User Interface Design Frameworks

A Comparative Analysis of Popular C++ GUI Libaries

Kevin Masson

Student of Computer Science

University of Lethbridge

Lethbridge Alberta Canada

kevin.masson@uleth.ca

1 Introduction

When a programmer wishes to create a graphics interface for their program, there is no easy way to do this within C++. This issue is due to the language not having any sort of native graphics library. This problem, however, has led to the development of numerous third-party graphics libraries that are available to users. This then leads to the question of which libraries are best to use? Furthermore, what is the criteria to judge such a metric? This is the line in inquiry I follow within this research paper to consider how a graphical user interface is created within C++ and to compare the options of widely used libraries to create such interfaces within C++.

2 Background

I aimed to first consider the reasoning behind the lack of a native graphics library within C++. When comparing other object-oriented programming language competitors, such as Java and Python have built-in graphics library easy for programmers to use with the Abstract Window Toolkit and Graphics for each language respectively. This is a question even Bjarne Stroustrup, the creator of C++ himself, has noted as a problem of the language. He explains in his FAQ regarding C++ that “it doesn't have a standard GUI, and that is indeed a major problem.” In which every project requires a unique graphics library and often takes time and confusion from not being a standard [1]. However, Stroustrup notes that “C++ has many commercial and open-source GUIs” and “providing a GUI is both a technical and political problem. There are lots of GUIs with lots of users, and generally they wouldn't like some other GUI to be declared standard” [1]. This seems to be the heart of the issue with implementing graphics with C++ as it would cause issues for one library to be accepted as *the* C++ graphics library and would demean the work of many programmers creating other libraries. Considering the options Stroustrup has noted for open-source options, the answer seems to be there is simply no reason it is needed when there exists many open-source options already made.

3 Criteria

Once deciding upon which libraries to use, the user then must weigh their options of what works better for their program. Graphical user interface could be created through a lower-level library such as Simple DirectMedia Layer for more intricate design [2]. However, it is much easier and more likely the programmer should opt to use highly abstracted GUI widget libraries. This is because these libraries are much less concerned with the individual generation of graphical objects or it’s event loop and gives the programmer a more understandable syntax and layout for generating objects and events. As there are too many libraries that exist for C++ to compare within completely in this paper, I will try to focus on comparing three popular libraries: QT, GTK, and wxWidgets. This is due to the fact that all three are cross-platform, meaning that any work done on each will not need to be ported to other architectures, as well as each library having proven, popular software written on them [3, 4, 5]. Then the criteria I will look at to compare each library will be through it’s: ease of use, modularity, and its licensing structure.

4 Evaluation

When comparing the ease of use, its important to consider the complexity versus the tools the libraries provide to help remedy its complexities. Each framework uses event driven architecture which require a main application object with at least one window object being created, and the programmer adds widgets and functionality to the main application parent object. What differs however is the intricacies surrounding how each framework is written. GTK is natively written in C with simply C++ wrappers being available such as GTKmm [6]. Due to this There can be some issues in getting the code to execute the way you expect it to, as it is translating code in a way you might not expect it to. wxWidgets has a similar execution, however it is natively written in C++ and will more naturally be translating your code into ways you expect it to [7]. QT is also native C++, however it was created before many standard library classes were the norm such as std::string. Thus, QT requires one to use primarily their own QObjects to function which adds complexity to the program. As well, QT requires external processing for handling events through .pro files so it is a more bloated system to work with. However, QT has many first-party programs they provide to aid in working within these objects. QT provides QTDesigner, which is a program that easily automates UI code into a fast visual prototype [9]. This prototype can be used by their proved IDE QTCreator, which can easily interact with the objects generated and automatically generates necessary code to implement events. As such, QT has more of a learning curve but more provided tools to easily automate the processes.

Modularity I define as the number of options each library provides for your program's needs without needing external libraries. While all three allow for basic widget options, such as windows, buttons, lists, dialogue, etc., I researched the extra options that may be unique to each framework. GTK is unique in the sense that its available for C++ but is natively able to be written in C [10]. This is useful for being able to easily use the same skillset for working on Object-Oriented languages and non-Object-Oriented languages. GTK, as well, is tailored more towards being built over X11 and then ported to windows and, as such, has an easier time creating GUI for Linux systems, especially being the main display for the Linux GNOME environment with native API support [11]. wxWidgets has the benefit of using native widgets whenever possible to the user's system. This allows for windows and widgets to generally look more natural on your system (i.e. a wxWidgets program will *look* like a Linux program when run on Linux, and *look* like an OSX program when run on Mac) [5]. GTK and QT attempt to circumvent this by emulating native widgets for the platform, but not being able to look exactly like such. However, wxWidgets does not allow for custom themes to look different from system windows and has a lack for modularity from that [11]. QT has the ability of custom theming, but also includes many more addons compared to GTK and wxWidgets [11]. QT has first-party addons for code integration, such as Bluetooth, 3d, SQL, and ActiveX [9]. As well, QT attempts to extend the C++ language using their own *QT Markup Language (QML)*. This is a relatively less complex syntax to easily generate QT code and is parsed by a pre-compiling step before compilation to understand the code [9]. In this sense, QT is the much more robust framework when considering modularity.

Licensing of use of graphical libraries is a big topic when considering utilizing one of these libraries for your own work. Each of these tools have licensing guidelines of what is allowed for your program to use when released, in terms of for-profit and open source. GTK is featured under the GNU Lesser General Public License (LGPL) [6]. This allows any user to freely use the library and use legal copyright, closed-source, proprietary advantages of it without requiring any fees on the programmer's part. wxWidgets contains its own licence called the wxWindows License, which contains all open source, free aspects of the LGPL but with the ability to further ability for derived works in binary form may be distributed on user's own terms [5]. This allows for users to release the work as a binary with any restrictions you wish on viewing the code. QT falls under the LGPL as well but has commercial licenses available. These commercial licences allow for the programmer to modify QT as needed without disclosing modifications and allows extra tools and solutions for embedded development. As well some specific modules, especially newer ones, are under the General Public License, which requires you to also release your code open source, unless you purchase a commercial license [12]. Considering this, when using QT for any commercial products, one must consider exactly what they wish to use with QT and how open source they wish their software to be.

5 Conclusions

When considering each of the three GUI frameworks, one must consider what their intentions are for the use of the framework as well as their intentions for their finished software. If the intention is for the user to create a GUI in C++, QT seems to be the most robust, supported, and extendible option while also remaining open source. wxWidgets biggest draw for use versus QT is its use of native widgets to make your software appear to look just as intended upon the native architecture. However, even this isn't necessarily a drawback for QT, as this issue is balanced by the fact that the user can implement custom theming for their own software. GTK's main draw is the fact that it stays relatively simple and true C++ without extra functionality, as well as it's ability to work natively with C programs. QT's main hindrance is it's licensing, which utilizes the same licensing freedom as GTK and wxWidgets, however the presence of GPL and commercial licensing causes the user to need to be careful what they are incorporating and how to avoid breaking the terms of the license.

References

[1] Bjarne Stroustrup. Bjarne Stroustrup's FAQ. (July 2021). Retrieved March 20, 2023 from https://www.stroustrup.com/bs_faq.html

[2] Simple DirectMedia Layer 2.0. Retrieved March 20 2023 from https://wiki.libsdl.org/SDL2/FrontPage

[3] List of QT Applications. Retrieved March 20 2023 from https://wiki.manjaro.org/index.php/List_of_Qt_Applications

[4] GTK. Retrieved March 20 2023 from https://www.gtk.org/

[5] Commercial Applications using wxWidgets. wxWiki (March 2016). Retrieved March 20 2023 from https://wiki.wxwidgets.org/Commercial_applications_using_wxWidgets

[6] GTKmm Retrieved March 28 2023 from https://www.gtkmm.org/en/index.html

[7] Julian Smart and Kevin Hock. 2005. Cross Platform GUI Programming with wxWidgets. Crawfordsville, Indiana.

[8] QObject Class. QT Documentation. Retrieved March 28 2023 from https://doc.qt.io/qt-6/qobject.html

[9] QT Features, Framework Essentials, Modules, Tools & Add-Ons. Retrieved March 28 2023 from https://www.qt.io/product/features

[10] GTK Documentation. Retrieved March 28 2023 from https://docs.gtk.org/gtk4/overview.html

[11] wxWidgets Compared To Other Toolkits. wxWiki (October 2018). Retrieved March 28 2023 from https://wiki.wxwidgets.org/WxWidgets_Compared_To_Other_Toolkits

[12] QT Open Source Licensing. Retrieved April 2 2023 from https://www.qt.io/faq/tag/qt-open-source-licensing