0.1 Tools

Stopmotion has become an application with a relatively large amount of classes and allot of code lines. To navigate through these classes effectively and recompile repeatedly without having to stare at the roof for minutes at a time, it's necessary to use different tools speeding this up. We have used the following tools helping us with that:

Make

Make is an intelligent tool which controls the generation of executables of a program from its source files. It builds the executable based on information described in a makefile following a strict syntax. The makefile should therefore be well written and has to be error free for make to interpret it.

The advantage of using make is that it automatically figures out which files it need to recompile, and which ones it doesn't need to recompile. It determines if a source file is dependant on an another source file and recompiles them if one of them has changed. Make uses the timestamp for the file to figure out when it was last modified and uses this to find out if it need to be recompiled. That means it doesn't actually check the contents of the file for differences.

Ccache

As described above, make only checks the timestamp for a file to find out if it needs to be rebuilt. That means a file will be recompiled if you opens and saves it without making any changes. If the file also has dependencies to other files, make will decide to recompile them too. It would be really nice if we only could recompiled the files which actually has its contents changed. Ccache can help us with that.

Ccache uses wrappers for both gcc and g++ and has a cache containing the contents of previously compiled files. The wrapper compiler acts exactly like its "original" compiler, so the source code is interpreted and compiled the same way. The only difference is that Ccache compares the content of a file with the content of the same file found in cache before it compiles it. Recompiling only happens if the file is different from the file found in cache. This often results in a five to ten times speedup on recompile. In our case we have measured it to be 11 times faster on its maximum.

QMake

Writing makefiles by hand is time consuming, boring and very error prone. QMake is a tool created by Trolltech which auto-generates the makefiles and takes care of getting the right dependencies for the running platform and compiler. Since it doesn't have super cow powers it needs to be fed with some input from a project file (.pro). This file can be auto-generated by qmake itself by running *qmake -project* in the top-level directory containing the source code. QMake will then scan through all of the files in the running directory and sub-directories and generate the file. It's also possible to edit the file by hand if you need to add additional information. The project file format is simple and human readable and a very basic example file can be found in section 1 on

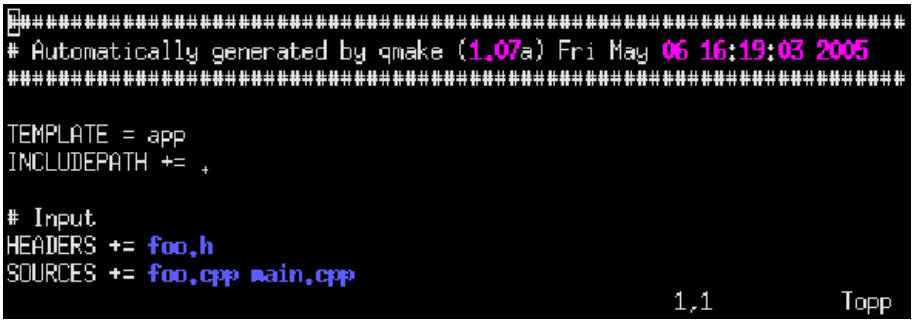

Figure 1: A very basic qmake project file

page 2. When the project file satisfies all of your needs, the makefile can be generated with executing *qmake <filename>.pro*.

Qmake will auto-generate makefiles and do a lot of job for us, but in our case we still have to do some work on the .pro file. It's not sufficient to let qmake auto-generate the .pro file because of the dependencies tied to Stopmotion. It's necessary to add some include paths and linking parameters which are specific for these libraries. By the way, this isn't good supported by the format if you use *pkg-config* or similar tools. In our case, when using *sdl-config*, we had to use *sed* and *grep* like this to get it work: *INCLUDEPATH += $$system(sdl-config –cflags | sed -e 's/-I//g')*. We also had to list some other files to get them included with the tarball generated with make dist.

In addition the .pro file is also used when generating the translation files, but this is explained in section **??**.

KDevelop

KDevelop has a separate QMake template which is excellent to use when programming Qt applications. KDevelop takes care of editing the .pro file with a tool called QMake Manager. This tool will automatically add information needed in the .pro file when adding new header files or source files to the project, and it will not override the manual settings such as the include path mentioned above.

In addition to this there's a lot of short cut keys such as F8 for compiling, Shift-F9 for executing the program, Ctrl-Shift-s for getting a Doxygen template for documentation etc. It's easy and quick to move between header files and theirs source files, and a lot of external programs such as debugging tools are available from the menus, if they're installed on the system.