université de BORDEAUX	**UE : 4TPM101U** / Initiation à l'informatique **Épreuve :** TP noté 2023 **Date :** 06 janvier 2023 **Durée :** 1h20 Documents : Autorisés	Collège Sciences et Technologies

— Sont autorisés : le polycopié du cours, vos notes de cours, ainsi que vos fichiers Python de TP.
— **Les questions sont largement indépendantes.**
— Téléchargez le fichier `bibimages.py` sur Moodle et placez-le dans le même dossier que votre fichier courant.
— Placer l'instruction `from bibimages import *` au début de votre fichier.
— Les fonctions de manipulation d'images sont rappelées en annexe.

Exercice 1 : Représentation des couleurs

Une couleur au format RGB est définie par un triplet (r, g, b) d'entiers, chacun compris entre 0 et 255. Il y a donc 256^3 couleurs possibles. La couleur (r, g, b) s'exprime comme un entier compris entre 0 et 16777215 par la fonction mathématique $f(r, g, b) = r \times 256^2 + g \times 256 + b$.

Question 1.1. Écrire une fonction `RGBVersInt(r: int, g: int, b: int) -> int` qui convertit une couleur exprimée au format RGB en un entier n compris entre 0 et 16777215 via la transformation décrite par la fonction f.

Question 1.2. Écrire une fonction `IntVersRGB(n: int) -> couleur` qui convertit un entier compris entre 0 et 16777215 en une couleur `(r,g,b)` via l'inverse de la fonction f.

Exemple : l'appel `IntVersRGB(3289650)` renvoie le triplet `(50,50,50)`, car $f(50, 50, 50) = 3289650$.

Exercice 2 : Tracé de segment

On souhaite tracer une ligne oblique entre deux points (x_1, y_1) et (x_2, y_2). On rappelle qu'une telle ligne, que l'on appelle le segment $[(x_1, y_1), (x_2, y_2)]$, est définie (lorsque $x_1 \neq x_2$) par l'ensemble des points (x, y) vérifiant $x \in [x_1, x_2]$ et

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

Question 2.1. Étant donné (x, y) sur le segment $[(x_1, y_1), (x_2, y_2)]$, exprimer $a, b \in \mathbb{R}$ tels que $y = ax + b$.

Le paramètre a est appelé le *coefficient directeur* du segment.

Question 2.2. Pour tracer le segment $[(x_1, y_1), (x_2, y_2)]$, on propose l'algorithme suivant : pour chaque abscisse x entière comprise entre x_1 et x_2, tracer le point de coordonnées (x, y), où $y = \lfloor ax + b \rfloor$.

Écrire la fonction `tracerSegment(img: image, x1: int, y1: int, x2: int, y2: int)` qui trace un segment blanc sur l'image `img` entre les points `(x1,y1)` et `(x2,y2)` via l'algorithme présenté ci-dessus.

Indication : Pour calculer la partie entière inférieure $\lfloor\ \rfloor$, on pourra utiliser la fonction `int(x: float) -> int`.

Question 2.3. Créer une image de taille 500×300, et afficher le segment d'extrêmités $(x_1 = 150,\ y_1 = 10)$ et $(x_2 = 350,\ y_2 = 30)$.

Que remarque-t-on pour le segment d'extrêmités $(x_1 = 100,\ y_1 = 10)$ et $(x_2 = 115,\ y_2 = 290)$?

On se propose à présent d'implémenter l'algorithme de tracé de segment de *Bresenham*, pour un meilleur rendu.

On va supposer pour simplifier que $x_2 - x_1 > y_2 - y_1 > 0$, autrement dit que la pente a du segment est comprise entre 0 et 1.

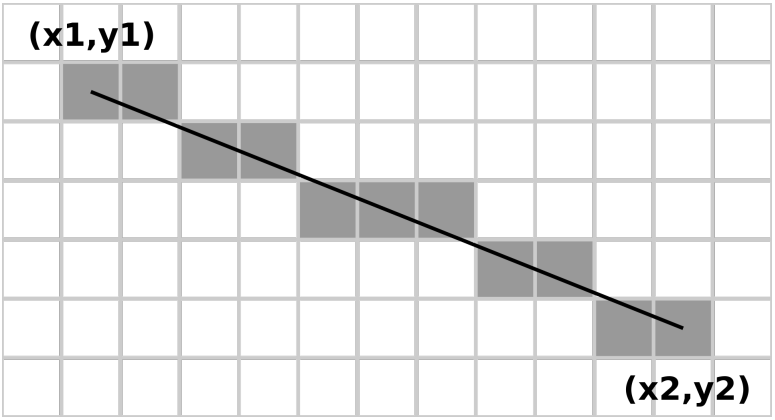


FIGURE 1 – Résultat du tracé de segment par l'algorithme de Bresenham

Si un pixel de coordonnées (x, y) n'est pas parfaitement sur la droite définie par la fonction affine $f(x) = ax + b$ (qui vérifie $f(x_1) = y_1$ et $f(x_2) = y_2$), on peut définir **l'erreur** commise par $e = (ax + b) - y$.

Ainsi :

— Au départ de l'algorithme, $x = x_1$ et $y = y_1$, donc $e = 0$.

— À chaque fois que x est incrémenté de 1 (pour passer au pixel suivant), la valeur de $e + y$ augmente de a.

Nous allons donc définir une variable `e`, de type `float`, qui vaudra donc 0.0 initialement. Il nous faut la maintenir en dessous de 0.5 tout au long du tracé.

Après avoir tracé un pixel de coordonnées (x,y), on ajoute a à la valeur de e. Il y a deux cas de figure pour le point suivant à tracer :

— Si e ≤ 0.5, on peut laisser y inchangé ;

— Sinon, on ajoute 1 à y, et il faut alors soustraire 1 à e.

Question 2.4. Écrire une fonction `MaJErreur(a: float, e: float, y: int) -> (float, int)`, qui prend en paramètres le coefficient directeur a, le paramètre d'erreur e, et la valeur actuelle de l'ordonnée y. Cette fonction doit renvoyer le couple (e,y) mis à jour pour le prochain point, via l'algorithme décrit ci-dessus.

L'algorithme de Bresenham est donc le suivant :

— On démarre du point (x $= x_1$, y $= y_1$), que l'on trace, et de e $= 0.0$;

— À chaque étape, on incrémente x de 1, on met à jour e et y, puis on trace le nouveau point (x,y) ;

— Répéter ce processus jusqu'à atteindre l'abscisse x $= x_2$.

Question 2.5. Écrire une fonction `bresenham1(img: image, x1: int, y1: int, x2: int, y2: int)`, qui prend en paramètres une image img, les coordonnées entières x1, y1, x2, y2 des extrêmités du segment, et qui trace en blanc le segment $[(x_1, y_1), (x_2, y_2)]$ via l'algorithme de Bresenham.

On veillera à réutiliser la fonction `MaJErreur`.

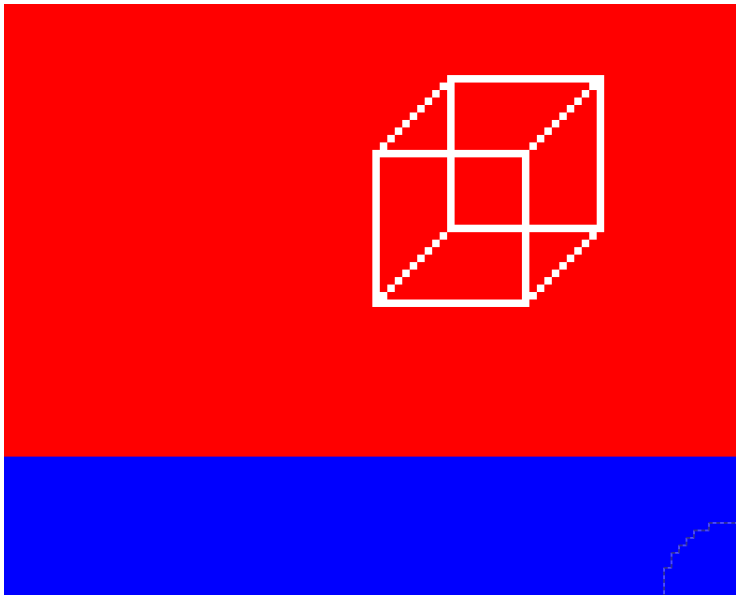
Question 2.6. On souhaite à présent adapter l'algorithme précédent à une pente a supérieure à 1. En réutilisant le code des fonctions précédentes, écrire une fonction `bresenham2(img: image, x1: int, y1: int, x2: int, y2: int)`, utilisant les mêmes paramètres et traçant le segment $[(x_1, y_1), (x_2, y_2)]$ via l'algorithme de Bresenham, si l'on suppose à présent que $y_2 - y_1 > x_2 - x_1 > 0$.

Indication : on pourra utiliser le même algorithme, en incrémentant cette fois y de 1, et en mettant à jour e et x.

Question 2.7. Créer une image de taille 500×300, et tracer le segment d'extrêmités $(x_1 = 100,\ y_1 = 10)$ et $(x_2 = 115,\ y_2 = 290)$ avec la fonction bresenham2. Comparer au segment obtenu à la question 2.3.

Exercice 3 : Pour les plus rapides

Reproduire l'image ci-dessous.



Indication : On pourra définir les fonctions `segmentHorizontalBlanc`, traçant un segment horizontal entre les points (`x1`,`y1`) et (`x2`,`y2`), `segmentVerticalBlanc`, `carreBlanc`, `rectanglePlein`, et réutiliser la fonction `tracerSegment` de l'exercice précédent.

Annexe : fonctions de manipulation d'images

`nouvelleImage(large:int, haut:int) -> image`	Retourne une image de taille $large \times haut$, initialement noire.
`afficherImage(img:image)`	Affiche l'image img.
`colorierPixel(img:image, x:int, y:int, (r,g,b): couleur)`	Peint le pixel (x, y) dans l'image img de la couleur (r, g, b)
`largeurImage(img:image) -> int`	Retourne le nombre de colonnes de pixels contenues dans img
`hauteurImage(img:image) -> int`	Retourne le nombre de lignes de pixels contenues dans img
`couleurPixel(img, x, y) -> couleur`	Retourne la couleur du pixel (x, y) dans l'image img