Dr. Edgar Huckert

08-2017   V1.0

SFML and conventional make files

I often write music programs: MIDI editors, converters and music notation editors. Normally I use C++ (occasionally also D) with *wxWidgets*. My goal is to write **portable** programs that run under Windows and Linux.

Music notation editors require a lot of graphic programming: you have to draw systems (5 horizontal lines), Text, symbols (like ff for fortissimo or different kinds of brackets) and notes with different kinds of flags or even partial rounded curves for the bindings between notes. I used with some succes the graphic methods contained in wxWidgets. The problem: the majority of these routines do something on the screen immediately after each call. This leads to a long series of synchronous calls which can be very time consuming. Even with wxWidgets I encountered some incompatiblities between Windows and Linux/GTK appear forcing you to make tricks.

I found that I could try a different approach for the screen output routines: using the portable *opengl* library or SDL something like SFML. My choice was finally SFML as it is the more modern approach with a higher level of abstraction. SFML ist basically a C++ API with bindings in many other programming languages. I manipulates rather the internal screen buffer before – after a series of calls - this buffer is redrawn. I have tested C++ and D (package DSFML).

You will find here some of my SFML test results with C++. The program is based on a short sample found in the SFML documentation. I enhanced it by adding oder modifying calls for

- drawing lines
- drawing rectangles
- drawing rotated texts
- drawing sprites
- simple animation using rotation
- basic performance measurement
- better error handling

I have not yet tested if and how SFML can be used for output to a printer.

My sample program uses nearly the complete set of **libraries** coming with SFML – only the *network* library is not used:

- window
- audio
- graphics

- system
- sfml

It is very strange that people requiring help in the internet forums are always **urged to use *cmake*** instead of the simple **make** build system. *Cmake* ist very complicated and requires a large installation. *Make* is very simple, very old and is normally bundled to the usual compilers (gcc/g++ in this case). Not to mention *Visual C* (Microsoft) and *Eclipse* with their huge environments. I have seen a lot of projects where people struggled more with *Visual C* or *Eclipse* than with the basic project problems. Worst of all build systems: *ant* used in JAVA projects.

Believe me: neither *cmake*, nor GUIs like *Eclipse*, nor *Visual C* are required in order to build SFML programs.

Make files describe the rules (also called „targets") to build programs by specifying dependencies between the program components. Consult Unix tutorials or books for an introduction to make files.

Here is a **conventional *make* file** for Windows using the MingW gcc/g++ compiler:

```
OBJS=sfml2.o
CFLAGS=-static -DWIN32 -Ic:\Huckert\src\SFML\SFML-2.4.2\include
The most important rule
LFLAGS=-Lc:\huckert\src\SFML\SFML-2.4.2\lib -lsfml-main -lsfml-
window -lsfml-graphics -lsfml-system -lsfml-audio

sfml2.exe: $(OBJS)
     g++ -o sfml2.exe $(CFLAGS) $(OBJS) $(LFLAGS)

sfml2.o:  sfml2.cpp
     g++ -c $(CFLAGS) -o sfml2.o sfml2.cpp

clean:
     del $(OBJS) sfml2.exe
```

And here is a very similar make file for Linux/Debian using the GNU gcc/G++ compiler:

```
OBJS=sfml2.o
CFLAGS
LFLAGS=-L/usr/lib/x86_64-linux-gnu -lsfml-window -lsfml-
graphics -lsfml-system -lsfml-audio

sfml2: $(OBJS)
     g++ -o sfml2 $(CFLAGS) $(OBJS) $(LFLAGS)

sfml2.o: sfml2.cpp
     g++ -c $(CFLAGS) -o sfml2.o sfml2.cpp
```

```
    clean:
          rm $(OBJS) sfml2
```

You will have to control and modify the flags (variables *CFLAGS, LFLAGS*), the source and object names and the pathes. You can invoke the build process by calling *make* with the „-f" flag on the command line (shell, „black" window):

```
     make -f sfml2.mak
```

The most important rule for working with *make* files: use real tabs instead of spaces for indentation! Indentation is an essential mechanism in *make* files. If you look at other *make* files as produced for large programs / packages the you will find that the *make* files are much more complex and sometimes unreadable. The reason is: these files are often generated by programs or scripts. The majority of them are more complex than really needed.

Under Windows I had to modify my execution path (see *setpath.bat* in the ZIP file). For whatever reason a message „sfml-graphics-2.dll missing" appeared on the screen when starting the executable.

Here is the source code for this SFML program. You will find the source, the *make* files and the sample data files in the ZIP file.

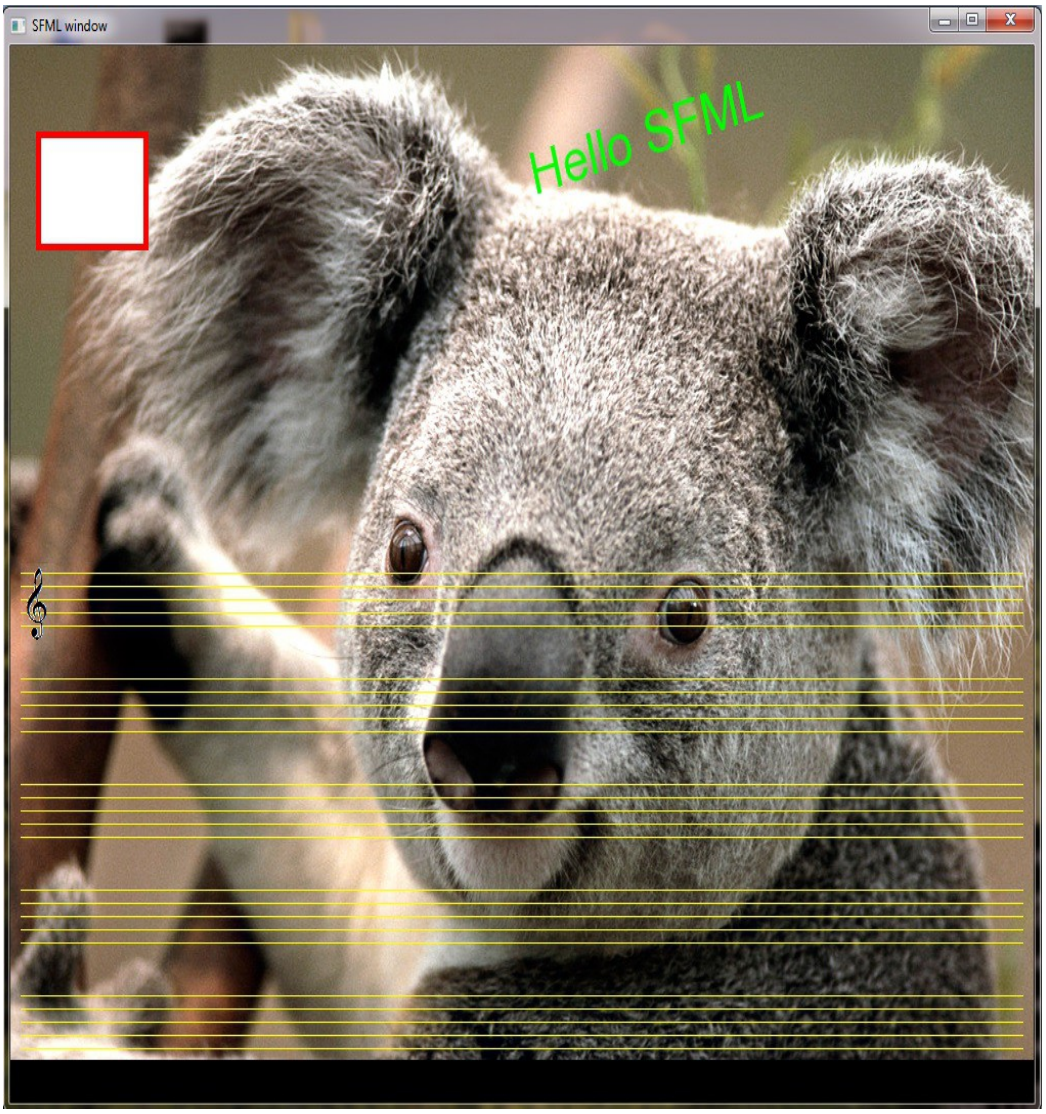
```
// module sfml2.cpp: version for Linux/Debian and Windows
// original sample program from the Internet:
//   https://www.sfml-dev.org/documentation/2.4.2/

// strongly enhanced by EH 08-2017
// compile on windows with g++ (Mingw):
//   see makefile sfml2.mak
// for execution: PATH must be set to include sfml-graphics-2.dll:
//  on Windows: set PATH=%PATH%;c:\huckert\src\SFML\SFML-2.4.2\bin
//
// compile on Linux/Debian with GNU g++ :
//   see makefile sfml2u.mak
//   libs must have been installed via: sudo apt-get install libsfml-dev

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

#include <iostream>
#include <inttypes.h>  // not really needed

// -------------------------------------------------------
// define five horizontal lines corresponding to a music system
sf::VertexArray makeNotenzeile(int xp1, int xp2, int yp)
{
  #define LDIST 10
  #define LINES 5
  // Horiz.line 1
  sf::VertexArray system(sf::Lines, LINES * 2);
  //
  int m = 0;
  for (int n=0; n < LINES; n++)
  {
    system[m].position   = sf::Vector2f(xp1, yp + (n * LDIST));
    system[m].color      = sf::Color::Yellow;
    system[m+1].position = sf::Vector2f(xp2, yp + (n * LDIST));
    system[m+1].color    = sf::Color::Yellow;
    m += 2;
  }
  //
  return system;
}   // end makeNotenzeile()

// ------------------------------------------------------
int main()
{
  int ret;
  sf::Clock clock;
  sf::Time elapsed;
  int64_t t64;
  unsigned long count = 0L;
  #define WINW 1000
  #define WINH 800
  //
  // Create the main window
  sf::RenderWindow window(sf::VideoMode(WINW, WINH), "SFML window");
  //
  // Load a sprite to display
  sf::Texture texture;
  //if (!texture.loadFromFile("cute_image.jpg"))
  sf::IntRect area(100, 100, 600, 450);  // defines the area to be loaded
  //ret = texture.loadFromFile("./Koala.jpg", area);
  ret = texture.loadFromFile("./Koala.jpg");
  if (! ret)
  {

    std::cout << "file Koala.jpg not found" << std::endl;
    return EXIT_FAILURE;
  }
  texture.setSmooth(true);
  sf::Sprite sprite(texture);
  //
  // load a second sprite (violin key)
  sf::Image violKey;
  ret = violKey.loadFromFile("./keyv2.bmp");
  if (! ret)
  {
    std::cout << "file keyv2 not found" << std::endl;
    return EXIT_FAILURE;
  }
  sf::Color col(255,255,255);
  violKey.createMaskFromColor(col, 20);
  sf::Texture violKeyTx;
  violKeyTx.loadFromImage(violKey);
  sf::Sprite violKeySp(violKeyTx);
  //
  // Create a graphical text to display
  sf::Font font;
  // EH: path added for font
  #ifdef WIN32
  ret = font.loadFromFile("c:\\windows\\fonts\\arial.ttf");
  #else
  ret = font.loadFromFile("/usr/share/fonts/truetype/dejavu/DejaVuSans-
Bold.ttf");
```

```
#endif
if (! ret)
{
  std::cout << "font DejaVuSans-Bold.ttf not found" << std::endl;
  return EXIT_FAILURE;
}
sf::Text text("Hello SFML", font, 45);
text.setPosition(500, 70);
text.setColor(sf::Color::Green);
text.setRotation(360.0 - 15.0);
//
// Load a music file to play
/*
sf::Music music;
if (!music.openFromFile("./nice_music.ogg"))
    return EXIT_FAILURE;
//
// Play the music
music.play();
*/
//
// rectangles are the replacement for a thick line
sf::RectangleShape rectangle;
rectangle.setSize(sf::Vector2f(100, 80));
rectangle.setOutlineColor(sf::Color::Red);
rectangle.setOutlineThickness(5);
rectangle.setPosition(30, 70);
//
// Define five music systems
sf::VertexArray system1 = makeNotenzeile(10, WINW - 10, (WINH / 2) + 0);
sf::VertexArray system2 = makeNotenzeile(10, WINW - 10, (WINH / 2) + 80);
sf::VertexArray system3 = makeNotenzeile(10, WINW - 10, (WINH / 2) + 160);
sf::VertexArray system4 = makeNotenzeile(10, WINW - 10, (WINH / 2) + 240);
sf::VertexArray system5 = makeNotenzeile(10, WINW - 10, (WINH / 2) + 320);
//
float rot = 0.0;
//
// Start the loop
while (window.isOpen())
{
    // Process events
    sf::Event event;
    while (window.pollEvent(event))
    {
        // Close window: exit
        if (event.type == sf::Event::Closed)
        {
          std::cout << "Exit required" << std::endl;
          window.close();
        }
    }
    //
    clock.restart();
    // Clear screen
    window.clear();
    //
    // Draw the sprite
    window.draw(sprite);
    //
    // Draw the string
    window.draw(text);
```

```
      //
      window.draw(rectangle);
      //
      // draw the music systems (5 horizontal lines per system)
      window.draw(system1);
      violKeySp.setPosition(10, (WINH / 2) - 8);
      window.draw(violKeySp);
      window.draw(system2);
      window.draw(system3);
      window.draw(system4);
      window.draw(system5);
      //
      // Update the window
      window.display();
      //
      // result on Windows: mostly < 1 ms, occasionally 5 ms
      elapsed = clock.restart();
      t64 = elapsed.asMicroseconds();
      //std::cout << "count=" << count << " t=" << t64 << std::endl;
      count++;
      if ((count % 100) == 0)
      {
        // rotate the sprite/texture
        // this works - but is not really need for tests
        rot = rot + 1.0;
        if (rot > 90.0)
          rot = 0.0;
        //sprite.setRotation(rot);
      }
  }
  return EXIT_SUCCESS;
}   // end main()
```

If you compile and start this sample program the look carefully at the pathes (complete file names) used in the source code. You probably must modify them.

And here is a screen shot for this program. This is from a Windows computer – but the output on Linux/Debian is exactly the same:



The ZIP file contains a second test program called *noten.cpp* with its associated make files. This program has been written in order to redesign the screen output driver for a music notation editor. This second test program uses C++ routines for recurring tasks (like loading symbols, drawing music systems etc.). When using sub programs (routines, methods) it is important to create SFML class instances on the **heap** (using *new* or *malloc()*) and not on the **stack!** You can see this in routine *loadSymbol().*