AGL Compositor update: Sep 2019

Daniel Stone
daniels@collabora.com
Hi, I'm Daniel

Graphics lead at Collabora
Open-source consultancy est. 2005
Wayland core developer
Outline and agenda

- Update on compositor/WM progress
- Window manager and shell architecture recap
- Flexible output management
- Future development and input manager
Update on progress
Compositor/WM progress

- AGL ivi-compositor project created and stood up
- Support for DRM/KMS, Wayland, X11 backends
- Initial home screen ported and available
- Basic window/output management functionality available
- Work beginning to integrate with UCB and make available
- Configured through weston.ini (like old compositor)

https://gitlab.collabora.com/agl/agl-ivi-compositor
Home screen progress

- Current AGL reference UI has ‘all in one’ home screen
- No separation of panels/dialogs/etc into windows at Wayland protocol level
- Ongoing work to separate these out and provide separate surfaces to compositor
- Should be presentable end of September
Window management / shell
Window management concept

• WM based on output/layer/surface (like IVI shell)
• New concept from Weston: surface view
  – Views position an output within a layer
  – Multiple views allow to show surface in different places
  – Crucial for remoting: can create new view for other display or ECU
  – Window manager always controls views!
Window management concept

• Not so different from previous IVI shell!
• Key difference: give OEMs power to manage windows themselves with full API
• Offer callback into OEM module for every window event
  – new window created
  – window content updated
  – window removed
Surface/view relationship

- Compositor creates layers for grouping
- Positions layers within compositor space
- Compositor creates views for each surface to display
- Positions views within layers
- AGL IVI compositor API to manage view creation and positioning
- Display of views handled by libweston

Managed by libweston

- `weston_surface`
  - `wl_surface`
  - width, height, current buffer

Managed by AGL

- `weston_layer`
  - x, y position for output
  - z position
- `weston_view`
  - x, y position in layer
  - z position in layer
Relationship between libweston and AGL views

libweston

**weston_desktop_surface**
(xdg_surface)
surface type
surface title

**weston_surface**
(wl_surface)
width, height
current buffer

**weston_view list**
stored in absolute list order
used by Weston core to display
final UI result

AGL compositor
store list of views and configuration

**agl_view**
position in layer
z position in layer
position before/after other view
shown/hidden

**weston_view**
(x, y) position in layer

recalculate total view list
consider ordering of layers and views

link to layer

create new view

surface change
Why two separate lists?

- Keep IVI concept of Z positioning
- Flexible positioning: allow views to be dynamically enabled/disabled
- Easy integration with OEM WM policy
  - AGL view API can be stable for OEM plugins
- AGL core compositor will maintain translation between two worlds: recalculate libweston list after WM changes
Animation framework integration

- Time-driven animations made available to WM
- Spring physics model provides simple easing
- Parameters are desired end state and time to achieve end state
- Intermediate frames driven by output repaint
- Available animators:
  - Move window
  - Zoom window
  - Fade window opacity
Animation framework example
Next developments for shell

- Collect additional OEM shell requirements through JIRA
- Example of pop-up dialog content such as warnings or status updates
- Integration with Web Application Manager (see afternoon session)
Output management
Output management status

- Current output configuration only handled by weston.ini:

  [output]
  name=HDMI-A-1
  mode=1920x1080
  rotate=270

  [output]
  name=HDMI-A-2
  mode=off
Output management status

- Current output configuration only handled by weston.ini:

```
[output]
name=HDMI-A-1
mode=1920x1080
rotate=270

[output]
name=HDMI-A-2
mode=off
```

Must be made dynamic!
**Output manager concepts**

- Based on Weston’s model with separate head/output
- ‘Head’ represents a display device: HDMI, eDP, DSI screens, or virtual output windows
- ‘Output’ represents a grouped area of pixels to be shown on a head
- Fully exposes capability of hardware and system as separate concerns
Output manager concepts

1. Display 1 (eDP)
   - weston_head
     - Display device
   - weston_output
     - Pixel data to display
   - Weston renderer
     - Composite scene for each output

2. Display 2 (DSI)
   - weston_head
     - Display device
   - weston_output
     - Pixel data to display
Output manager concepts

Display 1 (eDP)  
Display 2 (DSI)  

Output manager concepts

Display 1 (DSI)  
Display 2 (DSI)  

Weston renderer  
Composite scene for each output
Output manager benefits

• Supports complex use cases like clone mode, e.g. all RSE showing same content from one pixel pipe
• Based on runtime dynamic API: window manager can always make policy decisions and change configuration
• Dynamic output management allows for remote displays being added/removed
• Output manager can query head information even if disabled
• Output layout, positioning, etc determined by compositor
Output manager API

• List of weston_head available:
  
  ```c
  struct weston_head *head = NULL;
  while ((head = weston_compositor_iterate_heads(ivi->compositor, head)))
    /* XXX: do something with head */
  ```

• Properties available for heads:
  - name
  - connection status (connected, disconnected)
  - available modes (resolution)
  - EDID/CEA display information
  - content protection
Output manager API

- ‘Heads changed’ signal provided via standard Wayland signal/listener mechanism
- Compositor iterates properties of all heads and configures based on policy
- libweston applies new policy
- Further development required for example future complex usecases
Future development
Input bindings: hardkey & CAN

- Input manager currently only supports runtime application of key bindings
- Create helper module adding support for high-level CAN bindings allowing use of CAN inputs
- Create helper module showing example configuration of bindings (e.g. file on disk configuring actions to be taken when keys pressed)
Input bindings: touch gesture

• Touch gesture bindings currently only support set number of fingers
  - Example: three fingers on screen triggers binding

• Add example swipe gesture recognition to allow switching between applications

• Allow gesture navigation to be customised through configuration
Input manager design

Active client

Window manager

send normal event to focus client

Input manager

Vendor action (special event)

Input event (CAN high-level)

libweston core

Backend

Input source (touch, key)

Input source (CAN low-level)
Multiple backend support

• Presently being developed by ADIT with support from Collabora
• Multiple backends to support heterogeneous environment: some output via DRM/KMS, other output into virtualised display (safety-critical/IC domain), other output into remote display (second-screen/RSE)
• Requires capability for multiple simultaneous hardware backends
• Preliminary work being done before upstream
Miscellaneous items

• Support for overlapping outputs: required for efficient virtualisation / remote display to present same content to multiple displays without hardware assistance
• Advanced display configuration: allow shell to prepare UI for reconfigured output before output becomes active
• libwayland integration with SMACK to query client label
CIAT integration

- Much work gone into upstream Weston test suite recently
- GL support for headless renderer designed to allow aggressive testing on development or headless devices
- Not currently integrated with AGL testcases and CI
- After CES demo work complete, develop test plan and integrate tests into AGL infrastructure
Thankyou!

daniels@collabora.com