

# **Algorithmes pour la Programmation Graphique**

**Niveau L2, semestre 1, Licence Informatique & Vidéoludisme,  
Université Paris 8**

# Objectifs

- Généralités sur l'image et la synthèse d'images
- Introduction à l'Analyse Discrète Différentielle
- Introduction à la géométrie algorithmique
  - Découpage de l'espace, triangulation, intersection/collision, ...
- Automates cellulaires, courbes paramétriques, ...
- Spécification et réalisation d'un pipeline de rendu
  - Remplissage, coloriage par dégradé, textures, ombrage, transformations spatiales
- Utilisation du « moteur de rendu »



# Comment ?

- Présentation de concepts/notions
- Cas pratiques (implémentation)
  - On programmera en C
  - On utilisera GL4D 🤖
  - TP/DM notés
- Petit projet final (gros DM) utilisant les implémentations réalisées au cours du semestre ???
- Les rendus seront effectués par le biais du moodle du cours.

# L'image numérique 🙄

- On va regarder du côté de **wiki**
- Quel est l'objet de ce cours : le **Computer Graphics**
- L'image vectorielle
  - Le Vectrex  
<https://www.youtube.com/watch?v=dBLqZrdGhEA>
  - Rien depuis ?
- L'image Bitmap (matricielle 🤢)



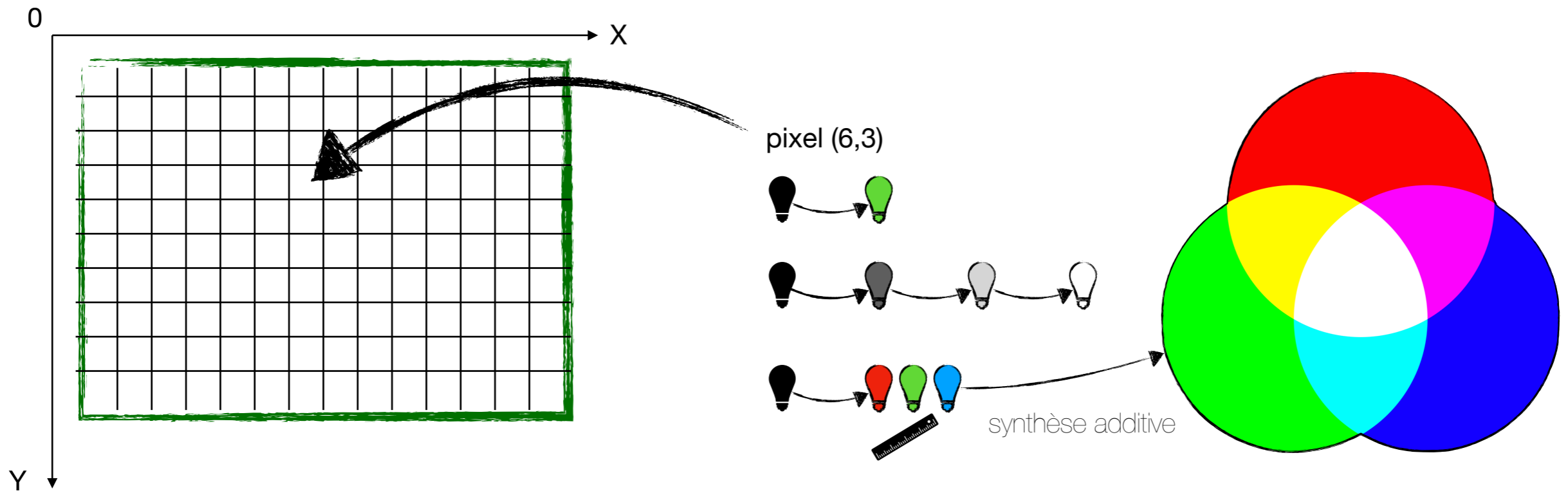
Tableau, grille, ... de points colorés -> les pixels

Monochrome (fin '70, début '80), palette de couleurs, 16M de couleurs ...

# Le pixel

- ***Picture Element***

- Le pixel = allumé/éteint (bit, d'où Bitmap) | intensité | intensités
- Physique (1px  $\varphi \rightarrow$  1px logique) ou Simulé ( $X$ px  $\varphi \rightarrow$  1px logique,  $X \in \mathbb{R}^*$ ,  $\Rightarrow$  interpolation)



# Le pixel

pixel

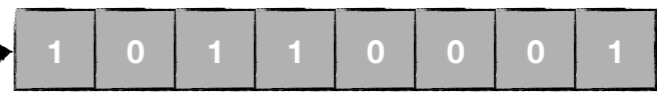


**un bit (on/off)**  
Bitmap



1 octet -> 8 pixels

**un byte (8 bits)**  
d'intensités de blancs  
Grayscale 256 niveaux

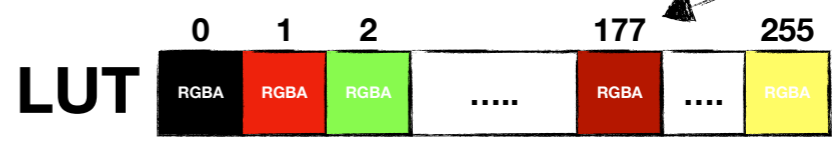


1 octet -> 1 pixel gris, intensité = 177 / 255

**un byte (8 bits)**  
de couleurs indexées  
IndexColor 2 à 256



1 octet -> 1 pixel n°177, aller voir la LUT  
LUT = Look Up Table ou table d'indirection



**deux bytes (16 bits)**  
d'intensités de blancs  
Grayscale 65K niveaux



2 octets -> 1 pixel gris, intensité = 45489 / 65535  
256 fois plus de détail dans la variation

**deux bytes (16 bits)**  
5 bits / composante R,G,B  
1 bit : transparent / opaque  
16 bits color



2 octets -> 1 pixel ici opaque,  
R = 22/31, G = 6/31, B = 24/31



le pixel

**3 bytes (24 bits)**  
8 bits / composante R,G,B  
16M colors



3 octets -> 1 pixel,  
R = 255/255, G = 127/255, B = 0/255

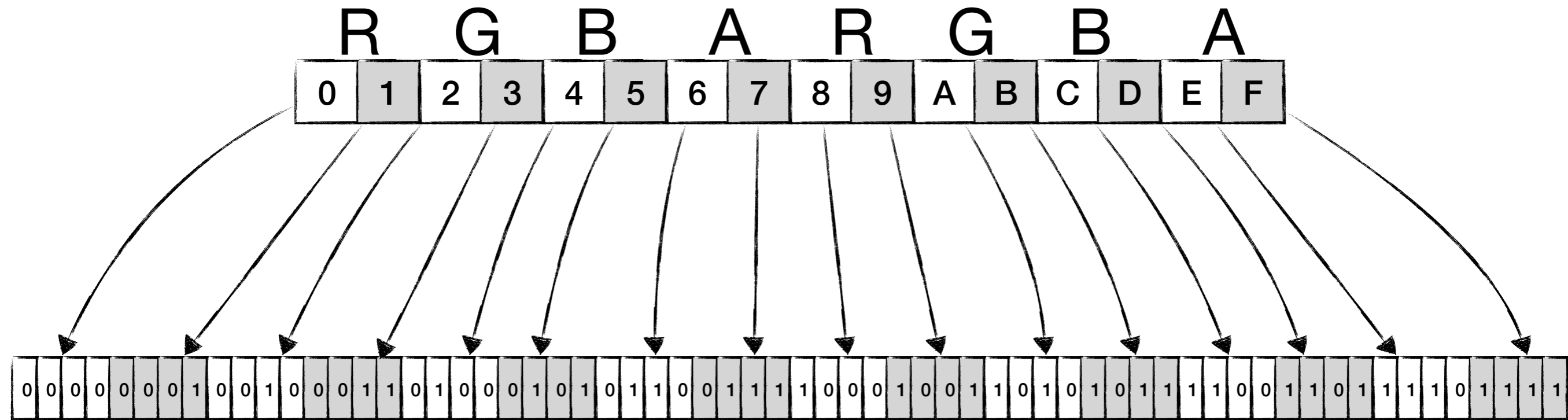


le pixel

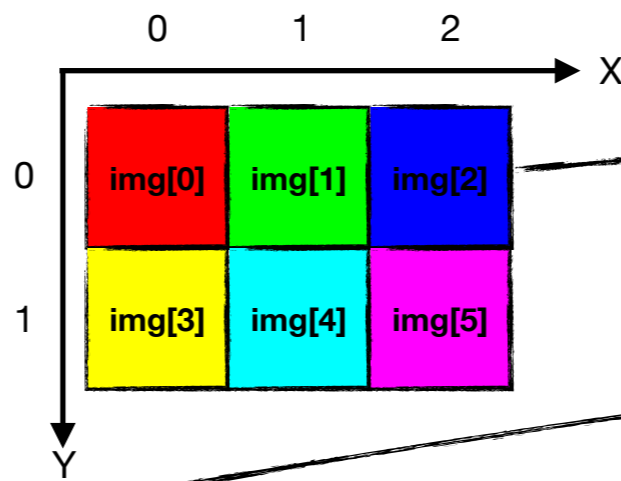
**4 bytes / 1 int (32 bits)**  
8 bits / comp. R,G,B et A  
16M colors + alpha

//  
+ 256 niveaux d'alpha pour le mélange de couches d'images

# Le pixel



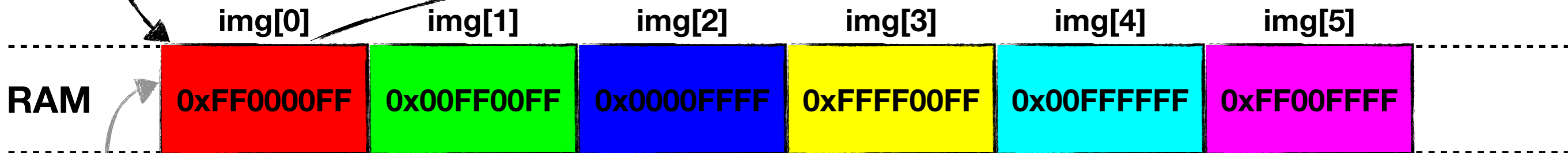
# Le pixel



Ce qu'on imagine

En RAM

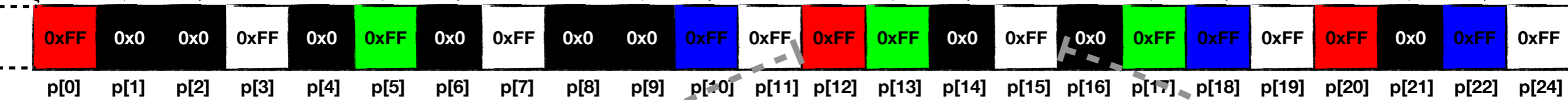
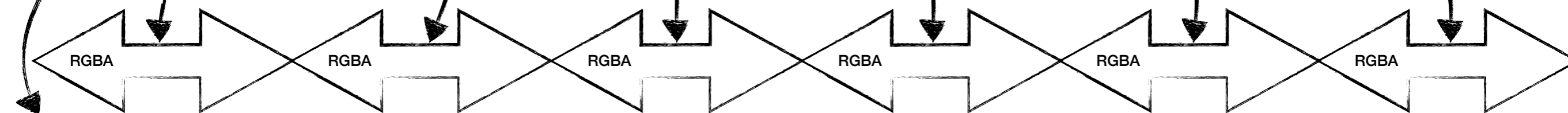
`int32_t* img` (cf. `stdint.h`)



RAM

aussi, car la même chose

`int8_t* p = (int8_t*)img`





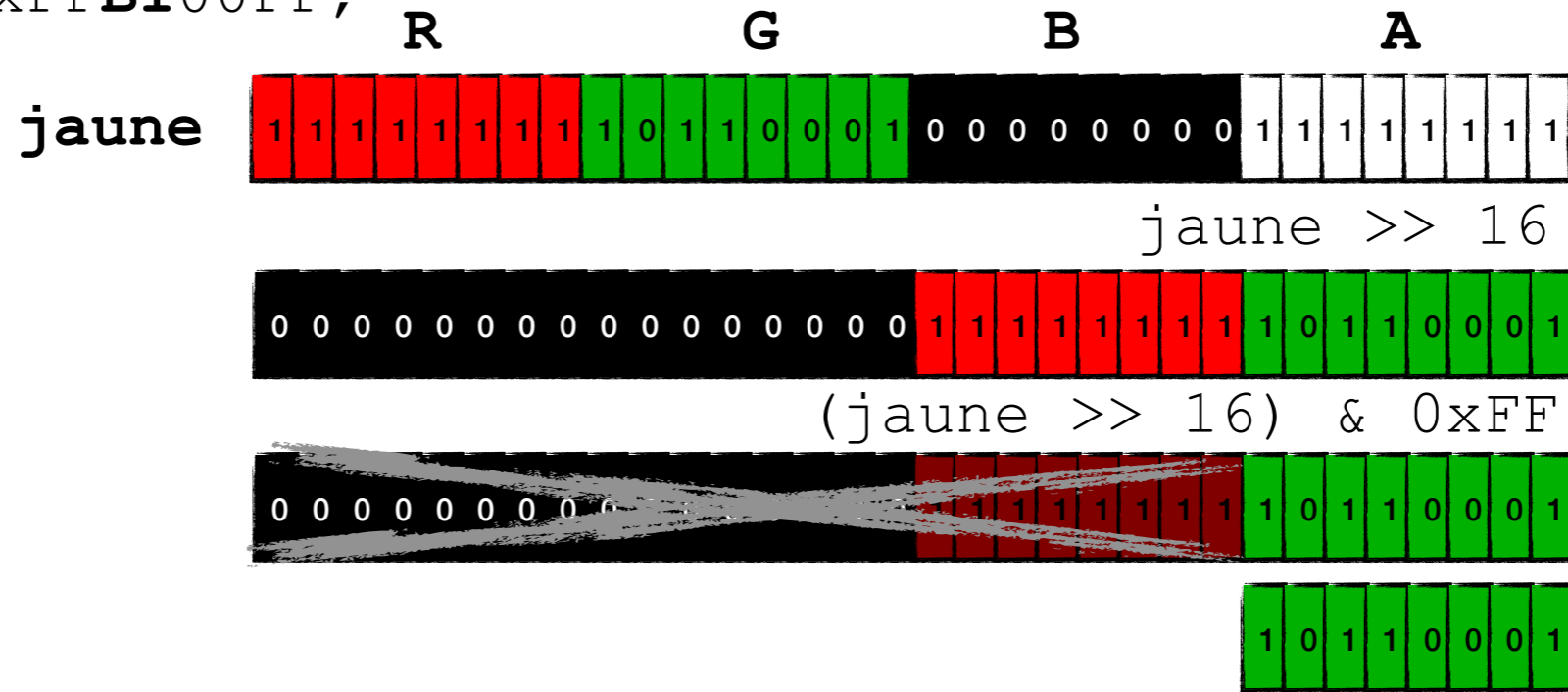
# Le pixel



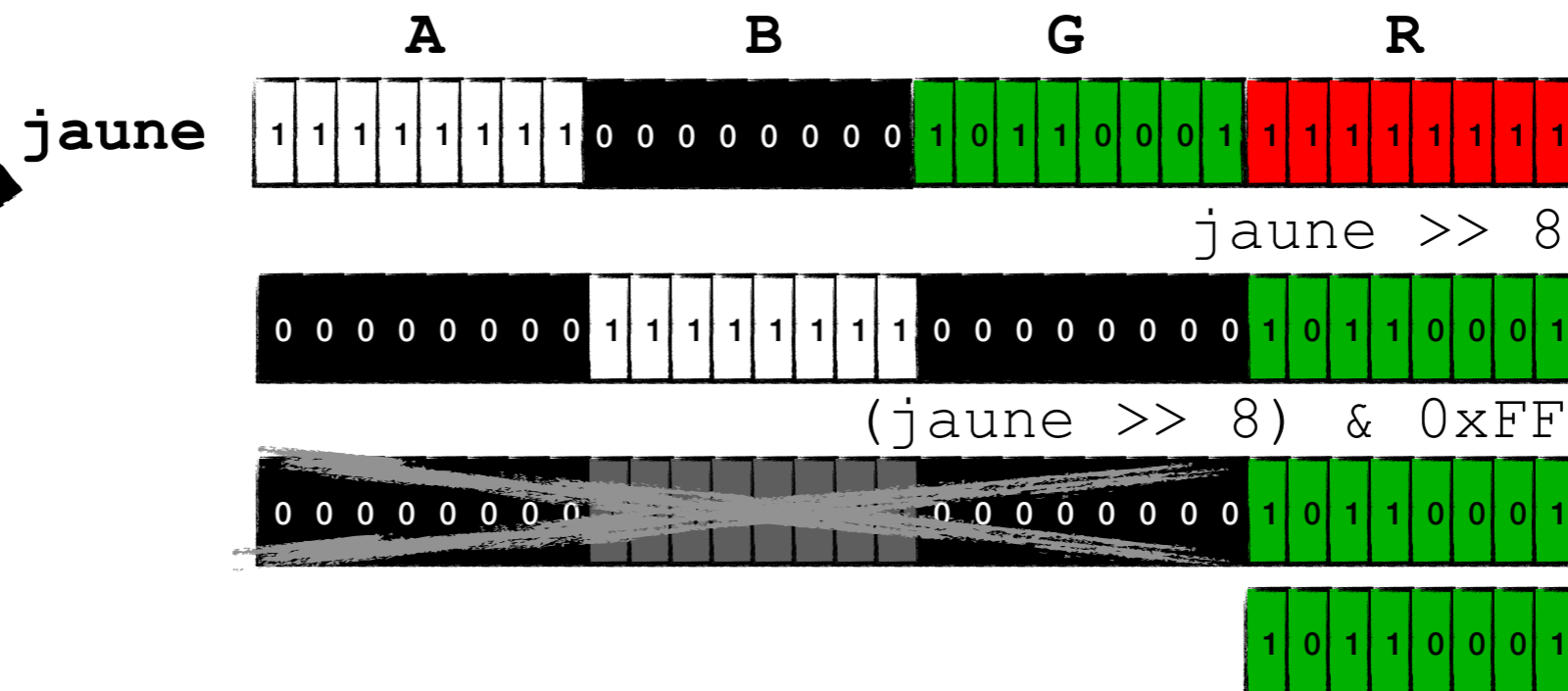
*Récupérons la composante verte de ce jaune*

```
int32_t jaune = 0xFFB100FF;
```

**BIG-ENDIAN**

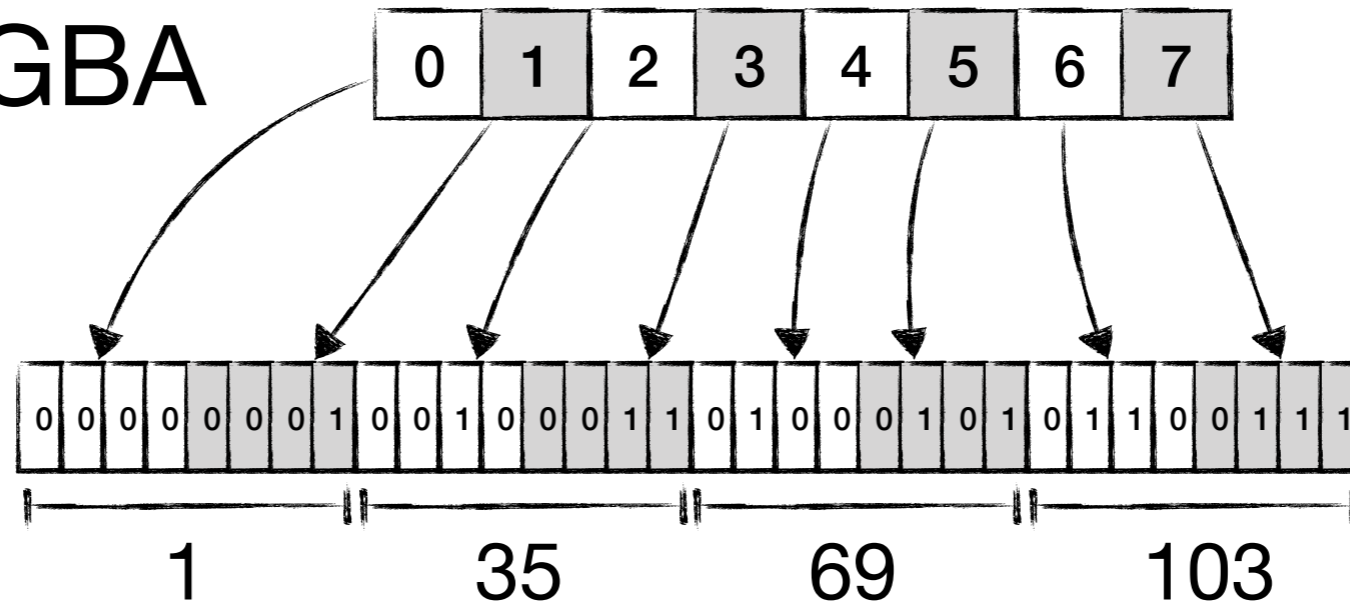


**LITTLE-ENDIAN**



# Le pixel — un peu de pratique

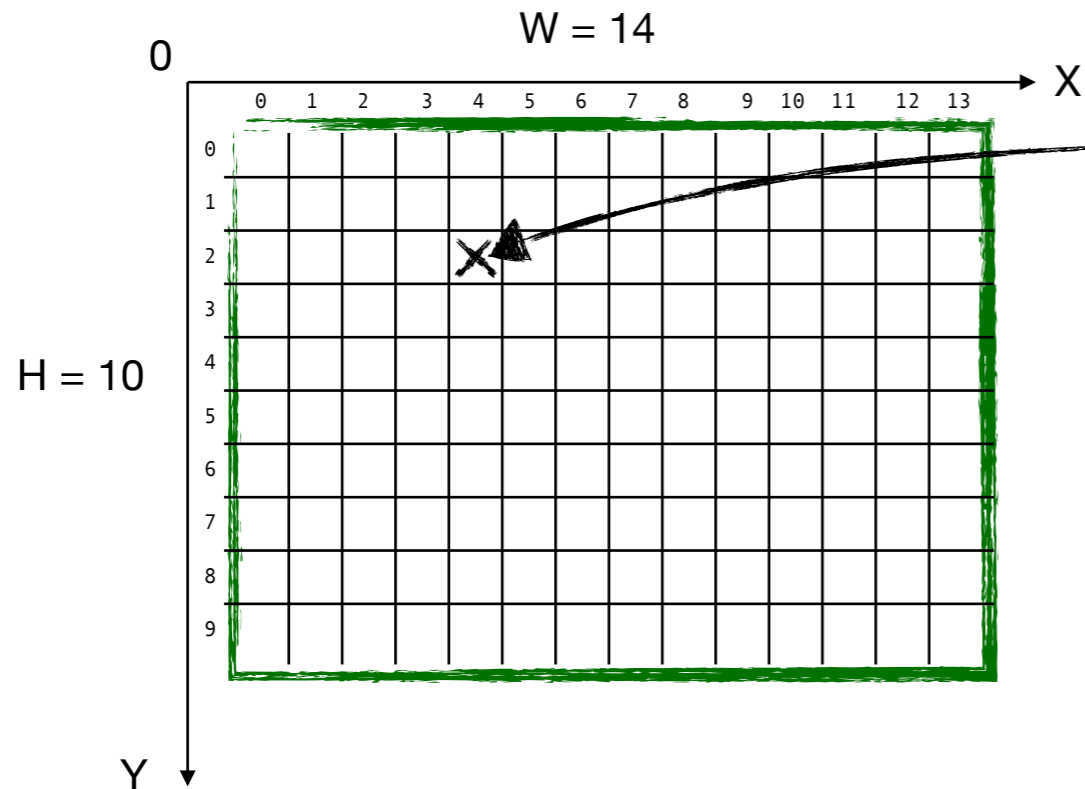
1 RGBA



```
#include <stdint.h>
#include <stdio.h>
int main(void) {
    char imgc[] = { 1, 35, 69, 103 }, * pc;
    int32_t imgi = { 1 << 24 | 35 << 16 | 69 << 8 | 103 }, * pi, i;
    pc = (char *)&imgi;
    pi = (int32_t *)&imgi;
    for(i = 0; i < 4; ++i)
        printf("imgc[%d] = %d \t pc[%d] = %d\n", i, imgc[i], i, pc[i]);
    printf("imgi = %d \t *pi = %d\n", imgi, *pi);
    return 0;
}
```



# Le pixel dans la grille

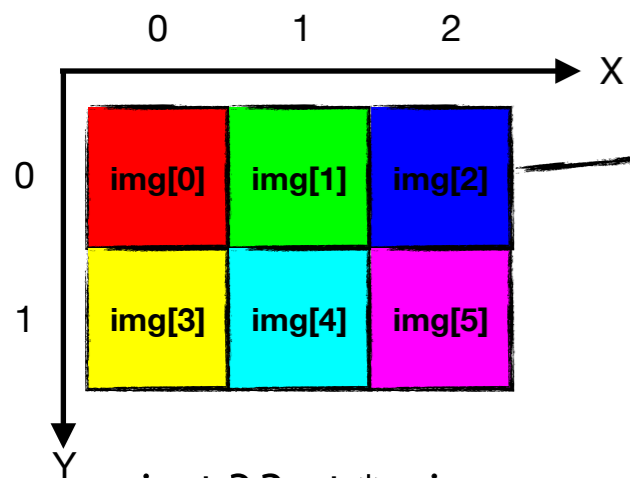


Case d'indice  $i = 32$

En même temps, ses coordonnées dans la grille sont  $x = 4$  et  $y = 2$

Comment avec 4 et 2 arriver à 32 et inversement ???

$(2 \times 14 + 4 = 32)$  ... trouvez comment faire :  
 $(x, y) \rightarrow i$  et  $i \rightarrow (x, y)$



Ce qu'on imagine

Ce qu'on a en RAM

`int32_t* img`

`img[0]`

`img[1]`

`img[2]`

`img[3]`

`img[4]`

`img[5]`

RAM

`0xFF0000FF`

`0x00FF00FF`

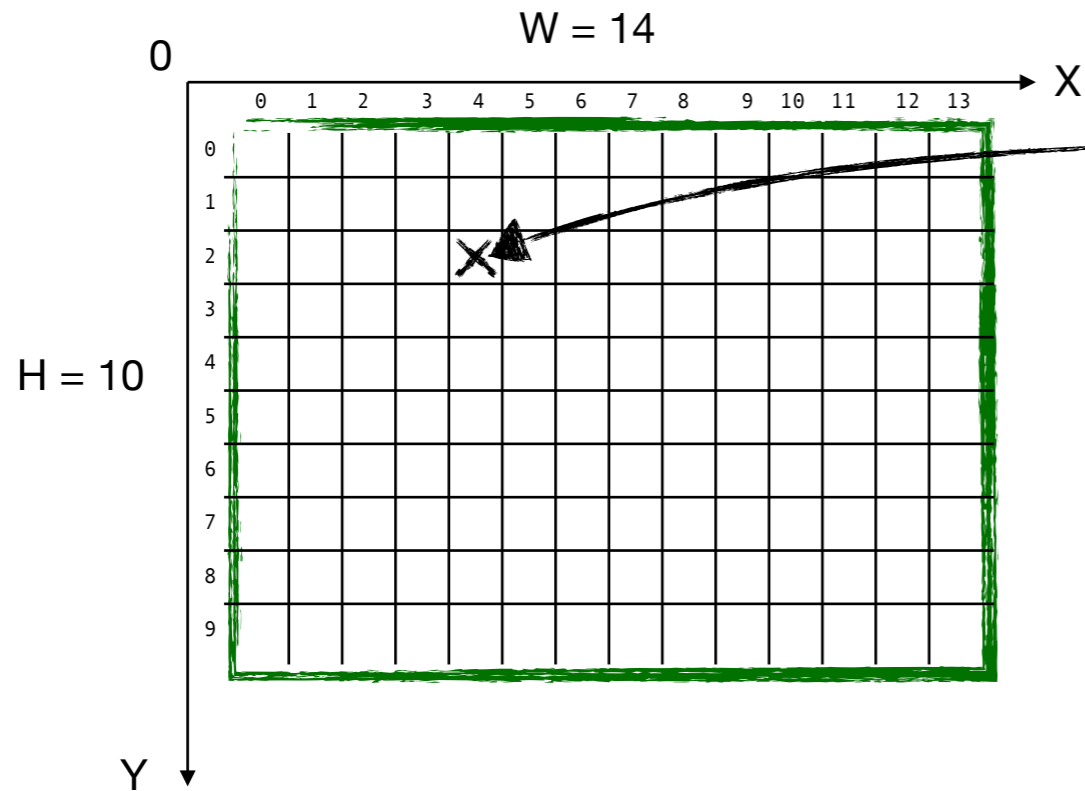
`0x0000FFFF`

`0xFFFF00FF`

`0x00FFFFFF`

`0xFF00FFFF`

# Le pixel dans la grille



Case d'indice  $i = 32$

En même temps, ses coordonnées dans la grille sont  $x = 4$  et  $y = 2$

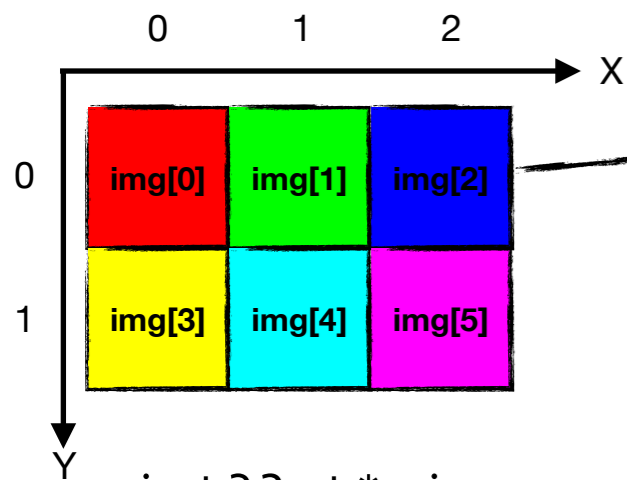
Comment avec 4 et 2 arriver à 32 et inversement ???

$(2 \times 14 + 4 = 32)$  ... trouvez comment faire :

$(x, y) \rightarrow i$  et  $i \rightarrow (x, y)$

$$\text{Pour } (x, y) \rightarrow i = y \times W + x$$

$$\text{Pour } i \rightarrow \begin{cases} x = i \% W \\ y = \lfloor i / W \rfloor \end{cases}$$



Ce qu'on imagine

Ce qu'on a en RAM

`int32_t* img`

`img[0]`

`img[1]`

`img[2]`

`img[3]`

`img[4]`

`img[5]`

RAM

`0xFF0000FF`

`0x00FF00FF`

`0x0000FFFF`

`0xFFFF00FF`

`0x00FFFFFF`

`0xFF00FFFF`

# Pratique (rapide)

Téléchargez le code

[https://expreg.org/amsi/C/APG2223S1/code/00\\_basiques/static\\_2d\\_array-1.0.tgz](https://expreg.org/amsi/C/APG2223S1/code/00_basiques/static_2d_array-1.0.tgz)

PUIS

```
/*  
* Exercice : modifier la fonction afficher pour qu'elle ne soit plus "rigide"  
* et faire que le parcours se fasse en une simple boucle de 0 à n - 1 où n = w * h.  
* Attention à ne pas oublier le saut de ligne à fin de chacune des lignes de l'image.  
*/
```

# La droite (rappels)

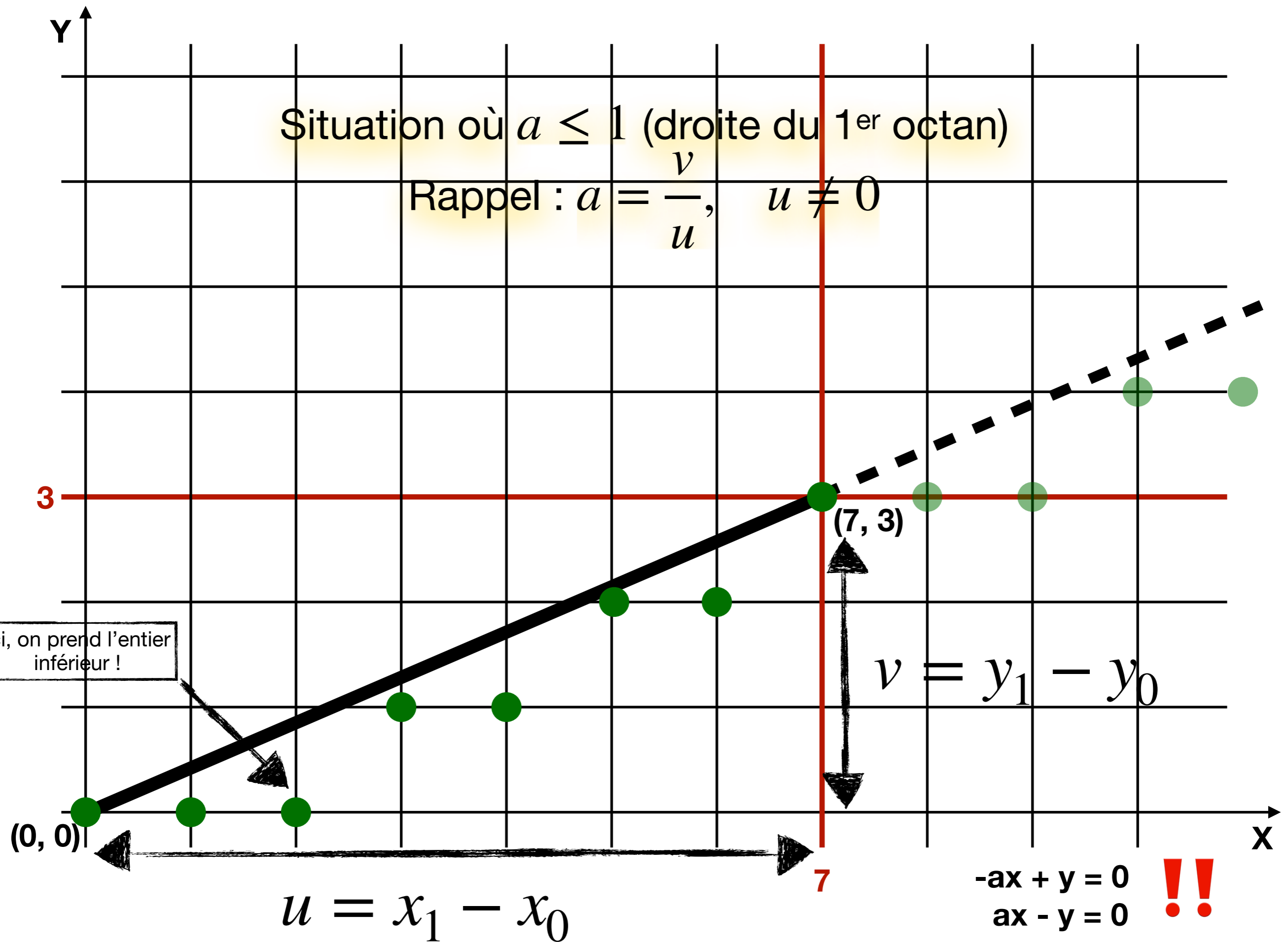
- L'équation ? (Ou les équations)
  - $(a, b, c) \in \mathbb{R}^3, ax + by + c = 0 \quad (1)$
  - Sinon sous la forme d'une fonction affine  
 $y = f(x) = ax + b \quad (2)$   
où  $a$  est le coefficient directeur (ou la pente) et  $b$  est l'ordonnée à l'origine (où la droite coupe l'axe des ordonnées - ou axe des  $y$ )
  - MAIS on peut simplifier :  $y = ax + b \iff y - b = ax$   
on pose  $y' = y - b$  (soit  $y = y' + b$ ) alors on utilise simplement  $y' = ax$
  - En résumé, on peut toujours ramener cette fonction affine vers le cas particulier d'une fonction linéaire puis refaire un changement de variable au moment du dessin ( $y = y' + b$ ), soit décaler  $y$  de  $b$  au moment de colorier le « pixel ». Partons donc sur  $y = f(x) = ax \quad (3)$

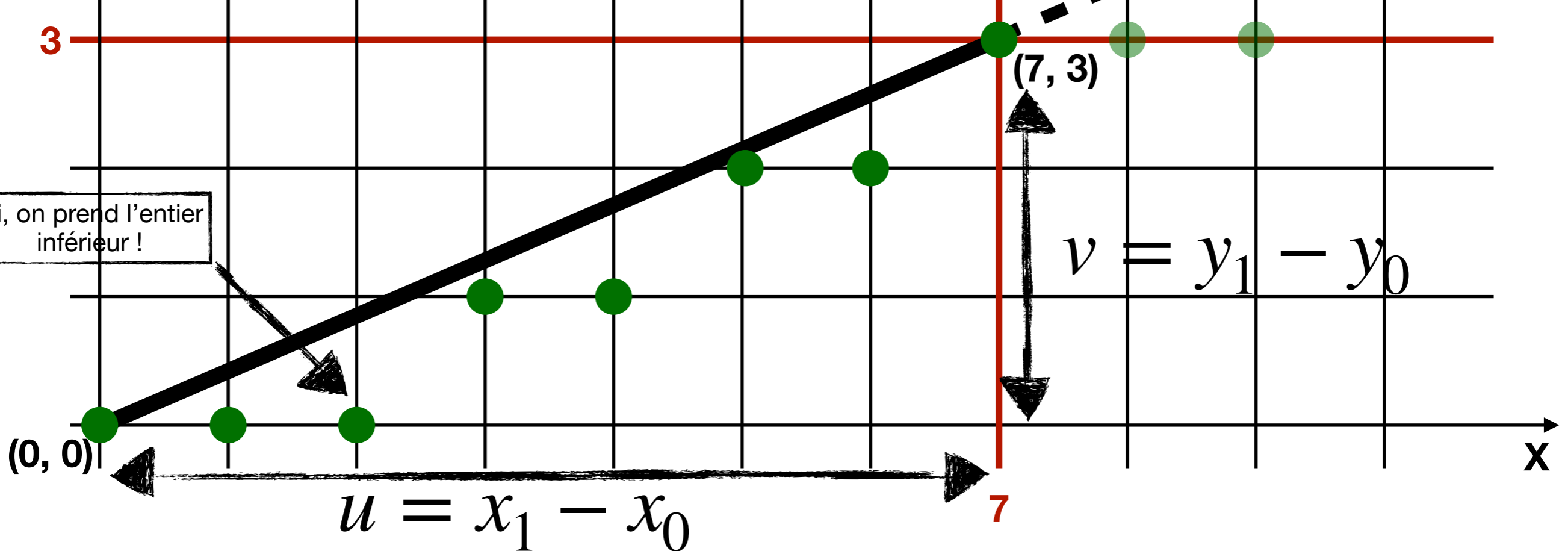
# La droite (rappels)

- Du coup, la pente  $a$  (ou coefficient directeur) ?  
Qu'exprime-t-il ?
  - Combien de déplacements  $dv$ , effectués verticalement, pour combien de déplacements  $du$ , effectués horizontalement.
  - $a$  est donc le rapport  $dv$  sur  $du$  !  
$$a = \frac{dv}{du}$$
 ou plus généralement noté  $a = \frac{v}{u}$

# Le segment de droite

- N'est pas une droite !
  - On part de  $(x_0, y_0)$ , pour arriver à  $(x_1, y_1)$  ou inversement :)
  - On peut en déduire que le déplacement vertical est  $v = y_1 - y_0$  et que le déplacement horizontal est  $u = x_1 - x_0$
  - Donc  $a = \frac{v}{u} = \frac{y_1 - y_0}{x_1 - x_0}$  (attention à  $u = x_1 - x_0 = 0$ )





$$a = \frac{v}{u}, \quad u \neq 0, \text{ ici nous soulignons le fait que } y = ax \text{ est un rationnel, donc flottant (float)}$$

Partant de la relation  $y = ax$ , nous constatons ici que  $x$  démarre à 0 et **augmente de un en un** jusqu'à **atteindre**  $u$ . Un entier (`int`) suffit donc à stocker les valeurs pour  $x$ .

Concernant  $y$ , soit ① nous calculons  $y = ax$  pour chaque valeur de  $x$  (un entier fera l'affaire pour  $y$  — multiplication flottante *castée* dans un entier), soit ② nous considérons  $y$  comme un flottant qui démarre à `0.0f` et qui est incrémenté de  $a$  à chaque itération de  $x$  (ici si  $y$  est entier, le cast l'empêchera d'avancer, dans tous les cas sauf quand  $a = 1$ , donc non).

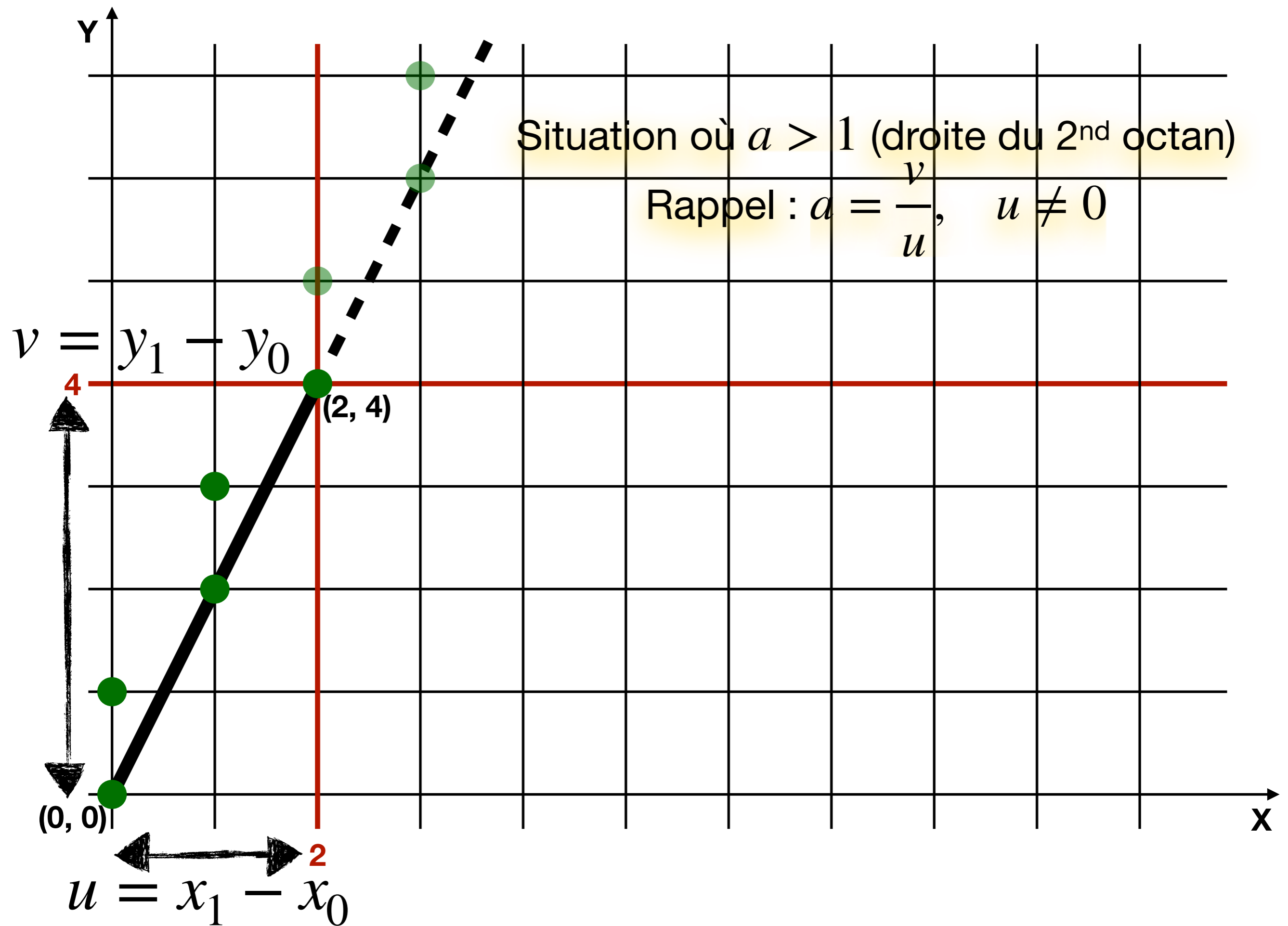
Le dessin du segment se fera en coloriant, avant incrément, chaque coordonnée entière donnée par  $(x, y)$  dans le cas ① et par  $(x, (\text{int})y)$  dans le cas ②.

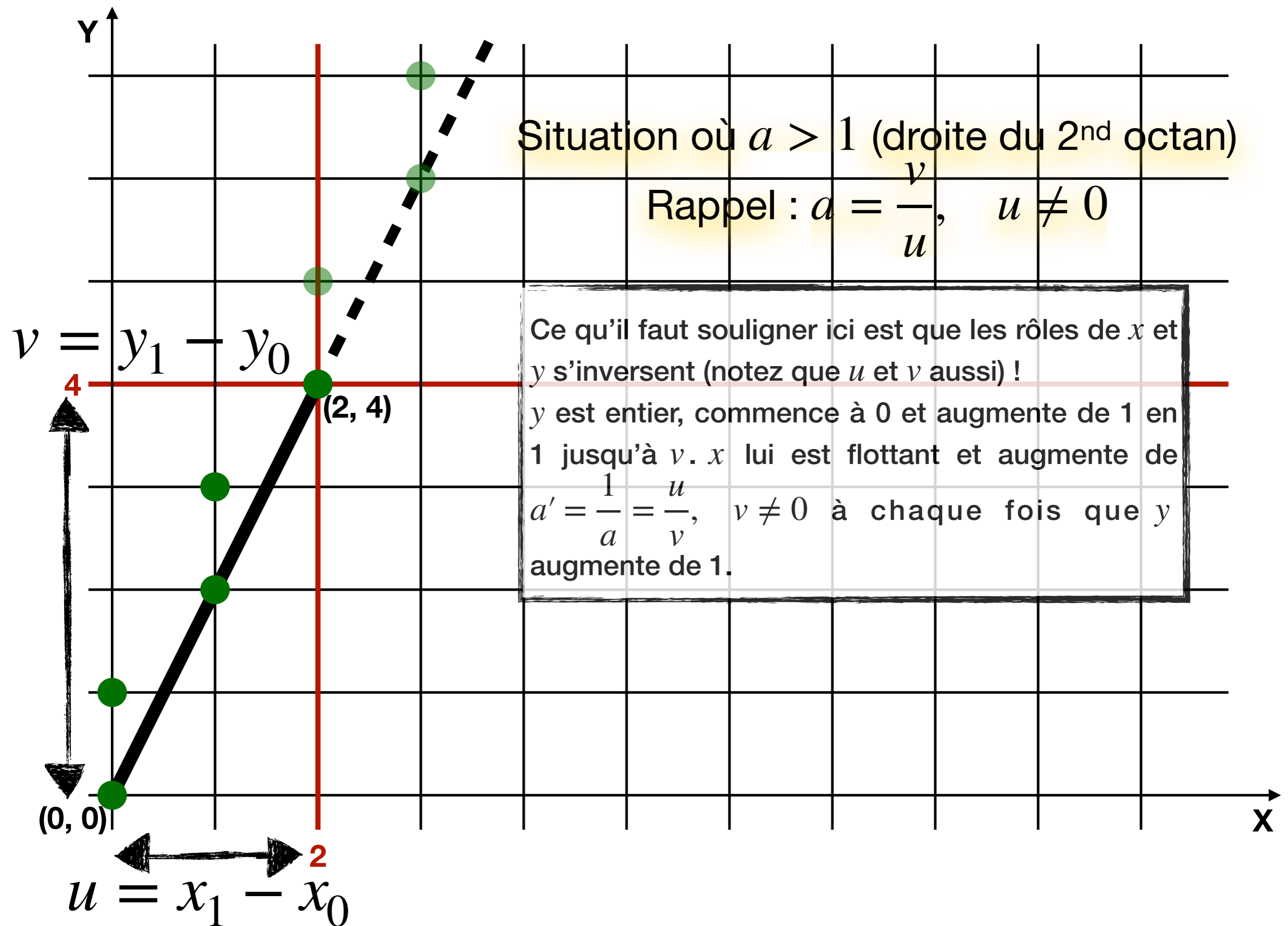


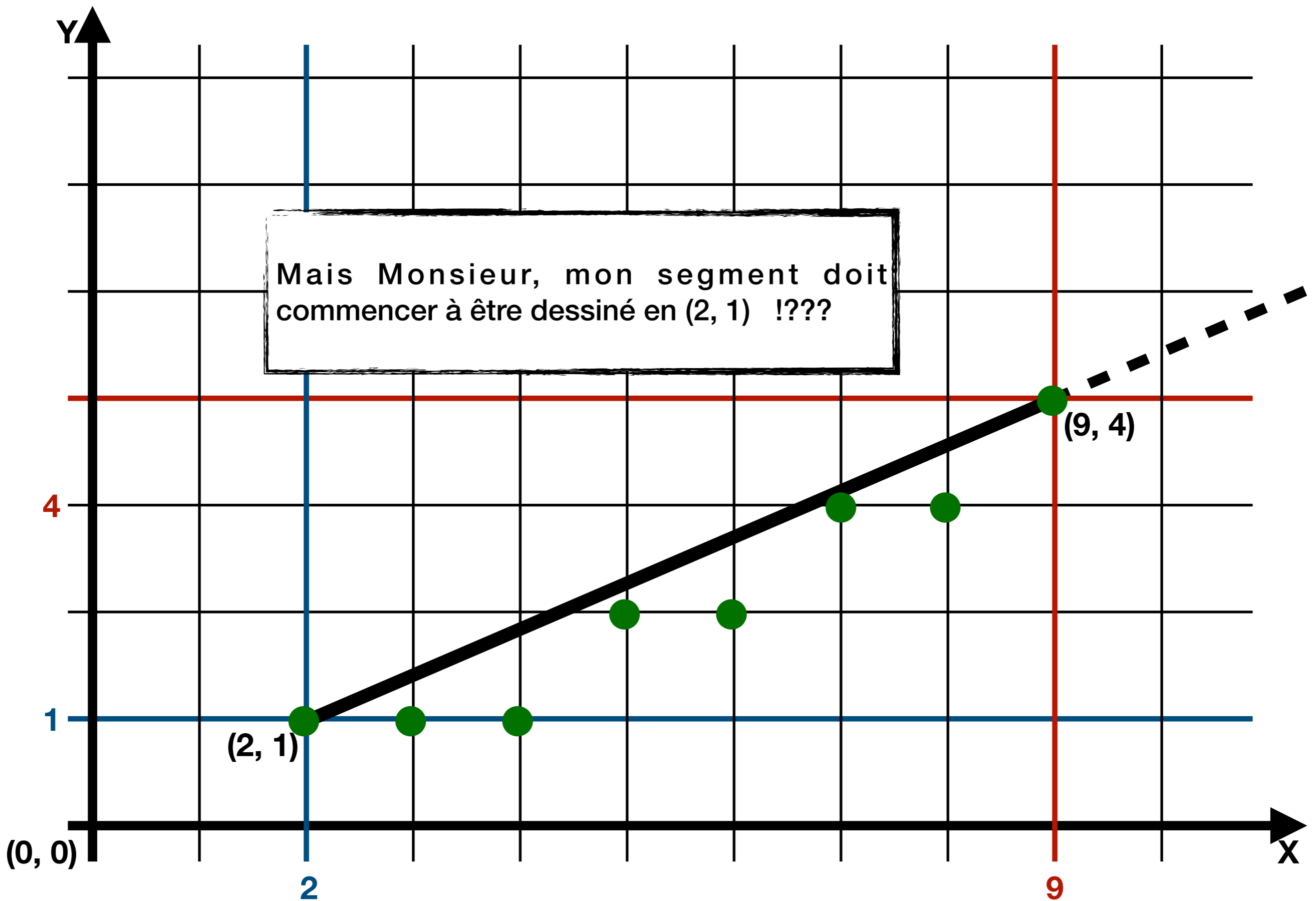
```

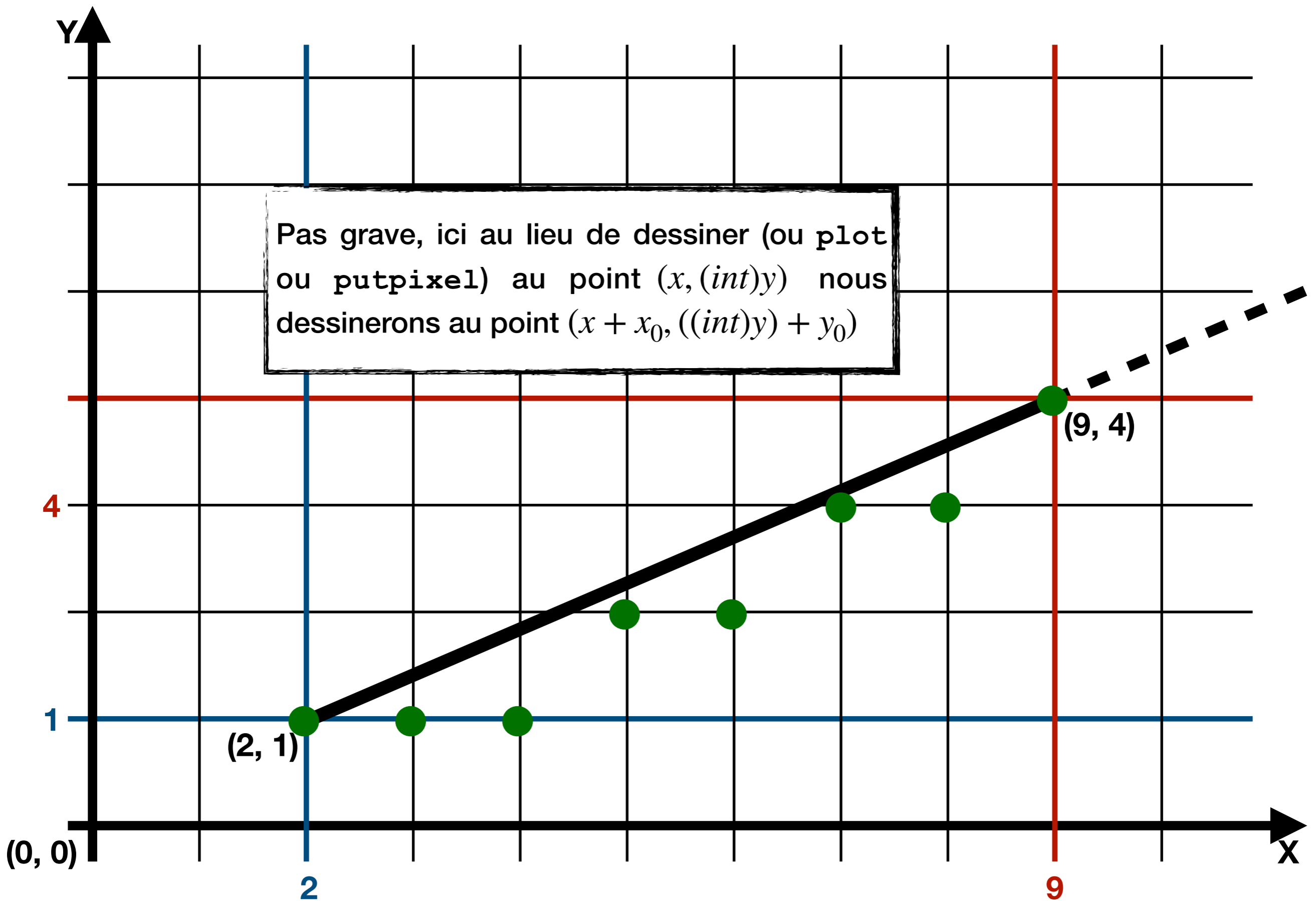
/* un début de fonction line
 * attention à remplacer <type> par le type utilisé pour un pixel
 * par exemple int32_t
 */
void line(int x0, int y0, int x1, int y1, int W, <type *> image, <type> color) {
    int u = x1 - x0;
    int v = y1 - y0;
    float a = v / (float)u; /* des vérifications à faire avant ... */
    if(a <= 1.0f) { /* pas nécessairement suffisant ... */
        float y = 0.0f;
        for(int x = 0; x <= u; ++x) {
            image[((int)y) * W + x] =color; /* plot(x, (int)y) */
            y += a;
        }
    }
}

```

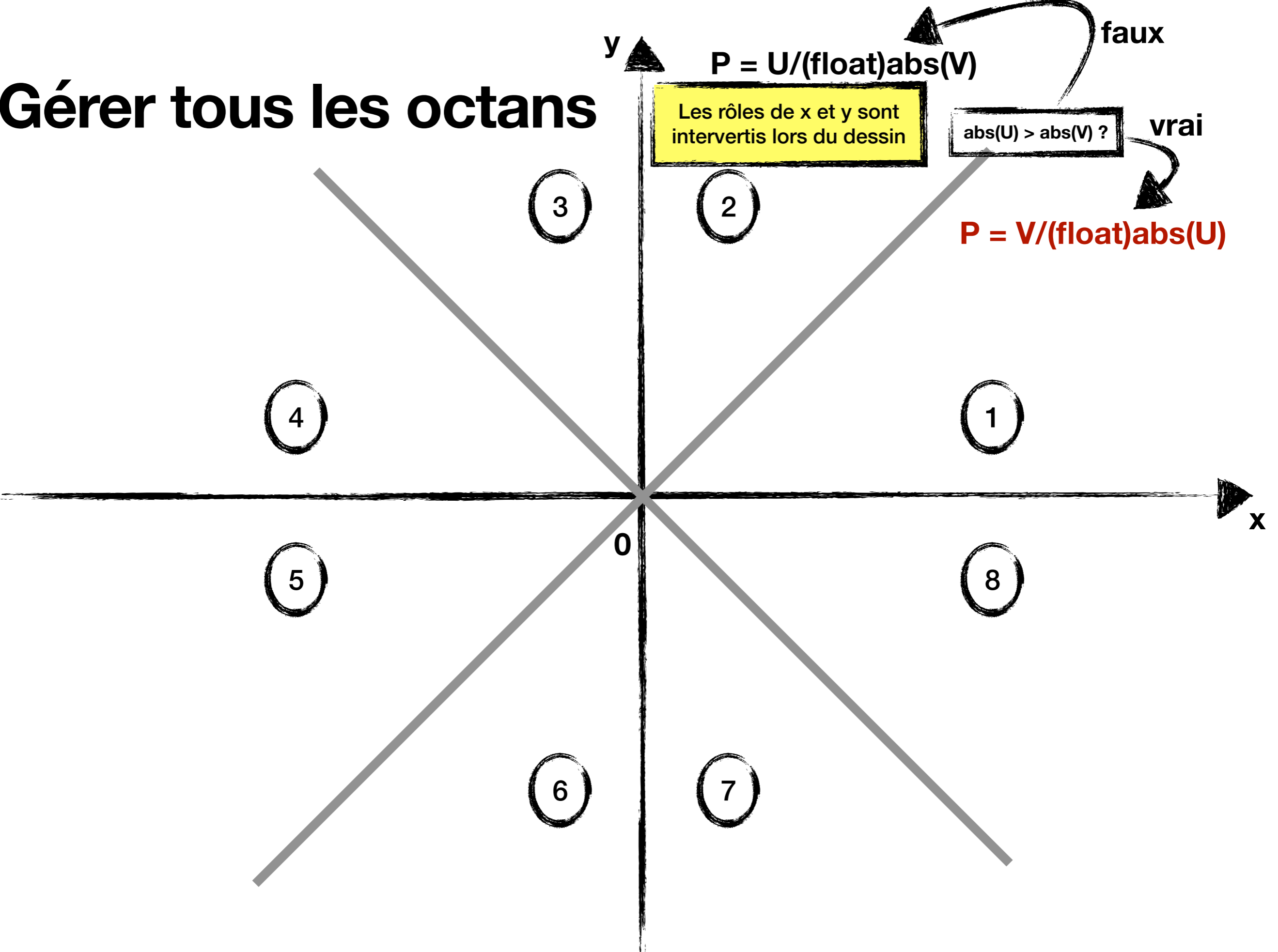








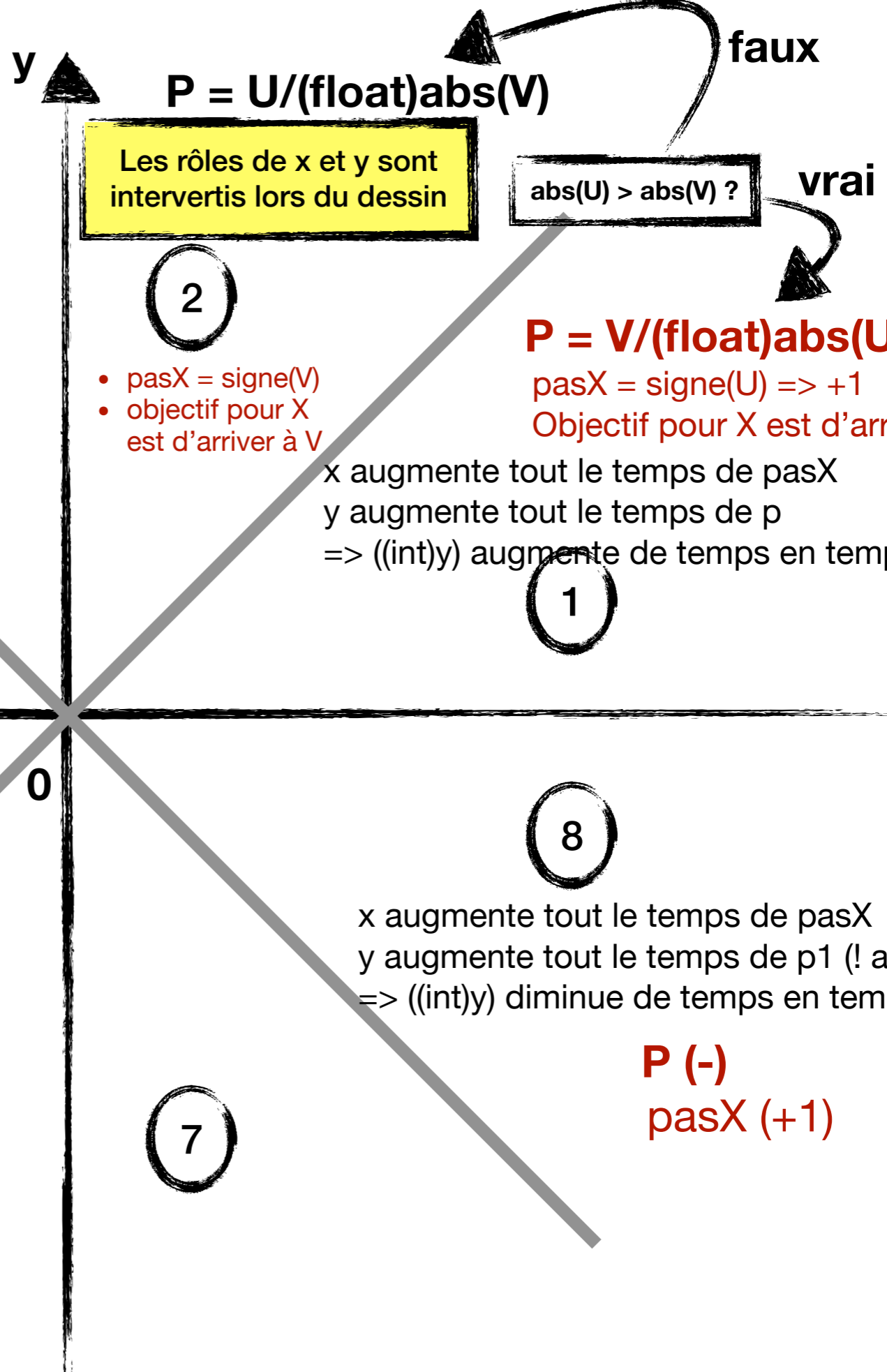
# Gérer tous les octans



# Et comment faire pour gérer tous les octans ?

- On gère par distinction entre premier et deuxième octan en utilisant les valeurs absolues de  $u$  et  $v$ .
- On a donc deux cas possibles :
$$\left\{ \begin{array}{l} |u| \geq |v| \rightarrow a = \frac{v}{|u|} \leq 1 \quad (1) \\ |u| < |v| \rightarrow a' = \frac{1}{a} = \frac{u}{|v|} \leq 1 \quad (2) \end{array} \right.$$
- On gère donc les octans manquants par symétrie, le pas entier de  $x$  devient négatif (et son extremum aussi) **et/ou** la pente devient négative pour gérer les segments sur <octan 4>, **<octan 5>**, ou <octan 8> à partir du cas (1). Le pas de  $y$  devient négatif (et son extremum aussi) **et/ou** l'inverse de la pente devient négative pour gérer les segments sur <octan 3>, **<octan 6>**, ou <octan 7> à partir du cas (2).

```
#define signe(a) ((a) < 0 ? -1 : 1)
```



Les rôles de x et y sont intervertis lors du dessin

abs(U) > abs(V) ?

faux

vrai

**P (+)**  
pasX (-1)

x augmente tout le temps de pasX  
y augmente tout le temps de p  
=> ((int)y) diminue de temps en temps de 1

- pasX = signe(V)
- objectif pour X est d'arriver à V

**P = V/(float)abs(U)**  
pasX = signe(U) => +1  
Objectif pour X est d'arriver à U

x augmente tout le temps de pasX  
y augmente tout le temps de p  
=> ((int)y) augmente de temps en temps de 1

4

1

5

x augmente tout le temps de pasX  
y augmente tout le temps de p (! au signe)  
=> ((int)y) diminue de temps en temps

8

x augmente tout le temps de pasX  
y augmente tout le temps de p1 (! au signe)  
=> ((int)y) diminue de temps en temps

**P (-)**  
pasX (-1)

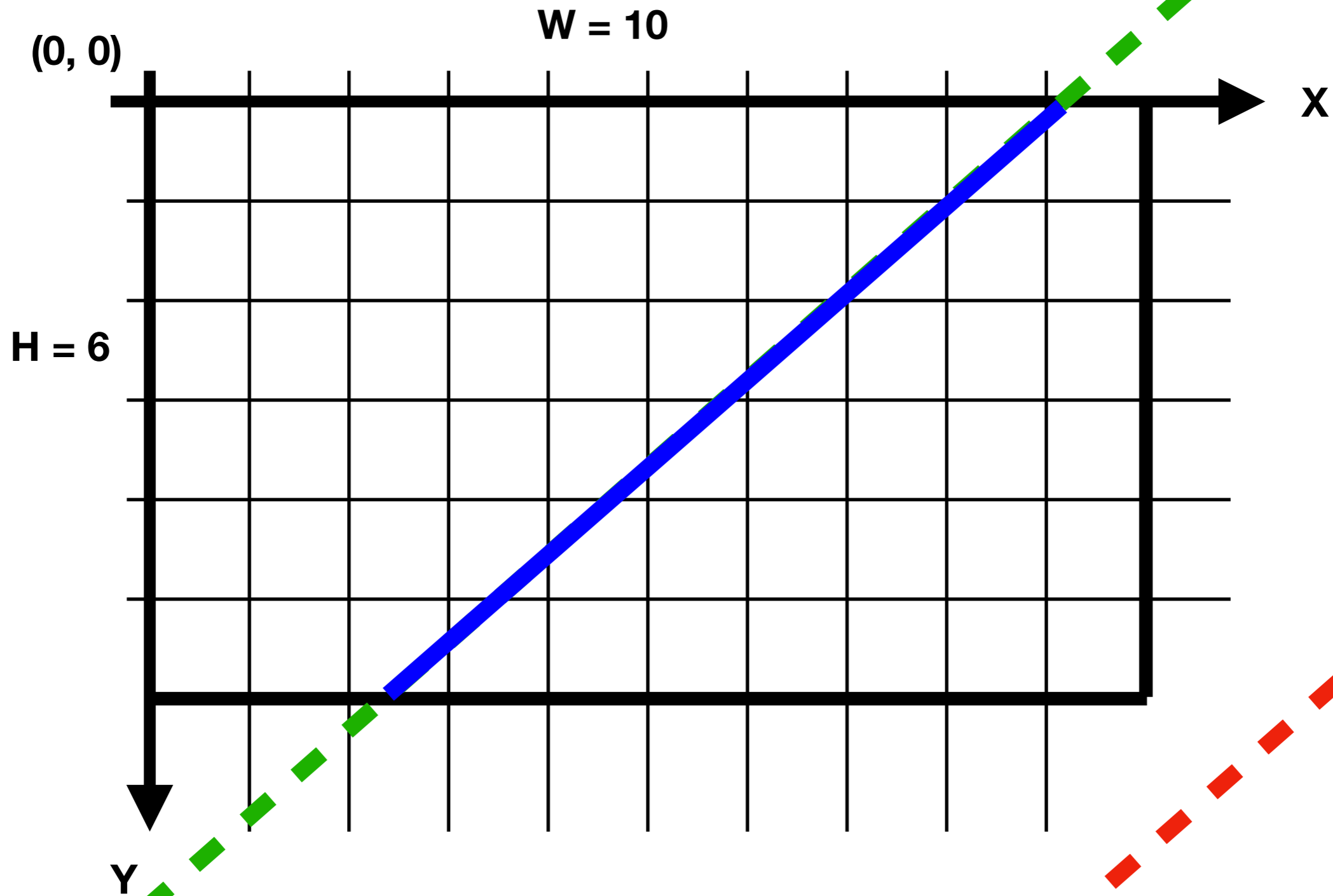
6

7

**P (-)**  
pasX (+1)



Penser aussi à éviter des situations problématiques (points en dehors de l'image)



# Pratique (à finaliser et rendre sur Moodle)

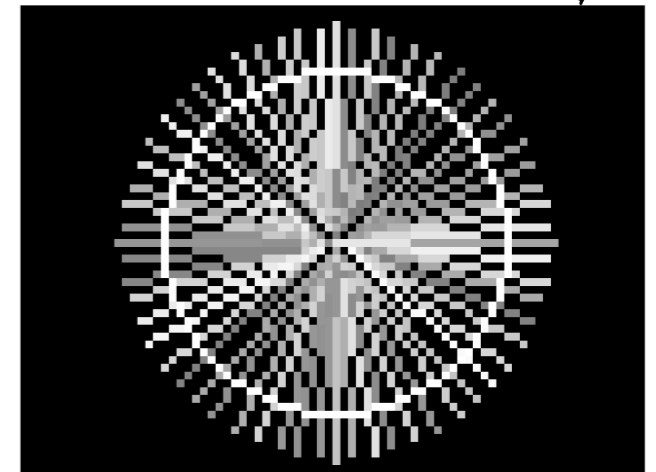
Téléchargez le code

[https://expreg.org/amsi/C/APG2223S1/code/00\\_basiques/drawLineInBMP-1.0.tgz](https://expreg.org/amsi/C/APG2223S1/code/00_basiques/drawLineInBMP-1.0.tgz)

PUIS (par étapes si besoin) réalisez ce qui attendu à l'exercice 4

```
/*
 * Exercice 1 : essayez de dessiner le segment (75, 10, 0, 50, 255),
 * ça ne fonctionne pas. Corriger drawLine.
 *
 * Exercice 2 : essayez de dessiner le segment (10, 10, 20, 50, 255),
 * ça ne fonctionne pas. Corriger drawLine.
 *
 * Exercice 3 : essayez de dessiner le segment (-10, -10, 120, 150,
 * 255), ça ne fonctionne pas. Corriger drawLine.
 *
 * Exercice 4 : dessiner (si ça marche) toutes les positions d'une
 * aiguille trotteuse en variant les intensités de gris. Vous pouvez
 * utiliser cos et sin (inclure math.h) tels que :
 *
 * for (float angle = 0.0f, rayon = 20.0f; angle < 2.0f * M_PI; angle += 0.5f)
 *     drawLine(image, W, W/2, H/2, W/2 + rayon * cos(angle), H/2 + rayon * sin(angle), rand()%256);
 *
 * Exercice 5 (BONUS) : l'équation d'un cercle est donnée par
 *
 * rayon * rayon = (x - x0) * (x - x0) + (y - y0) * (y - y0);
 * écrire drawCircle.
 */
```

Doit ressembler à ça



# Maintenant, on va plus Loin

## Analyse Discrète Différentielle (ADD)

### Bresenham 1965 => la droite :

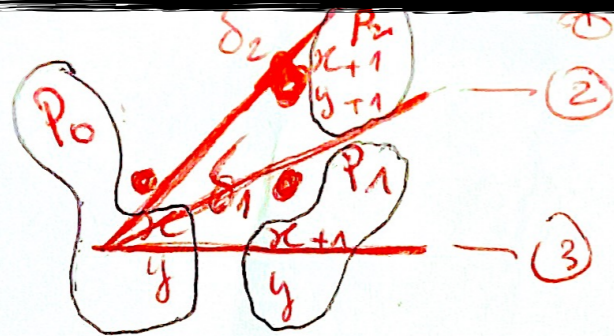
- [https://en.wikipedia.org/wiki/Bresenham%27s\\_line\\_algorithm](https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm)
- Mais aussi gl4dpLine (voir code source de GL4Dummies fichier gl4dp.c)

### Bresenham 1977 => le cercle :

- <http://public.callutheran.edu/~reinhart/CSC505/Week1/BresenhamCircles.pdf>
- Mais aussi gl4dpCircle (voir code source de GL4Dummies fichier gl4dp.c)



# Exposé en cours



- ①  $\delta_1 < 0$  &  $\delta_2 < 0 \rightarrow p_2$
- ②  $\delta_1 < 0$  &  $\delta_2 \neq 0 \rightarrow ?$
- ③  $\delta_1 > 0$  &  $\delta_2 > 0 \rightarrow p_1$



$\Rightarrow$

- ①  $\delta_1 + \delta_2 < 0 \rightarrow p_2$
- ③  $\delta_1 + \delta_2 > 0 \rightarrow p_1$
- ② ? mais est-ce que condition de ① et ② fonctionnellement  $\rightarrow$  oui!

$$\Delta = \delta_1 + \delta_2$$

$$\Delta = (uy - v(x+1)) + (u(y+1) - v(x+1))$$

eq:  $y - \frac{v}{u}x = 0$

ou  $uy - vx = 0$

$$\Delta = 2uy - 2vx + u - 2v$$

qd  $x=y=0$

$$\Delta_1 = u - 2v$$

$|x=1$

$$\Delta_0 = 0$$

$|x=0$

Comment varie  $\Delta$ ?

$x \rightarrow x+1$

\* si  $p_1$  choisi:  $\delta \Delta_1 = (2uy - 2v(x+1) + u - 2v) - (2uy - 2vx + u - 2v)$

$$\delta \Delta_1 = -2v \Rightarrow \text{inc } 1 = -2v$$

\* si  $p_2$  choisi:  $\delta \Delta_0 = (2u(y+1) - 2v(x+1) + u - 2v) - (2uy - 2vx + u - 2v)$

$$\delta \Delta_0 = 2u - 2v \Rightarrow \text{inc } 0 = 2u - 2v$$