A Review of the Simple and Fast Multimedia Library
by Vincenzo Felici

In this post I will be discussing the “Simple and Fast Multimedia Library” (SFML). First I will give a brief overview of what the library is, and my experience working with it for the first time. Then I will go into more specific features of the library. I will discuss some classes and features that I found particularly useful or well designed. Finally, I will mention a few of the aspects that I disliked about the library, or at the very least found confusing to use.

As the name implies, this library is designed around simple, high level use, with efficient handling of resources of various formats. It is good for creating simple interfaces for an application, and managing the graphics, audio, windowing, etc that is often required. In my case, I am learning it for use as an interface to a small game application. The SFML is compatible across Windows, Mac, and Linux systems, and can be used across multiple languages. This is useful in my case, as we have developers across multiple systems that will need to reference this library.

My first introduction to the library was the SFML website (https://www.sfml-dev.org/index.php). It is a well organized and very detailed site, providing most everything a new user would need. I am glad that there is an official website to reference when learning the SFML, especially one that is well organized. The tutorials tab is particularly helpful, which includes multiple tutorials for setting up and using each module. These tutorials are good for a beginner, as they are simple but still provide insight as to how and why the example code is written. The tutorials show the development process of a function, explaining why a class should be used one way and not another. I much prefer this to tutorials that would only tell the user what to write. Being a website, the tutorials include links to the API and other sections of the site. This made it easy to find other tutorials and detailed class and function descriptions, as compared to a physical format or third party video tutorial. Another important section is the API tab. This lists the entire API for the SFML, each with their own descriptive page. It makes it simple to understand the functions for each class and how they inherit from other classes of the library. Another way to learn the SFML library is by reading some books written on using the library. The site links multiple SFML books. I personally read sections of “SFML Game Development” by Artur Moreira, Henrik Hansson, and Jan Haller. The book gives a base description of the modules, and then goes into more detail on how they use it for creating their game application. Reading this book was a great way of finding example usage of the library, and provided a bit more depth and detail than the website tutorials provide. Finally, if the user still has questions even after reading through tutorials and examples, there is a FAQ and Forums section where they

can ask questions. I'm glad this exists on an official website, as it's easier to search for SFML related topics here than on a third party website.

The SFML is broken down into five main modules, which are the System, Window, Graphics, Audio, and Network modules. Each is separated into its own sub-library, to keep their code separate and non dependent on each other. This allows a user to include only the modules they want without any excess. One module can run properly without having to include or reference large sections of the rest of the library. I'm using the SFML to create an interface for a small game application, so I am primarily using the graphics and window modules. It is nice to need not work around the other modules while developing this portion of the interface.

The main purpose of SFML is to load and use multiple forms of media files. However, loading is typically an expensive task. There will likely be many media files to load, and each may be individually quite large. Stored outside of main memory, these large files will take relatively long to load. Update functions need to be called every frame of an application, and therefore are called many times per second. Trying to access secondary memory multiple times a second for every resource in the application would be extremely expensive. So, the SFML splits its resource loading into a structure of two classes, which I will refer to as heavyweight and lightweight. Heavyweight classes are meant to do the expensive tasks a low number of times. They load in the resource from secondary memory once for the application to then grab reference to later. Once this reference is grabbed, it can then be used throughout the application by lightweight classes. These lightweight classes are meant to be instantiated repeatedly throughout many parts of the application. But rather than loading their resources directly from memory, they can ask for a reference from their corresponding heavyweight class. Let's look at sf::Font and sf::Glyph as an example. In this case, sf::Font is the heavyweight class while sf::Glyph is the lightweight class. A font file would include the details for how to display each character that can be used, and is therefore relatively large. It would be expensive to load in this entire set of information each time we wanted to display a character on screen. So, an sf::Font is instantiated once and its loadFromFile function is called once to load the font file from memory. Then, whenever text needs to be displayed on screen, an sf::Glyph can be instantiated and rendered. This would occur every frame, and likely multiple times per frame. The sf::Glyph can ask the instantiated sf::Font for a reference to the correct character to display instead of loading it on its own. This pattern is easy to use and is the same across most of the SFML resource classes. The differing one being sf::Audio which streams in sound samples rather than loading entirely at once, and is therefore handled a bit differently.

There is a good amount of generic code in SFML as well. Mainly their positioning related classes, such as sf::Vector2 and sf::Rect. I think this is to allow for measuring coordinates and operating on them with datatypes of the user's choosing. They've made typedefs for common template types that help code look cleaner and shorter. sf::Vector2<T> already has sf::Vector2f, sf::Vector2i, and sf::Vector2u for float, int, and unsigned int.

Mainly following the book I read, it felt fairly simple to use SFML to produce the basic components of a game engine and interface. The majority of the resource loading tasks are done by the library, with a need to just implement a way to organize the loaded information. One is able to create a window and poll for various window and input events. There are classes for rendering shapes with textures. Updates and draw calls can be handled in the window's event polling loop. The functions dealing with rendering are built to be high level and straightforward to use. For example, using sf::View. Treating it like a camera, one would want to determine where it is on the window, move it around the game world, or maybe even rotate and zoom. These functions are already provided, and only a few short parameters need to be passed in. sf::View includes a move, rotate, and zoom functions, and allows setting of size and position relative to the render window. It is nice to have these prewritten, high level functions rather than trying to write them ourselves, even if they are relatively simple.

I find it strange that there isn't a pre-made, generic resource loader and holder class for SFML. In the book, they explained how most of SFML's resource loading and storage is done in a similar way across resource types. They explain a way of making loading and requesting references generic, so that an inherited class can ask to load by resource type and identifier. This seems to work and be straightforward for everything other than sf::Audio classes, so I'm not sure why it isn't already included in SFML. I imagine most programs using SFML would need to load resources the same way. So, making a generic loader would make things simpler to read and avoid implementation of multiple similar classes. If it were included in the SFML, the user could still choose to use the non generic forms if they wanted to.

There aren't many complaints I have with this library. Mainly the lack of a generic resource loading class, but that isn't all too difficult to implement. I did have a few issues with class parameters when trying to learn SFML. Some function parameter requirements no longer match the tutorial page and forum answers. I seemed to have issues with the sf::FloatRect constructor which says it can take four parameters as left, top, width, and height. However, it only seemed to work when I gave it a sf::Vector2f position with a sf::Vector2f size. This was a strange issue as it was described with four parameters in the tutorial, api, and some forum posts, but I could only have it work with

the two parameter layout. Maybe this is an issue with my IDE, or perhaps there is an issue with the documentation.

Overall, I found SFML to be a simple to use library that covered the major needs of an interface for a small game application. It is designed to be simple and efficient. It has modules for multiple resource types that may be needed. It is relatively easy to get into, as the site provides many resources for learning about SFML.