TP3 : Récursivité

Pour ce TP, l'objectif est d'utiliser la recursivité dans un programme qui permet une approche visuelle. Comme prétexte pour ce TP, nous allons nous intéresser à la représentation du triangle de Sierpiński (encore une fractale, mais d'un autre type), dont un exemple est donné figure 1. Le TP2 sera utile pour recycler quelques fonctions ou principes.

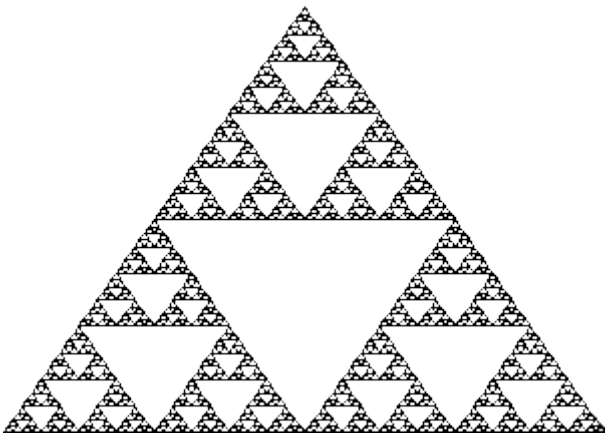

Figure 1: représentation du triangle de Sierpiński

Le triangle de Sierpiński est, dans sa représentation la plus commune, un triangle équilatéral délimité, à chaque sommet, par un triangle qui lui est identique à l'échelle près (2 fois plus petit).

1) Structures de données de base

Bien que ce ne soit pas strictement nécessaire, il est conseillé de récupérer la définition de `struct color` du TP2. Cela allègera les prototypes de fonctions.

Écrivez la définition de la structure `picture` qui sera utilisée pour mémoriser votre image pendant sa création. Elle doit contenir les champs suivants :

- width : la largeur de l'image en pixels
- height : la hauteur de l'image en pixels
- pixels : une représentation du contenu de l'image sous forme d'un tableau de composantes de couleur.

Écrivez la fonction `new_pic()` qui récupère deux entiers, une largeur et une hauteur exprimées en pixels, et qui renvoie un struct picture correspondant aux dimensions souhaitées. Cette fonction est très proche de la fonction `new_mandel()` du TP2.

Écrivez la fonction `save_pic()` qui recoit un `struct picture` et un nom de fichier sous forme de chaine de caractères, et qui crée le fichier image correspondant. Cette fonction peut être écrite en simplifiant la fonction `save_mandel()` du TP2.

Écrivez la fonction `set_pixel()` qui reçoit un `struct picture`, deux entiers (des coordonnées x et y en pixels) et une couleur. Cette fonction assigne la couleur donnée au pixel correspondant aux

coordonnées x et y dans l'image fournie. Pour suivre la convention des formats d'images, le pixel de référence (0,0) sera en haut à gauche. Selon votre choix d'utiliser ou non le type `struct color`, vous entrerez votre couleur sous forme d'un `struct color` ou de trois `char` codant les trois composantes rouge, verte et bleue.

Testez les différentes fonctions que vous avez produites jusqu'ici en créant une image de taille 10x10, dont tous les pixels sont noirs à l'exception des quatres pixels correspondant aux quatre coins de l'image qui prendront la couleur de votre choix (à condition de ne pas choisir noir).

2) Tracé du premier triangle

Écrivez la fonction `draw_line()` qui reçoit un `struct picture`, quatre entiers (les coordonnées de deux points) et une couleur. Cette fonction trace une ligne de la couleur donnée entre les deux points dont les coordonnées sont fournies.

Notons x1, y1 et x2, y2 les coordonnées des deux points définissant le segment à tracer. Le nombre de pixels à tracer est : n = max(|x1-x2|, |y1-y2|) + 1. Il ne reste alors qu'à dessiner les n pixels en calculant leurs coordonnées dans une boucle allant de 0 à n-1.

Testez la fonction `draw_line()` en recréant l'image de la figure 2 (dimensions 10x10), basée sur :

- un fond blanc,
- une ligne rouge entre (2,2) et (7, 7),
- une ligne bleue entre (2,7) et (7,2),
- deux lignes vertes entre (1,2) et (1,7), et entre (8,2) et (8,7)
- deux lignes magenta entre (2,1) et (7,1), et entre (2,8) et (7,8)

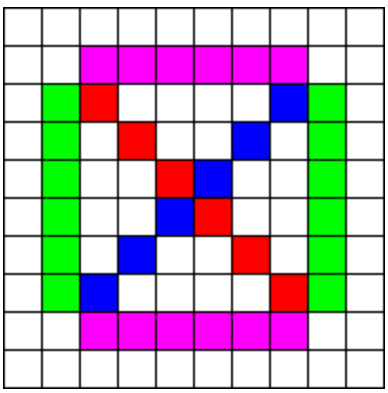

Figure 2: image test pour le tracé de ligne

La composition des couleurs les plus courantes est donnée dans la table 1.

Écrivez la fonction `sierpinski()` qui reçoit un `struct picture`, trois doubles (les coordonnées d'un point et une taille) et une couleur. Cette fonction trace un triangle equilateral dont un des sommets est aux coordonnées fournies, et dont le côté mesure *taille*.

Testez votre fonction en créant une image de dimensions 400 x 350 contenant un triangle de largeur 400 dont un des sommets est en (x=0, y=349). Il s'agit de la représentation de départ.
En toute logique, les coordonnées des autres sommets auront pour coordonnées (*taille*-1, 349) et (*taille*/2, 349 – *taille**√3 / 2).

Table 1: composition des couleurs principales

Couleur	Composante Rouge	Composante Verte	Composante Bleue
Noir	0	0	0
Rouge	255	0	0
Vert	0	255	0
Jaune	255	255	0
Bleu	0	0	255
Magenta (violet)	255	0	255
Cyan (bleu clair)	0	255	255
Blanc	255	255	255

3) Application de la récursivité pour le tracé

Cette partie est destinée à accompagner les étudiants vers un algorithme récursif. Chacun peut donc, selon son aisance, passer directement à la création du triangle complet, ou suivre les étapes détaillées.

a) utilisation d'une fonction intermédiaire.

Pour commencer, **écrivez** la fonction `sierpinski_div()` qui recoit les mêmes arguments que `sierpinski()`, mais qui cette fois trace trois triangles, chacun deux fois plus petit que la taille demandée. Chacun de ces trois triangles sera tracé en utilisant la fonction `sierpinski()` aux coordonnées :

- (*x0*, *y0*) : triangle en bas à gauche
- (*x0+taille/2*, *y0*), : triangle en bas à droite
- (*x0+taille/4*, *y0-taille*√3 /4*) : triangle du haut

Vérifiez le comportement de la fonction `sierpinski_div()`

b) adaptation en fonction de la taille demandée.

Modifiez la fonction `sierpinski_div()` pour que le triangle tracé soit composé :

- d'un seul triangle si la taille demandée est inférieure à 300 (comportement identique à `sierpinski()`)
- de trois triangles si la taille demandée est supérieure à 300 (comportement d'origine)

En regardant bien, lors du tracé de la figure en trois triangles, la fonction `sierpinski_div()` peut appeler indifféremment `sierpinski()` ou `sierpinski_div()`. En effet, lorsqu'elle est appelée en interne, la fonction de tracé des « sous-triangles » reçoit une taille inférieure à 300, dans ce cas, les deux fonctions se comportent de façon identique. Ce test sur la taille sera appelé **condition d'arrêt**.

Vérifiez le comportement de la fonction `sierpinski_div()`

c) faible récursivité

En fusionnant `sierpinski()` et `sierpinski_div()`, transformez la fontion `sierpinski()` en une fonction récursive dont la condition d'arrêt est taille<300.

Pour qu'une fonction récursive produise un résultat correct, il faut remplir les deux conditions suivantes :

- La fonction ne doit pas s'appeler elle-même lorsque la condition d'arrêt est vraie.
- À chaque fois que la fonction s'appelle elle-même, elle doit le faire avec des paramètres qui la rapprochent de la condition d'arrêt.

Gardez le même programme, mais abaissez la condition d'arrêt à *taille<150*, puis à *taille<75*.

d) Figure complète

Determinez la condition d'arrêt pour tracer le triangle complet[1].
Il n'y a plus qu'à …

4) (Optionnel) Un peu de couleur...

Pour améliorer le rendu et mettre en évidence le caractère infini des subdivisions, il est interessant de tracer les triangles avec une couleur d'autant plus sombre qu'ils sont petits.

Modifiez vore programme pour permettre ce résultat.

5) (Optionnel) De la compréhension du phénomène...

Histoire de s'amuser un peu (sic!), modifiez le motif du dessin, tout en gardant les règles d'héritage identiques (par exemple, vous pouvez tracer un carré, un rond, ou juste une croix…)

La figure est perturbée, mais le schéma principal reste le même, ce qui prouve que les règles d'héritage de la récursivité sont plus importantes que les dessins unitaires par eux-mêmes.

1 En toute rigeur, la structure est infinie et ne peut donc pas être tracée complètement, mais en informatique, la précision de la représentation des images est limitée à la taille du pixel. On considèrera donc que l'image est tracée complètement si les détails de l'ordre du pixel sont visibles, sans considérer les détails plus petits qu'un pixel.