Wayland on Android

Project introduction and status on June 18th, 2012

Pekka Paalanen
ppaalanen@gmail.com

Collabora

Outline

Collabora

What is Wayland?

Wayland is a new **protocol** for server–client communication in a graphical windowing system, aiming to become a de-facto standard.

- core is extremely simple and small
- designed to be extendable and avoid round-trips
- does not contain any rendering interfaces
- has a well-defined concept of a complete frame
- core includes input, selection (cut & paste), and drag & drop
- does not expose any direct client-to-client communication, except passing file descriptors for selection and d&d data

The design of Wayland

- Start with the minimal set of protocol that modern graphical apps need: user input and graphics output.
- Combine the server, compositor, and window manager into a single process, to minimize IPC.
- Clients render everything themselves, including decorations, and send complete frames.
- "Every frame is perfect."
- Window management by protocol extensions with policy.
- Transition from and co-existence with X is well planned.

What is Android?

Android is an operating system aimed for embedded-like multimedia devices (smartphones, tablets, laptops, in-vehicle infotainment, televisions, . . .).

- runs on lots of fun devices
- completely non-standard windowing system
- crippled "standard" C library (Bionic)
- proprietary IPC: Binder (RPC)
- monolithic GNU Make based build system
- many proprietary closed source drivers from hardware vendors

Why bring Wayland to Android?

Device manufacturers (especially ARM SoC vendors) concentrate on enabling Android. Generic Linux systems may come later, but likely never.
We want to leverage all that hardware enablement, to be able to offer

- well-known, stable, and superior free open source software technologies, and
- an alternative to an Android-only operating system: Android hardware, freedesktop.org middleware, familiar FOSS user applications.

Project goals

A proof of concept Wayland stack running on an Android device.

- Weston running with hardware GL ES 2 on the framebuffer, and working input.
- Working native Wayland clients, software rendered.
- Working native Wayland clients using hardware accelerated GL rendering, and zero-copy buffer submission (i.e. Wayland EGL platform).

The target is Android 4.0 (4.0.1_r1.2 for now) on a Samsung Galaxy Nexus device.

Outline

Collabora

Building autoconf/automake projects

Android build environment requires lots of compiler and linker options, that the Android build system (ABS) sets.
We introduce a special configure step, that must be ran before the real build. There we

- run `autoreconf`
- run `configure` with all the ABS flags, pkg-config overrides, runtime test overrides, etc.
- generate all `$(BUILT_SOURCES)` and
- generate `Android.mk` files with Androgenizer in the automake Makefiles.

All libraries should have also the uninstalled version of `.pc` files.

Bionic missing features encountered

Implemented by fallbacks:

- `SOCK_CLOEXEC`, `F_DUPFD_CLOEXEC`, `MSG_CMSG_CLOEXEC`
- `epoll_create1()`, `EPOLL_CLOEXEC`
- `accept4()`, `strchrnul()`, `mkostemp()`

Still missing, just hacked out:

- `execinfo.h`, `backtrace()`
- timerfd API, signalfd API
- `eaccess()`, `euidaccess()`
- `locale.h`, `localeconv()`

Warning: Thread-Local Storage

Thread-specific data can **only** be used through `pthread_key_t` objects. There are less than 60 slots available.

The keyword `__thread` must not be used. The compiler will happily compile it, but the runtime does not support it. Beware of false positives in autoconf tests.

Outline

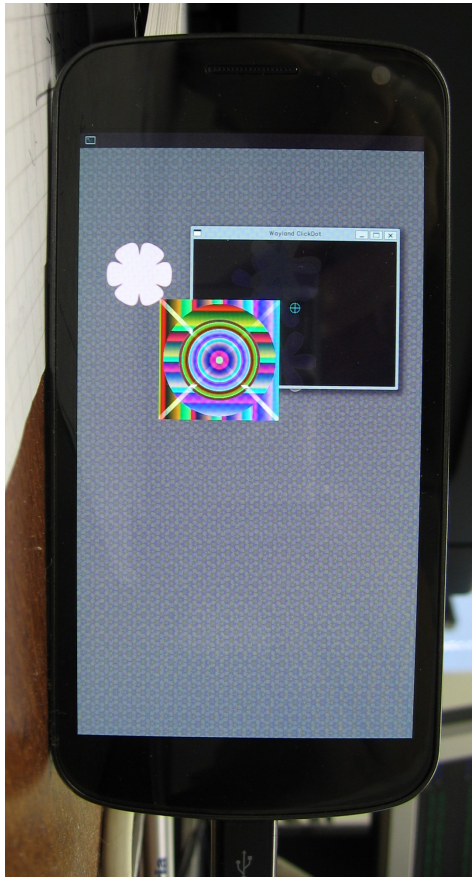
Weston compositor

Weston is runnable and can handle clients.

- graphics output into framebuffer with presumably hardware accelerated GL ES 2
- shared memory (i.e. software rendered) clients work
- hardware GL clients work (as in one ad hoc test app, no standard API!)
- to do: input, timers, signal handling

External libraries needed by Weston: libwayland, libffi, pixman, mtdev, libxkbcommon

Weston demo clients



Ported and working clients:

- simple-shm
- weston-desktop-shell
- clickdot
- flower

Also simple-egl builds (the needed stubs are added to the Android libEGL), but does not run.

Needed external libraries in addition to Weston's dependencies: cairo

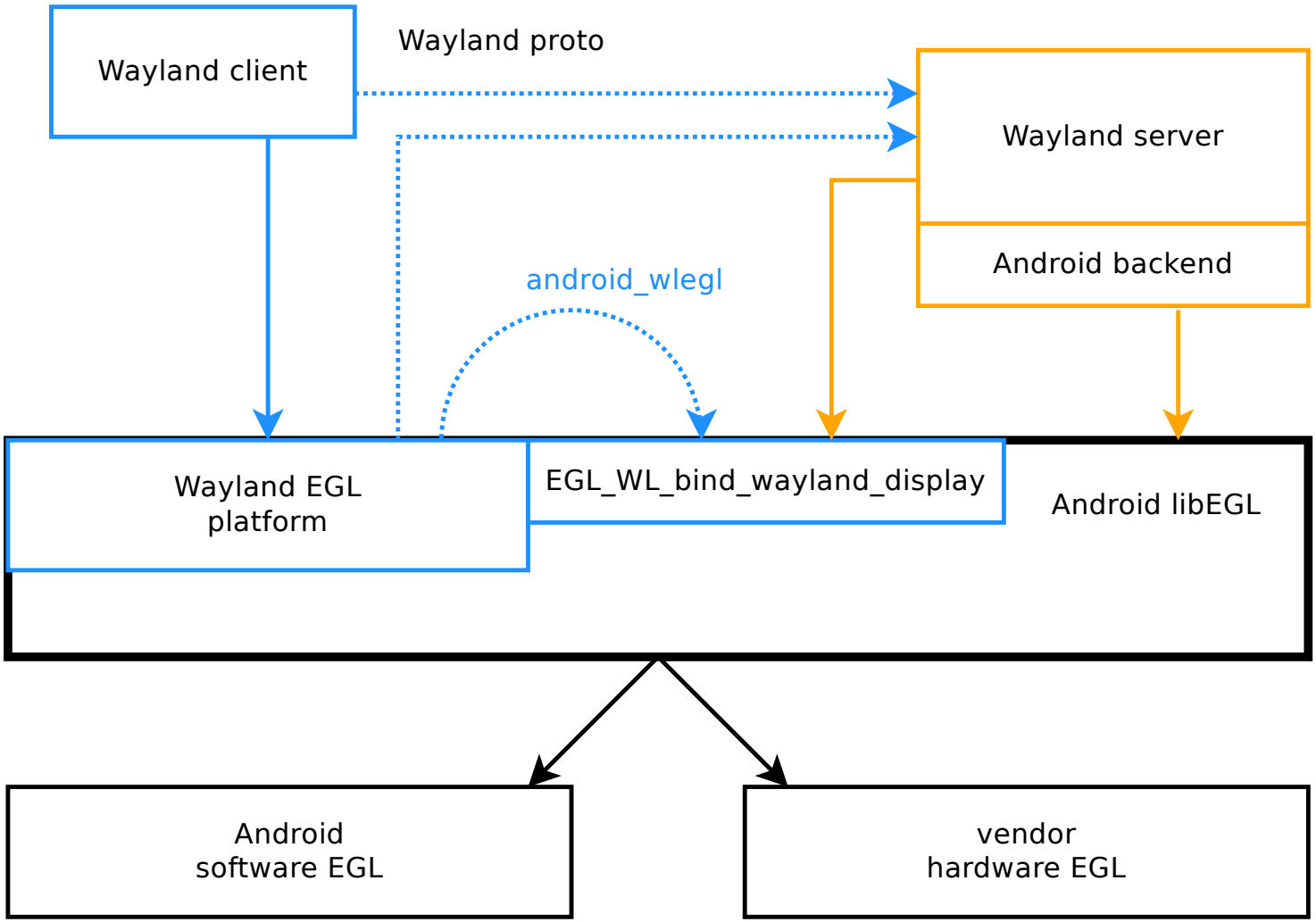
EGL Wayland support

Implementations in Android's libEGL (open source wrapper library).
Wayland protocol extension: `android_wlegl`

- used to implement EGL_WL_bind_wayland_display extension
- passing Android hardware graphics buffers is tested to work
- design similar to `wl_drm` of Mesa
- the design may need to be reversed, if Android does not allow clients to allocate buffers

Wayland EGL platform (the standard client side API) is not yet implemented.

Android libEGL and Wayland



Outline

On my plate

Input support for the Weston Android backend:

- Split udev code from evdev code in Weston. (≈done)
- Reimplement the evdev device opening logic from Android.

The Wayland EGL platform—enables clients to use libwayland-client and wayland-egl along with the standard EGL API for graphics.

Android backend graphics enhancements:

- Do we have vsync, and how does it work?
- Can we know when a buffer turns into light on the screen?
- taking advantage of hardware overlays

Questions?

Thank you!

http://www.collabora.com/services/android/
http://ppaalanen.blogspot.fi/