TP 8 : PGCD et Fractales

Notions : Récursivité

1 Calcul du PGCD

Le calcul du pgcd de 2 nombres a et b peut se faire de manière simple, en utilisant la récursivité selon l'algorithme 1 suivant :

Algorithm 1 PGCD de a et b

Si $a \geq b$ **Alors**
  Si b==0 **Alors**
    Retourner a
  Sinon
    Retourner le PGCD de b et du reste de la division de a par b
  Fin Si
Sinon
  Retourner le PGCD de b et de a
Fin Si

2 Un arbre fractal

Il existe des courbes du plan qui présentent des auto-similarités et sont infiniment imbriquées dans elles-mêmes : certaines parties sont semblables à la courbe complète. Ces sont des fractales. Elles sont de longueur infinie, bien que contenues dans une surface d'aire finie. Quelques unes de ces fractales sont définies par un processus récursif très simple, à partir d'un segment initial auquel on applique récursivement des transformations linéaires (rotation, homothétie, translation).

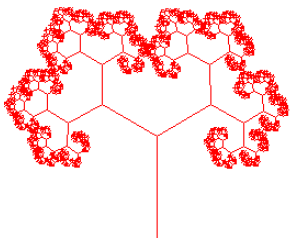

FIGURE 1 – Un arbre fractal.

Par exemple, la courbe de l'arbre (figure 1) est définie de manière récursive par le processus suivant :

1. Pour commencer, on prend un segment AB (en noir sur la figure ci dessous),
2. on crée le segment BM_1 par une homothétie de valeur r, suivie d'une rotation de θ degrés autour du centre A puis d'une translation $\overrightarrow{OB}$ du segment initial,
3. on crée le segment BM_2 par une homothétie de valeur r, suivie d'une rotation de $-\theta$ degrés autour du centre A puis d'une translation $\overrightarrow{OB}$ du segment initial,
4. on répète, jusqu'à atteindre l'ordre désiré, les étapes 2 et 3 sur les nouveaux segments en rouge sur les exemples :
   (a) BM_1 d'une part,
   (b) BM_2 d'autre part.

On modélise ces transformations par 3 fonctions : f_1 pour le segment BM_1, f_2 pour le segment BM_2 et f_3 pour le segment AB. Ce processus s'appelle IFS, pour Iterated Function System. En prenant les points $A(0,0)$ et $B(x,y)$, les fonctions s'écrivent :

$$\begin{aligned} f_1(x,y) &= r.\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \\ f_2(x,y) &= r.\begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \\ f_3(x,y) &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

La figure 2 représente les trois premiers ordres de la courbe de l'arbre pour $r = \sqrt{2}$ et $\theta = \pi/4$.

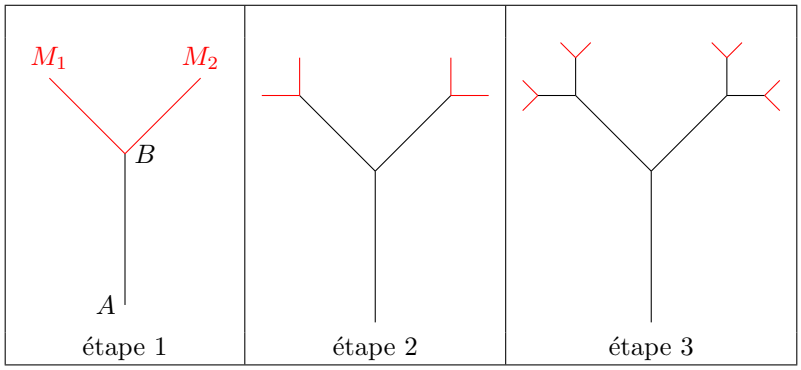

FIGURE 2 – Les 3 premiers ordres de la courbe.

Si les points initiaux sont $A(x_a, y_a)$ et $B(x_b, y_b)$, la fonction $f_1(x,y)$ donne le point $M_1(x_1, y_1)$, la fonction $f_2(x,y)$ donne le point $M_2(x_2, y_2)$. On conserve le segment AB grâce à la fonction $f_3(x,y)$.

Les calculs des coordonnées des points $M_1(x_1, y_1)$ et $M_2(x_2, y_2)$ s'écrivent :

$$\begin{aligned} x_1 &= x_b + r * \cos(\theta) * (x_b - x_a) - r * \sin(\theta) * (y_b - y_a) \\ x_2 &= x_b + r * \cos(\theta) * (x_b - x_a) + r * \sin(\theta) * (y_b - y_a) \\ y_1 &= y_b + r * \sin(\theta) * (x_b - x_a) + r * \cos(\theta) * (y_b - y_a) \\ y_2 &= y_b - r * \sin(\theta) * (x_b - x_a) + r * \cos(\theta) * (y_b - y_a) \end{aligned}$$

Pour dessiner l'arbre fractal, il faut donc utiliser une procédure récursive qui prend en paramètres les points entre lesquels il faut tracer la courbe, ainsi que le niveau (ou ordre) que l'on veut atteindre. Il faut tracer la droite entre les deux points. Si le niveau est atteint, on s'arrête. Sinon, on crée les deux nouveaux segments et on relance la procédure récursive de dessin sur chacun de ces segments.

---

tracer la droite entre les deux points A et B
Si le niveau n'est pas atteint **Alors**
  calculer les coordonnées de M_1 et de M_2
  incrémenter le niveau
  relancer la procédure récursive sur chacun des nouveaux segments BM_1 et BM_2
Sinon
  ne rien faire ! !
Fin Si

2.1 Version en mode texte

1. Ecrivez une fonction qui affiche les coordonnées des points M_1 et M_2 calculés à chaque niveau, sans utiliser de fonction graphique.

2. Ecrivez un programme qui appelle cette fonction.

Les valeurs des premiers points M sont les suivantes pour $A(100, 0)$ et $B(100, 100)$ pour $r = 2/(1+\sqrt{5})$ et $\theta = \pi/4$ en arrondissant les coordonnées :

— étape 1 : (56,143) et (143,143)

— étape 2 : (17,142) (55,181) et (143,180) (180,143)

3 Programmation multi fichier et makefile

Lorsque on développe un logiciel, il est courant de travailler avec plusieurs fichiers source pour mieux structurer notre développement. Lors d'une modification, il faut alors recompiler les fichiers modifiés et réaliser l'édition des liens pour créer un fichier exécutable. Bien souvent, le nombre de fichiers en jeu est important et il est alors pratique d'utiliser un utilitaire qui permet d'automatiser toutes ces opérations de compilation. Cet utilitaire est la commande `make` qui utilise un fichier de configuration `Makefile` afin de savoir quels sont les fichiers à compiler pour créer un exécutable.

Vous allez maintenant séparer votre code en plusieurs fichiers et réaliser un fichier `Makefile` qui sera utilisé par la commande `make` pour compiler. Voici les fichiers à construire, dans un nouveau répertoire, à partir du programme que vous venez d'écrire :

1. un fichier d'entête `fractale.h`, qui contient les prototypes des fonctions et les définitions des pseudo constantes de type `#define M 6`.
2. un fichier source `fractale.c`, qui contient le code source des fonctions. Il est identique à celui que vous venez de faire, si ce n'est qu'il ne contient pas de fonction principale `main`.
3. un fichier source `programme.c`, qui contient le code de la fonction `main`, identique à votre fonction précédente. Ce fichier doit inclure le fichier d'entête `fractale.h`, mais aucun code autre que `main` [1].
4. un fichier `Makefile`, dans lequel vous allez définir les règles utiles à la compilation de vos deux fichiers sources pour construire un exécutable :
   (a) définir une variable CFLAGS pour les options de compilation `CFLAGS=-c -g -Wall`
   (b) définir une variable LDFLAGS pour les options d'édition des liens `LDFLAGS=-lm` pour la bibliothèque mathématique [2]
   (c) définir la règle de construction de votre exécutable, baptisé `programme`.

   ```
   programme : fractale.o programme.o
    gcc $(LDFLAGS) fractale.o programme.o -o programme
   ```

   Cette règle indique que l'exécutable `programme` dépend des fichiers `programme.o` et `fractale.o`. Si l'un des deux est plus récent que `programme`, alors il faut exécuter la commande située en dessous (`gcc $(LDFLAGS) fractale.o programme.o -o programme`) qui réalise l'édition des liens de `fractale.o` et de `programme.o` pour créer l'exécutable `programme`.
   (d) définir la règle de compilation du fichier source `fractale.c`.

   ```
   fractale.o : fractale.c
    gcc $(CFLAGS) fractale.c
   ```

   Cette règle indique que le fichier `fractale.o` dépend du fichier `fractale.c`. Si ce dernier est plus récent que `fractale.o`, alors il faut exécuter la commande située en dessous (`gcc $(CFLAGS) fractale.c`) qui réalise la compilation de `fractale.c` pour créer le fichier binaire `fractale.o`.
   (e) définir la règle de compilation du fichier source `programme.c`

   ```
   programme.o : programme.c
    gcc $(CFLAGS) programme.c
   ```

Sauver votre fichier `Makefile` puis compiler avec la commande `make`

1. dans cet exemple basique bien sûr
2. qui est en fait le fichier `libm.a`

4 Version graphique

4.1 Bibliothèque phelma

Vous devez utiliser la bibliothèque graphique phelmagtk pour réaliser l'affichage graphique. Ce sont simplement des fonctions qui sont destinées à simplifier le dessin dans une fenêtre graphique, en se basant sur la bibliothèque `gtk-3`.

Vous pouvez créer une fenêtre graphique dans laquelle votre fonction récursive va dessiner quand cela sera utile. Cette fenêtre, créée dans le programme principal, sera passée comme paramètre à la fonction récursive pour que celle-ci puisse dessiner dedans.

Plusieurs étapes sont nécessaires pour dessiner dans une fenêtre graphique, car il faut initialiser le système graphique, une variable pour manipuler la fenêtre graphique, créer une fenêtre graphique. Ensuite, on peut alors dessiner, afficher une image, du texte dans cette fenêtre.

1. déclaration d'une variable de type `GtkWindowPhelma* f1;`
2. initialisation du systeme graphique `ph_gtk_init`.
3. création d'une fenêtre vide par la fonction `ph_gtk_create_windows`.
   Cette fonction prend en paramètres deux entiers `nbc` et `nbl`, qui indiquent respectivement le nombre de colonnes (largeur) et de lignes (hauteur) de la fenêtre à créer, ainsi qu'un paramètre indiquant si la fenêtre peut être redimensionnée `PH_RESIZE` ou pas `PH_NO_RESIZE`. Elle retourne un pointeur vers la fenêtre créée ou `NULL` si cette ouverture est impossible.
4. dessin d'un trait dans la fenêtre graphique : `ph_gtk_draw_line`
   Cette fonction prend 6 paramètres. On l'appelle de la façon suivante :
   `ph_gtk_draw_line(f1, xdep, ydep, xarr, yarr, color);`
   Elle trace une droite de couleur `color` dans la fenêtre `f1`, entre les points `(xdep,ydep)` et `(xarr,yarr)`.

Vous retrouvez ces trois étapes dans le programme contenu dans le fichier `exemple.c`. Ce dernier réalise les actions suivantes :

1. il crée une fenêtre vide,
2. il appelle une fonction qui trace une ligne cyan entre les points de coordonnées $(x = 10, y = 0)$ et $(x = 300, y = 50)$ sur la fenêtre passée en paramètre.
3. il met l'écran à jour,
4. il attend un retour chariot pour se terminer.

FIGURE 3 – Dessin d'un trait dans la fenêtre graphique (l'origine (0,0) se trouve en haut à gauche).

Pour compiler, il faut copier le fichier `Makefile` (voir le site des tp) et utiliser la commande unix `make exemple` pour créer un exécutable. On lance l'exécution par la commande `./exemple`.

```
#include <phgtk.h>

void dessinerquelquechose(GtkWindowPhelma* f) {
    /* Affichage : dessiner des objets sur la fenetre f1
      Ligne noire entre les points x=10,y=0) et (x=300,y=50)
      Attention, la ligne n'est pas encore visible a l'ecran */
  couleur=PH_CYAN ;
  ph_gtk_draw_line(f1,10,0,300,50,couleur);
    /*
      On peut aussi dessiner des carres, des cercles
      et meme des fractales recursives
    */
}

int main (int ac, char **av) { unsigned int couleur ;
          /* Variable permettant de manipuler une fenetre */
  GtkWindowPhelma* f1=NULL;
```

```
  /* Initialisation du systeme graphique */
  if ( 0 != ph_gtk_init()) {
    fprintf(stderr, "%s\n", "Initialisation graphique impossible");
    exit(EXIT_FAILURE);
  }
     /*
       Creation d'une fenetre de dimension 300x200,
       */
  f1=ph_gtk_create_windows(300,100,PH_NO_RESIZE) ;

         /* On verifie que la fenetre est ouverte ;
         sinon, arret du programme */
  if ( f1== NULL) { printf("Erreur phelmagtk\n"); exit(EXIT_FAILURE);  }

  dessinerquelquechose(f1);

  /* Met a jour l'ecran : la ligne apparait */
  ph_gtk_show_window(f1);

  /* on attend, sinon le programme se termine et la fenetre disparait trop vite*/
  printf("Appuyer sur Return ou ENtree dans le terminal pour continuer\n");
  ph_gtk_display_until_return_pressed_in_terminal();

  /* On detruit la fenetre */
  f1=ph_gtk_destroy_windows(f1);
  /* Et on quitte */
  ph_gtk_quit();
}
```

4.2 Travail à réaliser

1. Quels sont les paramètres de la fonction dont le rôle est de dessiner la courbe précédente à l'ordre n sur une fenêtre à l'écran ?
2. Ecrire une fonction puis un programme dessinant la courbe précédente à l'ordre n, la valeur de n étant saisie au clavier.

   Votre nouvelle fonction `main` doit donc initialiser le système graphique, créer une fenêtre graphique et appeler votre fonction récursive de fractale. Ne pas oublier de bloquer en attente le programme ensuite, afin d'avoir le temps de voir le dessin.

   Cette fonction récursive prend maintenant un paramètre supplémentaire, la fenêtre graphique dans laquelle elle devra dessiner. Attention, le point origine de coordonnées (0,0) est en haut à gauche ! Vous ne pouvez pas dessiner si vous avez des coordonnées négatives. Pensez donc à donner des coordonnées de départ cohérentes.
3. Adapter le fichier `Makefile` à votre programme.

5 Pour ceux qui veulent aller plus loin, récursivité et jeu : Les Monstres et les globes

Dans les problèmes précédents, la récursivité est contrôlée par un paramètre qui décroit (ou croit selon le cas), ce qui permet d'arrêter les appels récursifs de manière certaine.

Dans certains problèmes, en particulier de jeu, on explore de manière récursive des solutions potentielles, mais sans paramètre de contrôle. Le principe est le même, on considère un mouvement possible qui donne une solution partielle, et on regarde si ce mouvement amène à une solution par un appel récursif. Si ce n'est pas le cas, on annule ce mouvement (on appelle cela du backtracking) et on cherche un autre mouvement.

Il faut cependant prendre garde à ne pas retomber plusieurs fois sur la même solution partielle : dans ce cas, il y a un risque de boucle infinie entre les différentes solutions partielles. On stocke de ce fait les

solutions partielles déjà examinées auparavant, et avant un mouvement, on regarde si on n'est pas déjà tombé sur cette solution partielle.

Règle du jeu

Il y a trois monstres et trois globes, chacun de trois tailles différentes (petit, moyen, grand). Au départ, les globes sont répartis aléatoirement entre les monstres. Le but est que chaque monstre porte le globe correspondant à sa taille.

Les globes sont transférés entre les monstres en respectant les trois règles suivantes :

1. un seul globe peut être transféré à la fois,
2. si un monstre détient plus d'un globe, seul le plus grand peut être transféré,
3. un globe ne peut pas être transféré à un monstre détenant un globe plus grand

Représentation des monstres et des globes

Une solution pour représenter monstres et globes est d'utiliser une matrice dont les éléments valent 1 ou 0. Les lignes sont les indices des monstres (0 pour petit, 1 pour le moyen et 2 pour le grand). Les colonnes correspondent à la taille des globes (0 pour petit, 1 pour le moyen et 2 pour le grand). La valeur 1 à l'indice i,j signifie que le monstre i porte le globe j. Une situation est donc donnée par cette matrice dont 3 éléments sur 3 lignes différentes valent 1, les autres valant 0. La situation gagnante est donnée par la diagonale à 1.

Tableau des situations déjà examinées

Dans un cadre général, on utiliserait ici soit une table de hachage, soit une liste. Dans le cas qui nous intéresse, une situation contient au plus 9 bits d'informations : présence ou absence d'un globe pour chaque monstre. On peut donc coder une configuration avec un entier sur 9 bits : 3 bits pour le monstre 0 (présence ou absence des globes 0,1,2), 3 bits pour le monstre 1, 3 bits pour le monstre 2. Ainsi, l'entier en base 2 $0b001100010$ indique que le grand monstre 2 porte le moyen globe 1 (les trois bits de poids faible, les plus à droite : 010), le moyen monstre 1 porte le grand globe 2 (les trois bits de poids 3 à 5 : 100) et le petit monstre 0 porte le petit globe 0 (les trois bits de poids 3 à 5 : 001). De meme, l'entier en base 2 $0b011100000$ indique que le grand monstre 2 ne porte pas de globe (les trois bits de poids faible, les plus à droite : 000), le moyen monstre 1 porte le grand globe 2 (les trois bits de poids 3 à 5 : 100) et le petit monstre 0 porte les petit et moyen globe (les trois bits de poids 3 à 5 : 011).

Un tableau d'octet [3] dont les indices sont au moins sur 9 bits [4] suffit donc à stocker toutes les situations possibles, et sera alors utilisé pour conserver toutes les situations déjà examinées pendant la recherche. Si une situation a déjà été examinée, alors la valeur du tableau pour l'indice correspondant à cette situation sera mise à 1. Pour calculer l'indice correspondant à une configuration, nous vous donnons la fonction `int monsterToIndex(char monster[][NUMBER])` qui calcule l'entier correspondant à la situation `monster`.

```
int monsterToIndex(char monster[][NUMBER]) {  int i,j,s=0;
  for (i=0; i<NUMBER;i++)
    for(j=0;j<NUMBER; j++) {
      s<<=1;
      s |= monster[i][j];
    }
  return s;
}
```

Pour résoudre le problème, vous aurez sans doute besoin des fonctions suivantes (4 ou 5 lignes pour chaque fonction) qui vous faciliteront l'écriture de la procédure récursive.

— `int indexMaxGlobes(int n, char monster[][NUMBER])` : retourne l'indice du plus grand globe du monstre n, ou -1 s'il n'a pas de globe

3. En réalité, on pourrait utiliser 1 seul bit pour indiquer que la configuration est vue et stocker 8 configurations sur un octet

4. déclaré par `char alreadyExamined[1<<NUMBER*NUMBER]`

— `int isMovePossible(int dep, int dest, char monster[][NUMBER])` : retourne 1 si le monstre *dep* peut donner un globe au monstre *dest*

— `void moveGlobles(int dep, int dest, char monster[][NUMBER])` : bouge un globe du monstre *dep* vers le monstre *dest*

— `void printMonster(char monster[][NUMBER])` : affiche les monstres et leurs globes

— `int success(char monster[][NUMBER])` : retourne 1 si on a réussi (diagonale à 1) ou 0 sinon

— `int mag(char monster[][NUMBER], char alreadyExamined[])` la fonction récursive qui cherche la solution, un peu plus longue.

Algorithm 2 Positionner les globes sur les monstres : tableau des monstres, tableau des situations deja vues

```
1: Si chaque monstre porte son globe Alors
2:    /* C'est gagne, */
3:    Retourner 1
4: Sinon
5:    Si le tableau des monstres a deja ete examine Alors
6:       /* C'est pas gagne, on a deja vu ce cas */
7:       Retourner 0
8:    Sinon
9:       Mettre le tableau des monstres dans le tableau des situations deja vues
10:      Pour toutes les paires i,j de monstres Faire
11:         Si le monstre i peut donner son globe au monstre j Alors
12:            Effectuer le mouvement de globe entre i et j
13:              /* Il faut maintenant continuer recursivement a deplacer les globes */
14:            Si positionner les globes sur les bons monstres == 1 Alors
15:               /* C'est reussi, */
16:               Afficher le mouvement
17:               retourner 1
18:            Sinon
19:               /*
20:                  La ligne suivante restaure la configuration des monsters
21:                  On annule le mouvement précédent de i vers j
22:               */
23:               Remettre le globe du monstre j au monstre i
24:               /* et on passe a la paire i,j suivante */
25:            Fin Si
26:         Fin Si
27:      Fin Pour
28:   Fin Si
29: Fin Si
```

6 Erreurs fréquentes

Quelques erreurs communes