Kesäkoodi final report

Riku Salminen

Part one: XCB and OpenGL

My original Kesäkoodi project was adding support for OpenGL 3 extensions to XCB-GLX, the XCB bindings to the GLX protocol. The goal of the project was to enable OpenGL-based graphics in XCB applications.

The GLX API is a standard programming interface to enable using OpenGL applications in the X Windowing System and the GLX protocol is the underlying protocol for X client/server communication. The GLX API is closely coupled with Xlib, the standard programming interface to the X Windowing System. While XCB is trying to be an alternative to using Xlib, XCB-GLX is **not** intended as a replacement for GLX. I had to learn it the hard way.

The GLX API is not implemented by a single software library, but instead different OpenGL implementations have different GLX implementations. The GLX implementation is responsible for two things: setting up client-local OpenGL state and negotiating with the X server using the GLX protocol. XCB-GLX on the other hand can only be used to negotiate with the X server but it does not (and can not) set up the private client-local OpenGL state and thus cannot be used to set up OpenGL rendering.

Although GLX and XCB-GLX do very different things, it's very easy to be mistaken. Looking at the GLX and the XCB-GLX functions, one can easily get the impression that XCB-GLX is to GLX what XCB is to Xlib, that is, a replacement API. The GLX functions have a nearly one-to-one correspondence to the XCB-GLX functions, just the way XCB functions look almost like Xlib functions. It's easy to make the same mistake I made, and looking at various discussions in the Internet, I am not the only one to make this mistake.

The start of my XCB project went pretty fluently, I was able to set up an XCB development environment pretty fluently and got hacking straight away. I was able to add some of the OpenGL 3 GLX protocol extensions to XCB-GLX and use them. I started to notice that things were wrong when I had a simple XCB + XCB-GLX + OpenGL app set up just like it's supposed to be, but I was not get any drawing to happen.

With next to no documentation available on the subject, all I could do was reverse engineering. I started doing a side by side comparison of "equivalent" XCB/XCB-GLX and Xlib/GLX applications. I used the xtrace X client/server communication debugger, and noticed that something's fishy and that my Nvidia GLX implementation was doing something that I didn't expect it to. I contacted several other XCB developers across the Internet (in mailing lists as well as private e-mail correspondence) to discuss my findings. I figured out that what I was trying to do was effectively impossible.

While struggling with the GLX vs. XCB-GLX impedance mismatch and trying to figure out a solution to set up OpenGL in an XCB application, I had a breakthrough. Although the GLX API is closely tied to the Xlib library, it is possible to use OpenGL with XCB. The solution is to use a hybrid XCB/Xlib setup, where Xlib is used to negotiate with GLX and XCB is used for windowing, event handling and everything else. This is possible if you use an XCB-based version of Xlib, which is likely if you have a modern Unix-based operating system. New versions of Xlib are built using XCB for the network communication and it is possible to mix XCB and Xlib together. This provides a way to incrementally port Xlib applications to XCB, but it can also be used to work around the tight Xlib/GLX coupling.

Once I had realized that my project with XCB and OpenGL was impossible to complete, I wanted to finish the project as quickly as possible and move on to another project. I went on to document my

findings the best I could. I wrote a draft documentation in this blog, complete with a few clarifying diagrams. I asked the XCB mailing list for feedback on the documentation and after receiving some encouragement, I posted the documentation in the XCB wiki. The final documentation can be found here: http://xcb.freedesktop.org/opengl/. Unfortunately, I could not upload any images to the XCB wiki and I had to leave out the diagrams I had spent lots of time and effort doing.

This concludes my one month ordeal with XCB and OpenGL. The rest of my Kesäkoodi I would go on and spend on another project, Libwm. What I achieved was not much. I wrote the XCB/OpenGL documentation to the XCB wiki, which does present the first working XCB/OpenGL solution I have found in the Internet, although it's not Xlib-free like I wanted. I also posted a bug report to a bug I found while hacking on the project. All in all, I was very disappointed with my work and embarrassed of looking like a fool working on a project that was impossible to begin with.

Part two: Libwm

After coming to the conclusion that my original Kesäkoodi project, XCB and OpenGL had come to a dead end, I quickly documented my findings and published my XCB/Xlib/GLX hybrid setup example in the XCB wiki. After that I could move on to other projects. I made a few suggestions for new projects, my favorite suggestion being Libwm, an abstraction library for window and render context management for OpenGL applications. I was glad to find out that my supervisors approved of my choice. I got cracking right away.

First, I had to write a new project plan. I set my goal on making the first release of Libwm. I had seven weeks of time left to finish my goal. I decided to set my schedule so that I'd have four weeks of time to achieve the primary goal, the Libwm 0.1.0 release. The three remaining weeks I wanted to have as a worst case scenario-buffer time or if I'd make release in time, I could improve the library and add new features to it.

I had started the Libwm project early in the spring and it had about two months of work and 7000 lines of code under it's belt. I felt that the project was about half-way from being a complete usable software library and coding-wise I knew pretty much what I had to do to get there. The bigger part of getting the library in release shape was documentation and project management related. I decided to schedule my time half coding, half project management. I wanted to do the most useful project management tasks first, the ones that would help me with the coding process.

Once I had project plan and a schedule, I started off by creating task issues in the Libwm issue tracker in Google Code. I had a few dozen task TODO list written on a piece of paper earlier this year and I compiled the issues list based on that and comments I had previously added to the source code. I also set up CMake and Doxygen-based API doc generation and wrote the skeleton for the API docs by adding a brief annotation comment for all public classes, structures, functions and their parameters and return values. I had plenty of coding time on my schedule so I went to work.

I started off by splitting the source code in two parts, the windowing and the rendering context part. The windowing part would consist of Win32 and Xlib code and the context management would be done with their respective OpenGL-glue API's WGL and Xlib. I had tried doing this earlier in the project, but I ended up having a simpler solution by merging them together. However, the lessons learned when struggling with XCB turned out to be very helpful for putting together a big picture about OpenGL and Windowing system communication. I also wanted to keep things well organized so adding EGL and OpenGL ES support would be easier to do in the future. Overall, the library design improved a whole lot. My overall design paradigm for Libwm has been trying to find the commonality in the underlying API's and abstracting it without breaking the underlying fundamental concepts instead of providing a monolithic and intrusive framework like many of Libwm's competitors do.

The next project management task I had was to improve the CMake build system and configure CPack to build distribution files. Everything didn't quite work out as planned but I got a decent system set up to build and package Libwm. I couldn't get proper distribution files built for .deb-based systems or Mac OS X, because of difficulties and little details in creating a distributable shared library package. Everything was pretty much following the 80-20 rule and I decided that I was happy with the result and not to waste any more time on CPack.

I did some small but important improvements to Libwm event handling code, but I quickly realized that I'm running out of small fixes, I have plenty of time on my schedule and some big coding tasks that I originally intended to postpone until after the release. They were the ones that I also had the most negative feedback about, adding support for full screen and backwards compatibility code for GLX 1.2. I took the rest of the week to get that done, and it went pretty fluently. The biggest issue was doing GLX 1.3 and GLX 1.2 code binary compatibility. To make Libwm-based applications run on a GLX 1.2-only version of the OpenGL library (libGL.so or equivalent), I used dlopen to load the GLX 1.3 functions dynamically at run-time if glXQueryVersion reported GLX 1.3 or higher.

The third week of the coding process was entirely spent coding. I did some refactoring and redesign behind the scenes and added small less-significant things like hiding the mouse cursor and creating windows that can not be resized. I felt that the library was almost complete, but needed some review and testing.

The fourth week was the planned release week. I spent most of the time writing user documentation in the Libwm Wiki. I also stumbled across a few bugs and overall I did not feel confident enough to make the release quite yet and decided to wait a little longer. The fifth week was spent on finalizing the library and on Thursday 13th of August I uploaded the official 0.1.0 release tarballs to the Libwm home page in Google code and added a version tag to the Mercurial source repositories.

After the release was done, I had a few weeks left. I had scheduled the last week for writing the final documentation but instead I decided to spend the entire period coding. When I started the Libwm project, the intention was to write it to provide a comfortable platform for writing OpenGL 3.x-based applications across platforms and provide backward compatibility workarounds. Adding support for OpenGL 3.0 (or the ARB_create_context extensions) didn't make it to the first 0.1.0 release and that was intentional. Now that I had the release done, I didn't hesitate. I quickly wrote support for OpenGL 3, multisampling and shared-exponent (sRGB) pixel formats, among other stuff. I released the second version of Libwm, 0.2.0 just a few weeks after the first release.

Now that Libwm is a full-featured and capable software library, the next step is to start using it. Some practical use and real world testing will surely improve the library and help to squish out bugs that may be there.

The next natural step for Libwm is to support a wider range of platforms, including new windowing and rendering context API's. The most interesting API to support would be EGL, a new glue API that can be used to set up OpenGL ES, OpenVG and other graphics API's on mobile platforms and I hear it's coming to the desktop too. Adding native Mac OS X Cocoa support is also interesting and would enable OSX users to run Libwm applications without the X server and with EGL could run Libwm on the iPhone. There's also plenty of new features that can be added to Libwm, including but not limited to multi display support, better input handling, support for multi-GPU systems and a lot more.

I thank COSS and the Kesäkoodi crew and all the sponsors for the financial support and making Libwm possible. I am very satisfied with the final product and it wouldn't have been possible without the ability to work full time on this open source software library.