Arc++ Graphics
Rendering, Compositing and Window Management

David Reveman
reveman@google.com
Arc++ is Android running alongside ChromeOS and providing ChromeOS users with Play Store Android apps without developer action.
Background

App Runtime for Chrome (ARC), was launched September 2014 with a few curated applications. Due to the heavy involvement of Google Engineering team on curating those apps, in April 2015, we decided to open the program for anyone to submit an app. Despite doing so, there has not been a lot of developers submitting applications.
Arc++ Goals

Full Play Store access
Minimize Android Framework changes
Keep Chrome OS secure
Maintain Chrome’s update model
Graphics Overview
Rendering

Hardware-accelerated Canvas API

OpenGL ES 2.0 since Android 4.0

OpenGL ES (GLES) directly

Everything is rendered onto a "surface."

Producer side of a buffer queue that is consumed by SurfaceFlinger

The Gralloc HAL is used to allocate buffers for each buffer queue

ARC++

Gralloc and the GLES driver are using the Direct Rendering Manager (DRM)

Apps still have access to a fully accelerated GLES implementation

Apps using a different rendering API works too
Compositing

Android windows are backed by surfaces

- Surfaces are sent to SurfaceFlinger for compositing onto the display
- SurfaceFlinger uses GLES for compositing

HWComposer HAL

- Allows the OEM to handle some surfaces using overlay hardware

ARC++

- All surfaces are handled by HWComposer
- Forwarded to Chrome for compositing together with the rest of the ChromeOS UI
- Chrome compositor uses overlay hardware when available
Window Management

Shared between Android and Chrome

Chrome
- Maximize, minimize and full-screen
- Application switching, multiple profiles, screen magnifiers, etc.

Android
- Absolute positioning
- Resizing
DRM/KMS

Direct Rendering Manager “DRM”
- Used for rendering and buffer allocation on both Android and ChromeOS
- Allows for efficient sharing of graphics resources (DMA Buffers) between the two platforms
- Chrome is DRM-Master and can program a display-controller
- Android doesn’t need mode-setting capabilities in ARC++
  - Render-node access to the GPU is sufficient

DMA Buffer Sharing and PRIME
- Linux kernel internal API designed to provide a generic mechanism to share DMA buffers
- PRIME file descriptors are used for buffer sharing
- Unix operating systems provide a safe way to pass them through an Unix domain socket using the SCM_RIGHTS semantics
Gralloc

Provides a type of shared memory on Android that is also shared with the GPU

- Written to directly by regular CPU code
- Used as a OpenGL texture

ARC++

Gralloc implementation that allocates buffers through the DRM API

Android “surfaces” are backed by DRM buffers and can be shared with Chrome using PRIME file descriptors
Ozone/Freon

Chrome platform abstraction layer beneath the Aura window system that is used for low level input and graphics

Supports underlying systems ranging from embedded SoC targets to new X11-alternatives on Linux such as Wayland

ChromeOS devices that run ARC++ uses the Freon graphics stack

Under the Freon driver model, the Chrome browser talks directly to the kernel's DRM/KMS APIs

GpuMemoryBuffer

Allows platform independent code in Chrome such as the compositor take advantage of low-level graphics buffers

Designed to support Chrome’s multi-process architecture and security model

Implementation backed by DRM buffers and PRIME file descriptors

Extended for ARC++ with support for importing of foreign graphics buffers

Used by ARC++ to have Gralloc allocated DRM buffers imported into Chrome as GMBs
## Pixel Formats

For a DRM buffer to be imported into Chrome the pixel format needs to be supported by Chrome. Chrome's support for pixel formats has been extended for ARC++.

<table>
<thead>
<tr>
<th>Format</th>
<th>FourCC</th>
<th>Compositing</th>
<th>HW Overlay</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL_PIXEL_FORMAT_RGBA_8888</td>
<td>DRM_FORMAT_ABGR8888</td>
<td>Enabled</td>
<td>Enabled (Minnie)</td>
</tr>
<tr>
<td>HAL_PIXEL_FORMAT_RGBX_8888</td>
<td>DRM_FORMAT_XBGR8888</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>HAL_PIXEL_FORMAT_BGRA_8888</td>
<td>DRM_FORMAT_ARGB8888</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>HAL_PIXEL_FORMAT_RGB_565</td>
<td>DRM_FORMAT_RGB565</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
<tr>
<td>HAL_PIXEL_FORMAT_YV12</td>
<td>DRM_FORMAT_YVU420</td>
<td>Enabled</td>
<td>Disabled</td>
</tr>
</tbody>
</table>
Exosphere

Chrome component that provides the functionality needed for an external client to connect to Chrome
- Protects Chrome from malicious clients
- Efficient delivery and integration of graphics frames

Built on top of the GpuMemoryBuffer framework
- Takes advantage of Chrome’s ability to import graphics buffers

Shared Window Management
- Some WM operations handled by Chrome (Ash) some by Android WM

<table>
<thead>
<tr>
<th>WM Operation</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimize</td>
<td>Ash</td>
</tr>
<tr>
<td>Maximize</td>
<td>Ash</td>
</tr>
<tr>
<td>Fullscreen</td>
<td>Ash</td>
</tr>
<tr>
<td>Close</td>
<td>Both</td>
</tr>
<tr>
<td>Activation</td>
<td>Both</td>
</tr>
<tr>
<td>Position</td>
<td>Android WM</td>
</tr>
<tr>
<td>Size</td>
<td>Android WM</td>
</tr>
</tbody>
</table>
Chrome Compositor

Responsible for compositing contents from web pages and ChromeOS UI

- Clients produce frames in the form of a set of quads
- Each quad is typically backed by a GLES texture
- GpuMemoryBuffer framework allows these textures to ultimately be backed by GMBs
- Quads are rendered or scheduled as overlays by the compositor instance in the browser process

ARC++

- Exosphere produces frames in the form of a number of quads
- Quads are backed by GMBs that have been created by importing the graphics buffers allocated by Android
Synchronization

Preventing Android from reusing a buffer before GPU has finished reading from it
Wayland

Protocol for a compositor to talk to its clients as well as a C library implementation of that protocol

ARC++

- Graphics, window management and input related communication between Android and Chrome

Benefits

- De facto protocol for secure compositor communication by Linux apps
- Provides well established interfaces for most of the communication needed between Android and Chrome
- New interfaces are easily added as needed for ARC++
- Existing world of Wayland clients provides an excellent set of examples and integration tests
- Offers maximum code reuse and minimizes the attack surface if we were to enable more generic container support
Wayland Interfaces

In addition to the core wayland protocol, Chromium implements the following interfaces:

- wp_viewporter
- xdg_shell_v5 (zxdg_shell_v6 experimental)
- wl_drm
- zwp_linux_dmabuf_v1
Chromium Wayland Interfaces

**cr_alpha_compositing**

Allows the client to specify the blending equation and alpha value used for compositing

**cr_gaming_input**

Provide secure access to gaming input devices for a given seat (limited to gamepads today)

**cr_aura_shell (aka remote_shell)**

Extension to xdg_shell that provides additional shell functionality needed for ARC++

**cr_secure_output**

Allow surfaces to be marked as only visible on secure outputs

**cr_stylus**

Allows a wl_pointer to represent an on-screen stylus

**cr_vsync_feedback**

Informs the client about vertical synchronization timing in a precise way and without unnecessary overhead
Future Wayland Interfaces

Explicit synchronization
  Releasing buffers
  Presentation timing
Protected buffers
  Digital rights management
Building and Testing

Exosphere code:
src/components/exo/

Wayland bindings code:
src/components/exo/wayland/

Wayland extensions:
src/third_party/wayland-protocols/

$ gn args out/exosphere
Contents of args:
target_os = "chromeos"
use_xkbcommon = true

$ ninja -C out/exosphere exo_unittests chrome

$ ./out/exosphere/chrome --enable-wayland-server
Questions?
THANK YOU