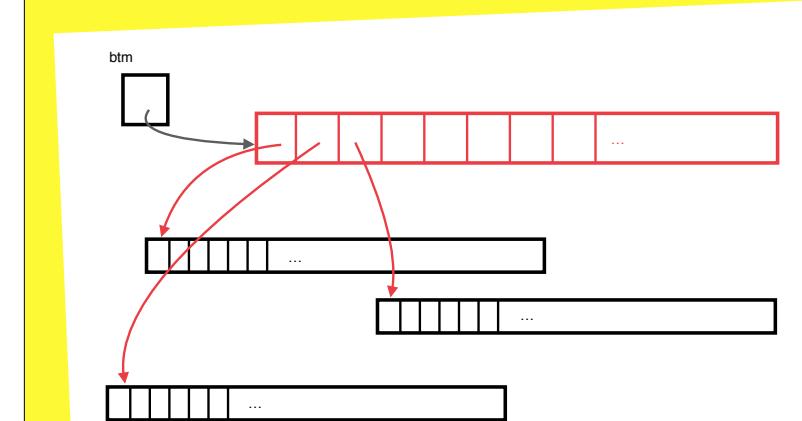
```



```

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

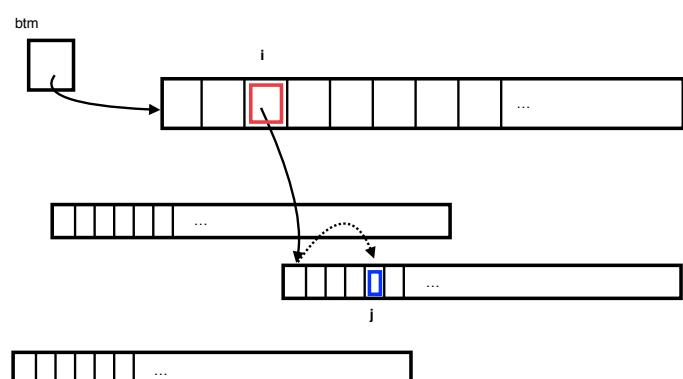
```



```

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 :

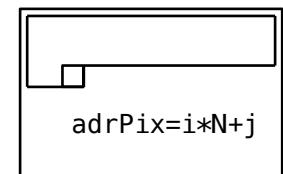
- tableau 1D dynamique de  $M \times N$  pixels

```

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

```

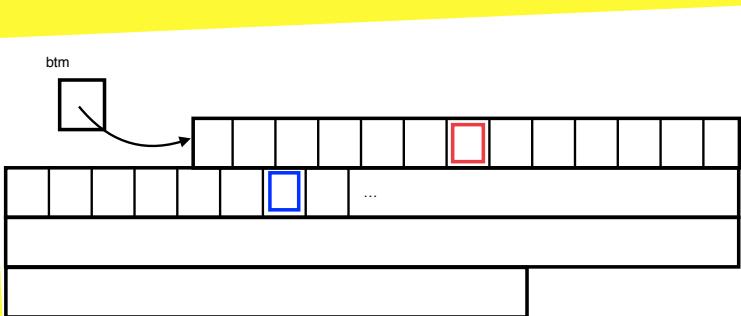
$M \times N$



```

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      ● un bloc mémoire
- copie => itération      ● copie simple
- expression du voisinage      ● perte notion voisins
- boucles imbriquées      ● parcours linéaire
- double déréférencement      ● calcul d'adresse

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

- plusieurs blocs mémoire      ● un bloc mémoire
- copie => itération      ● copie simple
- expression du voisinage      ● perte notion voisins
- boucles imbriquées      ● parcours linéaire
- double déréférencement      ● calcul d'adresse

## En pratique

M lignes de N colonnes de pixels :

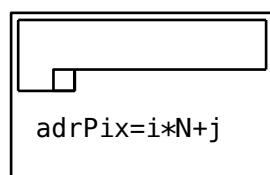
- tableau 1D dynamique de  $M \times N$  pixels

```

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

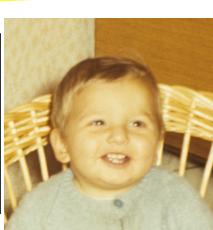
```

$M \times N$



# Exemple : inversion vidéo

```
for y in range(0,height) :  
    for x in range(0,width) :  
        (r,v,b)=img.getpixel((x,y))  
        img.putpixel((x,y),  
                    (255-r,255-v,255-b))
```

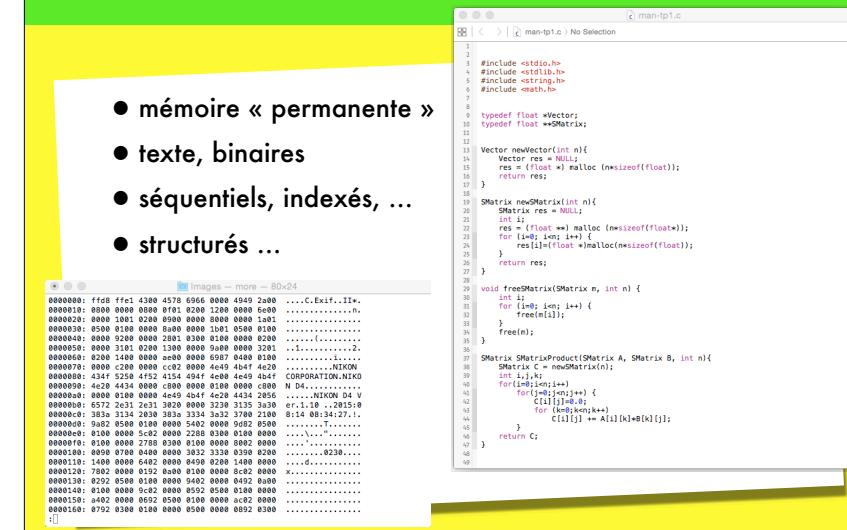


```
int i;  
for(i=0; i<3*width*height; i++) {  
    bitmap[i] = 255 - bitmap[i];  
}
```

# 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);

# Fichiers



```
#include <cmath>  
#include <string.h>  
#include <math.h>  
  
typedef float *vector;  
typedef float **matrix;  
  
vector newVector(int n){  
    Vector res = NULL;  
    res = (float *) malloc (n*sizeof(float));  
    return res;  
}  
  
matrix newMatrix(int n){  
    Matrix res = NULL;  
    int i;  
    res = (float **) malloc (n*sizeof(float*));  
    for (i=0; i<n; i++) {  
        res[i] = (float *) malloc (n*sizeof(float));  
    }  
    return res;  
}  
  
void freeVector(vector m){  
    int i;  
    for (i=0; i<m; i++) {  
        free(m[i]);  
    }  
    free(m);  
}  
  
Matrix MatrixProduct(Matrix A, Matrix B, int n){  
    Matrix C = newMatrix(n);  
    int i,j,k;  
    for (i=0; i<n; i++) {  
        for (j=0; j<n; j++) {  
            C[i][j]=0.0;  
            for (k=0; k<n; k++) {  
                C[i][j] += A[i][k]*B[k][j];  
            }  
        }  
    }  
    return C;  
}  
  
int main(){  
    FILE *f1,*f2,*f3;  
    f1=fopen("A.txt","r");  
    f2=fopen("B.txt","r");  
    f3=fopen("C.txt","w");  
    int i,j,k;  
    float a,b,c;  
    for (i=0; i<3; i++) {  
        for (j=0; j<3; j++) {  
            fscanf(f1,"%f",&a);  
            fscanf(f2,"%f",&b);  
            c=a*b;  
            fprintf(f3,"%f ",c);  
        }  
        fprintf(f3,"\n");  
    }  
    fclose(f1);  
    fclose(f2);  
    fclose(f3);  
    return 0;  
}
```

# Accès aux fichiers

- Ouverture f = open(filename)
- avec mode = "r|w|a|x[b|t]" — exemple "rb"
- avec mode f = open(filename, 'w')
- fermeture f.close()

# 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`

# Accès aux fichiers «texte»

- lecture intégrale `texte = f.read()`
- écriture : `f.write('texte')`
- mode ligne : `for line in f # à splitter`
- une ligne : `line = f.readline()`
- toutes les lignes : `lines = f.readlines()`

# 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)
- - : moins portable (représentation des nombres)

- `f.write(chr(img[i]).encode('ascii'))`
- `img[i]=f.read(1)`
- **pickle**
  - `data=pickle.load(f)`
  - `pickle.dump(data, f, ...)`

## 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));
```

P5  
800 600  
255  
@ @ @ @ @ @ @ @ @ @ ...

## Exemple : lire le « pgm »

```
buffer = f.readline()
if buffer != "P5\n" :
    error("not binary pgm file\n");
buffer = f.readline()
(width,height)=buffer.split(" ")
buffer = f.readline()
bitmap=f.read(width*height)
```

P5  
800 600  
255  
@ @ @ @ @ @ @ @ @ @ ...

## 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));
```

P6  
800 600  
255  
@ @ @ @ @ @ @ @ @ @ ...

## 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);
```

P5  
800 600  
255  
@ @ @ @ @ @ @ @ @ @ ...

## E/S : écrire le « pgm »

```
P5  
800 600  
255  
@ @ @ @ @ @ @ @ @ @ ...
```

```
f = open(nomFichier, "wb")  
f.write("P5\n"+str(largeur)+" "+str(hauteur)+"\n255\n")  
f.write(bitmap) #bytarray  
f.close()
```

## E/S : écrire le « ppm »

```
P6  
800 600  
255  
@ @ @ @ @ @ @ @ @ @ ...
```

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

## Exemple : inversion vidéo

```
int i;  
for(i=0; i<width*height; i++) {  
    res[i] = 255 - bitmap[i];  
}
```



```
int i;  
for(i=0; i<3*width*height; i++){  
    res[i] = 255 - bitmap[i];  
}
```

# 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;  
}
```

```
def inversion_videoNB(img, xs, ys):  
    res = Image.new("L",img.size,"black")  
    for y in range (0,ys):  
        for x in range (0,xs):  
            col=img.getpixel ((x,y))  
            res.putpixel((x,y),255-col)  
    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

```
// 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

```
IPI — bash — 80x13
pc17:IPI tellier$ gcc -c effets.c -Wall -ansi
pc17:IPI tellier$ gcc -c ppm.c -Wall -ansi
pc17:IPI tellier$ gcc -c main.c -Wall -ansi
pc17:IPI tellier$ gcc -o appli main.o effets.o ppm.o
pc17:IPI tellier$
```

- `gcc -o appli main.c effets.c ppm.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 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];
}
```

## Modules Python

```
from effets_photom import *
from effets_geom import *
from filtrages import *
```

## 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; //%sy
    c=random()*sx/(float)RAND_MAX; //%sx
    adr = (l*sx+c)*3;
    if (l<sy-1)
        if (gris(res,adr)<gris(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)*3;
    if (l<sy-1)
        if (gris(res,adr)<gris(res,adr+sx*3))
            for(k=0;k<3;k++)
                res[adr+sx*np+k]=res[adr+k];
}
```

# Sous-échantillonnage

Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

- Filtrage
- Pixelisation

Pxl	Pxl	Pxl
Pxl	Pxl	Pxl

# Relief



Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

# Floutage



Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	<b>Pxl</b>	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

# Filtrage



# Floutage



The diagram illustrates the memory layout of a 2D array of pixels. It consists of two rows of memory slots. The top row contains eight slots, each labeled "Pxl". A red arrow points from the first slot in the top row to the first slot in the bottom row, indicating that the first row is a pointer to the second row. The bottom row also contains eight slots, each labeled "Pxl", representing the actual pixel data.

Tr

Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl
Pxl	Pxl	Pxl	Pxl	Pxl	Pxl

# 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 ppm (P5)

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



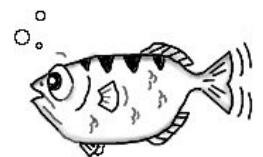
# Doxygen

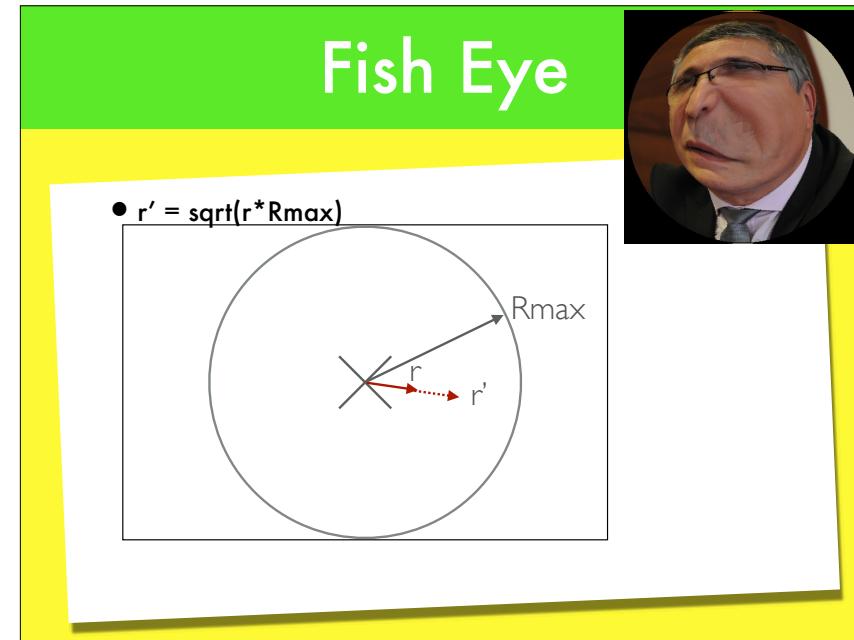
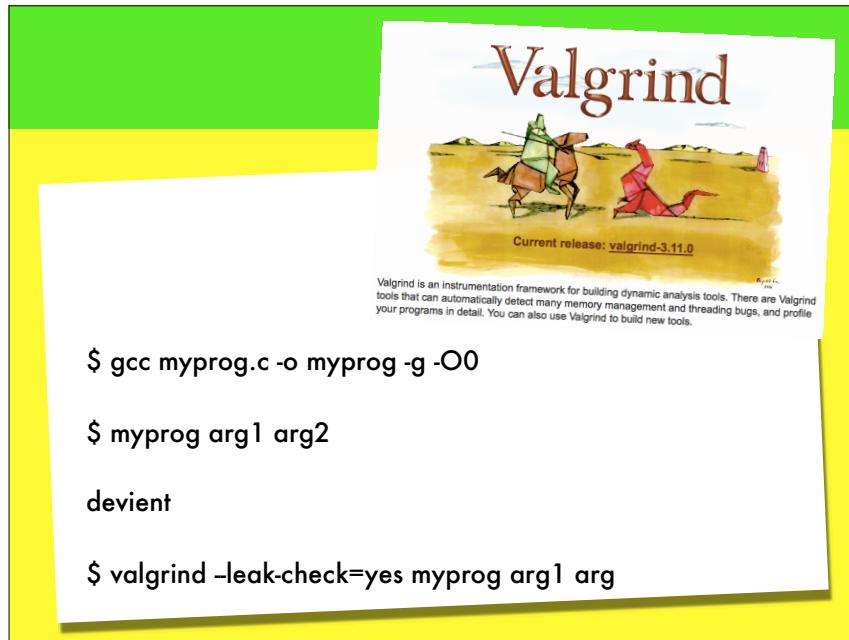
## • Dans les commentaires

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

# GDB the GNU Project Debugger

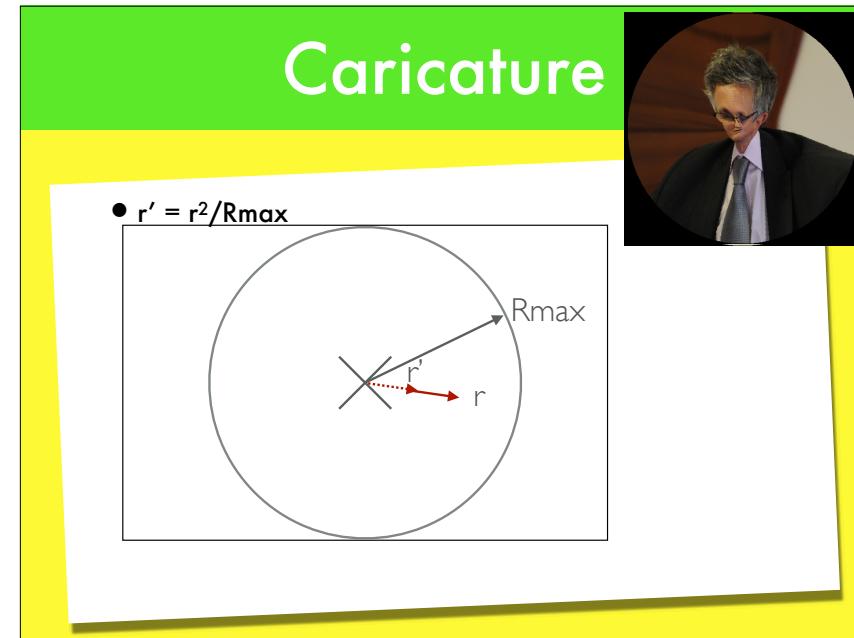
```
$ gcc myprog.c -o myprog -g
$ gdb myprog arg1 arg2
(gdb) run
(gdb) break
(gdb) print
(gdb) up
(gdb) next
(gdb) step
(gdb) watch
(gdb) where
(gdb) list
(gdb) down
(gdb) delete
(gdb) cont
(gdb) ...
(gdb)
```





## 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=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];
    }
```

## Exemple : effet « Givre »

```
Pixel* effetGivre(Pixel *img, int sx, int sy, int np) {
    int i,j,k,adr,k1,k2,c;
    Image tmp = (Pixel*)malloc(sx*sy*np*sizeof(Pixel));
    for(i=1;i<sy-1;i++)
        for(j=1;j<sx-1;j++) {
            c = rand();
            if (c % 2) k1 = -1; else k1 = 1;
            c = rand();
            if (c % 2) k2 = -1; else k2 = 1;
            adr = ((i+k1)*sx+(j+k2))*np; // incrémental ?
            for(k=0;k<np;k++)
                tmp[(i*sx+j)*np+k]=img[adr+k];
        }
    return tmp;
}
```

## Exemple : effet « Givre »

- Principe : « déplacer » aléatoirement les pixels vers une position voisine
- En pratique, on fait le contraire : pour chaque pixel de l'image destination, on calcule son origine dans l'image de départ : pas de pixels non initialisés.

## Effets photométriques

