Programmation
TP 6

Juliusz Chroboczek

25 octobre 2023

Il était jadis beaucoup plus facile qu'aujourd'hui d'afficher des graphiques à l'écran de l'ordinateur. L'écran était constitué d'une matrice de *pixels (picture elements)*, typiquement 200 lignes de 320 colonnes, et l'instruction `PLOT` permettait d'allumer individuellement chaque pixel.

Pour faire des graphiques aujourd'hui, il faut passer par une bibliothèque graphique compliquée qui parle à un système de fenêtres asynchrone qui parle à une couche d'abstraction matérielle qui interagit avec le matériel graphique. Pour vous permettre d'afficher des graphiques simples sans comprendre la phrase précédente, je vous fournis une petite bibliothèque qui permet de contrôler individuellement les pixels d'une fenêtre, comme au temps de grand père.

La bibliothèque que je vous fournis permet de manipuler une fenêtre consistant d'une matrice de pixels. Chaque pixel est identifié par une paire de coordonnées (x, y) positives ; l'axe des ordonnées est orienté vers le bas (le contraire de la convention utilisée en mathématiques). Chaque pixel a une couleur, définie par un triplet (r, g, b) d'entiers compris entre 0 et 255, et qui représentent, respectivement, l'intensité de la lumière rouge, verte et bleue [1]. Le noir est donc représenté par le triplet $(0, 0, 0)$, le blanc par $(255, 255, 255)$, et le rouge pur par $(255, 0, 0)$.

La bibliothèque contient trois fonctions :

- `int gr_init(int width, int height)`, qui prépare la bibliothèque pour manipuler une fenêtre de taille `width` × `height` entièrement remplie de pixels blancs ;
- `int gr_draw_point(int x, int y, int r, int g, int b)`, qui affecte la couleur `(r, g, b)` au pixel de coordonnées `(x, y)` ;
- `init gr_display()`, qui affiche la fenêtre et passe la main au système de fenêtres ; cette fonction retourne lorsque l'utilisateur ferme la fenêtre.

Un programme écrit avec cette bibliothèque a la structure suivante :

```
#include <gtk/gtk.h>
#include "graphiques.h"

int main() {
    gr_init(..., ...);
    gr_draw_point(...);
```

1. https://en.wikipedia.org/wiki/RGB_color_model

```
    gr_draw_point(...);
    ....
    gr_display();
    return 0;
}
```

Exercice 1. Si vous administrez votre propre système, installez d'abord le paquet `libgtk-3-dev` (il est déjà installé dans les salles de TP). Téléchargez l'archive

`https://www.irif.fr/~jch/enseignement/graphiques.tar.gz`

Compilez le programme d'exemple à l'aide de la commande `./compile` et testez-le. Lisez le programme d'exemple `exemple.c` et le script `compile`. Ne lisez pas le fichier `graphiques.c`.

Exercice 2.

1. Écrivez un programme qui affiche en noir le segment qui joint les points de coordonnées (100, 100) et (100, 200).
2. Écrivez un programme qui affiche en rouge le segment qui joint les points de coordonnées (100, 100) et (200, 100).
3. Écrivez un programme qui affiche en vert le segment qui joint les points de coordonnées (100, 200) et (200, 300).
4. Écrivez un programme qui affiche en mauve le segment qui joint les points de coordonnées (200, 100) et (100, 200).
5. Écrivez un programme qui affiche en amarante le rectange plein qui a des coins aux coordonnées (100, 100) et (300, 250).

Exercice 3. Écrivez un programme qui ouvre une fenêtre de taille 800×400 puis qui affiche une sinusoïde. Pour cela, pour chaque entier x allant de 0 à 799, vous afficherez le point de coordonnées

$$(x, 200 + 200\cos(x \times 2\pi/800)).$$

Vous pourrez vous servir de la constante `M_PI` et de la fonction `cos` ; il faudra inclure l'entête `math.h` et ajouter l'option `-lm` à la fin de la ligne de commande de compilation.

Exercice 4. Écrivez un programme qui affiche un graphique représentant le temps de vol de la n-ième suite de syracuse en fonction de n, pour n allant de 1 à 200. Il faudra probablement doubler les pixels pour que le graphique soit lisible.

Exercice 5. Écrivez un programme qui affiche un cercle.

Exercice 6.

1. Écrivez un programme qui affiche un segment horizontal de longueur 256 dont le i-ème pixel (en comptant à partir de 0) est de la couleur $(i, 0, 255 - i)$.
2. Écrivez un programme qui affiche un carré plein de taille 256×256 rempli d'un dégradé tel que :
   - le coin nord-ouest est noir ;

– le coin nord-est est rouge ;
– le coin sud-ouest est vert.

Exercice 7. L'entête `complex.h` définit le type « `complex double` » des nombres complexes à virgule flottante. Il définit aussi la constante « `I` » (l'unité imaginaire), ce qui permet de construire des nombres complexes arbitraires, par exemple `2.0 + I * 3.0`.

Pour chaque nombre complexe c, on définit la suite $(z^{(c)})$:

$$z_0^{(c)} = 0;$$
$$z_{n+1}^{(c)} = (z_n^{(c)})^2 + c.$$

1. Écrivez une fonction

   ```
   int mandelbrot(complex double c);
   ```

   qui retourne le nombre d'itérations n nécessaires pour que $z_n^{(c)}$ ait un module supérieur à 2 ; si le module reste inférieur à 2 au bout de 1000 opérations, votre fonction retournera -1. Pour éviter de calculer des racines carrées (ce qui prend du temps), vous pourrez utiliser le test suivant :

   ```
   creal(z) * creal(z) + cimag(z) * cimag(z) < 4
   ```

On dit qu'un nombre complexe c est dans l'*ensemble de Mandelbrot* [2] si la suite $z^{(c)}$ est bornée. Une bonne approximation de l'ensemble de Mandelbrot est donnée par l'itération ci-dessus : un point est dans l'*ensemble de Mandelbrot approché* si $|z^{(c)}|$ n'a pas dépassé 2 en 1000 itérations.

2. Écrivez un programme qui créé une fenêtre de taille 400×400 et qui colore le pixel de coordonnées (x, y) en noir si et seulement si le nombre

   $$\frac{x - 200}{100} + \frac{y - 200}{100} i$$

   est dans l'ensemble de Mandelbrot approché, en blanc sinon.
3. Modifiez votre programme pour que la couleur d'un pixel qui correspond à un nombre qui n'est pas dans l'ensemble de Mandelbrot en une couleur qui dépend du nombre d'itérations nécessaires pour que le module dépasse 2.

2. `https://en.wikipedia.org/wiki/Mandelbrot_set`