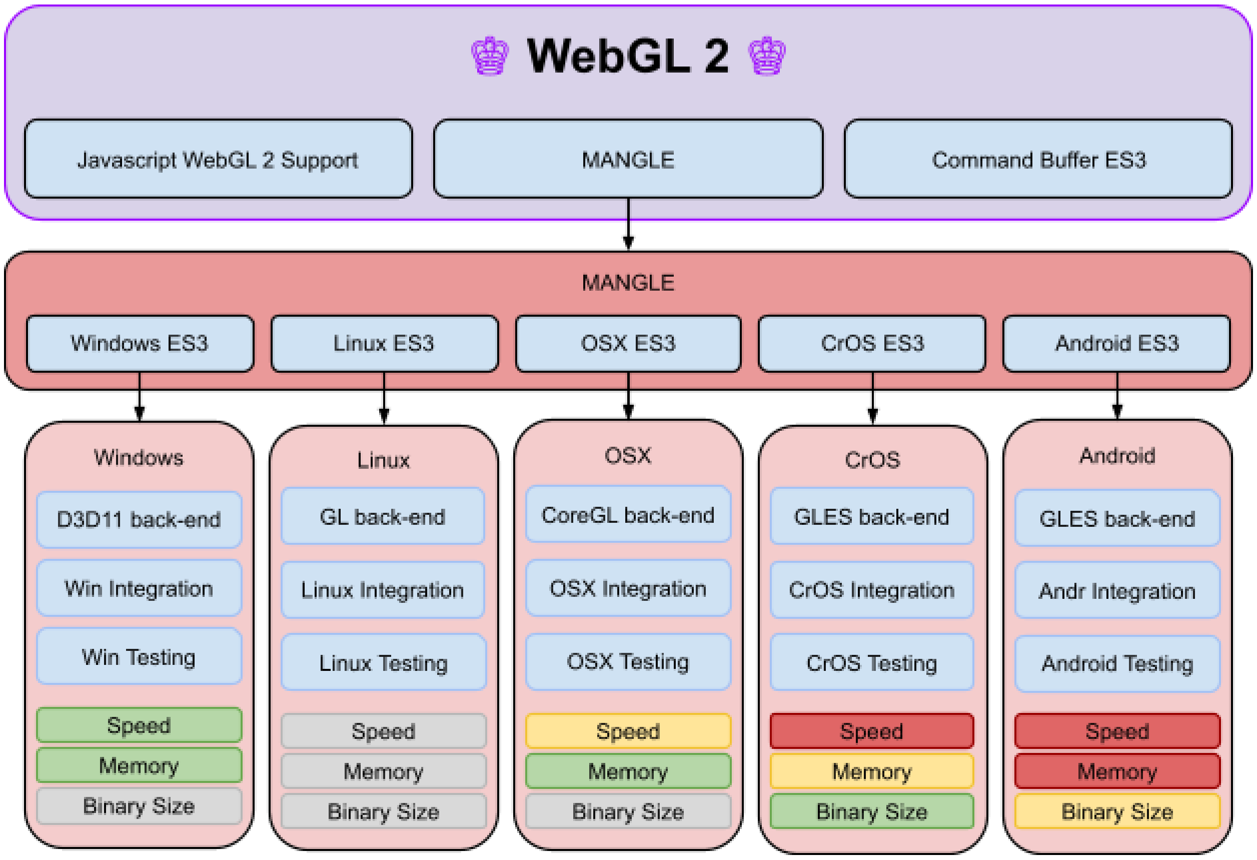
MANGLE+WebGL 2 Roadmap

Written Q2 2015



Objectives: Javascript WebGL 2 support/Command Buffer ES3

These tasks are underway, led by the folks in mountain view (bajones@ and zmo@). The ANGLE team won't necessarily be involved in this directly. The command buffer ES3 objective would mean pass-through messaging, sending graphics commands to ANGLE from the Renderer process without additional validation or translation.

Objective: Back-end Tasks

Back-ends tasks require that we implement a Renderer for a particular platform, such as D3D11 on Windows or OpenGL Core profile on OS X. The Windows D3D11 back-end is mostly complete, as well as the compatibility profile GL backend as of this writing. On Mac OS X, we'll need to implement the core profile GL back-end: OS X has discontinued compatibility profile support past GL 2.1, so for versions 3 and up, which we need for ES3, we'll need to use core profile. On ChromeOS and Android we'll need GLES support. The progress of the GL back-end is described <here>.

Key Result: standalone test passing rate
ANGLE currently has two maintained standalone rendering test suites, the angle_end2end_tests, and the dEQP tests. dEQP also has separate targets for GLES2 conformance and GLES3 conformance. We can measure the completion of a back-end by the pass rate (percentage) of these three test suites, where angle_end2end_tests has a mix of GLES2 and GLES3 tests.

Prerequisite Tasks
ES3 back-end tasks require us to complete the EGL port on each platform before we can run rendering tests, and also require that the tests are ported before we can run them. This means OSWindow and family, since they enable both dEQP and the end2end_tests.

Objective: Integration Tasks
Integration here means integration into Chromium, with ANGLE running as the primary graphics engine, instead of desktop GL. We'll need ANGLE running under Chrome on every platform before we can switch over entirely. This is a prerequisite for WebGL 2 because ANGLE will perform core GL translation on OSX, and save us avoiding implementing all the validation (and state tracking) associated with the 3.x GLES objects and entry points.

Key Result: WebGL CTS passing rate
The WebGL tests give a good numeric measure of how well ANGLE is implementing OpenGL. There are separate tests for WebGL 1 and WebGL 2, so the passing rate of each respectively should give a good measure of how conformant we are for each API.

Prerequisite Tasks
It would be a good idea to complete the majority of the back-end tasks for a platform before we move on to Chrome integration. It's not strictly necessary to port OSWindow and friends, but we will need to port EGL, and giving us access to the standalone tests gives much easier debugging and removes the complication of integration with Command Buffer and javascript. We'll also need the completed command buffer and javascript support for running WebGL 2.

Additional Notes on Integration
See MANGLE + Chrome-integration notes for the discussion document which has notes on several of the key ANGLE+Chrome integration topics, such as EGLImage support, Chromium mailboxes, and video decode. We'd like to also port the GPU driver workarounds into ANGLE, to keep the GL translation logic contained in one module and thin out the command buffer. Some platforms will also likely involve turning on specific OpenGL/EGL extensions. We'll likely have to add extensions to each platform as we encounter them.

Objective: Testing Tasks
Each platform will require that we add WebGL 2 targeted testing to the GPU/GPU FYI bots. This means the standalone tests, and the WebGL tests. Currently we have WebGL testing on every

platform, and angle_end2end_tests on Windows. Soon we will add angle_end2end_tests on Linux, and the dEQP GLES2 tests on Windows.

Key Result: WebGL 2, angle_end2end_tests and dEQP-GLES3 on every platform

The result of this task should be green tests on every platform (excluding for the moment Linux/Intel).

Prerequisite Tasks

This will first require the platform port to be up and running (the back-end task). It will also require us to add the necessary steps to integrate our test build with Chromium, add isolates, and update test expectations as we add the test steps to the bot recipes. It's worth mentioning ANGLE's desire to get try bots up and running as well.

Objective: Performance Tasks

MANGLE is in large part driven by the desire to replace command buffer, to avoid doing similar kinds of work (validation, state tracking, driver workarounds) on two levels in Chromium. Eventually we'd like to treat command buffer as an RPC layer on top of ANGLE, with little in the way of workarounds and translation at that level. Eventually this could lead to improved performance. In the meantime, we need to make sure that for most platforms, and Android especially, we don't have significant regressions. We also need to make sure we're aware of binary size and memory use, so we don't regress significantly on platforms that care about these metrics. Again, Android is the biggest concern here.

Key Result: chromium-based performance tests

The angle_perftests give standalone performance benchmarks, and are good enough for measuring performance within ANGLE, but when we're talking about performance regressions on Android, we need to measure our performance relative to the status quo, which is a direct conversation to the driver.

Prerequisite Tasks/Design

Thus we will need to select or create specific benchmarks[1] with ANGLE+Chromium and measure performance against that. The same would go for memory use and binary size, though standalone perf tests would help. Before we can turn on MANGLE on Android, we need to ensure we have no regressions, hence the following design could be useful: first implement the ANGLE back-end and integration, and enable it behind a flag for the automated testing. Then, start duplicating parts of command-buffer code into ANGLE, and add a flag to skip the redundant code within command buffer. Once the performance of Chromium+ANGLE with the "skip redundant work" flag is within a certain multiplier of the original performance of Chromium+driver, we can decide to remove the redundant code and switch to ANGLE.

[1] See the telemetry tests here: http://www.chromium.org/developers/telemetry and the perf dashboard at https://chromeperf.appspot.com/.

Risks

One high-level issue is that MANGLE currently implicitly does context virtualization internally; when running on top of OpenGL it only allocates a single OpenGL context. Chromium's command buffer also has a virtual GL context implementation (which will be removed once MANGLE is in place), but it's only used on those platforms where it yielded a performance gain. On Android specifically, Chromium used virtual GL contexts everywhere initially, but the trajectory is toward using the driver's real contexts. Some reason supporting virtual contexts: fixes driver bugs or performance issues with context switches, and Chrome relying on glFlush() ordering, which is not guaranteed in EGL. Real context might give better isolation between WebGL/other contexts, and allow us to take advantage of newer features such as context priority. We'll need to carefully consider this design decision.

Platform Priorities

Some key tasks that we can attack independently:

- ES3 support on Windows (~91% passing dEQP, ~3% failures)
    - Mostly stalled except for Nvidia contributions
- Complete Linux integration (cwallez@ working on it)
- Complete Linux testing (cwallez@ also working on it)
- Complete OS X back-end (cwallez@ started)
- dEQP testing on Windows (mostly complete, jmadill@ working on it)
- ChromeOS support (not started)
- Android support (not started)
- Porting cmdbuf functionality to ANGLE extensions (user data, workarounds) (not started)

It might be a good idea to investigate Android support early, but move its implementation to the end due to the performance complexity. We could add the performance testing harnesses around it before we start digging into making ANGLE competitive with native, while also optimizing other platforms.