Report

# Streaming OpenGL commands with Qt

| | |
|---|---|
| Version: | 1.0 |
| Date: | 19 March 2015 |
| Classification: | Confidential |
| Status: | Draft |
| Filename: | Document1 |

## Authors

| Name | Chapter/Section | Create date | Visa |
|---|---|---|---|
| Christoph Speck<br>Oleksiy Kasilov | | 11.01.2016 | |

## History

| Version | Initial | Date | Description | Chapter/Section |
|---|---|---|---|---|
| 1.0 | | 11.01.2016 | Creation | all |
| | | | | |

## Table of contents

# 1 Introduction

## 1.1 Purpose

During the keynote of the QtDeveloper Days 2013, the CEO of QNX mentioned that they have developed the so-called GLCast for an automobile manufacturer. With this technology, OpenGL commands are transferred via a network. Thus, a screen can be easily streamed and displayed across multiple devices. Every modern browser also supports HTML5 WebGL. This makes it possible to interpret the OpenGL commands inside a browser. This technology should allow to display and control any QML application inside WebGL enabled browsers.

In the flow of this project we developed a C++ library with client-server architecture that allows streaming OpenGL commands over the network. The server serializes and streams commands via a WebSocket and clients de-serialize the stream and execute the OpenGL commands locally.

In this document we present the idea behind the project, describe our considerations and decisions and summarise the outcome of our work. We also list a few problems that were faced in the course of the project as well as limitations of our solution. We also give some ideas for future work.

## 1.2 Scope

The work was divided into two phases (KCP projects).

### KCP-73

A proof-of-concept application was developed based on the Qt OpenGL ES 2.0 Cube example. The streaming of OpenGL commands was done on the application level. Therefore, before starting to receive and execute streamed commands a client had to initialize its OpenGL context exactly the same way as it was done on the server (i.e. create and initialize Qt widgets, shader programs, textures, etc). This made the client code heavily dependent on that of server and thus this solution was impractical.

### KCP-79

In this project we tried to develop a general solution which would allow potentially any Qt application (C++ or QML based) to be streamed and displayed in the Web browser.

## 1.3 Definitions, Acronyms and Abbreviations

| | |
|---|---|
| OpenGL ES | application programming interface (API) for advanced 3D graphics targeted at handheld and embedded devices such as cell phones, personal digital assistants (PDAs), consoles, appliances, vehicles, and avionics. OpenGL ES is one of a set of APIs created by the Khronos Group[i]. There are three OpenGL ES specifications that have been released by Khronos so far: 1.0 1.1 2.0 3.0 3.1 3.2 EGL |

| | |
|---|---|
| EGL | an interface between Khronos rendering APIs (such as OpenGL, OpenGL ES or OpenVG) and the underlying native platform windowing system. EGL handles graphics context management, surface/buffer binding, rendering synchronization, and enables "high-performance, accelerated, mixed-mode 2D and 3D rendering using other Khronos APIs."[2] EGL is managed by the non-profit technology consortium Khronos Group. |
| QPA | Qt Platform Abstraction. See details under https://wiki.qt.io/Qt_Platform_Abstraction |
| KCP | Kompetenzcenter Project: bbv innovation projects. |

## 1.4 Summary of the project objectives and goals

Advantages:

- No client application deployment necessary
- Less network traffic than VNC or any other system that transfers rendered images.

Disadvantages:

- No useful information is shown if there is no network connection. Client has no logic of its own. Network latency is crucial for a proper running system.
- Huge network traffic compared with a dedicated client application tailored for a specific use.

Restrictions:

- There are many different OpenGL APIs. WebGL is based on OpenGL ES2.0. In order to maximize compatibility between Client and Server the later shall therefore use only OpenGL ES2.0.

# 2 Architecture and design

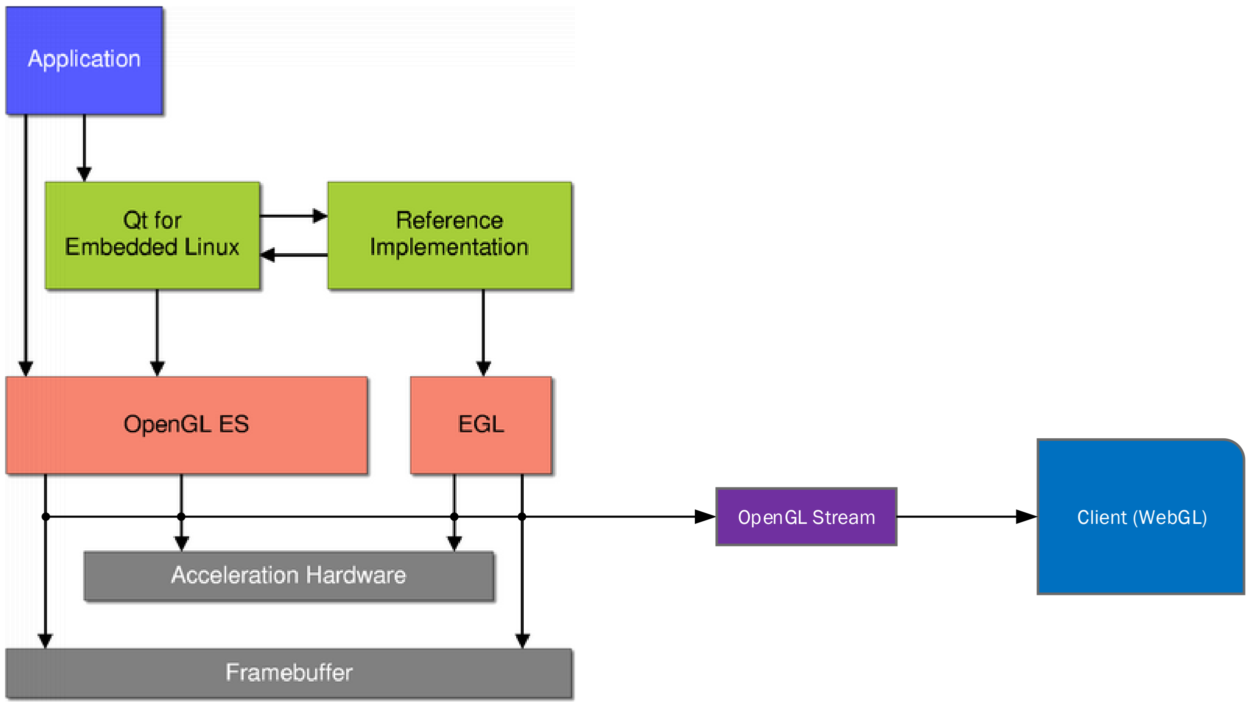
Our ultimate goal is to have a thin client application whose only task is to parse the stream of OpenGL commands and execute them on the local hardware. In order to achieve this goal we have to intercept all OpenGL calls. We have considered two possible approaches.

## 2.1 The Qt platform abstraction (QPA)

Qt 5 introduced a new platform abstraction layer (QPA). This layer allows to create QPA plugins for specific platforms by subclassing various QPlatform* classes. The root class QPlatformIntegration is responsible for Window System integration and could be used as an entry point for streaming OpenGL commands. The QPA is a fairly new concept and currently has very limited documentation. Although our solution is based on another principle, this idea must be revisited in the future as soon as there is more QPA documentation and examples available from Qt community.

## 2.2 OpenGL API interception

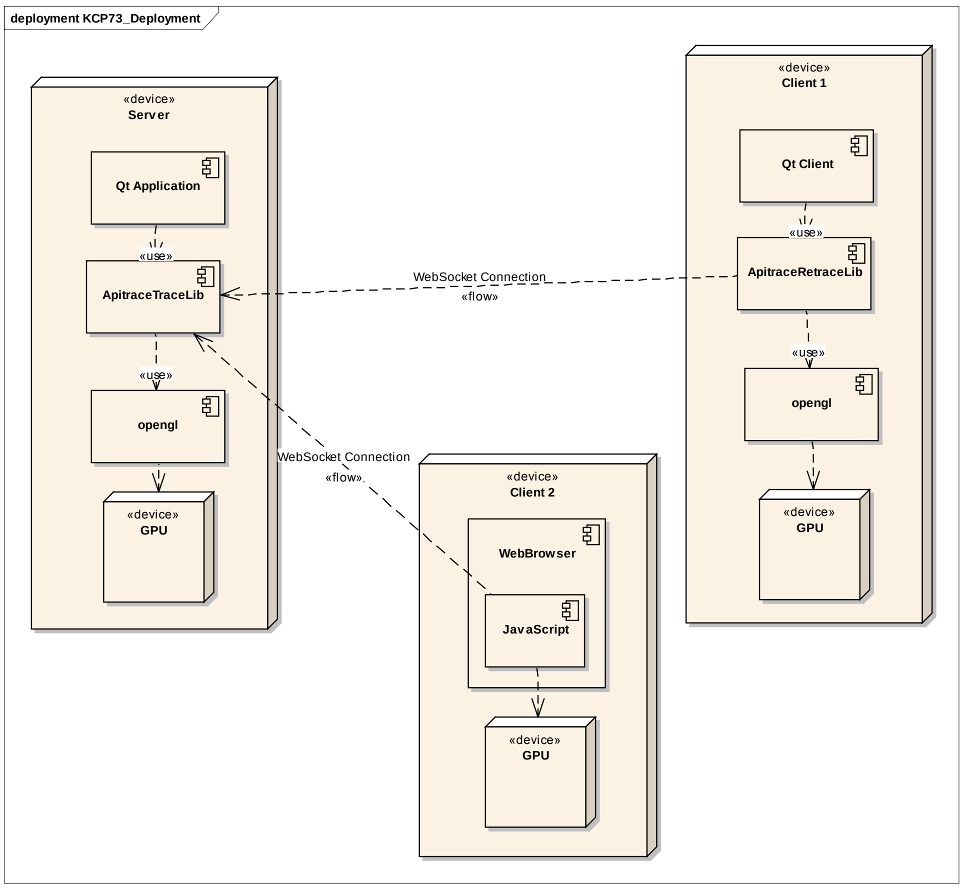
A picture below shows how any Qt application uses OpenGL ES as interface between software and acceleration hardware.



Most of the OpenGL calls are triggered internally from Qt libraries. At the application layer there is no way how to keep track of them. However, OpenGL is implemented as shared library and can be intercepted on Linux systems using the LD_PRELOAD environment variable. There are several open source projects which use this technique to trace API calls to shared libraries. Our solution is based on the apitrace tool which has shown to be a mature, actively developed and well tested project. However, apitrace aims to be a generic tool for tracing potentially any API calls. It currently supports: OpenGL, Direct3D and DirectDraw tracing on Windows, Linux, Mac and Android systems. Additionally it allows profiling and debugging of API calls. Being very flexible it imposes a lot of code overhead and additional latency for our application. Therefore, we have decided to adapt apitrace code to our needs and extract only needed functionality.

### 2.2.1 Trace/Retrace libraries

Based on the apitrace two libraries were created. ApitraceTraceLib captures calls to OpenGL library and transfers the graphic commands via WebSocket interface to connected clients. Clients can make use of ApitraceRetraceLib that connects to a server, and converts a received stream from WebSocket into calls to local OpenGL library.



We have built a very simple “thin” client in Qt which opens WebSocket, deserializes and executes received OpenGL commands. The integration of our tracing library into existing Qt application is very simple. The server application must be linked against AptraceTraceLib library and one line of code must be added into its main function. Client code remains the same for any application. This solution was tested with different Qt demo applications.

### 2.2.2 Building a JavaScript client

To enable a web browser to make use of the ApitraceRetraceLib the Qt C++ source code has been converted with Emscripten into JavaScript source code. Currently generated JavaScript file has size of about 6Mb.

# 3 Use cases

The approach described above is tested with 3 different applications. While all applications run correctly when a Qt Application is used on the client machine (Client 1 in graph above) all of them fail in web browser (Client 2).

## 3.1 Cube example

The Qt cube sample http://doc.qt.io/qt-5/qtopengl-cube-example.html produces only a black screen in the browser.

The first problem that the image texture file has not a power of 2 size can easily be fixed. We then even stripped the sample code from the texture file and just render a red cube without texture. The browser still complains with the following warnings:

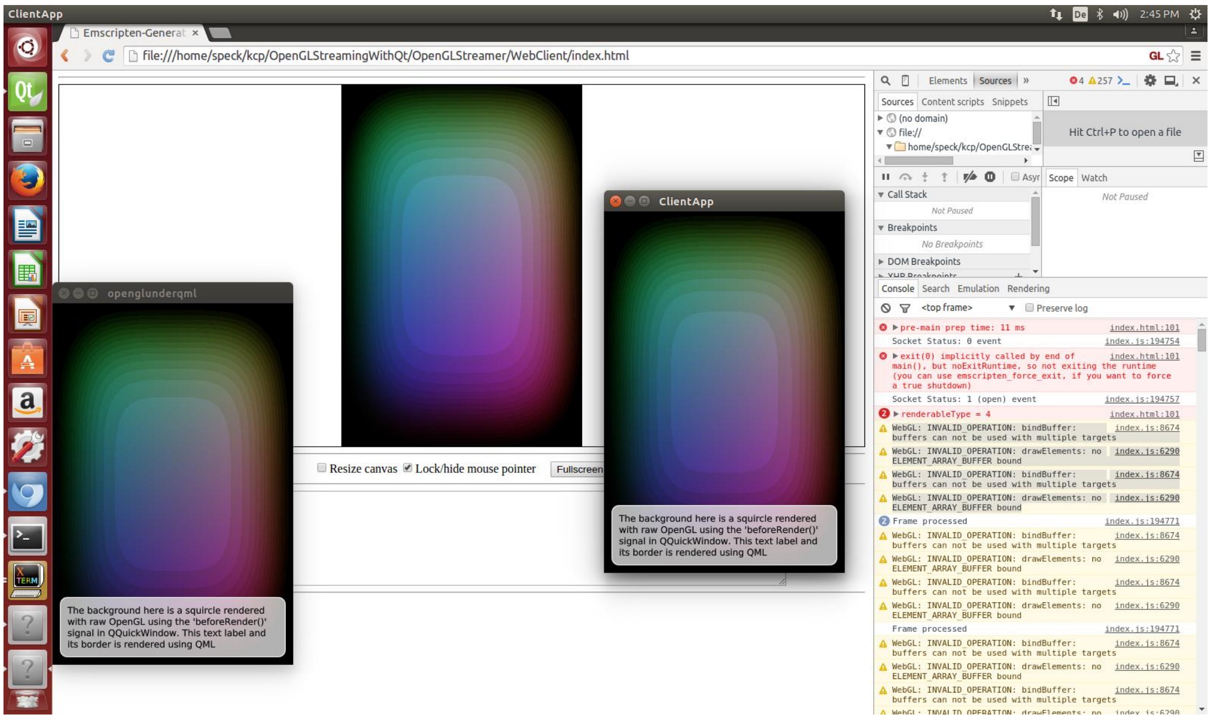
```
WebGL: INVALID_ENUM: renderbufferStorage: invalid internalformat

3VM106:1958 WebGL: checkFramebufferStatus: the internalformat of the attached
renderbuffer is not DEPTH_COMPONENT16

2VM106:1958 WebGL: INVALID_FRAMEBUFFER_OPERATION: clear: the internalformat of the
attached renderbuffer is not DEPTH_COMPONENT16

VM106:1958 WebGL: INVALID_FRAMEBUFFER_OPERATION: drawElements: the internalformat of
the attached renderbuffer is not DEPTH_COMPONENT16

VM106:1958 WebGL: checkFramebufferStatus: the internalformat of the attached
renderbuffer is not DEPTH_COMPONENT16

VM106:1958 WebGL: INVALID_FRAMEBUFFER_OPERATION: clear: the internalformat of the
attached renderbuffer is not DEPTH_COMPONENT16

VM106:1958 WebGL: INVALID_FRAMEBUFFER_OPERATION: drawElements: the internalformat of
the attached renderbuffer is not DEPTH_COMPONENT16

VM106:1958 WebGL: checkFramebufferStatus: the internalformat of the attached
renderbuffer is not DEPTH_COMPONENT16

VM106:1958 WebGL: INVALID_FRAMEBUFFER_OPERATION: clear: the internalformat of the
attached renderbuffer is not DEPTH_COMPONENT16
```

Debugging shows that there is a 640 x 480 px (image or viewport size) texture transmitted in each frame. We wonder why this large texture data is transferred in each frame.

## 3.2 SceneGraph

The Qt sample OpenGL under QML http://doc.qt.io/qt-5/qtquick-scenegraph-openglunderqml-example.html

Runs correctly when run in a Qt Client Application but when run in browser, the text label is missing.



The background here is a squircle rendered with raw OpenGL using the 'beforeRender()' signal in QQuickWindow. This text label and its border is rendered using QML

```
pre-main prep time: 11 ms                                  index.html:101
Socket Status: 0 event                                     index.js:194754
exit(0) implicitly called by end of                        index.html:101
main(), but noExitRuntime, so not exiting the runtime
(you can use emscripten_force_exit, if you want to force
a true shutdown)
Socket Status: 1 (open) event                              index.js:194757
renderableType = 4                                         index.html:101
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
Frame processed                                            index.js:194771
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
Frame processed                                            index.js:194771
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
```

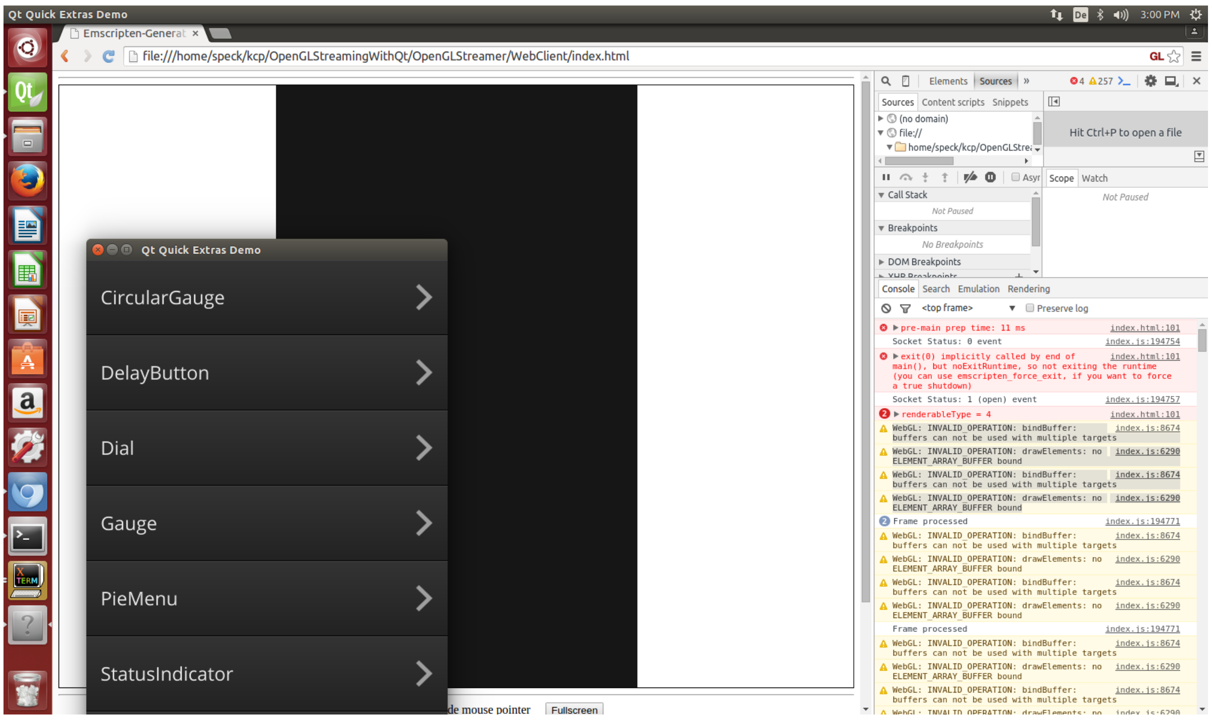
The bowser complains:

```
WebGL: INVALID_OPERATION: bindBuffer: buffers can not be used with multiple targets

index.js:6290 WebGL: INVALID_OPERATION: drawElements: no ELEMENT_ARRAY_BUFFER bound

index.js:8674 WebGL: INVALID_OPERATION: bindBuffer: buffers can not be used with
multiple targets

index.js:6290 WebGL: INVALID_OPERATION: drawElements: no ELEMENT_ARRAY_BUFFER bound
```

## 3.3 Gallery

The sample code: [http://doc.qt.io/QtQuickEnterpriseControls/qtquickenterprisecontrols-enterprise-controls-gallery-example.html](http://doc.qt.io/QtQuickEnterpriseControls/qtquickenterprisecontrols-enterprise-controls-gallery-example.html)

Produces only a black screen in the browser:



```
pre-main prep time: 11 ms                                  index.html:101
Socket Status: 0 event                                     index.js:194754
exit(0) implicitly called by end of                        index.html:101
main(), but noExitRuntime, so not exiting the runtime
(you can use emscripten_force_exit, if you want to force
a true shutdown)
Socket Status: 1 (open) event                              index.js:194757
2 renderableType = 4                                       index.html:101
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
2 Frame processed                                          index.js:194771
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
Frame processed                                            index.js:194771
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
WebGL: INVALID_OPERATION: drawElements: no                 index.js:6290
ELEMENT_ARRAY_BUFFER bound
WebGL: INVALID_OPERATION: bindBuffer:                      index.js:8674
buffers can not be used with multiple targets
```

The browser console shows:

```
WebGL: INVALID_OPERATION: bindBuffer: buffers can not be used with multiple targets

index.js:6290 WebGL: INVALID_OPERATION: drawElements: no ELEMENT_ARRAY_BUFFER bound

index.js:8674 WebGL: INVALID_OPERATION: bindBuffer: buffers can not be used with
multiple targets

index.js:6290 WebGL: INVALID_OPERATION: drawElements: no ELEMENT_ARRAY_BUFFER bound

index.js:8674 WebGL: INVALID_OPERATION: bindBuffer: buffers can not be used with
multiple targets

index.js:8882 WebGL: INVALID_ENUM: texImage2D: invalid texture format

5index.js:8544 WebGL: INVALID_VALUE: texSubImage2D: no pixels

index.js:6290 WebGL: INVALID_OPERATION: drawElements: no ELEMENT_ARRAY_BUFFER bound
```

## 4 Problem analysis and next steps

Given the test results shown above, here we list problems which we consider to be crucial for the further development of the project. The list is prioritized according to the implications on the project.

### 4.1 OpenGL Extensions

OpenGL is designed to be readily extensible. Using the OpenGL extension mechanism, hardware developers can differentiate their products and incorporate new features by developing extensions that allow software developers to access additional performance and technological innovations.

Qt code responsible for OpenGL rendering strongly depends on the extensions available on host system. Moreover, there are vendor specific extensions. Khronos group considers such extensions still to belong a certain version of OpenGL. One can therefore not rely on an OpenGL version. Even the base functions in OpenGL ES 2.0 depend on GPU HW and their driver implementations. The number of supported buffers may vary for example. An OpenGL program is required to query certain parameters and alter its behavior accordingly.

Another problem is that desktop OpenGL, OpenGL ES and WebGL each have their own extension registries. This means that neither desktop OpenGL or OpenGL ES extensions are automatically also WebGL extensions, although some amount of parity does exist.

This problem makes our solution very fragile if it should work one day. The way to solve this problem is to limit Qt OpenGL rendering code to a portable subset of OpenGL commands which are fully portable across different hardware. This would in practice imply very limited use of extensions, if at all. Feasibility of such solution must be discussed with Qt team.

## 4.2 Differences between OpenGL ES 2.0 and WebGL

WebGL is based on OpenGL ES 2.0 standard. However, there are certain changes which were introduced in WebGL in order to improve portability across various operating systems and devices. Due these differences a valid OpenGL ES 2.0 program is not guaranteed to be valid in WebGL context.

For a list of incompatibilities see here: https://www.khronos.org/registry/webgl/specs/1.0/#6

One of the major problem we faced was buffer object binding. In the WebGL API, a given buffer object may only be bound to one of the `ARRAY_BUFFER` or `ELEMENT_ARRAY_BUFFER` binding points in its lifetime. This restriction implies that a given buffer object may contain either vertices or indices, but not both.

The list of incompatibilities contains 28 items which must be analyzed. Emscripten has a special compatibility mode activated by `–s FULL_ES2=1` which provides a full OpenGL ES 2.0 environment and emulates client-side array that are missing in WebGL. This mode is not as efficient as the WebGL-friendly subset, because Emscripten cannot predict the optimal pattern for buffer creation/sizing/etc. It is also not recommended by Emscipten team. See for details: https://kripken.github.io/emscripten-site/docs/porting/multimedia_and_graphics/OpenGL-support.html

However, the idea behind this mode is to provide full compatibility with OpenGL ES 2.0 and the errors we observed in our use cases just suggest that Emscripten does not fully implement this mode yet. We have discussed these issues with Emscripten team and will post bug reports. Hopefully, some of the problems can be fixed in future releases of the framework.

## 4.3 Network latency and traffic

The selling factor of the project which could make this solution competitive with those based on Remote Frame Buffer protocol is lower network traffic. During our tests we have observed relative high traffic between server and clients. Depending on the application each frame had sizes between 2kB to 2Mb. The commands themselves should not take more the few hundreds bytes per frame. The buffers' and texture data take the major part of the traffic. However, content of the buffers does not change for each frame. Thus, by introducing caching mechanisms the overall network traffic could be significantly reduced.

## 4.4 Frame synchronization

Another important aspect which must be tackled in future work is frame synchronization. Server and clients can run on different hardware and handle different maximum frame rates. The frame rate generated on the server could be significantly higher than that each client can consume. Thus, clients cannot keep buffering every received frame. Depending on frame drawing speed clients must dynamically decide which frames will be dropped to guarantee smooth and reliable functioning of the application. However, if there are dependencies between OpenGL calls in different frames such that the next frame cannot be processed without executing the last one, those frames must be marked by the server to prevent clients from ignoring them. Reducing the frame rate produced by the server according to the client needs would also reduce unnecessary network load.

[i] https://www.khronos.org/