Wayland

Kristian Høgsberg Kristensen
kristian.h.kristensen@intel.com

Who am I

- Core X developer since 2004 (at Red Hat)
- AIGLX to enable GL compositors
- DRI2 lets GL applications work in a composited environment
- Not developing Wayland because I don't know how X works

What is Wayland

- New display server architecture
- Integrates server, WM and compositor
- No rendering API, all direct rendering

- Sounds ambitiuos, but we're largely just consolidating existing practices

Where do we come from

Breif review of how X used to work and changes we've made over the years.

- X used to do font management, rendering.
- Toolkits used to use sub-windows for widgets
- Modesetting and acceleration code was all tied up in X specific drivers
- Sharing the GPU between applications wasn't possible or grossly inefficient.
- The X server had input drivers to parse a mess of different input device serial formats.

What is a compositor?

Biggest change in recent years

- Application renders into their own private color buffer
- The *compositor* renders the final desktop by painting those buffers on the screen.
- In X applications rendered into the front buffer, divided up by the clipping rectangles.
- Composited desktop is a basic expectation/requirement today.
- Extra copy, more memory.

What is a compositor?

- Wayland makes the compositor the display server.
- Applications talk directly to the compositor.
- The compositor reads input from the input devices and distributes to the applications.
- The applications push their color buffers directly to the compositor, which renders the desktop.

How is Wayland feasible?

Server side:

- Reuses open source driver eco-system.
- Compositor uses KMS to bring up display and manage pageflipping and planes.
- Compositor renders using EGL/GLES2 to the KMS buffers.
- Read input from evdev devices.
- No hardware specific code in the server.

How is Wayland feasible?

Server side:

- Weston is a Wayland compositor written from scratch, the Wayland reference compositor.
  - Less than 10kloc, running this presentation!
- Existing X compositors (mutter, kwin, enlightenment etc) can be modified to also be Wayland compositors.
- Single process UIs on KMS/DirectFB can be made Wayland servers and incorporate content from external applications.

How is Wayland feasible?

Client side:

- Port toolkits: GTK+ 3, Qt 5, EFL, Clutter, SDL
- Rendering and fonts all client side in libs

All works in progress. Challenges:

- Port away from X rendering
- Client side decorations
- No grabs
- Only surface local coordinates
- Per-app X specific access

Driver Support

- Short version: "If you have KMS and can run DRI2, you can run Wayland"
- Longer version
  - Weston needs KMS, and an EGL stack that implements the Wayland compositor extension.
  - Wayland clients depends on the Wayland EGL platform.
  - Mesa and the Linux kernel provides all this for most Intel, nVidia and AMD chipsets.

The Big Plan

We expect that we can release Wayland 1.0 this year:

- 0.85, developer snapshot, protocol changes planned (released Feb 9, 2012)
- 0.90, starting beta, protocol frozen
- 0.9x, release candidates
- 1.0, first stable release
  - Marks beginning of stable protocol and API.
  - Not world domination.
  - Somewhere in second half of 2012.

Wayland 1.0

- First stable release of Wayland protocol and libraries.
- Toolkits can rely on a stable protocol and API form this point on.
- Changes and additions will be done in a backwards fashion.
- Release of Weston compositor 1.0 as well.

Questions?