Tableaux 2D par l'exemple

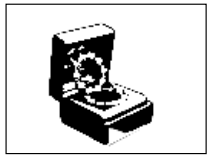
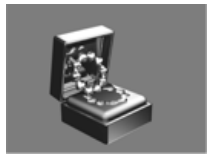
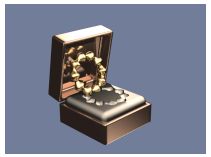
Les images numériques

Image = représentation 2D

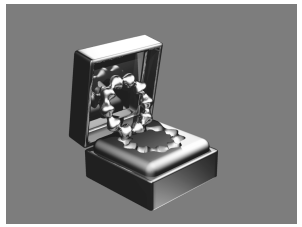
- monde réel, synthèse, dessin, visualisation de données, reconstruction, ...
- Unité ? Niveaux (de gris)
- Fichier, mémoire, organisation, affichage, création, amélioration, compression, ...

Niveaux de gris

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

Pixels

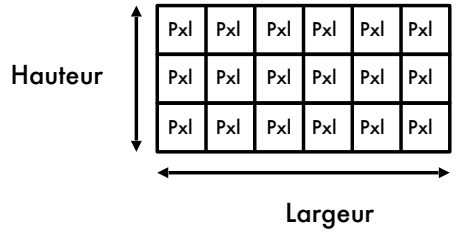



Le type pixel

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



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

Images numériques

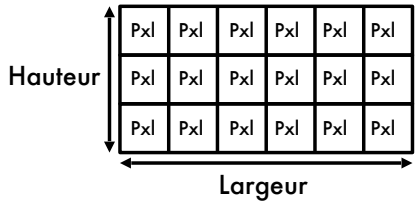
Quelle structure de données ?

Choix de la structure

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

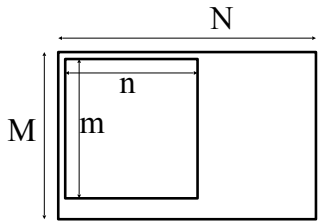
Matrices statiques

Taille d'image fixe : Hauteur et Largeur constantes




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

Matrices statiques



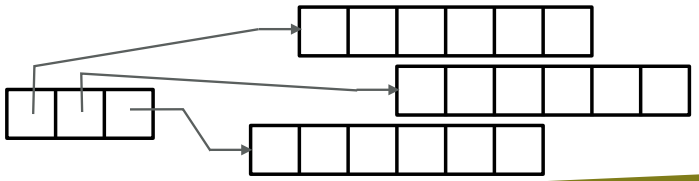

```
/* M et N constantes */
typedef Pixel Image[M][N];
Image bitmap;
int m, n; /* dimensions réelles <= dimensions
             maxi */
```

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);
```



Tableaux 2D dynamiques

M lignes de N colonnes de pixels :

- tableau 1D dynamique de M*N pixels

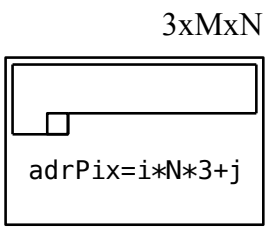
```
Pixel *btm = NULL;
btm=(Pixel *)malloc(M*N*sizeof(Pixel));
```

En pratique

M lignes de N colonnes de pixels (RVB) :

- tableau 1D dynamique de M*N*3 pixels

```
typedef unsigned char Pixel; /* comme gris */
int np = 3;
Pixel *btm = NULL;
btm=(Pixel *)
 malloc(M*N*3*sizeof(Pixel));
```



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 »

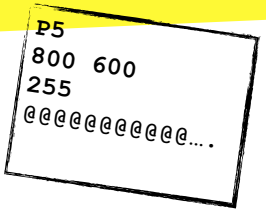
Exemple : effet « Givre »

- 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.

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;
}
```

E/S : lire le « pgm »




```
char buffer[128];
int width, height;
unsigned char *bitmap=NULL;
f = fopen(nomFichier, "rb");
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), f);
```

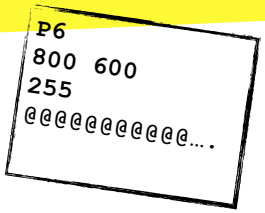
E/S : lire le « ppm »


```
char buffer[128];
int width, height;
unsigned char *bitmap=NULL;
f = fopen(nomFichier, "rb");
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), f);
```

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);
```

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

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

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)

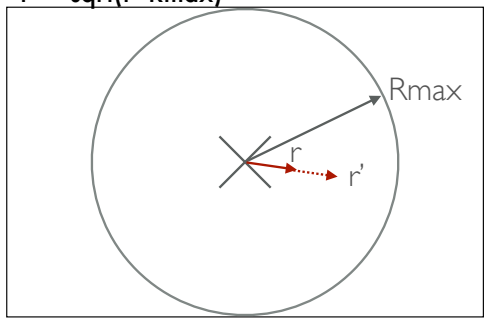
```
/* initialiser res : copie de image source */
for(i=0;i<N;i++) {
   l = rand()*sy/(float)RAND_MAX;
   c = rand()*sx/(float)RAND_MAX);
   adr = (l*sx+c);
   if (l<sy-1)
        if (res[adr]<res[adr+sx]) {
              res[adr+sx]=res[adr];
        }
}
```

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)*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];
      }
}
```

Fish Eye

- r' = sqrt(r*Rmax)

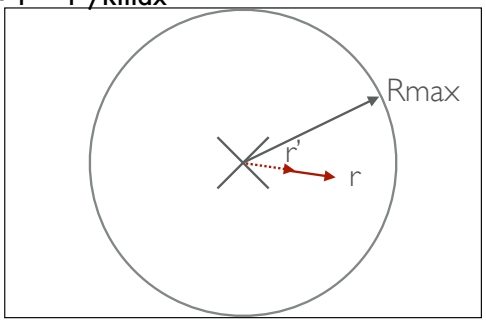


Fish Eye

```
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));
   a = atan2((double)(i-sy/2),(double)(j-sx/2));
   rr = r*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

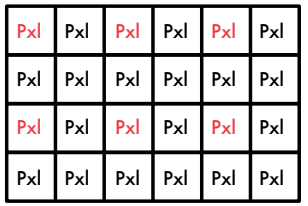
- $r' = r^2/Rmax$



Caricature

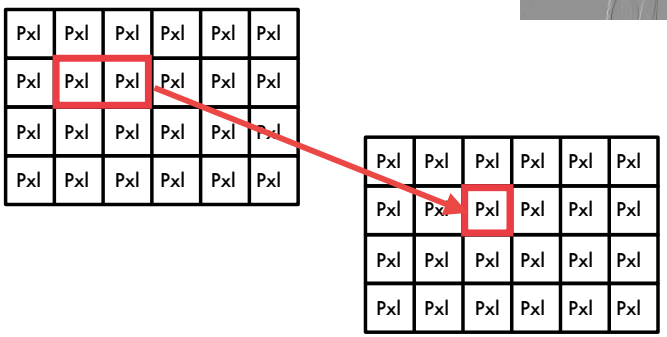
```
R=sqrt((sy/2)*(sy/2));
for(i=0;i<sy;i++)
  for(j=0;j<sx;j++) {
   r = sqrt((i-sy/2)*(i-sy/2) +(j-sx/2)*(j-sx/2));
   a = atan2((double)(i-sy/2),(double)(j-sx/2));
   rr = sqrt(r*R);
   x = sx*0.5 + rr*cos(a);
   y = sy*0.5 + rr*sin(a);
   if (y<0) y=0; else if (y>sy-1) y=sy-1;
   if (x<0) x=0; else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

Sous-échantillonnage



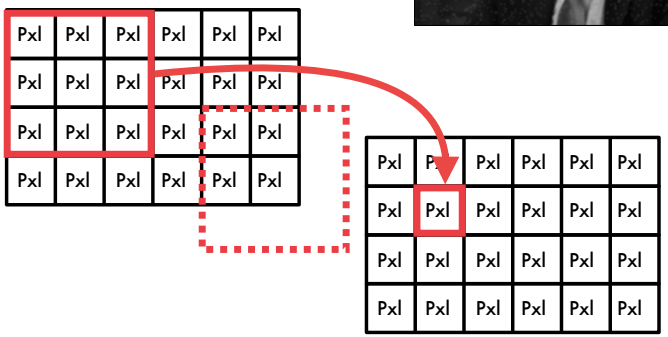
- Filtrage
- Pixelisation

Relief



Filtrage

Floutage



Filtrage médian