Programmation impérative

ENSIIE

Semestre 1 — 2020–21

Tableaux 2D

Tableaux en dimension $n \geq 2$

Besoin assez fréquent :

- matrices
- images
- grille

Plusieurs possibilités pour l'implémentation

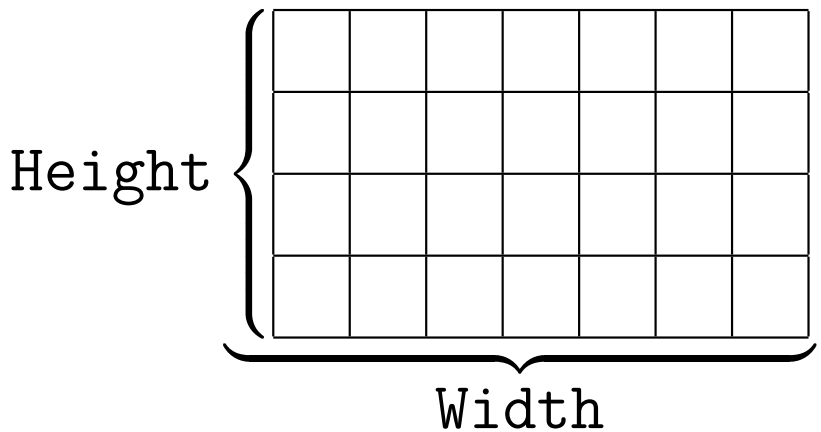
Tableaux statiques

Taille du tableau 2D connu à la compilation :

- hauteur et largeur constante

type_case `tab[Height][Width];`

`Height` et `Width` constantes

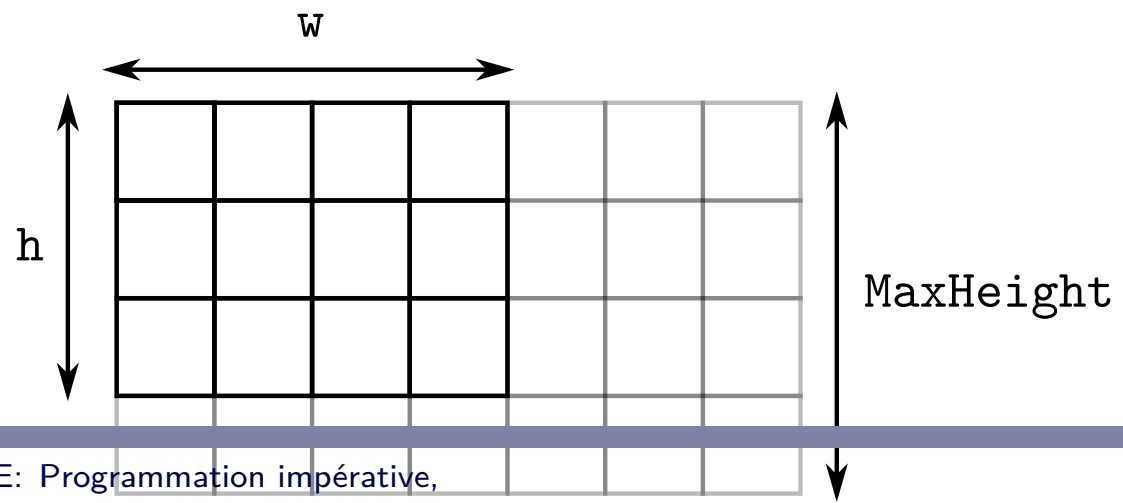


Sous-tableaux

```
type_case tab[MaxHeight][MaxWidth];
int h, w;    /* hauteur et largeur reelles */
```

Utiliser une structure pour regrouper les informations

```
struct array2D {
   type_case content[MaxHeight][MaxWidth];
   int h, w;    /* hauteur et largeur reelles */
};
```



Tableaux 2D dynamiques

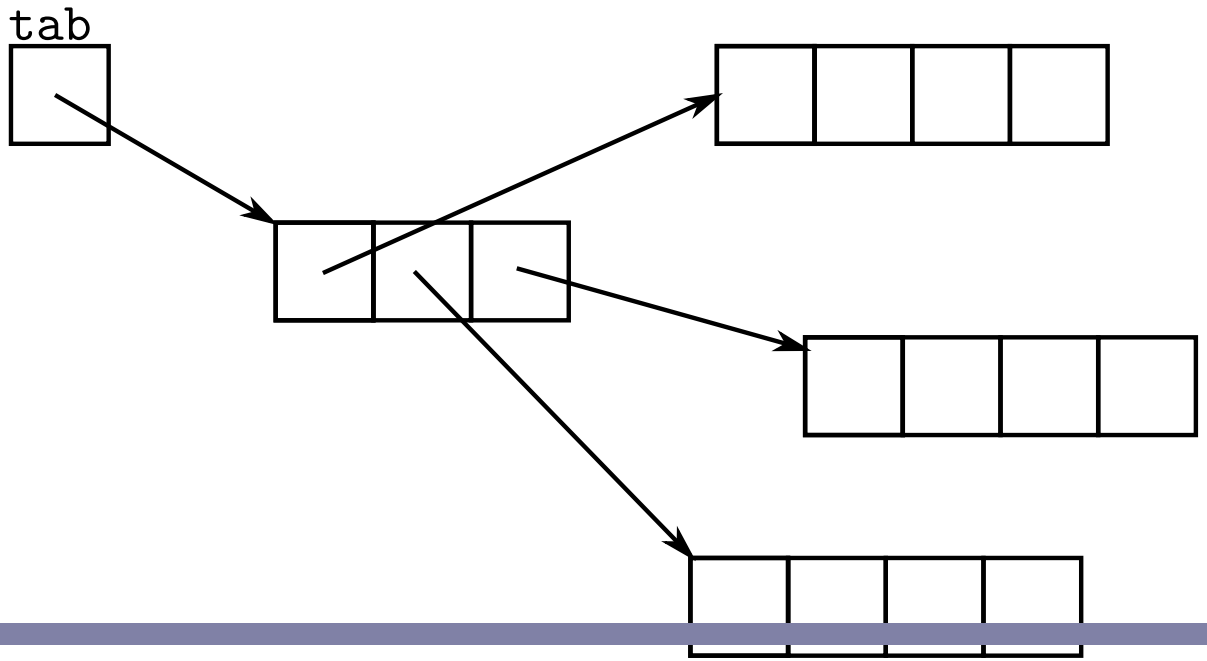
`h` lignes de `w` colonnes

```
type_case **tab = NULL;
tab = (type_case**) malloc(h * sizeof(type_case*
for (i = 0; i < h; i = i + 1)
  tab[i] = calloc(w, sizeof(type_case));
```

Tableaux 2D dynamiques

`h` lignes de `w` colonnes

```
type_case **tab = NULL;
tab = (type_case**) malloc(h * sizeof(type_case*
for (i = 0; i < h; i = i + 1)
  tab[i] = calloc(w, sizeof(type_case));
```

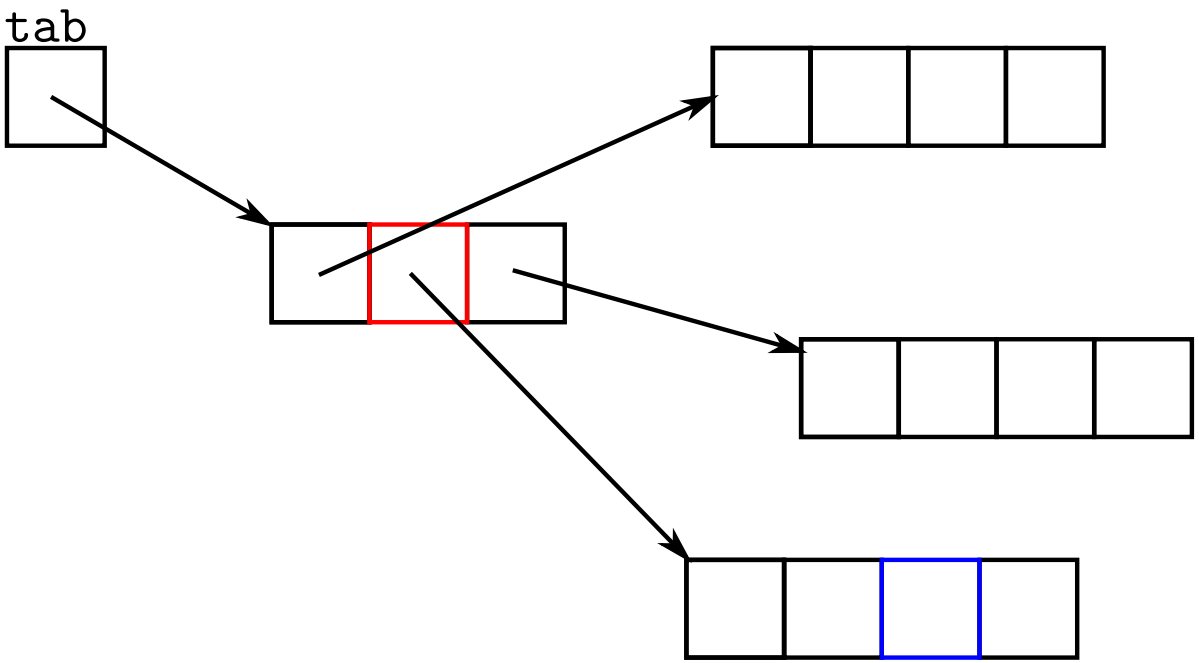


Accès tableau 2D

`tab`

`tab[i]` ligne d'indice `i`, adresse de la case `i,0`

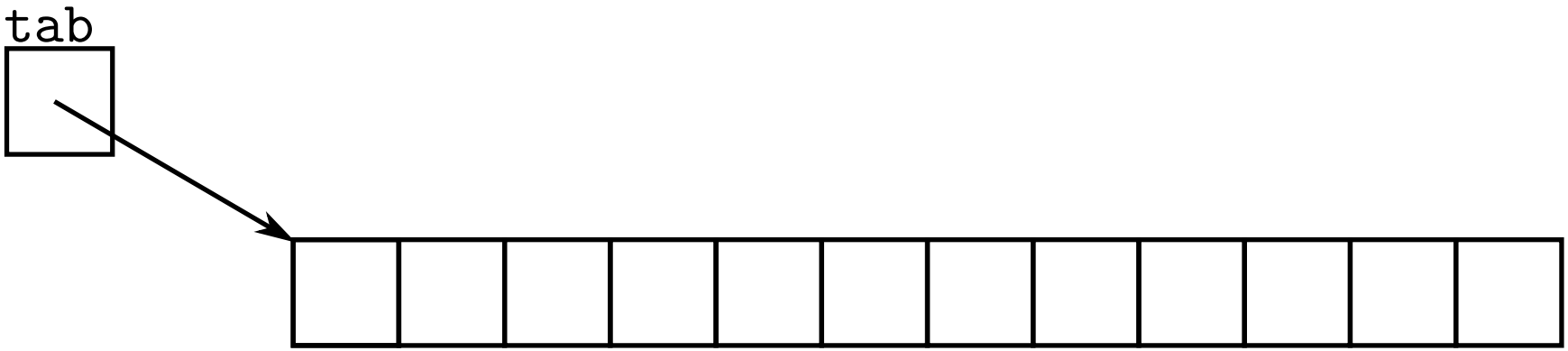
`tab[i][j]` case `i,j`



Autre solution

Tableau 1D dynamique de `h`×`w`

```
type_case *tab = NULL;
tab = (type_case*) calloc(h * w, sizeof(type_cas
```

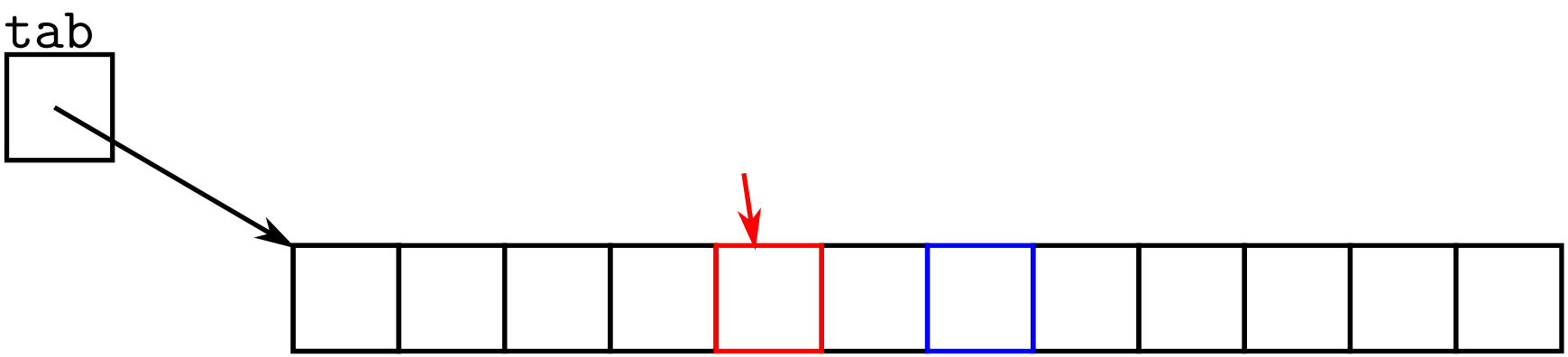


Accès dans représentation 1D

`tab`

`&tab[i*w]` “ligne” d’indice `i`, adresse de la case `i,0`

`tab[i*w + j]` case `i,j`



Tableaux 2D vs tableaux 1D

type_case `**tab;`

- plusieurs blocs mémoire
- copie $\Rightarrow$ itération
- expression du voisinage
- boucles imbriquées
- double déréférencement

type_case `*tab;`

- un seul bloc mémoire
- copie simple
- perte notion de voisin
- parcours linéaire
- calcul de l'indice

Calcul des indices

```
struct array2D {
   type_case *tab;
   int h;
   int w;
};
typedef struct array2D array2D;

type_case get(array2D i, int x, int y);

void set(array2D i, int x, int y, type_case v);
```

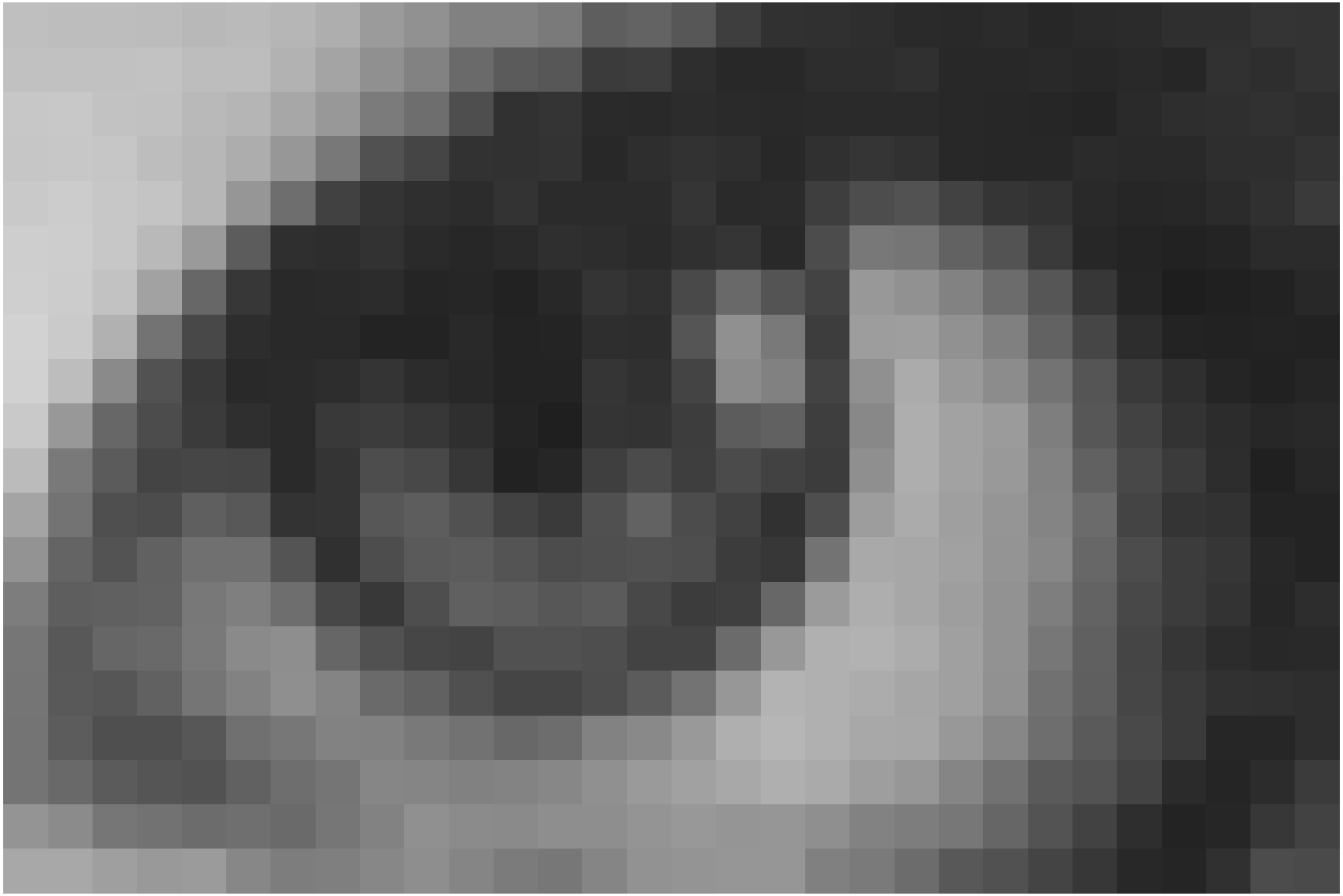
Image numérique

Image = grille 2D de pixels
Pixel → niveau de luminosité

- noir et blanc, niveau de gris, couleur



Pixel



Représentation d'un pixel

Images noir et blanc :

- 1 bit/pixel : 0 blanc 1 noir
- regroupé par 8 (1 octet)
- exemple 42 $\rightarrow$ 

Images niveaux de gris :

- 1 octet/pixel : 0 noir 255 blanc

Images couleurs :

- 3 octets/pixel : niveau de rouge, de vert, de bleu

Extensions :

- 4$^{\text{e}}$ octet pour niveau de transparence (α)
- niveaux sur plus de 1 octet/couleur $\rightarrow$ HDR

Type pixel en C

Niveau de gris :

```
typedef unsigned char pixel;
```

Couleur :

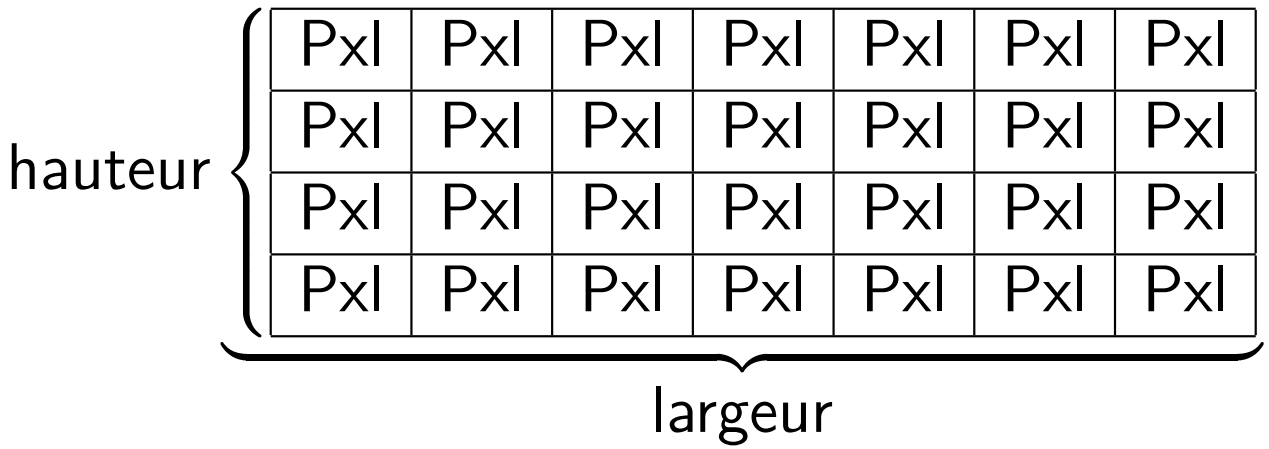
```
typedef unsigned char pixel[3];
```

ou alors

```
struct pixel {
  unsigned char r;
  unsigned char g;
  unsigned char b;
};
typedef struct pixel pixel;
```

Image = tableau 2D

Matrice de pixels Pxl_{ij}



Plus rarement :

- un tableau pour chaque plan (couleur)

Choix de la structure de donnée

- Affichage (matériel)
- Type de traitement (local, voisinage)
- Stockage (en mémoire, dans des fichiers)

Parenthèse : entrées/sorties

Interactions avec l'utilisateur/l'environnement

- arguments passés en ligne de commande
  Exemple :

  ```
  ./mon_programme -option1 --option2 arg1 arg2
  ```

- entrée et sorties standard
- fichiers

Arguments en ligne de commande

```
int main(int argc, char **argv) {
   ...
}
```

- `argc` nombre d'arguments plus 1
- `argv` tableau de chaînes de caractères
  - `argv[0]` nom du programme
  - `argv[1]` premier argument
  - `argv[2]` deuxième argument
  - ...

noms `argc` `argv` = convention

Fichiers

Descripteur `FILE *f;`

- Ouverture `f = fopen(filename, mode);`
  - mode `r|w|a[b][+]`
- Fermeture `fclose(f);`

Texte vs. binaire

Mode texte :

- lisible par un humain
- portable
- volumineux

```
{
    "parentMode": "user",
    "stylesheetName": "gnome-classic.css",
    "enabledExtensions": ["apps-menu@gnome
-shell-extensions.gcampax.github.com","pla
-shell-extensions.gcampax.github.com"],
    "panel": { "left": ["activities"],
               "center": [],
               "right": ["a11y"]
             }
}
```

Mode binaire :

- lisible par la machine
- portabilité ? (ex : boutisme)
- compact

```
......JFIF.....+.+.....C.........
.............................
.......................C.....
.............................
.............................
.............................
.............................
.............................
.............................
*
............................fH..
..............@b......r......q.
:@..........7 .....0...........
....4.........................
```

Lecture/écriture texte

- Entrées/sorties formatées :
  - lecture `fscanf(f, "format", ...)`
  - écriture `fprintf(f, "format", ...)`
- mode ligne
  - lecture `fgets(buffer, size, f)`
  - écriture `fputs(buffer, f)`
- Autres fonctions `fseek`, `ftell`, etc.

Lecture/écriture binaire

- lecture `fread(buffer, size, n, f)`
- écriture `fwrite(buffer, size, n, f)`

- Plus rapide (pas de conversion)
- Moins 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

Format PNM (portable pixmap)

Entête d'un fichier PGM (portable graymap)

```
P5
# nombre magique, 5 = graymap en binaire
500 500
# largeur hauteur
255
# valeur maximum pixel
```

puis hauteur$\times$largeur pixels

Exemple : lire un fichier PGM

```
typedef unsigned char pixel;
char buffer[128];
int width, height;
pixel *btm=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);
btm = malloc(width*height*sizeof(pixel));
fread(btm, sizeof(pixel), width*height, f);
```

Exemple : lire un fichier PPM

```
typedef unsigned char pixel[3];
char buffer[128];
int width, height;
pixel *btm=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);
btm = malloc(width*height*sizeof(pixel));
fread(btm, sizeof(pixel), width*height, f);
```

Écrire dans le PGM

```
f = fopen(nomFichier, "wb");
fprintf(f, "P5\n%d␣%d\n255\n", width, height)
fwrite(btm, sizeof(pixel), width*height, f);
fclose(f);
```

Écrire dans le PPM

```
f = fopen(nomFichier, "wb");
fprintf(f, "P6\n%d␣%d\n255\n", width, height)
fwrite(btm, sizeof(pixel), width*height, f);
fclose(f);
```

Traitement d'image

Idée générale :

- on applique une fonction de transformation sur la matrice de pixel

Exemples :

- inversion vidéo
- floutage, bruit
- filtre
- contraste, lumière, teinte
- changement de taille
- ...

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 qui changent d'échelle

Inversion vidéo

Modification en place vs. recopie

```
void videoInvBW(pixel *btm, int w, int h) {
  int i;
  for(i = 0; i < w * h; i = i + 1)
   btm[i] = 255 - btm[i];
}

pixel *videoInvBW(pixel *btm, int w, int h) {
  int i;
  pixel *res =
  (pixel*) malloc(w * h * sizeof(pixel));
  for(i = 0; i < w * h; i = i + 1)
    res[i] = 255 - btm[i];
  return res;
}
```

Transformation couleur vers niveaux de gris

Couleur vers niveaux de gris

Calcul de la luminance :

$$0,707R + 0,202V + 0,071B$$

```
pixel* imgNB= (pixel*) malloc(H*L);
for(i=0; i < w * h; i = i + 1)
   imgNB[i]= 0.707*img[3*i]
             +0.202*img[3*i+1]
             +0.071*img[3*i+2];
```

Éclaircir

Éclaircir

```
int v;
for(i=0; i < w * h; i = i + 1) {
  v = 1.25 * btm[i];
  if (v > 255) v = 255;
  res[i] = v;
}
```

Fonte

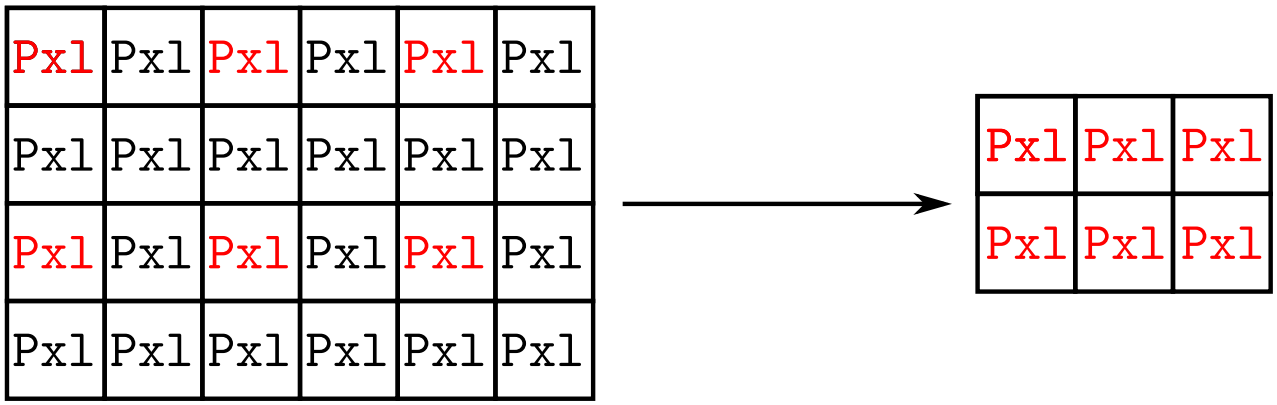
- Sélectionner des pixels au hasard
- Si plus sombre que voisin du dessous « glisser vers le bas »

Fonte

```
for (k = 0; k < N; k = k + 1) {
  i = rand()*(float)height/RAND_MAX;
  j = rand()*(float)width/RAND_MAX;
  if (i < height - 1)
    if (btm[i*width+j] < btm[(i+1)*width+j])
      btm[(i+1)*width+j] = btm[i*width+j];
}
```

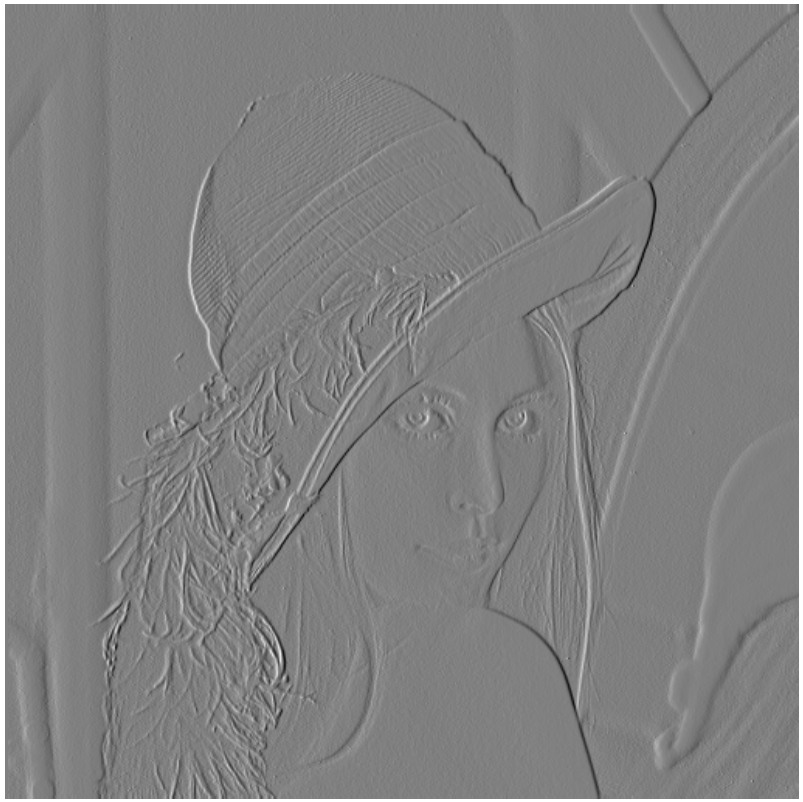
En pratique `N` vaut au moins `width * height`

Sous-échantillonnage



- pixelisation

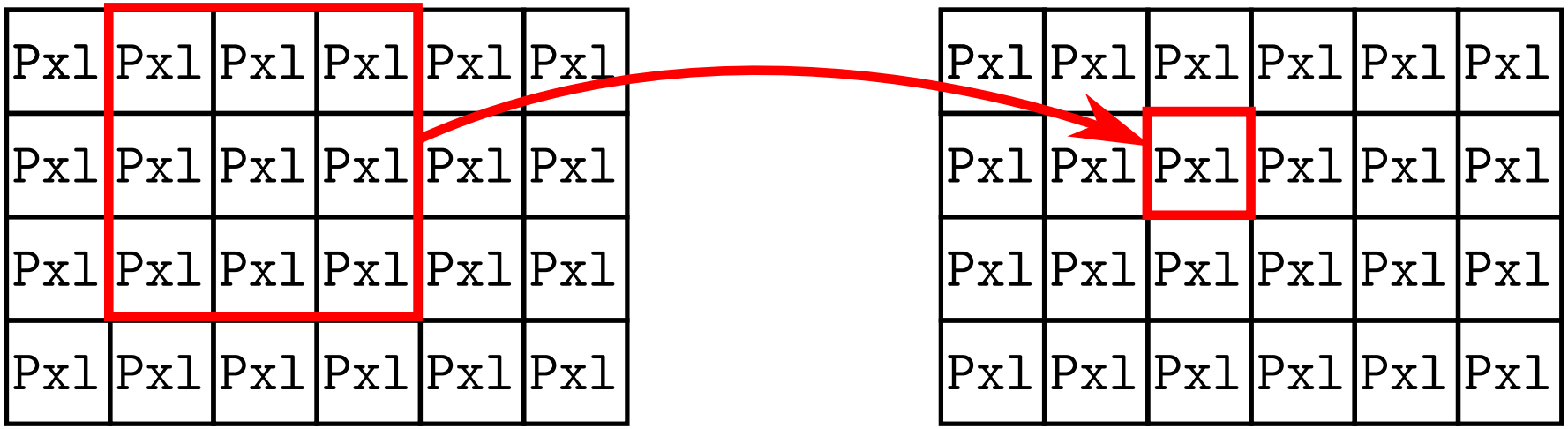
Relief



différence avec le voisin de gauche

Floutage

Floutage



Nouvelle valeur = moyenne du voisinage

Attention aux bords !

Filtrage médian

Filtrage médian