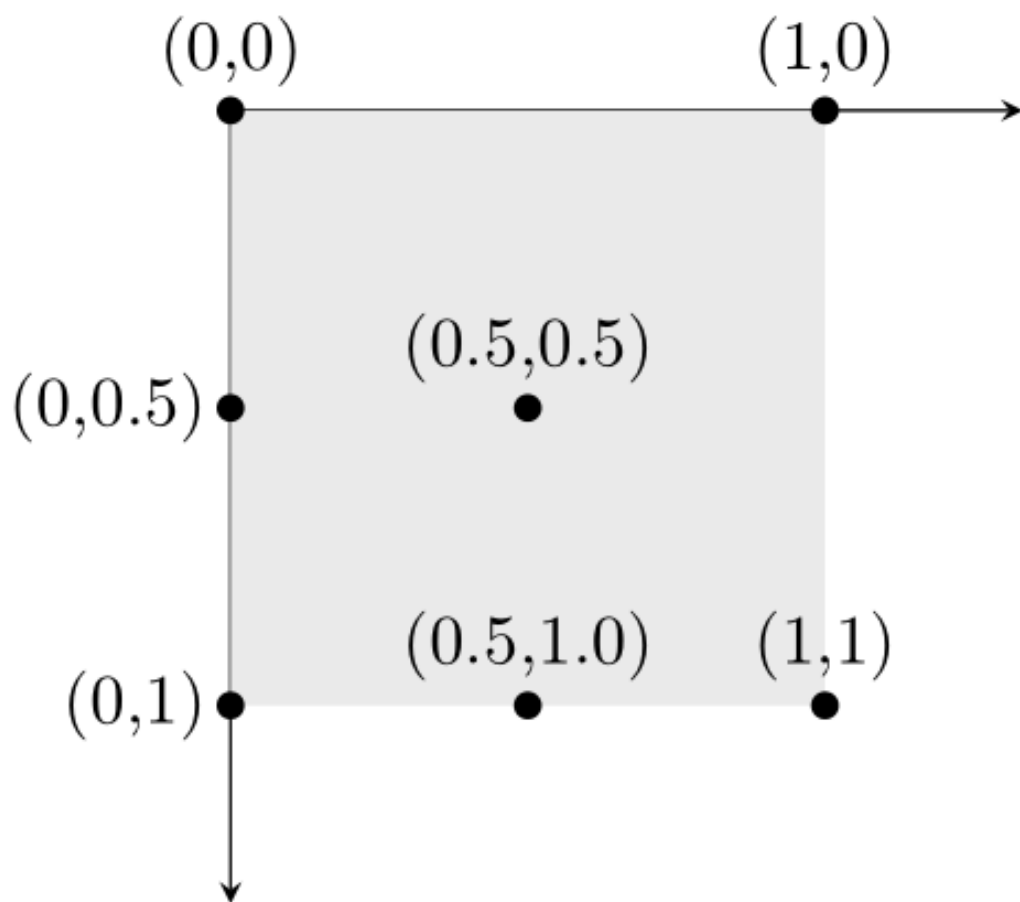


Art-C

October 7, 2021

1 Introduction

Ce TP prend la suite du TP4.5 dans son esprit mais va changer le point de vue. Dans le TP4.5 on a vu directement manipulé l'écran comme un tableau de pixels, avec ainsi des coordonnées entières. Ici on va considérer un écran logique qui correspond à la partie $[0,1]^2 \subset \mathbb{R}^2$. Contrairement à la convention en mathématiques, l'axe des y est orienté vers le bas. Voici donc quelques exemples de coordonnées dans ce système, la zone grisée correspondant à la zone du dessin :



L'avantage d'avoir des coordonnées indépendantes de la résolution c'est qu'elles vont nous permettre

de reproduire le dessin en plus ou moins grand selon la taille de la fenêtre.

1.1 Mise en place

On va procéder comme pour les TPs précédents, je vous indique donc les commandes à taper mais si vous avez compris vous devriez déjà les connaître. Ouvrez un terminal, assurez-vous d'être dans votre `home` en tapant `cd` si jamais il est déjà ouvert. **Vous pouvez aussi créer un répertoire TP pour placer les TP.** Et tapez maintenant :

- `curl -O http://marc.de-falco.fr/assets/tp05.zip`
- `unzip -d tp05 tp05.zip`
- `cd tp05`
- `make`
- `./tp05`

Déjà, si ça se passe mal appelez-moi, mais normalement vous devriez voir une fenêtre avec quatre petites zones qui vont correspondre à quatre *tableaux* qu'on va vous demander de réaliser. L'objectif est de vous approprier C donc utilisez au maximum, les variables, les boucles, ...

1.2 Paramètres de lancement

Vous pouvez lancer le programme en tapant juste `./tp05` mais il est aussi possible de lui fournir de 1 à 3 arguments. Par défaut, il ouvre une fenêtre de 800×800 pixels mais dont l'image affichée est 2 fois plus petite, et la disposition initiale correspond aux quatre tableaux. Vous pouvez basculer entre quatre tableaux et chacun des tableaux en utilisant la touche **Espace**.

Il est aussi possible d'appeler le programme ainsi :

- `./tp05 pixels` où `pixels` est un nombre entier qui va indiquer la résolution de la fenêtre, par défaut c'est 800
- `./tp05 pixels zoom` où `zoom` est le facteur de grandissement, par défaut c'est 2. Pour être avoir un pixel dessiné = un pixel affiché, il faut passer 1. Plus `zoom` est petit et plus ça va prendre du temps pour afficher en cas d'animations.
- `./tp05 pixels zoom config` où `config` vaut soit 0 pour quatre tableaux, soit `i` dans 1,2,3,4 pour afficher le tableau correspondant. Par défaut, c'est 0.

Vous pourrez ainsi adapter la commande à lancer en fonction des capacités de vos ordinateurs et de la question à traiter.

Astuce il est possible de retrouver les commandes précédemment entrées dans le shell à l'aide des touches fléchées haut et bas.

1.3 Organisation du code

Le code est organisé sur plusieurs fichiers pour la première fois. On a ainsi :

- `tp05.c` qui contient la fonction `main` et se charge de gérer la création de la fenêtre et l'affichage
- `primitives.c` qui contient les primitives de dessin qu'on va utiliser dans ce TP et qui seront décrites au paragraphe suivant
- `reproductions.c` qui est le fichier que vous allez modifier et qui contient les quatre fonctions de dessins appelées dans `tp05.c`

On a également `primitives.h` et `reproductions.h` pour permettre de déclarer les fonctions pouvant être appelées ailleurs.

1.4 Primitives

Tout d'abord, notons que nous allons travailler avec des coordonnées en nombre flottant. Pour cela, on ne va pas utiliser le type `float` qui correspond à des flottants sur 32 bit mais le type `double` qui correspond à des flottants sur 64 bits. Ce sont en fait les nombres flottants utilisés par le type `float` dans Python. En C, contrairement à OCaml, on ne fait pas de différences sur les opérations. Il n'y a donc rien à faire mis à part déclarer des `double`. Les fonctions de mathématiques (`sin`, `cos`, `sqrt`, ...) utilisent aussi des doubles par défaut.

Dans le TP 4.5, on a vu que c'était pénible de passer les couleurs à chaque fonction avec trois arguments. Ici, on va se servir de la possibilité de définir ses types en C pour définir un type `couleur`. Pour le moment, vous n'avez pas besoin de savoir comment on fait. Tout ce que vous avez besoin de savoir c'est :

- qu'on déclare une variable de ce type en écrivant juste `couleur c`;
- qu'il y a une fonction `couleur rgb(unsigned int r, unsigned int g, unsigned int b)`; qui permet de créer une couleur. On peut donc écrire `couleur rouge = rgb(255, 0, 0)`;
- on peut aussi directement faire un appel à `rgb` dans un appel de fonction. Ainsi on peut écrire `ligne(0, 0, 1, 1, rgb(255, 0, 0))`; ou `ligne(0, 0, 1, 1, rouge)`; si on définit la couleur dans une variable. L'avantage étant d'éviter de se répéter et de donner un nom signifiant aux couleurs.

Pour dessiner, vous disposez des fonctions :

- `void ligne(double x1, double y1, double x2, double y2, couleur c)`; qui trace une ligne de couleur `c` entre les points `(x1,y1)` et `(x2,y2)`.
- `void cercle(double x, double y, double r, couleur c)`; qui trace une cercle de couleur `c`, de centre `(x,y)` et de rayon `r`. Ici le rayon correspond aux dimensions dans `[0,1]`. Ainsi, pour faire un cercle inscrit dans le carré du dessin, il devra être de rayon `0.5` et on écrira `cercle(0, 0, 0.5, rouge)`;
- `void cercle_plein(double x, double y, double r, couleur c)`; comme `cercle` mais on remplit l'intérieur.
- `void remplit(double x, double y, couleur c)`; qui s'explique assez facilement avec l'outil que vous avez dû utiliser dans un logiciel de dessin : le *pot de peinture*. On part du pixel situé en `(x,y)` et on peint tout autour de la couleur `c` jusqu'à rencontrer une couleur différente de la couleur initiale de `(x,y)`. Un exemple vous est fourni dans `tableau1`.
- `couleur melange(double t, couleur c1, couleur c2)`; ici `t` est dans `[0,1]` et va permettre de faire un mélange entre les deux couleurs. Si `t` vaut `0` on renvoie `c1`, si `t` vaut `1` on renvoie `c2` et pour les autres valeurs on fait une interpolation linéaire, c'est-à-dire un mélange entre `1-t` de `c1` et `t` de `c2`. Ça va permettre de faire des effets comme des dégradés.
- `double delta()`; on a vu qu'on allait dessiner dans un espace indépendant de la résolution mais si on veut faire des lignes qui se touchent, il faudrait connaître la taille d'un pixel dans ces coordonnées. C'est exactement ce que renvoie cette fonction : la valeur δ qui correspond à un pixel sur l'écran.

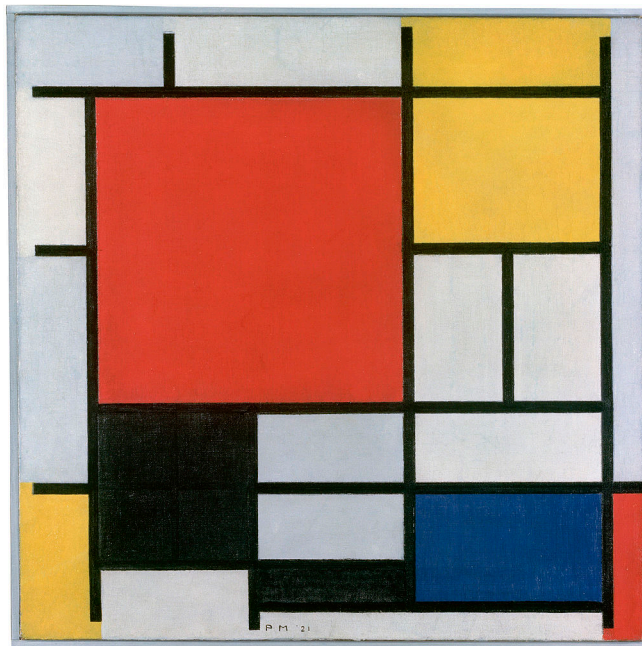
1.5 Exemples

Le fichier `reproductions.c` a déjà des programmes écrits pour vous permettre de comprendre le fonctionnement des fonctions précédentes.

1.6 Note de style

Dans la suite, on va vous demander de reproduire des tableaux avec des programmes. L'objectif n'est pas de faire douze mille lignes idiotes mais de trouver une logique. Parfois en utilisant des boucles, des sous-fonctions, des tableaux pour ne pas se répéter. Il y aura donc une note de style en lisant votre code.

2 Tableau 1 - Piet Mondrian



Recopiez votre fonction `tableau1` ici :

[]:

3 Tableau 2 - Victor Vasarely



Recopiez votre fonction `tableau2` ici :

[]:

4 Tableau 3 - Vassily Kandinsky



Recopiez votre fonction `tableau3` ici :

[]:

5 Animation

Vous avez pu voir que chaque fonction `tableaui` prend un paramètre `time`. Celui-ci est un nombre flottant dans $[0, 1]$ qui va de 0 à 1 dans une minute puis reprend la valeur 0. Vous pouvez vous en servir pour changer votre tableau en fonction du temps en calibrant l'animation sur une minute.

On pourra utiliser les fonctions de `math.h` comme `sin`, `cos`, ...

Par exemple, il suffit de considérer $\theta = 2\pi \frac{60\text{time}}{N}$ pour parcourir $[2k\pi, 2(k+1)\pi]$ en exactement N

secondes. Il suffit alors de calculer $\cos(\theta)$, $\sin(\theta)$ pour avoir un point qui tourne autour d'un cercle en N secondes.

Choisissez un des tableaux précédents et animez-le. Recopiez votre nouvelle fonction ici :

[]:

6 Tableau 4 : Vous

Voilà maintenant vous êtes libres de faire votre propre tableau et celui-ci devra être animé. Je vous ai mis un exemple, il n'a aucune raison de vous servir de base. Vous pouvez le supprimer ou le déplacer.

Recopiez votre fonction `tableau4` ici :

[]: