TD 6

Tutoriel SFML

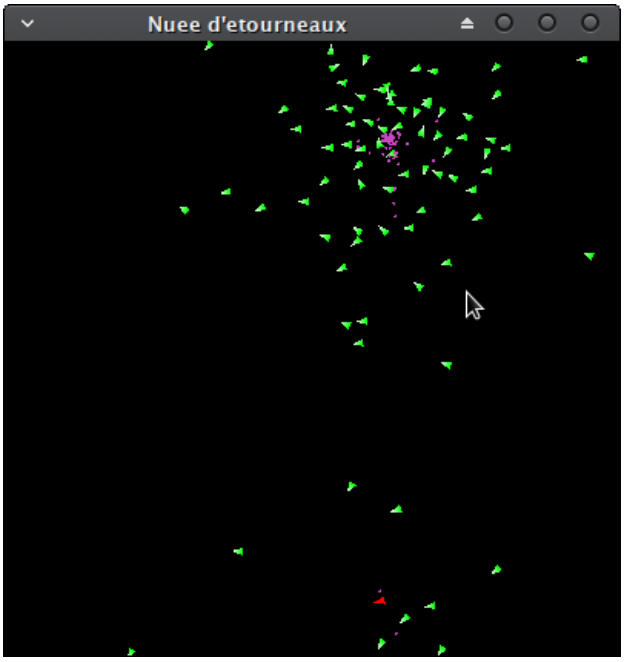
Ce tutoriel a pour but de prendre en main une bibliothèque graphique 2D : la SFML.

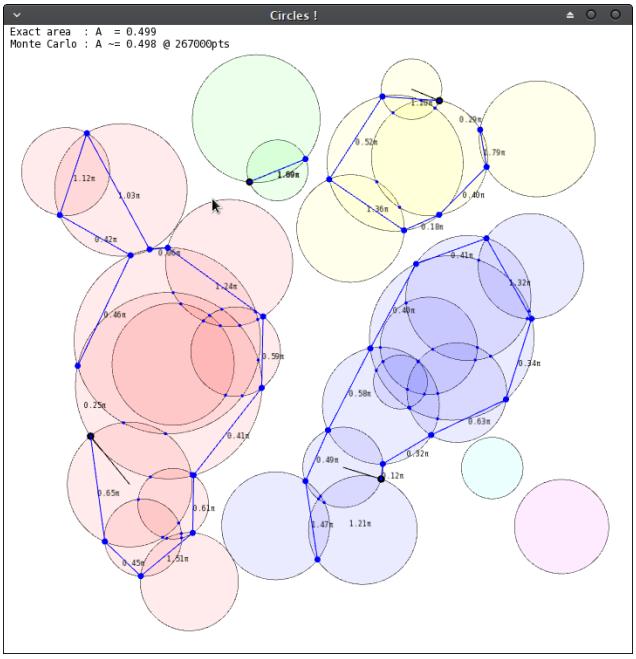
6.1 Introduction

La SFML est une bibliothèque C++ d'affichage 2D en temps réel, typiquement pour faire des petits jeux vidéos 2D. Mais c'est aussi très utile pour les simulations en physique !

Le principe est très simple : on a une fenêtre, une boucle qui s'exécute à chaque frame (par exemple 30 fois par secondes) et qui affiche des lignes, cercles, rectanges, textes... de toutes tailles et couleurs. Ce n'est toutefois pas conçu pour une utilisation scientifique, et il n'y a rien pour plotter des courbes ou des matrices par exemple. Enfin, il n'y a pas de support pour la 3D, il faudra utiliser d'autres bibliothèques pour cela.

Exemples de ce qu'on peut faire rapidement avec :





6.2 Installation de la SFML

Une bibliothèque (*library* en anglais) n'est rien d'autre qu'un ensemble de code cohérent, auto-contenu et disposant d'une interface claire et utilisable même pour quelqu'un qui ne maîtrise pas les rouages internes.

En C++, les fonctions et classes sont exposées à travers les fichiers d'en-tête (*headers*, `.h` ou `.hpp`), et l'implémentation est en général compilée sous forme d'un fichier objet. Pour utiliser une bibliothèque, il faut
- inclure le ou les fichiers d'en-tête dans le code aux endroits où on utilise la bibliothèque (`#include <mylib.h>` par exemple)
- ajouter le ou les fichiers objets lors de l'édition de lien (`g++ ... -lmylib -o ...` par exemple)

Pour les langages compilés, dont le C++ , il existe deux types de fichiers objets :
- les bibliothèques *statiques* (fichiers `.a`), qui ne sont rien d'autre qu'une archive de `.o` qui sont *copiés* et liés dans l'exécutable comme n'importe quel autre `.o` ; l'exécutable contient alors tout ce qui est nécessaire pour l'exécution, et est facile à distribuer
- les bibliothèques *dynamiques*, ou *partagées* (fichiers `.so` sur Linux, `.dylib` sur macOS, `.dll` sur Windows), qui sont des fichiers objets qui ne sont pas copiés dans l'exécutable, et qui doivent donc être livrés avec l'exécutable ou être déjà installés au niveau du système ; c'est très souvent le cas de la bibliothèque standard C++ , qu'il serait inefficace de re-livrer pour chaque programme [1].

Le compilateur ira chercher les fichiers d'en-tête et objets (et le chargeur de programme les bibliotèques paratgées) à des endroits prédéfinis et standardisés. Sur Linux et macOS, les fichiers d'en-tête sont en général dans `/usr/include` ou `/usr/local/include`, et les fichiers objets dans `/usr/lib` ou `/usr/local/lib`. Il est toutefois possible de dire au compilateur d'aller chercher ailleurs, comme on le verra.

Pour la plupart des langages de programmation, il y a trois façons d'installer une bibliothèque :

1. télécharger le code source, compiler la bibliothèque (si nécessaire), et placer les fichiers résultants au bon endroit (cet endroit varie suivant le langage)
2. télécharger la bibliothèque déjà compilée (on parle de fichiers binaires) et placer les fichiers au bon endroit
3. utiliser le gestionnaire de paquets du système ou du langage (lorsqu'il existe et lorsque la bibliothèque a été empaqutée par la communauté) faire ces opérations automatiquement (`pip install ...` en Python, `apt install ...` sur les systèmes Linux de la famille Debian/Ubuntu, `pacman -S ...` sur ArchLinux, `brew install ...` sur macOS...)

Ceux disposants de leur propre ordinateur préfèreront la dernière option. Sur les machines du $\hbar$, la SFML n'est pas installée et vous ne disposez pas des droits pour installer un paquet via `apt`. Il faut donc se rabattre sur une des deux premières options. La deuxième option est décrite sur la page `https://www.sfml-dev.org/tutorials/2.5/start-linux-fr.php`. La première est décrite dans la section 6.2.2 ci-dessous.

6.2.1 Installation sur un Linux de la famille Debian avec droits d'administrateur

On utilise le gestionnaire de paquets `apt`. Il suffit de faire `apt install libsfml-dev` avec droits d'administrateurs. Le postfixe `dev` est là pour indiquer que l'on souhaite installer les fichiers d'en-tête. On ne peut pas le faire sur les machines du $\hbar$ car on ne dispose pas des droits d'administrateurs.

6.2.2 Compilation et installation dans un dossier local

Il faut déjà télécharger le code source de la SMFL, qu'on peut trouver sur `https://www.sfml-dev.org/download-fr.php`. Choisir "Dernière version stable", puis "Tout OS" et "Code source". Une fois le téléchar-

1. C'est certes plus complexe. Mais il y a un gros avantage des bibliothèques partagées : si N programmes utilisent la même bibliothèque, le système d'exploitation n'a besoin de charger le `.so` qu'une seule fois en mémoire, ce qui ne serait pas le cas pour une édition de lien statique où, sans le savoir, le même code machine serait chargé N fois en mémoire. Ce joli mécanisme est exploité au maximum sur les systèmes Linux où la bibliothèques sont installées par le gestionnaire de paquet, et non livrées avec l'exécutable ; c'est moins le cas sur Windows et macOS. Autre avantage : les bibliothèques peuvent être mises à jour sans avoir à ré-installer le programme. Désavantage : il faut que l'ensemble soit cohérent, ce qui serait pénible sans un gestionnaire de paquets. D'une certaine manière, c'est aussi le cas dans l'écosystème Python, bien que les bibliothèques sont sous forme de code source et non de code machine.

gement effectuer, on ouvre un terminal et on se place dans le dossier de téléchargement.
Ensuite, on décompresse et on se place dans le dossier :

```
$ unzip SFML-*-sources.zip
$ cd SFML-*
$ ls
```

L'auteur de la bibliothèque SMFL a choisi `cmake` pour la construction de la bibliothèque. C'est un outil très populaire pour générer des `Makefile` facilement [2]. La bonne pratique est de créer un dossier `build` et de se placer dedans pour compiler la bibliothèque. On appelle alors `cmake` :

```
$ mkdir build && cd build
$ cmake -DSFML_BUILD_AUDIO=False -DBUILD_SHARED_LIBS=True -DCMAKE_INSTALL_PREFIX=$HOME/usr
↪  -DSFML_PKGCONFIG_INSTALL_PREFIX=$HOME/usr/lib/pkgconfig ..
```

avec les options suivantes :

- `SFML_BUILD_AUDIO=False` pour désactiver les fonctionnalité audio (il manque des bibliothèques sur le système des machines du $\hbar$ pour les utiliser)
- `BUILD_SHARED_LIBS=True` pour qu'une bibliothèque *partagée* soit générée
- `CMAKE_INSTALL_PREFIX=$HOME/usr` et `SFML_PKGCONFIG_INSTALL_PREFIX=$HOME/usr/lib/pkgconfig` pour que la bibliothèque soit intallée dans votre dossier utilisateur (`$HOME`), sous le préfixe `~/usr/` ; on ne peut en effet intaller la bibliothèque sous le préfixe standard `/usr/` car on n'a pas de droits d'administrateur sur les machines du $\hbar$...

Enfin, on compile et on installe les fichiers à l'endroit désiré :

```
$ make
$ make install
```

Regardez la sortie de `make install` et vérifiez que les fichiers ont étés installés au bon endroit. Passez un peu de temps à regarder l'organisation de `~/usr/`, ainsi que du répertoire système `/usr/`. En particulier, trouvez les fichiers objet de la SFML ainsi que les fichiers d'en-tête. La très grande majorité des bibliothèques sont organisées de cette sorte.

6.3 Premiers pas avec la SFML

On trouvera de bons tutoriels sur le site officiel : `https://www.sfml-dev.org/tutorials/2.5/index-fr.php`. Les sections qui nous concernent le plus sont "Ouvrir et gérer une fenêtre SFML", "Dessiner avec SFML", et "Dessiner des formes".

2. Il n'y a pas assez de temps pour introduire cet outil dans le cours, mais n'hésitez pas à l'utiliser pour vos projets !

Voici un exemple de `main()` typique :

```
#include <SFML/Graphics.hpp>

int main () {
    sf::ContextSettings settings; settings.antialiasingLevel = 8;  ← anti-aliasing pour faire joli
    sf::RenderWindow win (sf::VideoMode(400,400), "Mon super projet", settings);
    while (win.isOpen()) {  ← boucle principale
        win.clear();  ← 1. efface la fenêtre
        sf::Event event;
        while (win.pollEvent(event)) {  ← 2. traitement des évènements (souris, clavier...)
            if (event.type == sf::Event::Closed) window.close();
            if (event.type == sf::Event::KeyPressed) {
                switch (event.key.code) {
                    case sf::Keyboard::A:  dire_ahhh();  break;  ← touche A appuyée
                }
            }
        }
        Mon code de simulation (attention, afficher est lent : faire 100 ou 1000 pas de simulation par frame)
        sf::CircleShape cercle (10);  ← crée un cercle de 10 pixels de rayon
        cercle.setFillColor(sf::Color::Blue);  ← il sera bleu
        win.draw(cercle);  ← 3. dessine le cercle
        sf::VertexArray ligne (sf::LineStrip, 2);  ← crée un ligne (2 points (x1, y1) — (x2, y2))
        ligne[0] = sf::Vertex( sf::Vector2f(x1,y1), sf::Color(42) );
        ligne[1] = sf::Vertex( sf::Vector2f(x2,y2), sf::Color(24) );
        win.draw(ligne);  ← 4. dessine la ligne
        win.display();  ← 5. affiche la frame
    }
    return 0;
}
```

Créez un nouveau dossier de travail et créez un fichier `main.cpp` contenant l'exemple de la page "Dessiner avec SFML". À l'emplacement approprié dans le code, ajoutez

```
// construction d'un rectangle de largeur 120 et hauteur 50
sf::RectangleShape rectangle (sf::Vector2f(120, 50));
// change sa couleur en violet (R=1, G=0, B=1)
rectangle.setFillColor(sf::Color(255,0,255));
// définit sa position (haut gauche) à (x,y)=(150,50)
rectangle.setPosition(150, 50);
// dessin du rectange
window.draw(rectangle);
```

Note : Comme bien souvent dans les systèmes d'affichage, les positions (x, y) sont exprimées *en pixels par rapport au coin haut gauche*, contraire à l'usage des physiciens de placer l'origine en bas à gauche. Lorsque l'on augmente y, le rectangle descent sur l'écran !

Pour compiler ce programme de test, il va falloir dire au compilateur de prendre en compte la bibliothèque.

6.3.1 Avec une installation au niveau du système

C'est très simple. Comme la bibliothèque est installée à un emplacement standard, le compilateur sait déjà où chercher les fichiers d'en-tête et les fichiers objets. Il suffit juste de dire au linker de lier la bibliothèque à l'exécutable :

```
$ g++ -c main.cpp -o main.o  # compilation
$ g++ main.o -lsfml-graphics -lsfml-system -lsfml-window -o test.exe  # édition de liens
$ ./test.exe
```

Pour dire au linker de lier un fichier objet de bibliothèque, par exemple `libmachin.so`, on lui passe le flag `-lmachin`. La SFML est en fait séparée en plusieurs module, et pour un simple affichage graphique, il faut

- `sfml-system` (routines système)
- `sfml-window` (création de fenêtres dans l'interface graphique et contexte OpenGL)
- `sfml-graphics` (routines de dessin)

6.3.2 Avec une installation dans un dossier local

Dans ce cas, les fichiers de la bibliothèque ne sont pas dans un répertoire standard, et il faut dire au compilateur et au linker où ils te trouvent avec les flags `-I` et `-L`, le tilde `~/...` représentant le dossier personnel :

```
$ g++ -I$HOME/usr/include -c main.cpp -o main.o  # compilation
$ g++ main.o -L$HOME/usr/lib -lsfml-graphics -lsfml-system -lsfml-window -o test.exe  #
↪  édition de liens
$ ./test.exe
```

Essayez l'exécution. Pourquoi, à votre avis, le système d'exploitation refuse d'exécuter `test.exe` ? Pour résoudre le problème, le plus simple est sans doute de rajouter un flag `rpath` à l'édition de lien :

```
$ g++ -I$HOME/usr/include -c main.cpp -o main.o  # compilation
$ g++ main.o -L$HOME/usr/lib -lsfml-graphics -lsfml-system -lsfml-window
↪  -Wl,-rpath,$HOME/usr/lib -o test.exe  # édition de liens
$ ./test.exe
```

6.3.3 Animation

Ajoutez une varible globale `float anim;` que vous incrémentez de `0.01` à chaque itération de la boucle d'affichage. Faites osciller la position du rectangle en, par exemple, rajoutant `10 * sin(anim)` à la position x.

Correction

```cpp
#include <SFML/Graphics.hpp>
#include <cmath>
using namespace std;

int main () {
    // création de la fenêtre
    sf::RenderWindow window(sf::VideoMode(800, 600), "Mon super rectangle");

    float anim = 0;

    // on fait tourner le programme en boucle
    while (window.isOpen()) {
        // gestion des évènements (non nécessaire ici)
        // ...

        // effacement de la fenêtre en noir
        window.clear(sf::Color::Black);

        // c'est ici qu'on dessine tout
        // construction d'un rectangle de 120 par 50
        sf::RectangleShape rectangle (sf::Vector2f(120, 50));
        // change sa couleur en rouge (R=255, G=0, B=0)
        rectangle.setFillColor(sf::Color(255,0,255));
        // définit sa position à (x,y)=(150,50)
        rectangle.setPosition(150+20*sin(anim), 50);
        // dessin du rectange
        window.draw(rectangle);

        // affichage de tout ce qu'on a dessiné
        window.display();

        // animation
```

Correction

```
            anim += 0.01;
        }

        return 0;
    }
```