
Informatique Graphique

Sylvain Contassot-Vivier

Phuc Ngo

Christian Minich

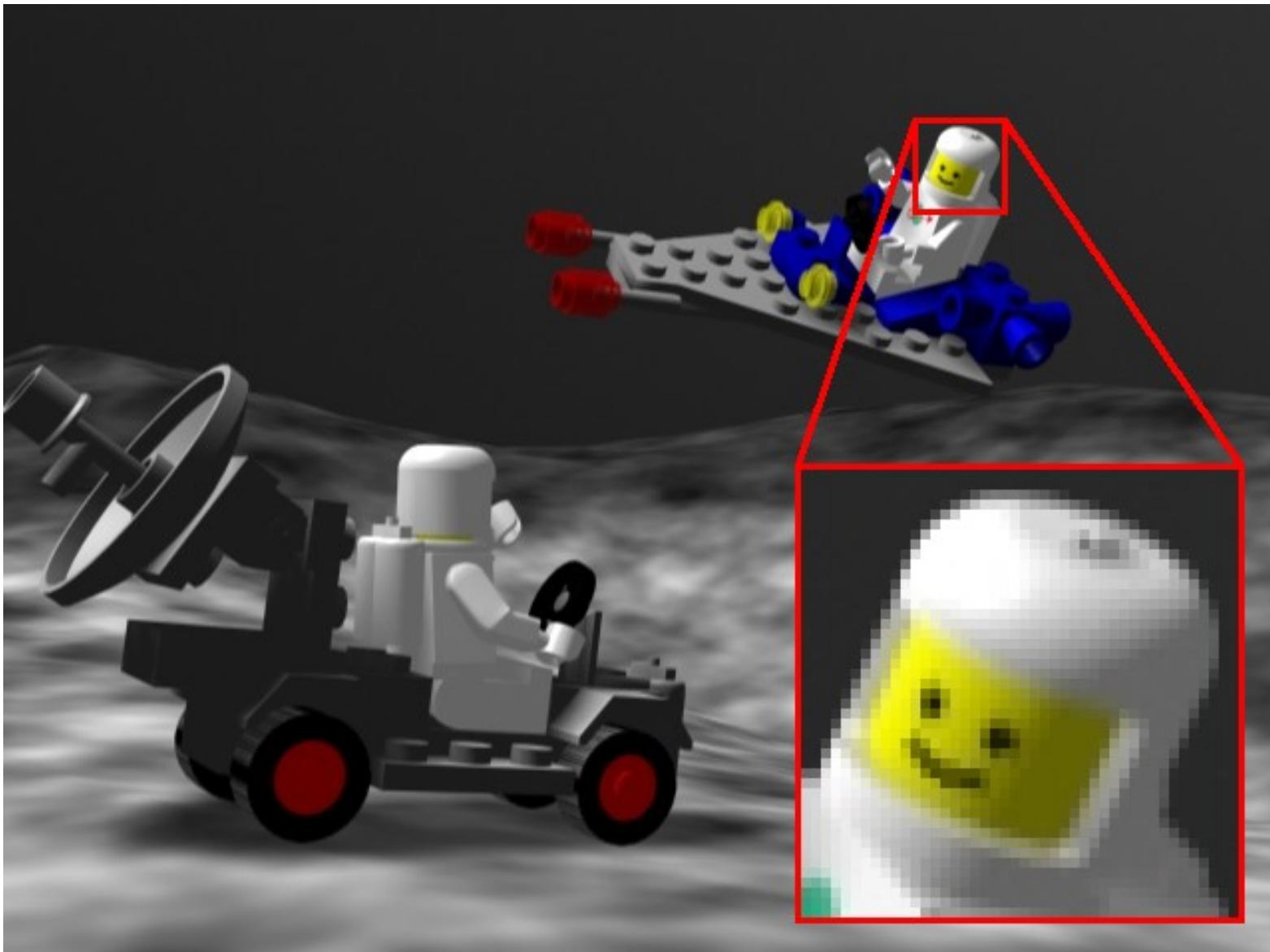
Image numérique

- Définition :
 - Contexte d'images numériques 2D
 - Écran subdivisé en grille régulière de *pixels*
- Types d'images numériques :
 - Deux grands types :
 - Images vectorielles
 - Images matricielles (bitmap)

Images matricielles

- Grille régulière de pixels :
 - Couleur varie d'un pixel à l'autre
 - Directement affichable sur les écrans
 - Encombrement indépendant du nombre d'objets dans l'image (mais de la taille de la grille)
 - Limitation du zoom
 - Pas de séparation sémantique des objets
 - Plutôt adaptée aux images naturelles :
 - Photos, dessins,...

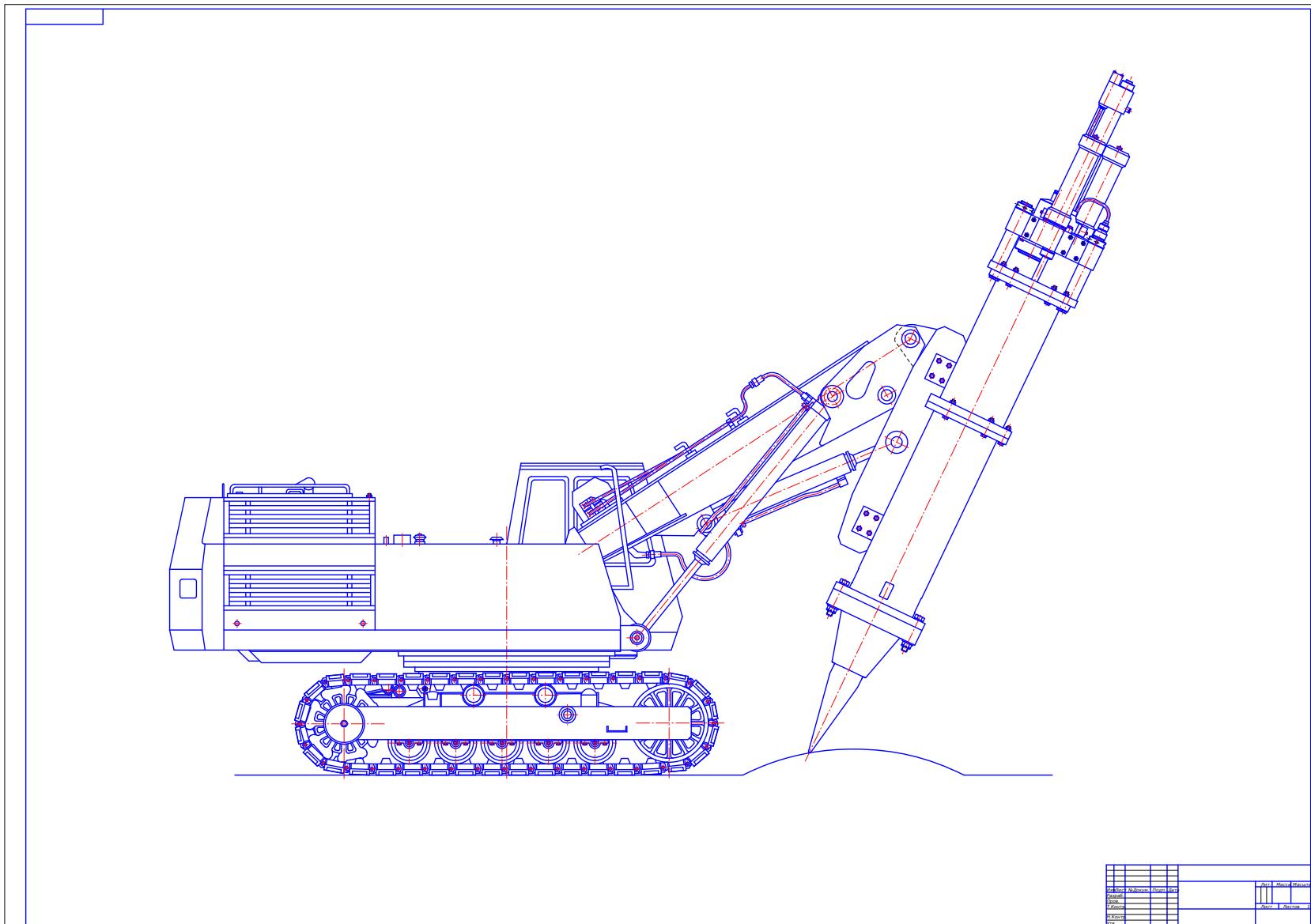
Image numérique 2D matricielle



Images vectorielles

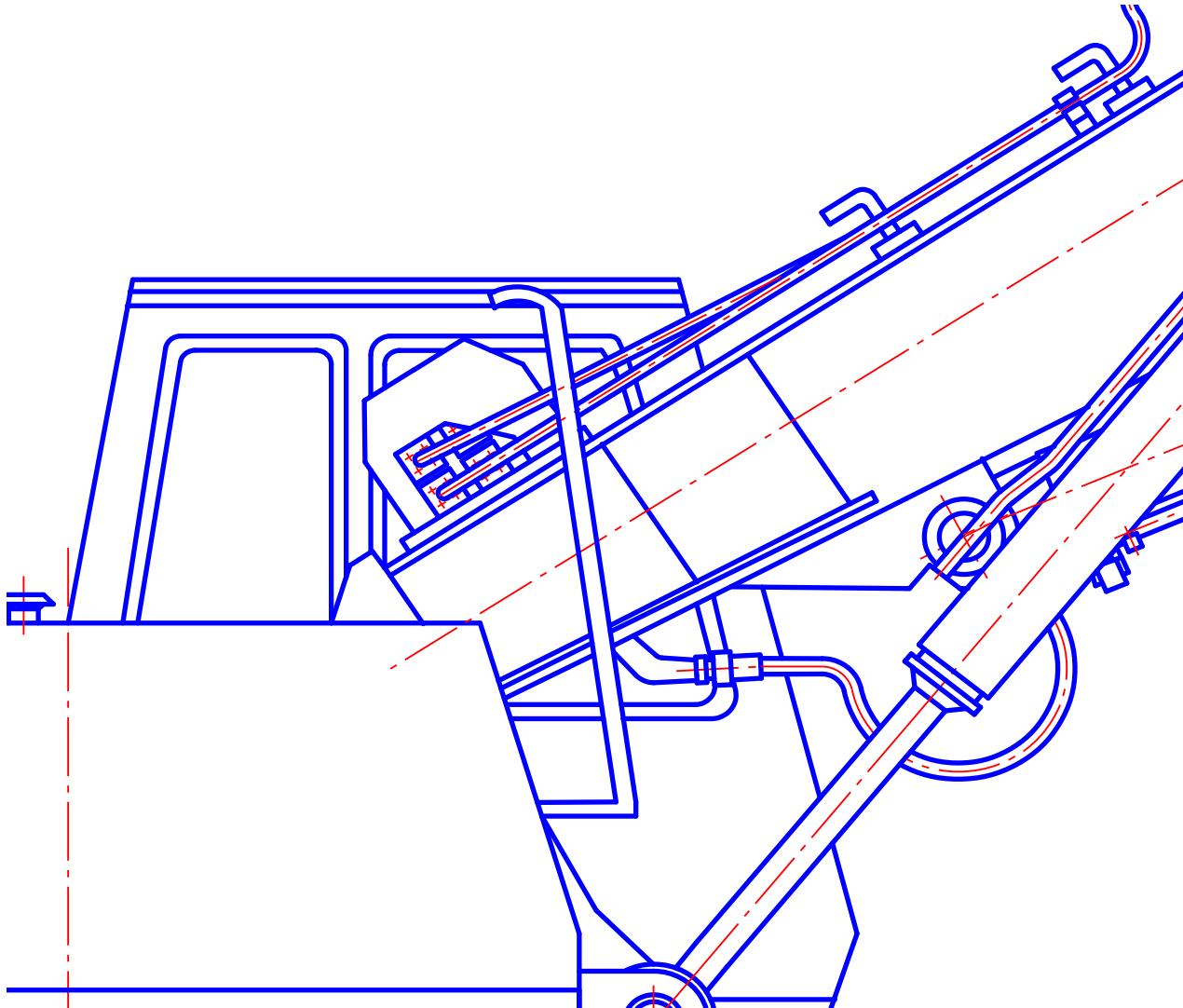
- Image composée d'une liste d'objets définis mathématiquement :
 - Lignes, rectangles, cercles,...
 - Peu encombrant si peu d'objets dans l'image mais peut devenir très encombrant si beaucoup d'objets
 - Précision illimitée (zoom arbitraire)
 - Difficulté à représenter des objets avec textures
 - Possibilité de modifier les objets indépendamment les uns des autres
 - Plutôt adaptée au dessin technique :
 - Plans, schémas,...

Image numérique 2D vectorielle



Source : Wikipedia → https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

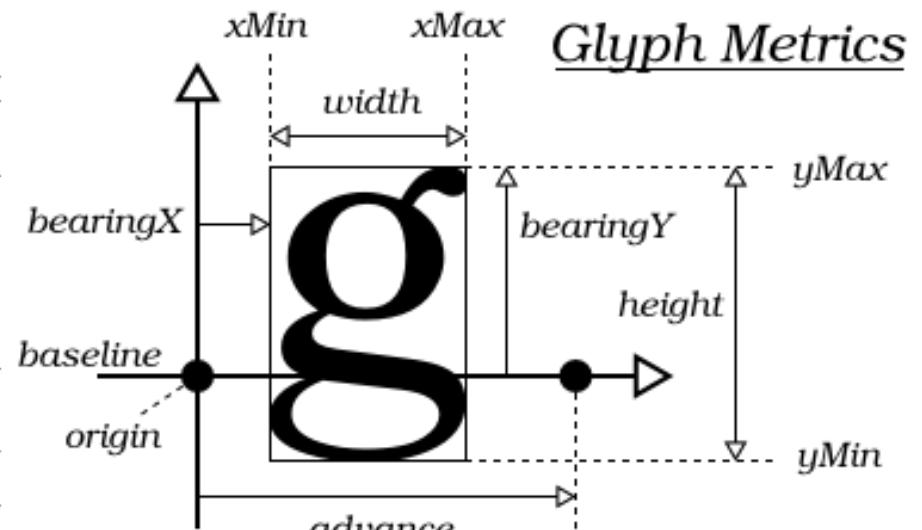
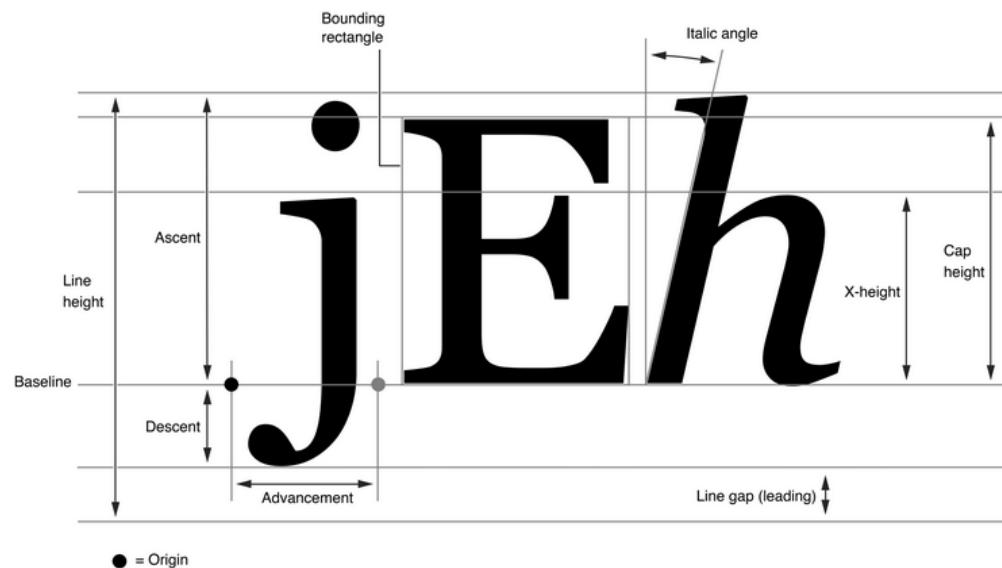
Agrandissement vectoriel



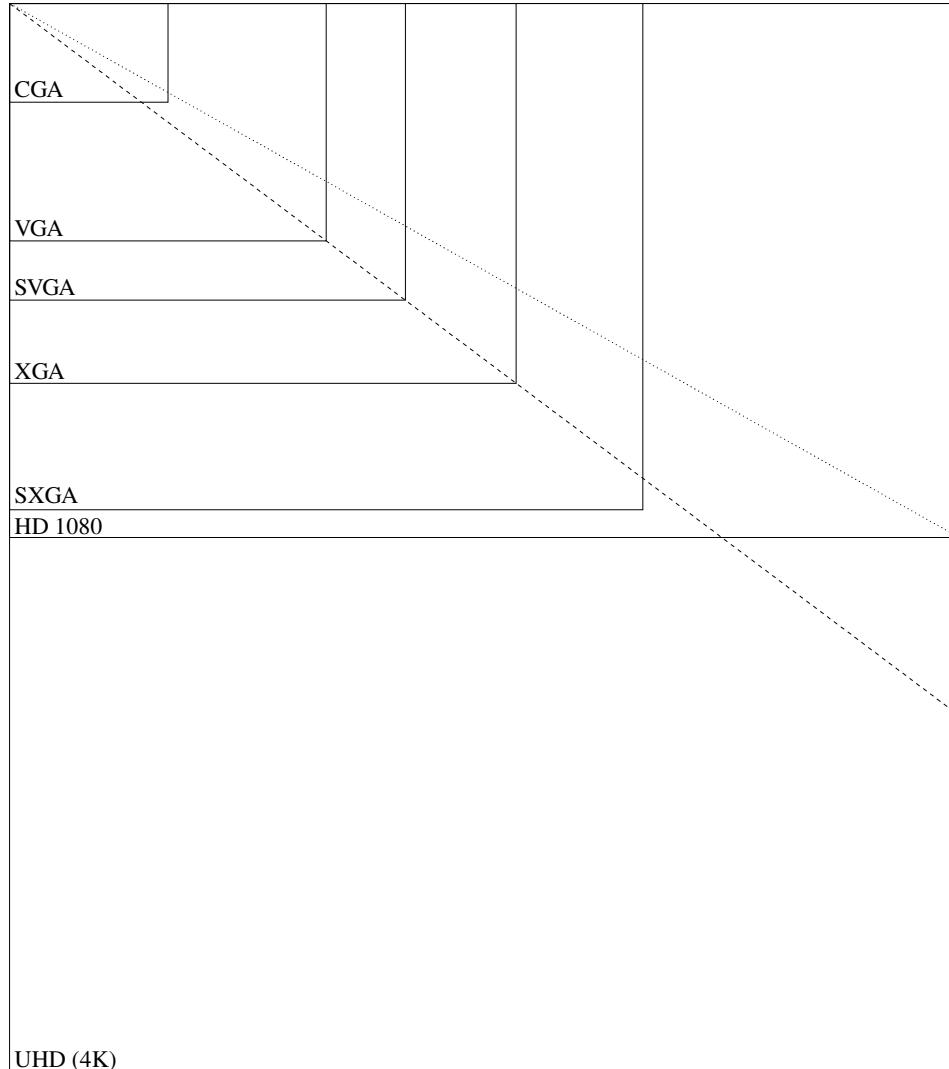
Source : Wikipedia → https://en.wikipedia.org/wiki/Scalable_Vector_Graphics

Police de caractères

- Police TrueType et FreeType



Principales résolutions standards

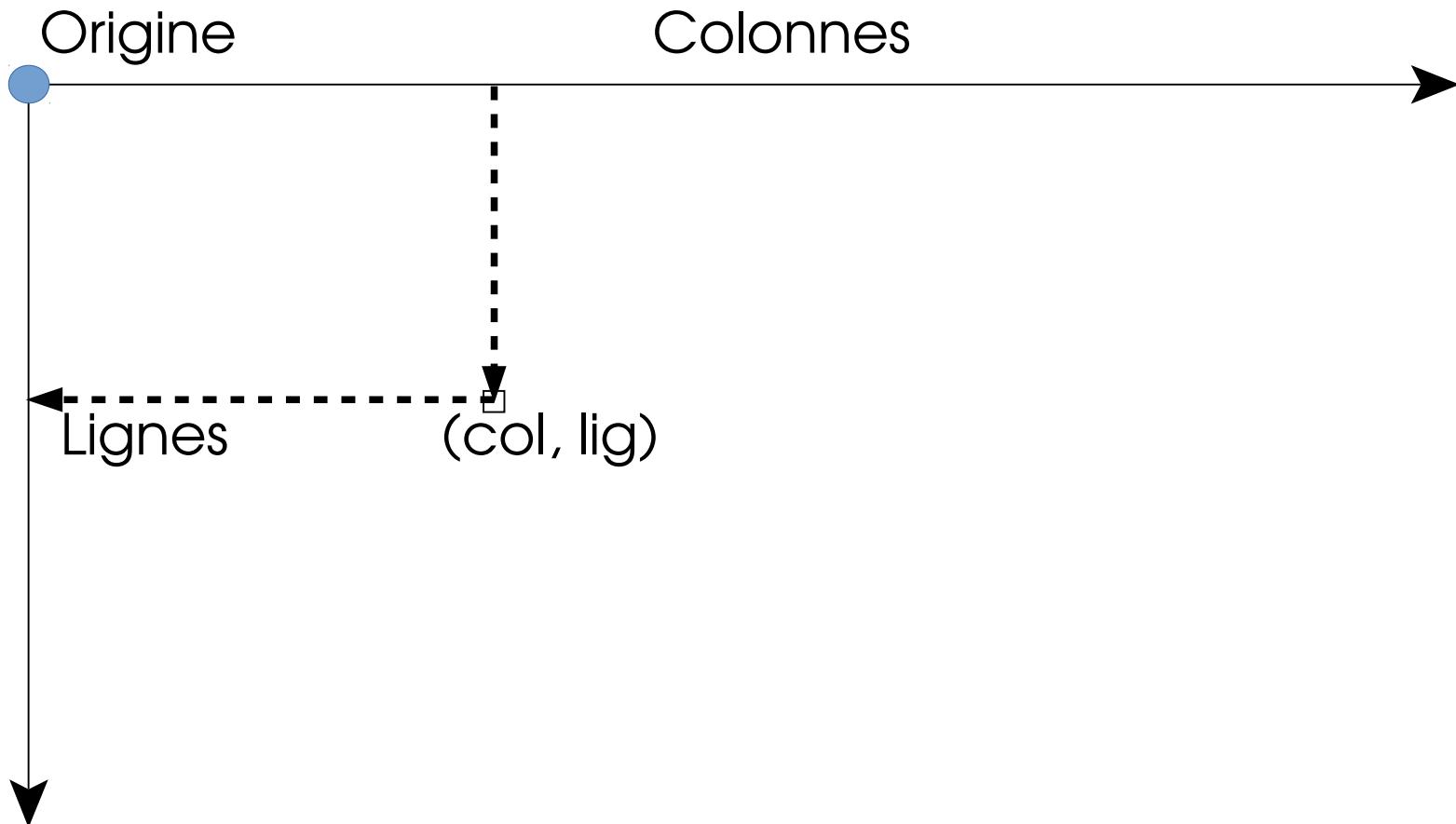


Nom	résolution	format (proportions)
CGA	320x200	(16 :10)
VGA	640x480	(4 :3)
PAL	768x576	(4 :3)
SVGA	800x600	(4 :3)
XGA	1024x768	(4 :3)
SXGA	1280x1024	(5 :4)
HD720	1280x720	(16 :9)
HD1080	1920x1080	(16 :9)
UHD (4K)	3840x2160	(16 :9)
UHD (8K)	7680x4320	(16 :9)

Format 4:3

Format 16:9

Repère de l'image



Quantification et codages binaires



1 bit par pixel



2 bits par pixel



4 bits par pixel



8 bits par pixel

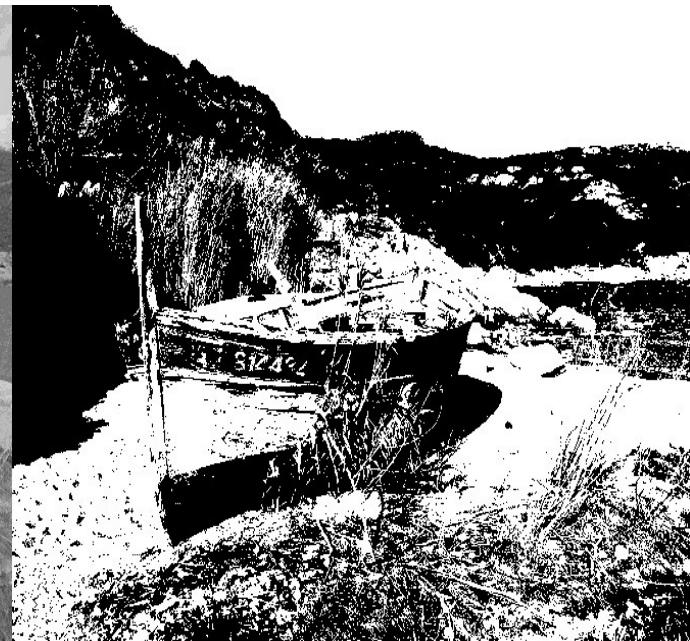
Exemple visuel en niveaux de gris



8 bits / pixel



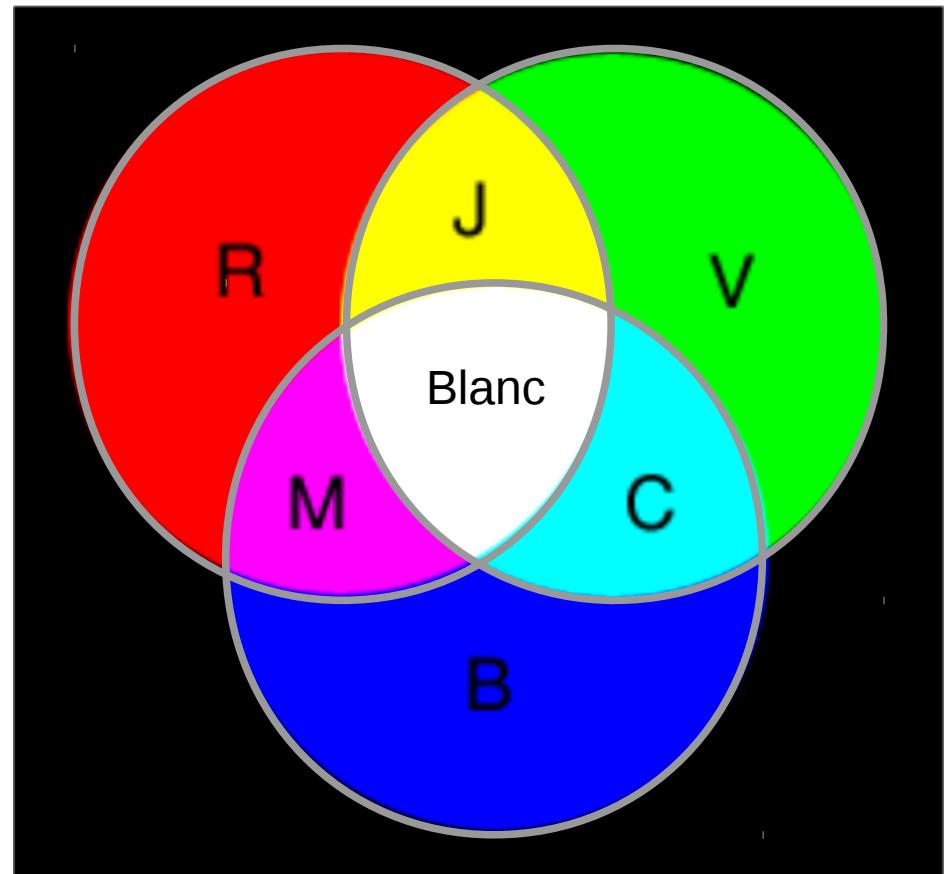
4 bits / pixel



1 bit / pixel

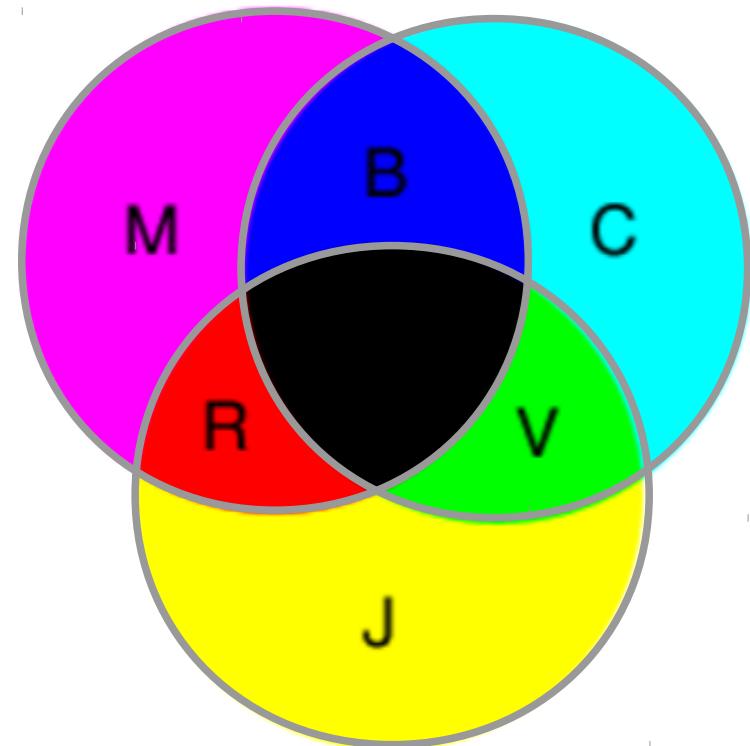
Modèle additif

- Couleurs primaires :
 - Rouge
 - Vert
 - Bleu
- Couleurs secondaires :
 - Jaune
 - Cyan
 - Magenta
- Relations :
 - $R+V = J$
 - $R+B = M$
 - $V+B = C$
 - $R+V+B = Blanc$

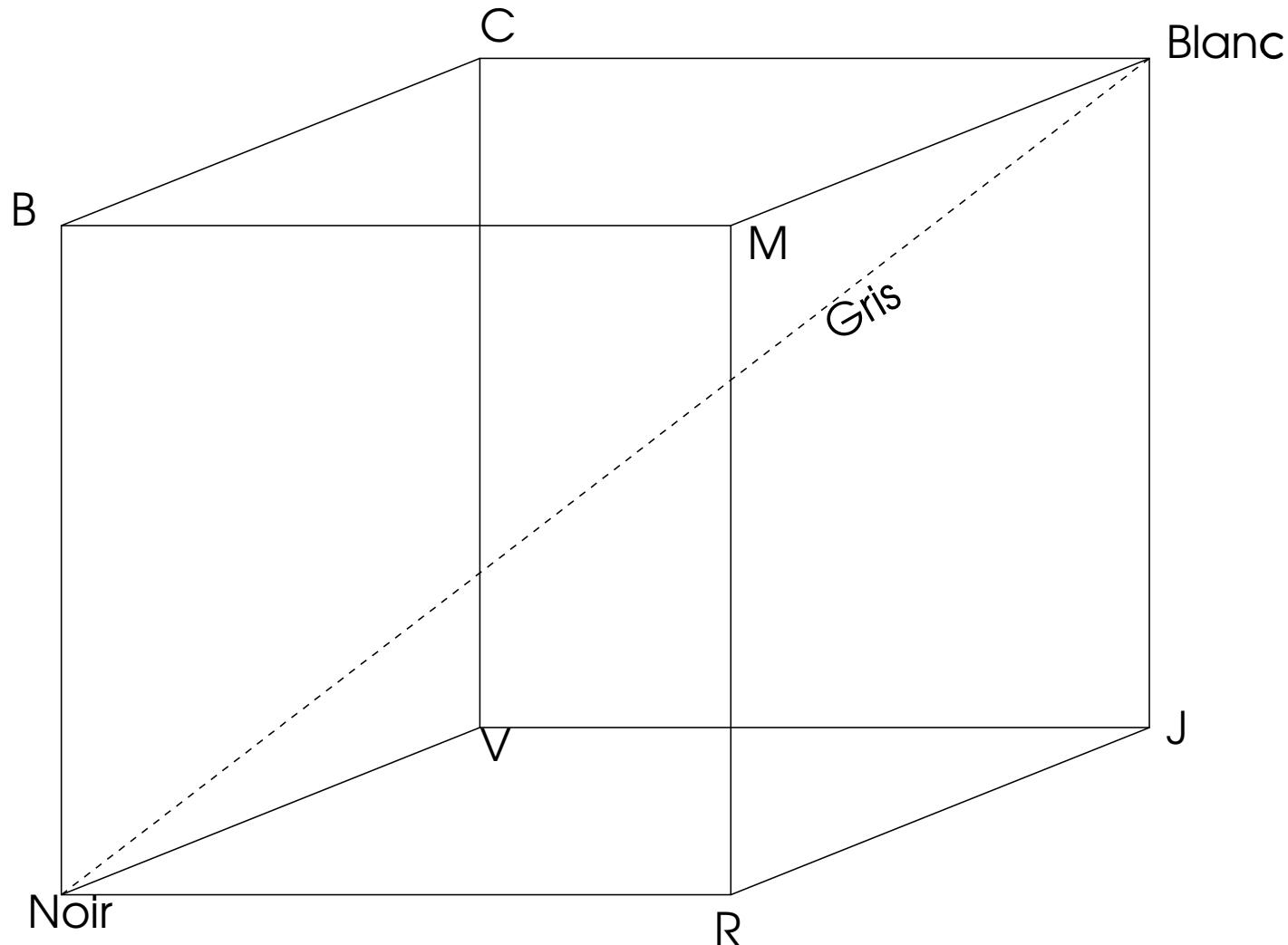


Modèle soustractif

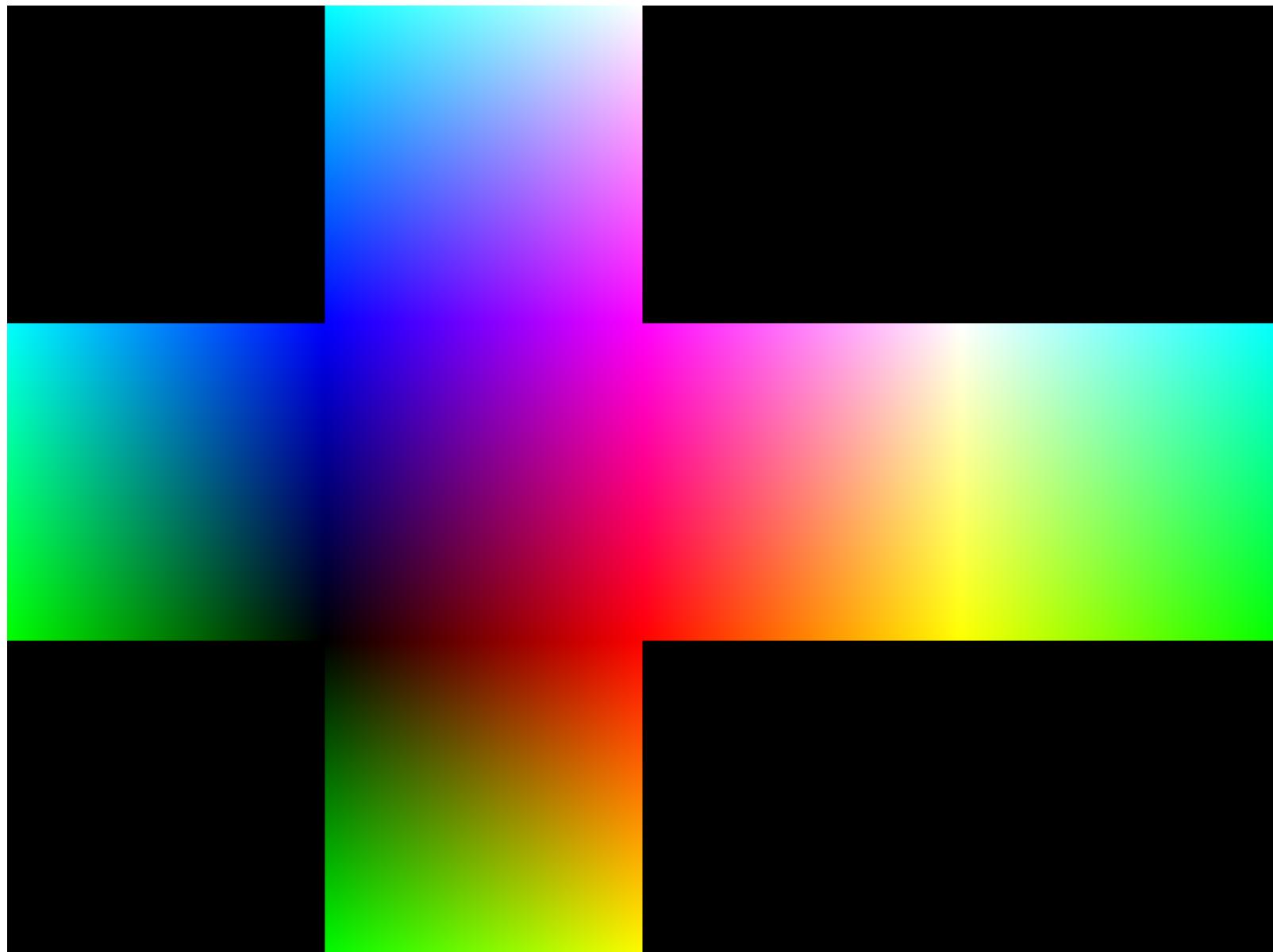
- Couleurs primaires :
 - Jaune
 - Cyan
 - Magenta
- Couleurs secondaires :
 - Rouge
 - Vert
 - Bleu
- Relations :
 - Blanc - (J+C) = V
 - Blanc - (J+M) = R
 - Blanc – (C+M) = B
 - Blanc – (J+C+M) = Noir



Cube des couleurs RVB



Cube déplié



Structure de données

- Définition du type Couleur :

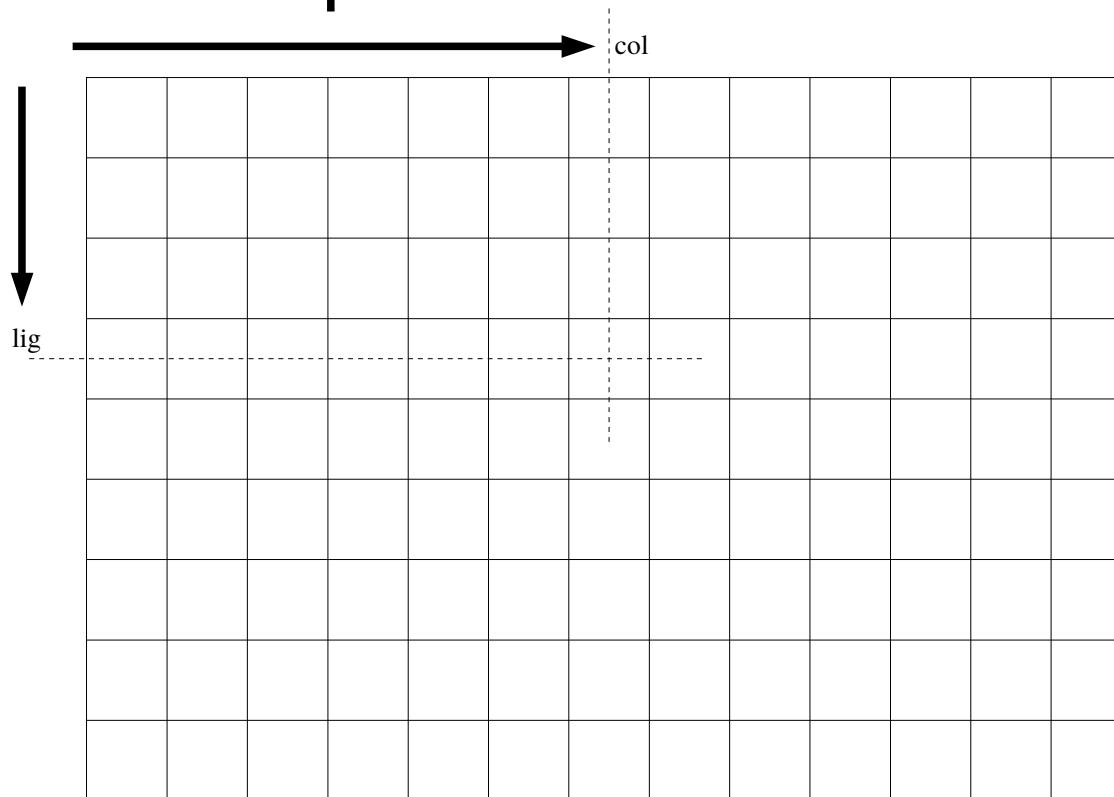
```
Couleur = structure
    R : entier // Composante rouge
    V : entier // Composante verte
    B : entier // Composante bleue
```

- Fonction de coloriage d'un pixel :

```
fonction ColoriePixel(col : entier, lig : entier, coul : Couleur) : Vide
// Colorie le pixel à la position (col,lig) dans l'image avec la couleur coul
```

Position d'un pixel

- La position d'un pixel est définie en son centre

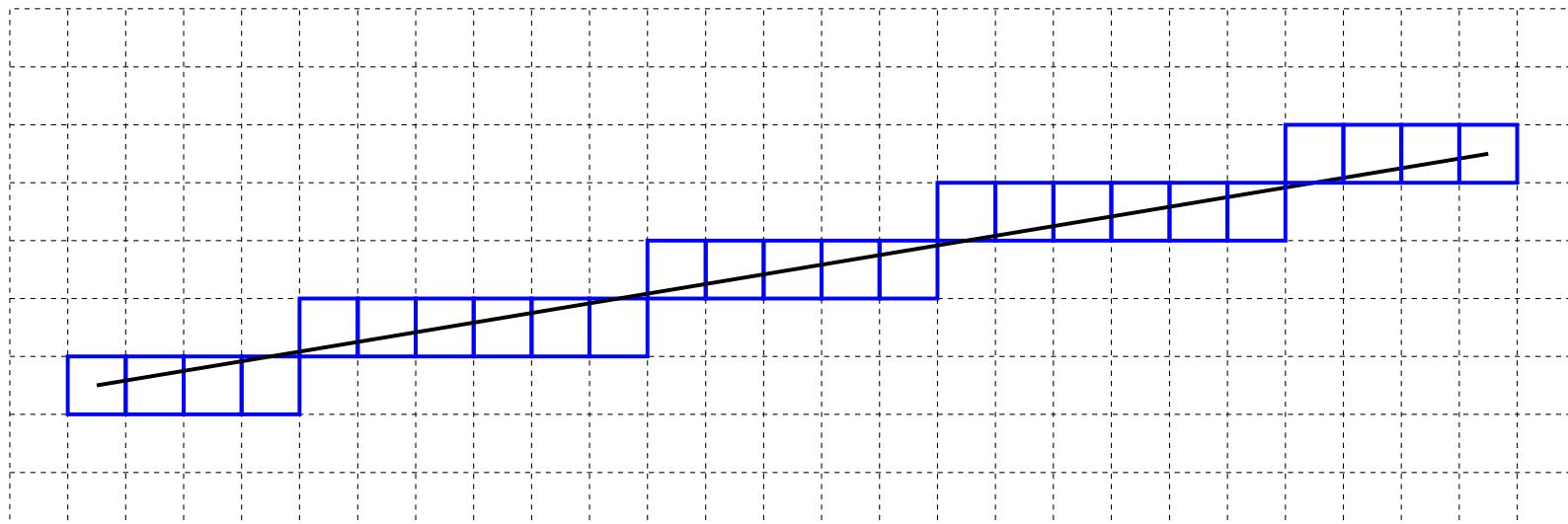


- Structure d'un point de l'image :

```
PointImage = structure
    col : entier // Colonne du point dans l'image
    lig : entier // Ligne du point dans l'image
```

Tracé de segment dans l'image

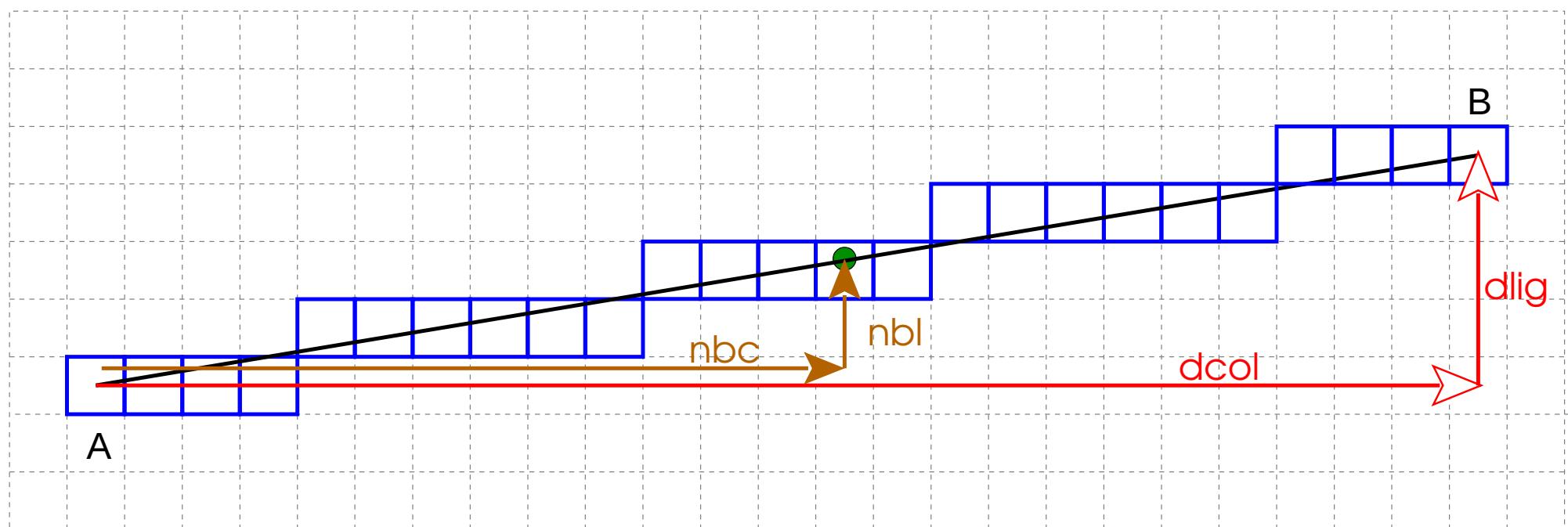
- On donne deux points A et B dans l'image :
 - $A = (A_c, A_l)$ et $B = (B_c, B_l)$
- On souhaite tracer le segment entre ces deux points dans l'image en respectant les règles suivantes :
 - Colorier uniquement des pixels traversés par le segment
 - Pas de trou dans le segment (assurer la continuité)
 - Pas plus de 2 voisins par pixel du segment



Pixels à colorier

Première version (analytique)

- On utilise l'équation de la droite :
 - On calcule la pente : $dlig / dcol$
 - À partir de l'extrémité gauche, on parcourt les colonnes et on calcule les lignes
 - On prend comme référence le point de départ



Algorithme analytique simple

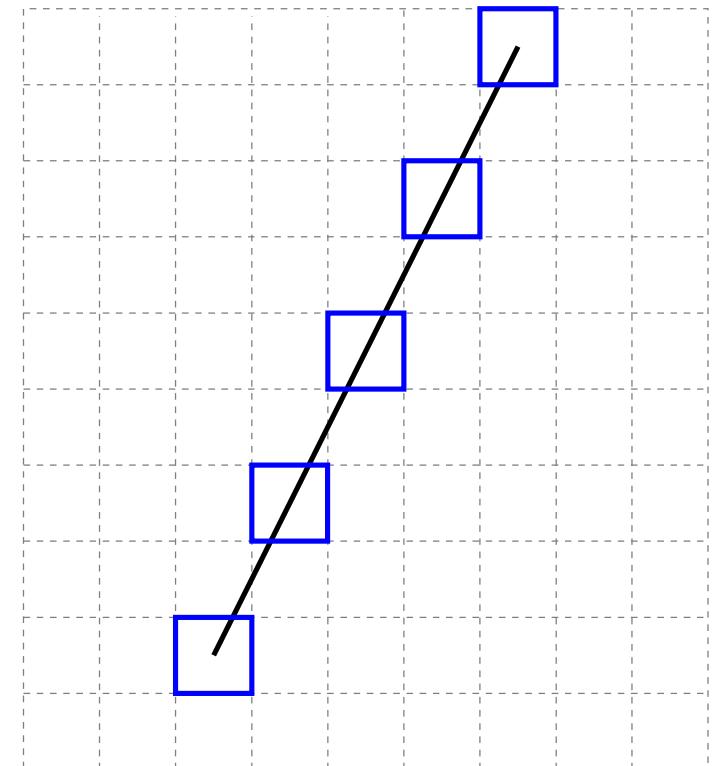
- Version considérant que le point A est l'extrémité gauche du segment

```
fonction DessineSegmentImage(A : PointImage, B : PointImage, coul : Couleur) : Vide
// Trace le segment entre les pixels A et B avec la couleur coul

dlig ← B.lig - A.lig // Variation en lignes
dcol ← B.col - A.col // Variation en colonnes
pente ← dlig / dcol // Pente de la droite AB
// Parcours des colonnes et calcul des lignes
pour col de A.col à B.col faire
    lig ← arrondi((col - A.col) * pente) + A.lig
    ColoriePixel(col, lig, coul)
fpour
```

Problèmes de la version analytique

- Modèle utilisé non général :
 - Segment vertical → pente infinie
- Nécessaire de trouver le point à gauche
 - Déterminer l'ordre de A et B
- Calculs inutiles si segment horizontal
 - Mauvaise performance
- Mais surtout, il peut y avoir des trous !!
 - Lorsque la pente $> 45^\circ$
 - Il faudrait colorier plusieurs pixels pour chaque colonne
 - Alors que l'on n'en colorie qu'un seul

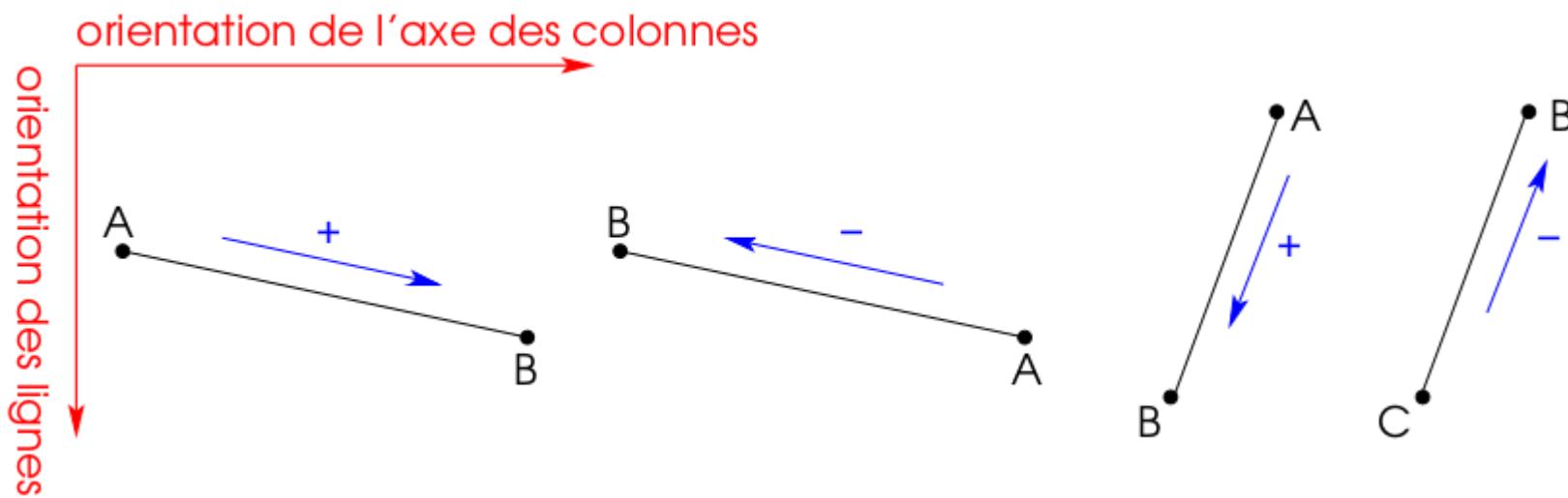


Solution générale

- On va séparer le problème en différents cas :
 - Segment horizontal :
 - Parcours des colonnes sur une même ligne
 - Segment vertical :
 - Parcours des lignes sur une même colonne
 - Segment de pente $\leq 45^\circ$:
 - Algo précédent
 - Parcours des colonnes et calcul des lignes
 - Segment de pente $> 45^\circ$:
 - Symétrie de l'algo précédent par échange des lignes et colonnes
 - Parcours des lignes et calcul des colonnes
 - Ordre des extrémités : segment [AB]
 - Parcours de A à B quelles que soient leurs positions respectives

Sens de parcours

- Dépend de la position des deux extrémités



Algorithme analytique général

```

fonction DessineSegmentImage(A : PointImage, B : PointImage, coul : Couleur) : Vide
// Trace le segment entre les pixels A et B avec la couleur coul

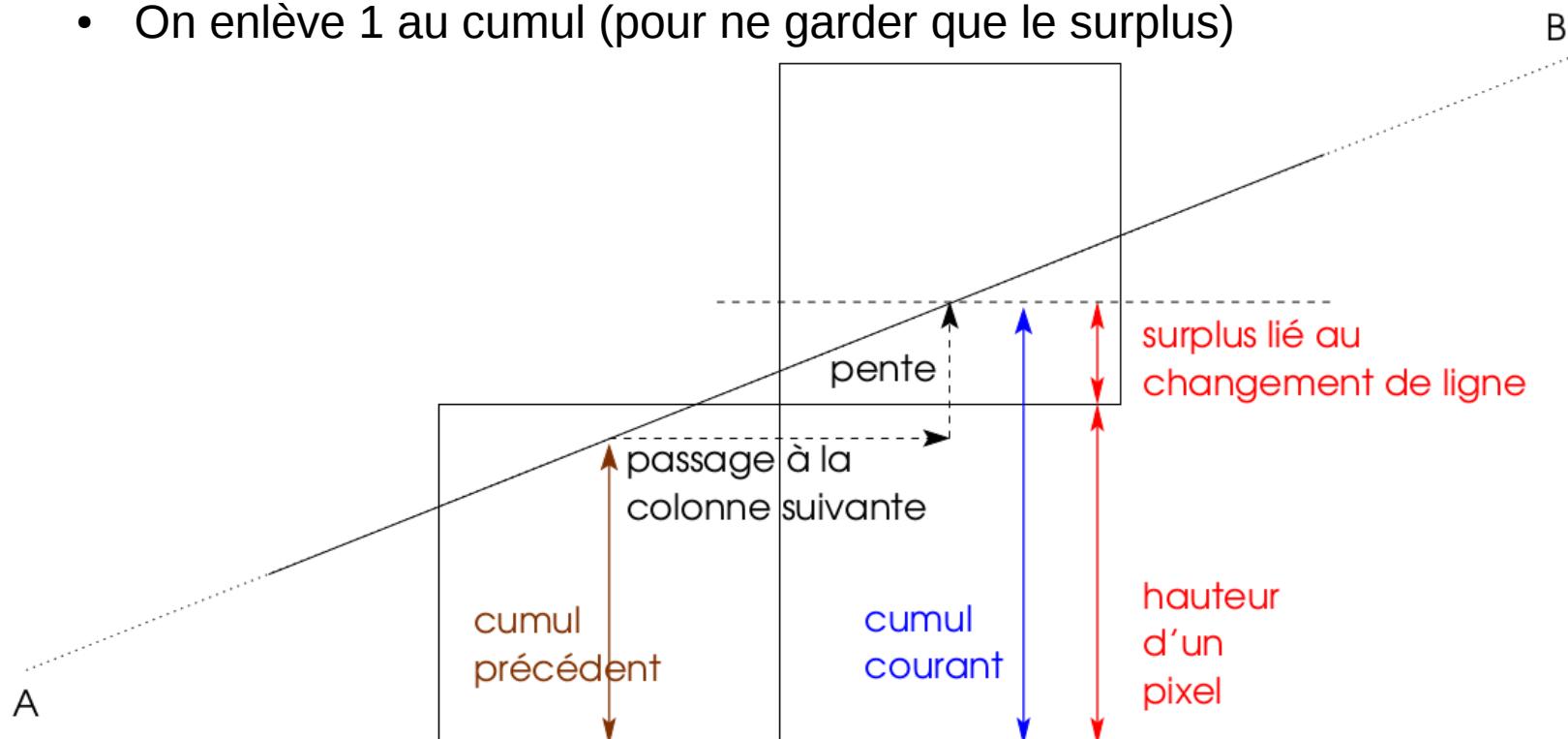
    dlig ← B.lig - A.lig // Variation en lignes
    dcol ← B.col - A.col // Variation en colonnes
    col ← A.col           // On commence le parcours du segment au point A
    lig ← A.lig
    sens ← 1              // Sens initial de parcours des lignes ou colonnes

    si dlig = 0 alors      // Segment horizontal
        si dcol < 0 alors    // Mise à jour du sens de parcours
            sens ← -1
        fsi
        tant que col ≠ B.col + sens faire
            ColoriePixel(col, lig, coul)
            col ← col + sens
        ftant
    sinon
        si dcol = 0 alors    // Segment vertical
            ... // Symétrique au segment horizontal en permutant lignes et colonnes
        sinon                  // Cas généraux
            si abs(dcol) ≥ abs(dlig) alors // Parcours des colonnes
                si dcol < 0 alors // Mise à jour du sens de parcours
                    sens ← -1
                fsi
                pente ← dlig / dcol // ATTENTION : la pente doit être calculée en réel
                tant que col ≠ B.col + sens faire
                    ColoriePixel(col, lig, coul)
                    col ← col + sens
                    lig ← arrondi((col - A.col) * pente) + A.lig // Calcul réel arrondi ↴
                                         ↴ au plus près
                ftant
            sinon // Parcours des lignes
                ... // Symétrique au parcours des colonnes
            fsi
        fsi
    fsi

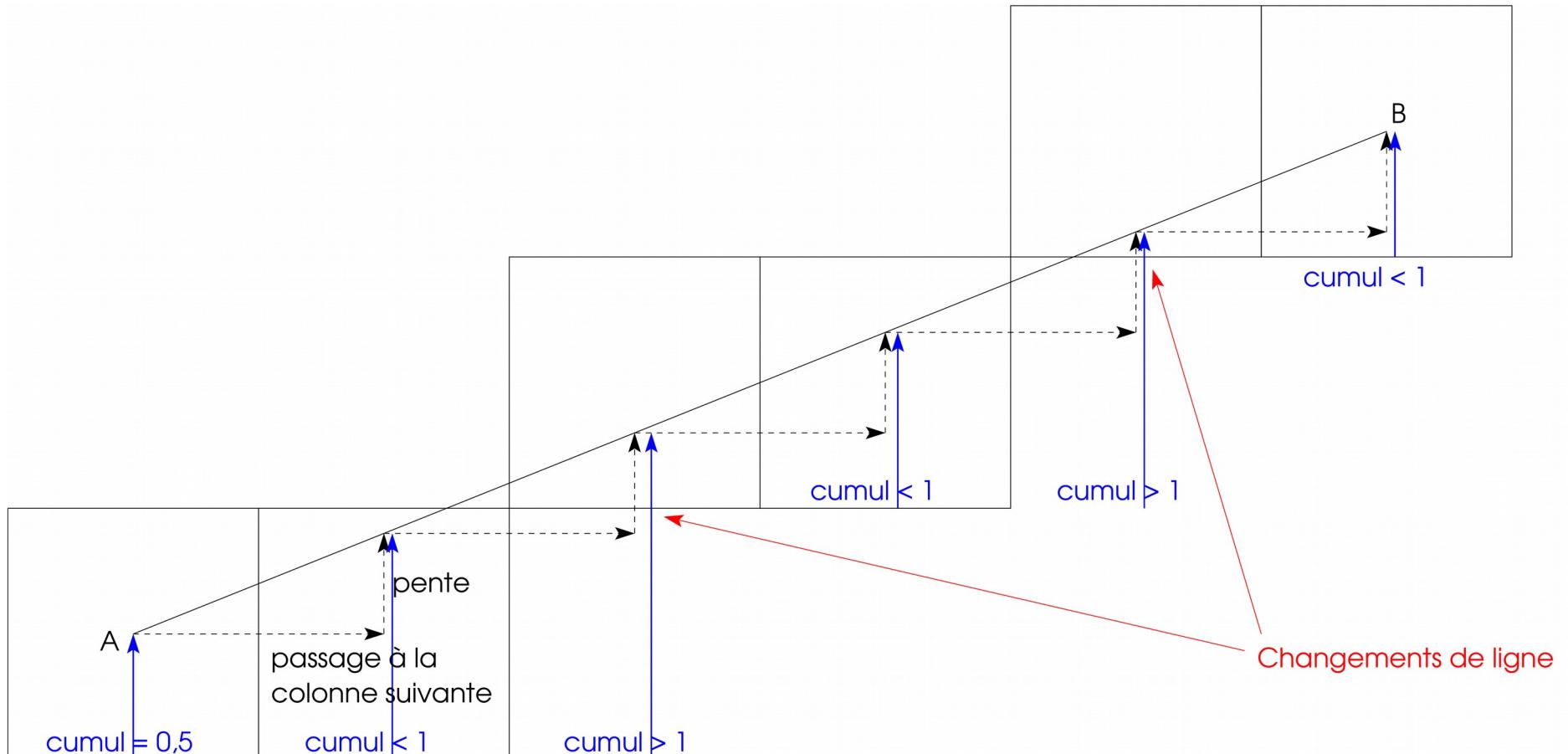
```

Version incrémentale

- **Principe :** déduire le y courant du y du pixel précédent
- Lors des déplacements unitaires selon l'axe de parcours :
 - On se déplace de la pente (<1) selon l'autre axe
 - Lorsque le cumul des déplacements dépasse 1 :
 - On change de position courante (± 1) sur le second axe
 - On enlève 1 au cumul (pour ne garder que le surplus)



Exemple de parcours



Algorithme incrémental

```

fonction DessineSegmentImage(A : PointImage, B : PointImage, coul : Couleur) : Vide
// Trace le segment entre les pixels A et B avec la couleur coul

    dlig ← B.lig - A.lig // Variation en lignes
    dcol ← B.col - A.col // Variation en colonnes
    col ← A.col           // On commence toujours au point A
    lig ← A.lig
    sensLig ← 1            // Sens initial de parcours des lignes
    sensCol ← 1            // Sens initial de parcours des colonnes
    cumul ← 0,5             // Le cumul commence au centre du pixel

    si dcol < 0 alors      // Mise à jour du sens de parcours des colonnes
        sensCol ← -1
    fsi
    si dlig < 0 alors      // Mise à jour du sens de parcours des lignes
        sensLig ← -1
    fsi
    si dlig = 0 alors      // Segment horizontal
        ... // Similaire à la version précédente
    sinon
        si dcol = 0 alors    // Segment vertical
            ... // Symétrique au segment horizontal en permutant lignes et colonnes
        sinon                  // Cas généraux
            si abs(dcol) ≥ abs(dlig) alors // Parcours des colonnes
                pente ← abs(dlig / dcol) // ATTENTION : la pente doit être un réel
                tant que col ≠ B.col + sens faire
                    ColoriePixel(col, lig, coul)
                    col ← col + sensCol
                    cumul ← cumul + pente // Ajout de la pente au cumul
                    si cumul > 1 alors // Test de changement de ligne
                        lig ← lig + sensLig // Changement de ligne dans le sens qui convient
                        cumul ← cumul - 1,0 // Réduction pour ne garder que le surplus
                    fsi
                    ftant
                sinon // Parcours des lignes
                    ... // Symétrique au parcours des colonnes
                fsi
            fsi
    fsi

```

Version entière

- La version précédente supprime les \times MAIS
 - Elle utilise des calculs réels coûteux, que l'on peut éliminer
- Les seuls variables réelles sont le cumul et la pente
- La pente est réelle via la division par *dcol*
 - ⇒ Il suffit de multiplier l'échelle d'un pixel par *dcol* !
- L'initialisation du cumul à *dlig/2* pose aussi problème
 - ⇒ Il suffit de multiplier l'échelle d'un pixel par 2 !
- Au final, on multiplie l'échelle d'un pixel par $2 * \text{dcol}$
- On obtient un gain de performance non négligeable :
 - 20 % sur version C, 260 % sur version python

Version entière (Bresenham)

```

fonction DessineSegmentImage(A : PointImage, B : PointImage, coul : Couleur) : Vide
// Trace le segment entre les pixels A et B avec la couleur coul

    dlig ← B.lig - A.lig // Variation en lignes
    dcol ← B.col - A.col // Variation en colonnes
    absdcol ← abs(dcol) // Valeur absolue de dcol
    absdlig ← abs(dlig) // Valeur absolue de dlig
    col ← A.col           // On commence toujours au point A
    lig ← A.lig
    sensLig ← 1            // Sens initial de parcours des lignes
    sensCol ← 1            // Sens initial de parcours des colonnes

    si dcol < 0 alors      // Mise à jour du sens de parcours des colonnes
        sensCol ← -1
    fsi
    si dlig < 0 alors      // Mise à jour du sens de parcours des lignes
        sensLig ← -1
    fsi

    si absdcol ≥ absdlig alors          // Parcours des colonnes
        cumul ← absdcol                  // Départ au centre du pixel
        tant que col ≠ B.col + sensCol faire
            ColoriePixel(col, lig, coul)
            cumul ← cumul + 2 * absdlig   // Ajout du déplacement vertical équivalent à un déplacement horizontal d'un pixel
            si cumul ≥ 2 * absdcol alors // Test de changement de ligne
                lig ← lig + sensLig       // Changement de ligne
                cumul ← cumul - 2 * absdcol // Réduction pour ne garder que le surplus
            fsi
            col ← col + sensCol
        ftant
    sinon // Parcours des lignes
        ... // Symétrique au parcours des colonnes
    fsi

```

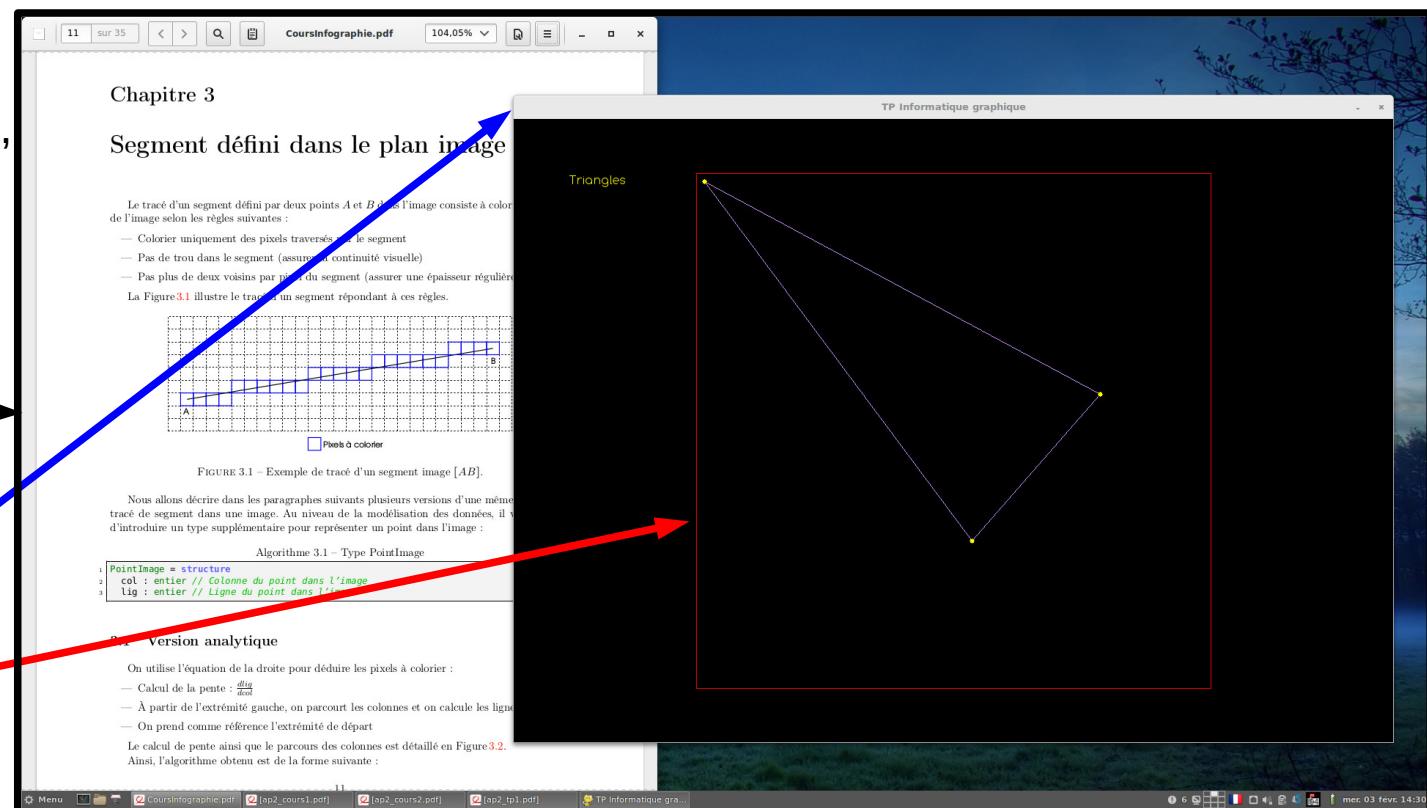
Hiérarchie de fenêtres

- Les écrans numériques ont une résolution fixe (nb de pixels), qui définit implicitement une zone rectangulaire sur l'espace N^2
- Dans les OS avec interface graphique, les applications affichent leurs informations dans des fenêtres indépendantes (images générées par l'application et affichées sur tout ou partie de l'écran)
- Enfin, dans une même fenêtre d'application, on peut avoir besoin de créer plusieurs zones de dessin (sortes de sous-fenêtres)
- On voit donc qu'il y a une hiérarchie de fenêtres (écran, fenêtre application, fenêtre dessin,...), dans laquelle chaque élément est défini dans le repère de l'élément supérieur

Copie d'un écran →

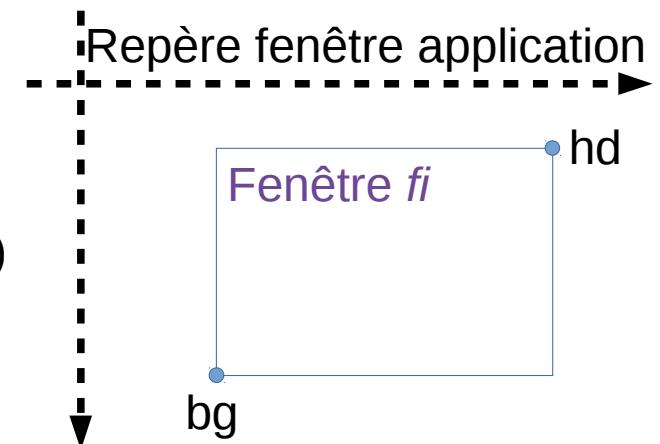
Fenêtre d'application
(notre repère global)

Fenêtre de dessin
dans l'application
(notre *fenêtre image*)



Fenêtre dans le plan image

- On s'intéresse ici à la définition d'une fenêtre de dessin dans le repère de la fenêtre d'application
 - Le niveau supérieur (avec l'écran) est géré par l'OS
 - La fenêtre de dessin (fenêtre *fi*) est définie :
 - dans le repère de la fenêtre application (notre repère de N^2)
(origine en haut à gauche, colonnes vers la droite et lignes vers le bas)
 - par deux points diagonalement opposés :
 - Choix : bas-gauche (*bg*) et haut-droite (*hd*)
- On crée pour cela une structure de fenêtre dans le plan image :



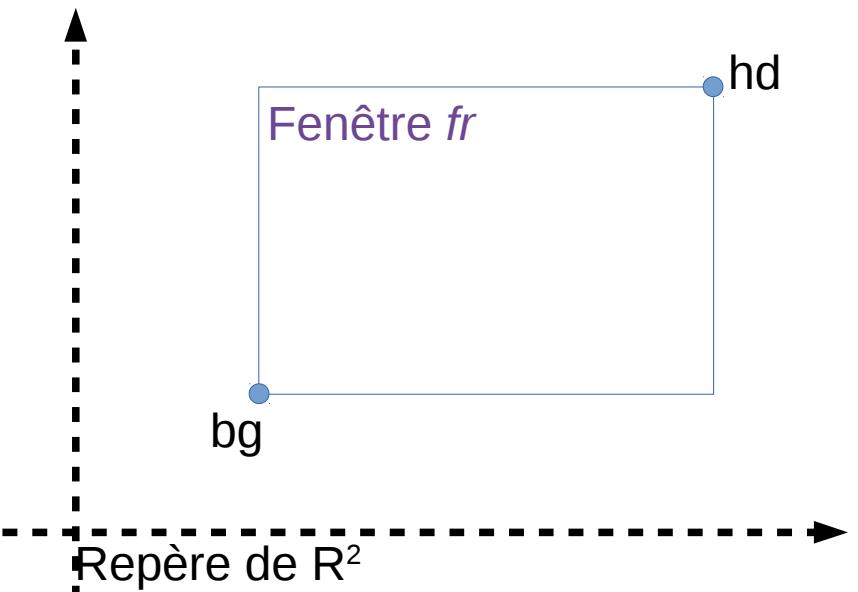
```

PointImage = structure
    col : entier // Colonne du point dans l'image
    lig : entier // Ligne du point dans l'image

FenetreImage = structure
    bg : PointImage // Sommet en bas à gauche de la fenêtre du plan image
    hd : PointImage // Sommet en haut à droite de la fenêtre du plan image
  
```

Fenêtre dans le plan réel

- De même que pour la fenêtre image, on peut définir une fenêtre dans le plan réel R^2 pour manipuler des objets continus
 - Cela définit la zone du plan (fenêtre *fr*) visualisée dans la fenêtre image *fi*
- On crée pour cela une structure de fenêtre dans le plan réel :



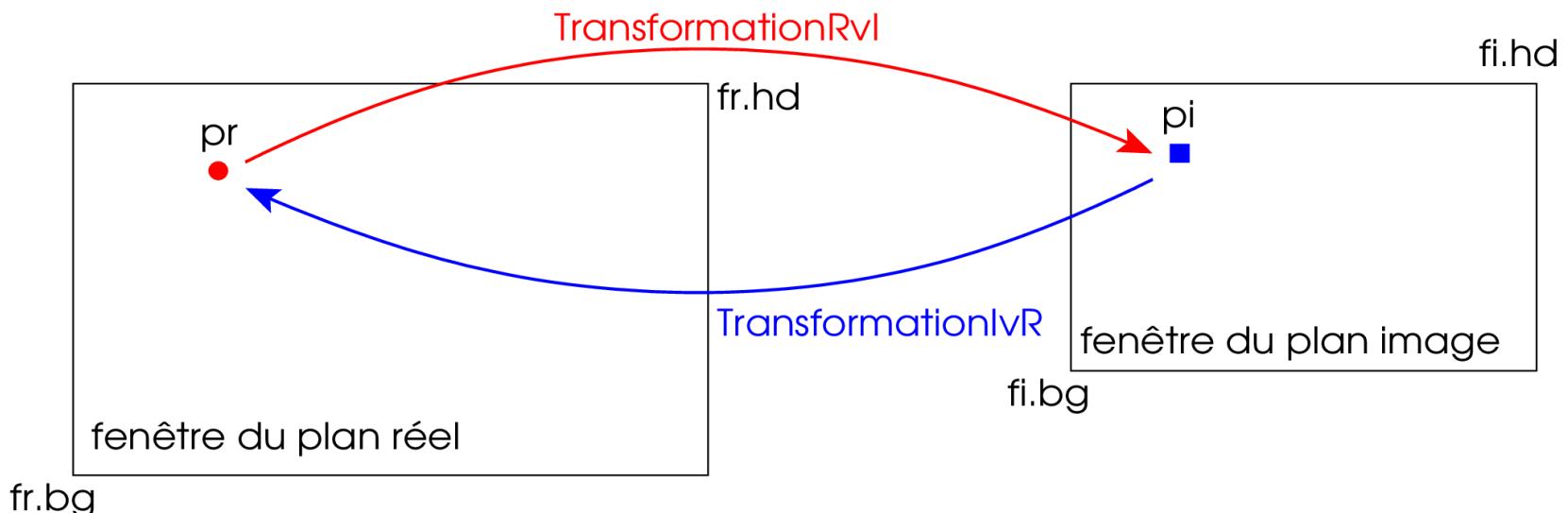
```

PointReel = structure
  x : réel // Abscisse du point
  y : réel // Ordonnée du point

FenetreReel = structure
  bg : PointReel    // Sommet en bas à gauche de la fenêtre du plan réel
  hd : PointReel    // Sommet en haut à droite de la fenêtre du plan réel
  
```

Transformations entre fenêtres

- On dispose donc de deux fenêtres définies dans deux espaces (et repères) différents
- On doit pouvoir passer de l'une à l'autre :
 - On calcule la relation de superposition des deux fenêtres :
 - un point dans une fenêtre a son correspondant dans l'autre fenêtre

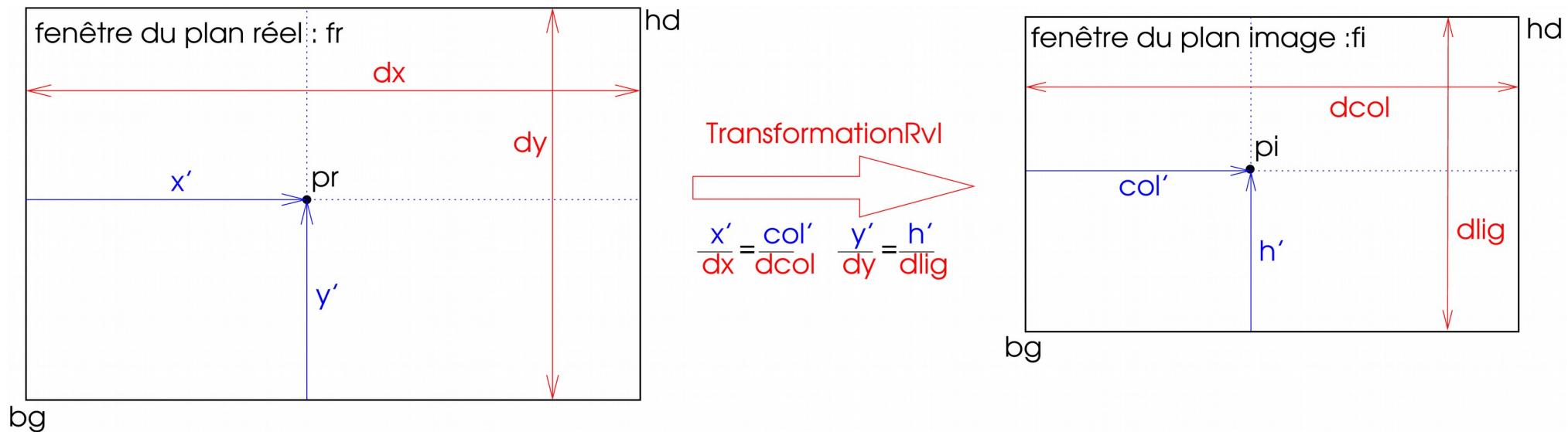


Transformations entre fenêtres

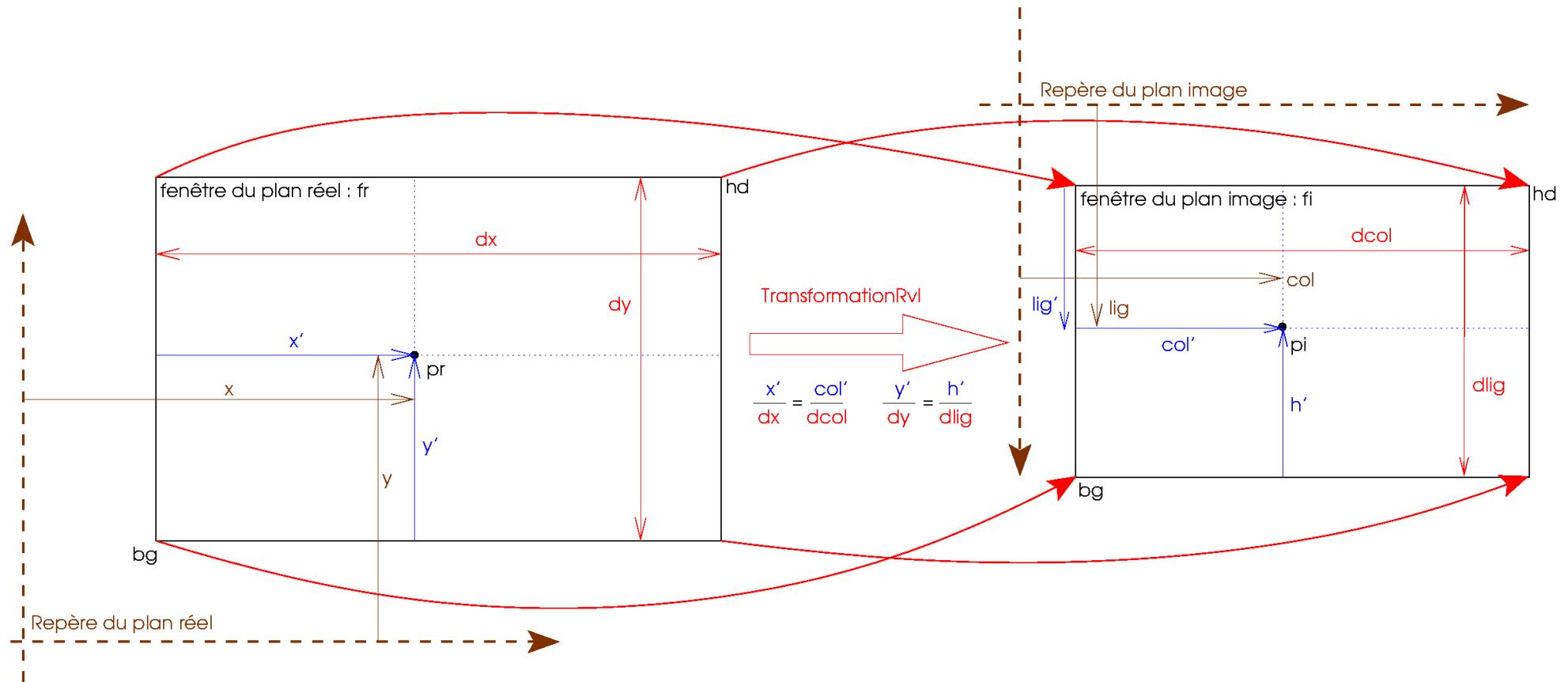
- Pour respecter la position d'un même point dans les deux fenêtres, il faut exprimer sa position relative à l'intérieur de chaque fenêtre :
 - Pourcentages de la largeur et de la hauteur
 - On a donc besoin des dimensions (largeur, hauteur) des deux fenêtres :
 - Dimensions de fr :
 - $dx = fr.hd.x - fr.bg.x$
 - $dy = fr.hd.y - fr.bg.y$
 - Dimensions de fi :
 - $dcol = fi.hd.col - fi.bg.col$
 - $dlig = fi.bg.lig - fi.hd.lig$
- // car axe vertical de haut vers bas*

Transformations entre fenêtres

- Positions relatives identiques entre les deux fenêtres



Positions dans les repères globaux



$$x = bg.x + x'$$

$$y = bg.y + y'$$

$$col = bg.col + col'$$

$$lig = bg.lig - h'$$

$$= hd.lig + dlig - h'$$

$$= hd.lig + lig'$$

Axe vertical inversé !!

Transformations entre fenêtres

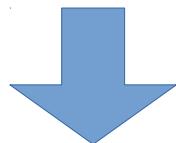
- Pour différencier les repères utilisés, on va noter :
 - pr : le point du plan réel dans le repère R^2
 - pr' : le point du plan réel dans le repère de la fenêtre fr
 - pi : le point du plan image dans le repère N^2
 - pi' : le point du plan image dans le repère de la fenêtre fi
- La transformation $pr \rightarrow pi$ se fait donc en trois étapes :
 - 1) Transfo de pr en pr' : passage du repère de R^2 à celui de fr
 - 2) Transfo de pr' en pi' : passage de fr à fi
 - 3) Transfo de pi' en pi : passage du repère de fi à celui de N^2

$$\begin{array}{ccc} 1 & 2 & 3 \\ pr \rightarrow pr' \rightarrow pi' \rightarrow pi \end{array}$$

Transformations entre fenêtres

- Étapes 1 et 3 et enchaînement des étapes :

- Point pr dans \mathbb{R}^2

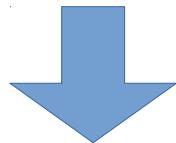


Étape 1

$$pr'.x = pr.x - fr.bg.x$$

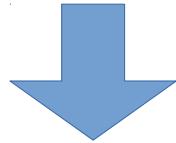
$$pr'.y = pr.y - fr.bg.y$$

- Point pr' dans fr



Étape 2

- Point pi' dans fi



Étape 3

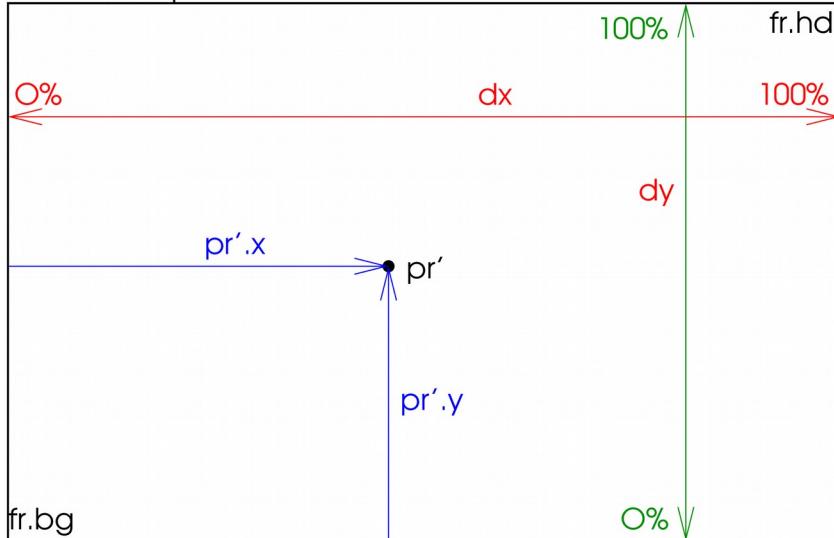
- Point pi dans N^2

$$pi.col = pi'.col + fi.bg.col$$

$$pi.lig = pi'.lig + fi.hd.lig$$

Étape 2 de la transformation

fenêtre du plan réel : fr

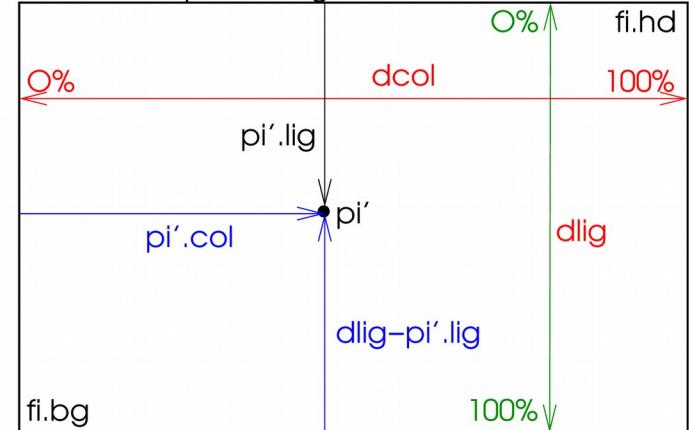


TransformationRvl

$$\frac{pr'.x}{dx} = \frac{pi'.col}{dcol}$$

$$\frac{pr'.y}{dy} = \frac{dlig - pi'.lig}{dlig}$$

fenêtre du plan image : fi



- On déduit les positions relatives (pourcentages) à partir du point bas-gauche (origine de *fr*) :

- De *pr'* dans *fr* :

Horizontale : $pr'.x / dx$ // 0 (=0%) ~ 1 (100%)

Verticale : $pr'.y / dy$ // idem

- De *pi'* dans *fi* :

Horizontale : $pi'.col / dcol$

Verticale : $(dlig - pi'.lig) / dlig$

// !! division réelle nécessaire !!

// soustraction car axe vertical inversé !

Étape 2 et fusion des étapes

- L'identité des positions relatives de pr' et pi' implique :

$$pr'.x / dx = pi'.col / dcol$$

$$pr'.y / dy = (dlig - pi'.lig) / dlig$$
- Et l'on en déduit les coordonnées de pi' :

$$pi'.col = pr'.x * dcol / dx$$

$$pi'.lig = dlig - pr'.y * dlig / dy$$
- Fusion des 3 étapes : pi en fonction de pr

$$pi.col = \text{arrondi}((\text{pr.x} - \text{fr.bg.x}) * \boxed{dcol / dx} + \boxed{\text{fi.bg.col}})$$

1) 2) 3)

$$pi.lig = \text{arrondi}(\boxed{dlig - (\text{pr.y} - \text{fr.bg.y}) * dlig / dy} + \boxed{\text{fi.hd.lig}})$$

2) 1) 3)

Formulation linéaire de la transformation

- On développe les expressions précédentes et on obtient une transformation de la forme :

$$\text{pi.col} = \text{arrondi}(A * \text{pr.x} + B)$$

$$\text{pi.lig} = \text{arrondi}(C * \text{pr.y} + D)$$

$$\text{avec : } A = d\text{col} / dx \quad B = fi.\text{bg.col} - A * fr.\text{bg.x}$$

$$C = -d\text{lig} / dy \quad D = d\text{lig} - C * fr.\text{bg.y} + fi.\text{hd.lig}$$

- Les coefficients A et C représentent respectivement les rapports d'échelles entre :
 - Les axes horizontaux des fenêtres *fr* et *fi*
 - Les axes verticaux des fenêtres *fr* et *fi*
- Les coefficients B et D correspondent à la conjonction des translations des deux fenêtres dans leurs plans respectifs

Structure et fonction de transformation

- Contient les deux fenêtres et les 4 coefficient A, B, C et D
- Nommage : préfixe ri = réel → image

```
TransfosFenetres = structure
    fr : FenetreReel // Fenêtre du plan réel liée aux transformations
    fi : FenetreImage // Fenêtre du plan image liée aux transformations
    riA : réel          // Rapport d'échelle horizontal de fr vers fi
    riB : réel          // Décalage horizontal de fr vers fi
    riC : réel          // Rapport d'échelle vertical de fr vers fi
    riD : réel          // Décalage vertical de fr vers fi
```

- Fonction de transformation :

```
fonction TransformationRvI(pr : PointReel, transfo : TransfosFenetres) : PointImage
// Transforme le point pr en un point image

// Point image résultant de la transformation
pi : PointImage

// Calcul du point pi
pi.col ← arrondi(transfo.riA * pr.x + transfo.riB)
pi.lig ← arrondi(transfo.riC * pr.y + transfo.riD)

// Renvoie le point image obtenu par la transformation du point réel
retourne pi
```

Version matricielle

- Étant donné que les transformations entre les fenêtres sont linéaires, on peut les exprimer sous la forme d'une matrice T_{ri} :

$$pi = \text{arrondi}(T_{ri} * pr) = \begin{pmatrix} pi.col \\ pi.lig \end{pmatrix} = \left[\begin{pmatrix} A & 0 & B \\ 0 & C & D \end{pmatrix} \times \begin{pmatrix} pr.x \\ pr.y \\ 1 \end{pmatrix} \right]$$

- Coordonnées homogènes :
 - Ajout à pr d'une 3^{ème} composante (=1) pour exprimer les translations sous forme matricielle
 - On peut aussi ajouter une 3^{ème} ligne à T_{ri} et à pi pour exprimer des transformations plus complexes

Transformation de *pi* en *pr*

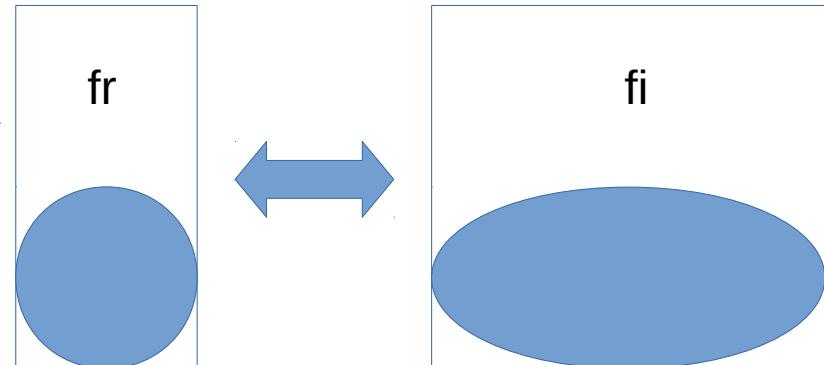
- Une fois que l'on a déterminé la transformation dans un sens, on peut facilement déduire sa réciproque, dans l'autre sens
- Pour déduire la transformation de *pi* vers *pr*, il suffit d'inverser les formules de *pr* vers *pi* en isolant les variables *x* et *y* :
$$riA * x + riB = col \Rightarrow x = (col - riB) / riA$$
$$riC * x + riD = lig \Rightarrow y = (lig - riD) / riC$$
- Ce qui permet de déduire les coefficients *irA*, *irB*, *irC* et *irD* de la nouvelle transformation (*ir* = image → réel, *ri* = réel → image) :
$$x = irA * col + irB \Rightarrow irA = 1 / riA \text{ et } irB = -riB / riA$$
$$y = irC * lig + irD \Rightarrow irC = 1 / riC \text{ et } irD = -riD / riC$$
- Enfin, *x* et *y* étant réels, l'arrondi à la fin du calcul n'est pas pertinent dans ce sens là

Proportions des fenêtres

- Lorsque les proportions sont différentes :

$$|dy/dx| \neq |dlig / dcol|$$

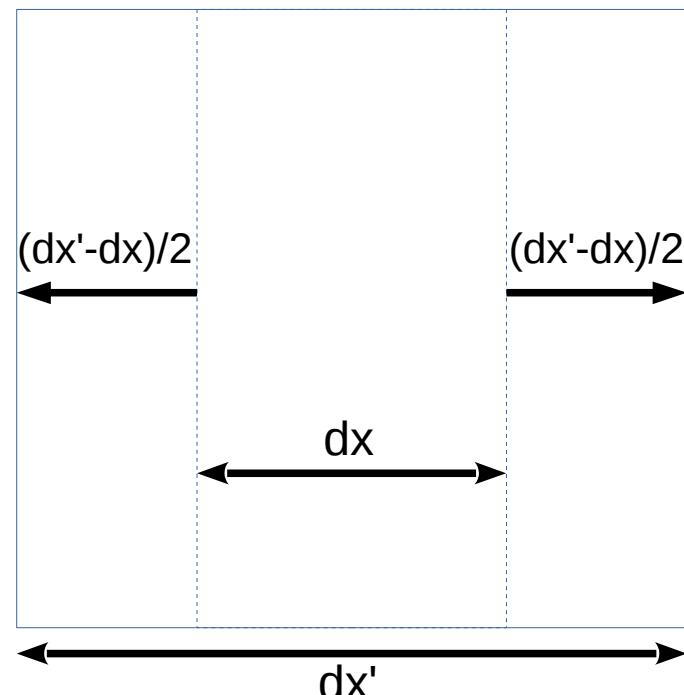
les objets sont déformés...



- Il faut alors modifier la taille de l'une des fenêtres pour retrouver les bonnes proportions
- 4 possibilités de changement, selon que l'on modifie dx , dy , $dcol$ ou $dlig$ afin de rétablir l'égalité des proportions
- Selon la modification effectuée, on obtient :
 - Un agrandissement de la largeur (ou hauteur) de l'une des fenêtres :
 - Toujours possible pour fr MAIS pas pour fi (surface d'affichage peut être limitée)
 - Une réduction de la largeur (ou hauteur) de l'une des fenêtres :
 - Toujours possible pour chaque fenêtre MAIS pas toujours souhaitable (moins d'informations affichées car zone visualisée plus petite)

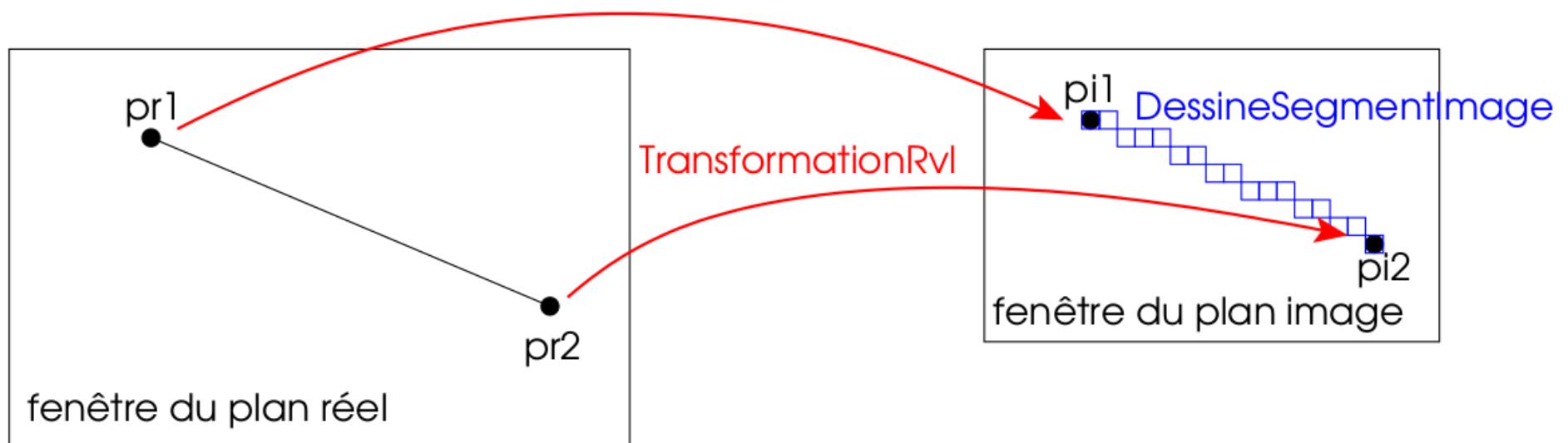
Proportions des fenêtres

- Il est alors préférable de toujours privilégier l'agrandissement de *fr* :
 - $dy/dx > dlig/dcol \rightarrow$ augmenter la largeur de *fr*
 - dx remplacé par $dx' = dy * dcol / dlig$
 - $dy/dx < dlig/dcol \rightarrow$ augmenter la hauteur de *fr*
 - dy remplacé par $dy' = dx * dlig / dcol$
- En général, on positionne la nouvelle fenêtre de sorte que l'ancienne soit au centre de la nouvelle
- Agrandissement horizontale :
 - $fr.bg.x' = fr.bg.x - (dx'-dx)/2$
 - $fr.hd.x' = fr.hd.x + (dx'-dx)/2$
- Symétrie en *y* pour l'agrandissement vertical



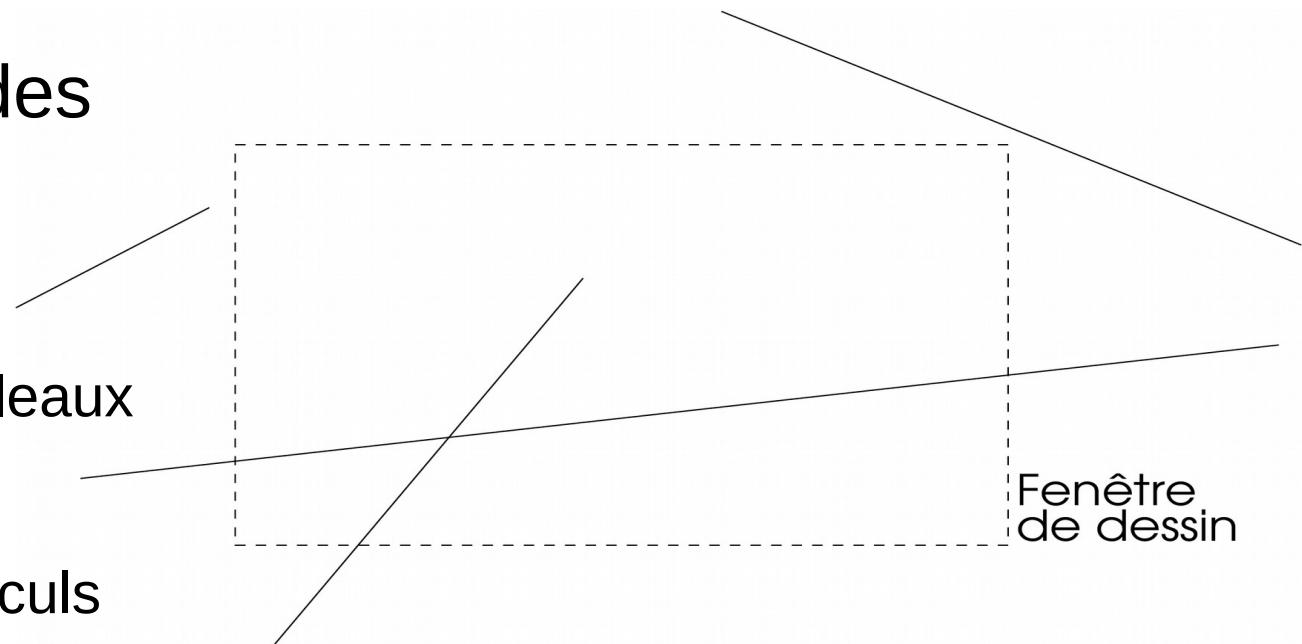
Tracé d'un segment du plan réel

- Transformation des deux extrémités
- Tracé du segment image obtenu



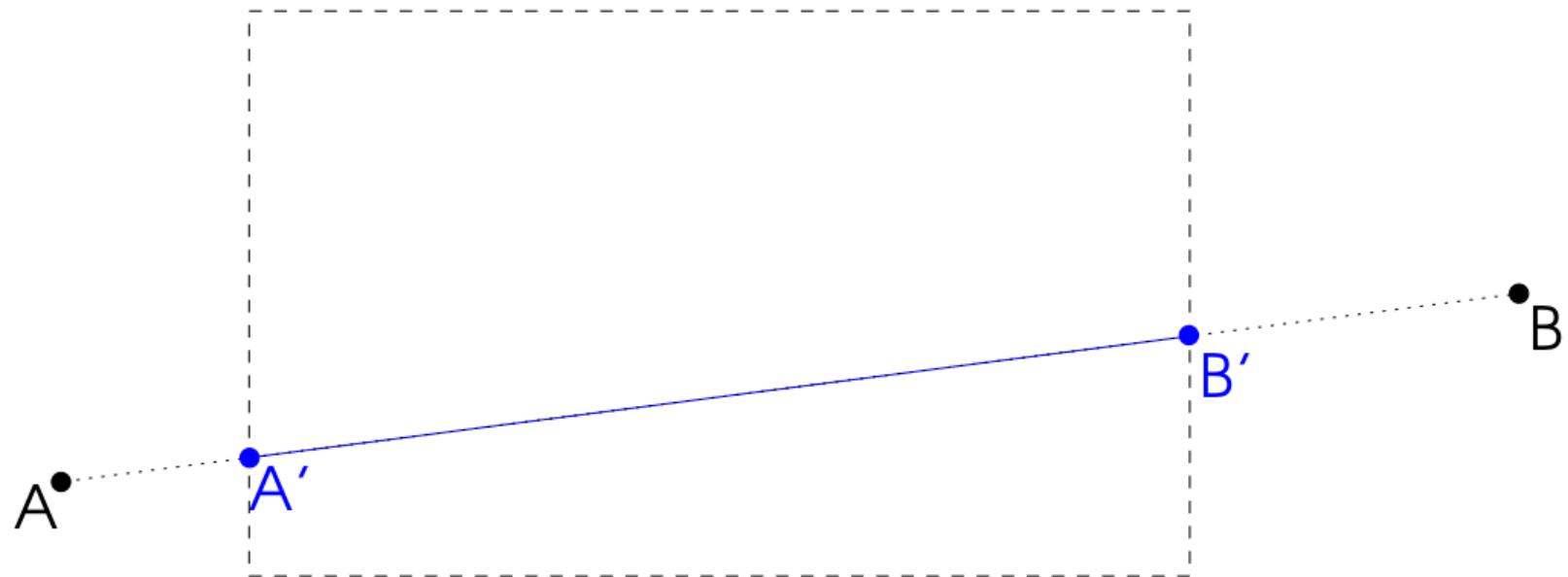
Découpage de segment

- Le problème lors du dessin d'un segment réel est qu'il peut ne pas être inclus totalement dans la fenêtre de dessin
- Cela peut poser des problèmes de :
 - Validité
 - Accès hors tableaux
 - Performance
 - Parcours et calculs inutiles
- Nécessité de découper les segments !



Principe du découpage

- Transformer le segment initial en un autre segment dont les extrémités sont dans la fenêtre
 - Si le segment est hors fenêtre, les extrémités doivent l'être aussi pour détecter qu'il ne doit pas être affiché



- On passe de $[AB]$ à $[A'B']$

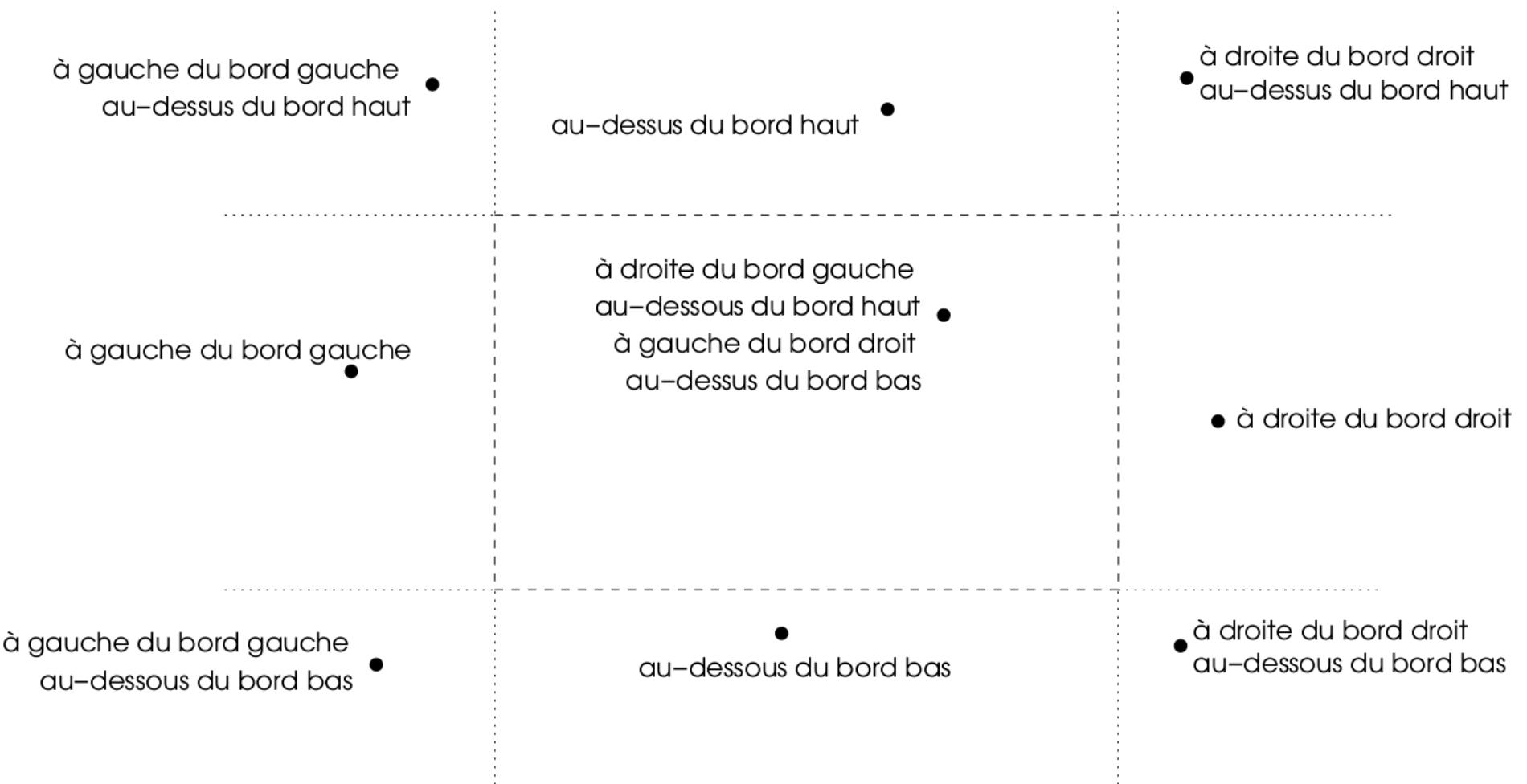
Algorithme général

- Le schéma général de l'algorithme est le suivant :

Pour chaque extrémité du segment :

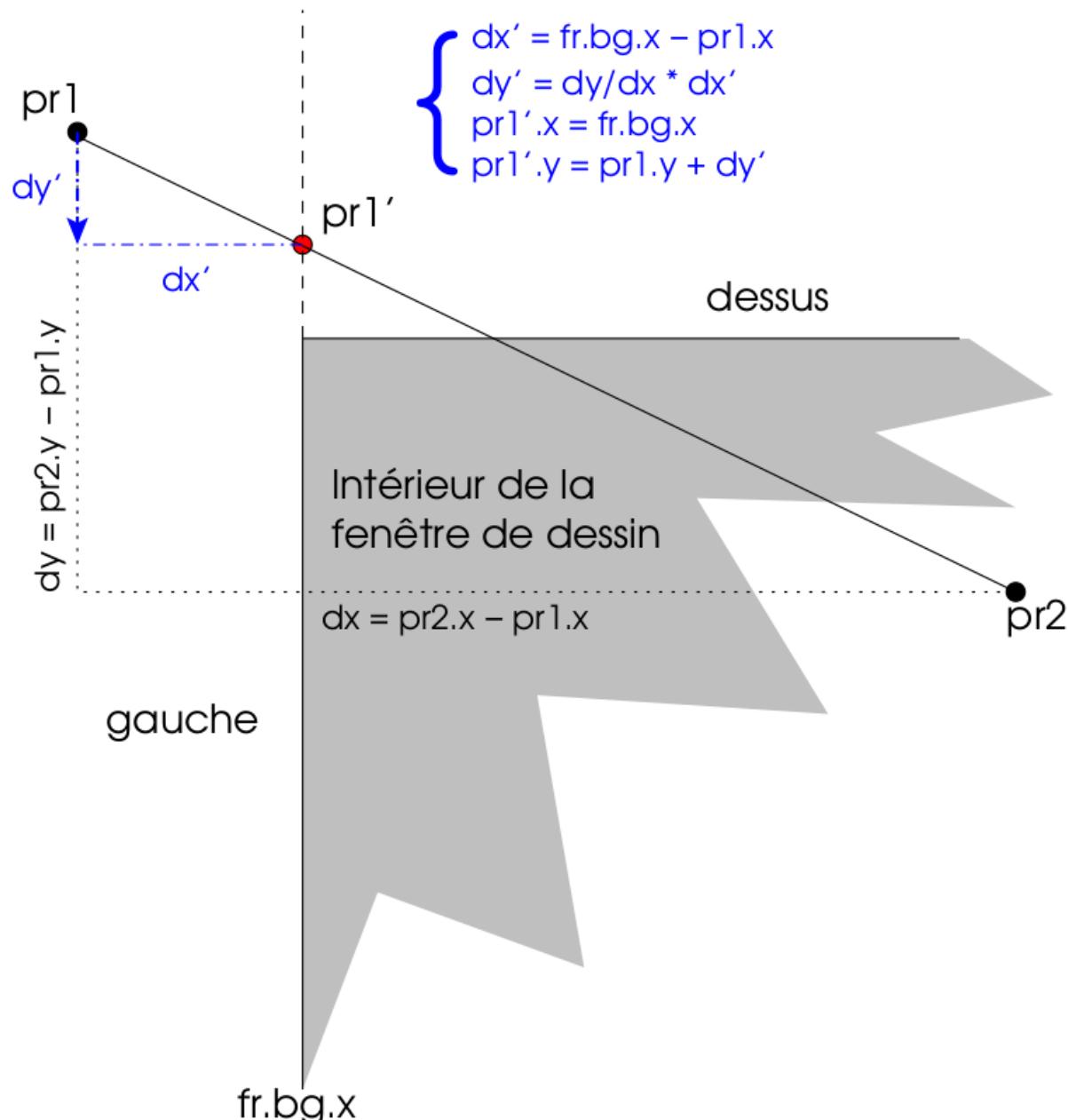
- Pour chaque bord de la fenêtre :
 - Si l'extrémité est à l'extérieur par rapport à ce bord alors :
 - remplacer l'extrémité par l'intersection du segment avec ce bord de la fenêtre

Test de la position d'un point



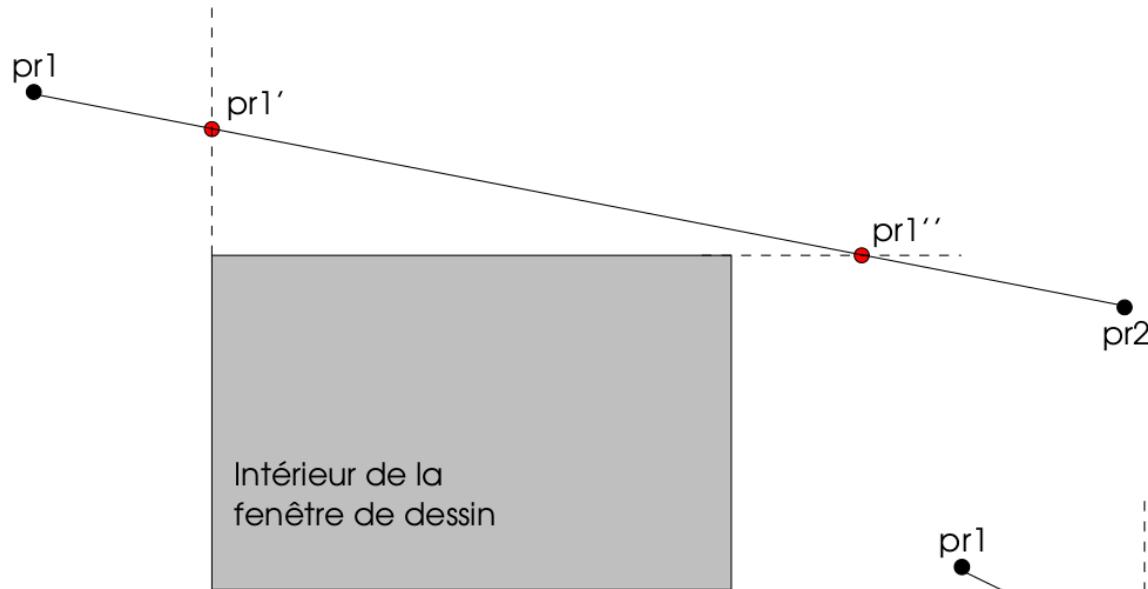
Projection d'un point sur un bord

- $pr1 \rightarrow pr1'$

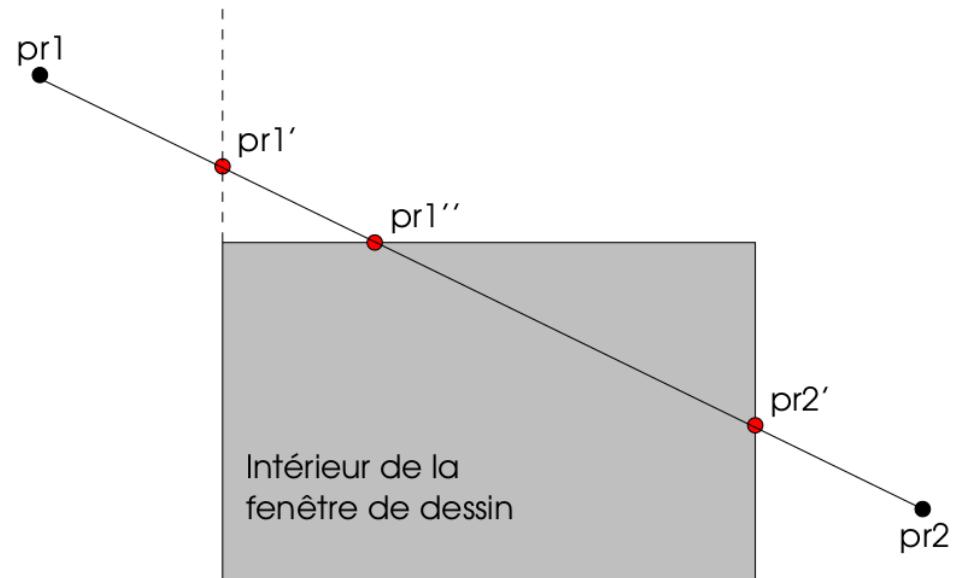


Processus de projection

- Projections sans intersection avec la fenêtre



- Projections avec intersections



Processus complet réel → image

