OVERVIEW

Allocator Review
What Changed?
Implementation in Nouveau
Results and Next Steps
HIGH-LEVEL GOALS

Arbitrate allocation properties across multiple devices and "engines"/usages

Support for broad set of engines - Not a "graphics" or "3D" API

Support Vendor-specific capabilities, including fully opaque capabilities

Support image usage “transitions”
ALLOCATOR’S SPOT IN THE ECOSYSTEM

- Application: Wayland Compositor
  - EGL/Vulkan Driver
  - Vendor1 Drv
  - DRM
  - Device 1
- Application: Wayland EGL Client
  - EGL/Vulkan Driver
  - Vendor2 Drv
  - Device 2
  - Device 3
  - ION Drv
  - ION

Userspace
Kernel
ALLOCATOR OBJECTS

ASSERTION
The desired width, height, and format of a surface

USAGE
A single desired application of a surface, such as rendering, on a single device

CONSTRAINT
An imposed surface limitation for a given assertion and usage

CAPABILITY
A supported surface feature for a given assertion and usage

CAPABILITY SET
A valid combination of constraints and capabilities
USAGE TRANSITIONS

Vulkan introduced the idea of explicitly transitioning between various surface uses.

Could be generalized across devices now that we can describe all usage explicitly.

Apps could query usage transition “meta-data” from allocator for usage pairs.

That meta-data could then be passed into GPU APIs to perform transitions.
WHAT CHANGED?
Where do DRM Format Modifiers fit into this?

Why not use existing APIs to describe usage and report capabilities?

Should the allocator build upon GBM or be a new API?

Is this project still a thing?
RE-EVALUATE HIGH-LEVEL GOALS

Arbitrate allocation properties across multiple devices and "engines"/usages

  Support only intersection of capabilities, not union of constraints

Support for broad set of engines - Not a "graphics" or “3D” API

  API-agnostic building blocks. Relies on Graphics APIs for some features

Support Vendor-specific capabilities, including fully opaque capabilities

  All participating driver components must be aware of capability to make use of it

Support image usage “transitions”

  Re-allocation still expensive, wasteful. Optimal steady-state still important
ALLOCATOR OBJECTS

**ASSERTION**
The desired width, height, and format of a surface

**USAGE**
A single desired application of a surface, such as rendering, on a single device

**CONSTRAINT**

**CAPABILITY**
A supported surface feature for a given assertion and usage

**CAPABILITY SET**
NEW ALLOCATOR CONCEPT MAPPINGS

**ASSERTION - VULKAN/GL/GBM IMAGE CREATION PARAMETERS**
The desired width, height, and format of a surface

**USAGE - VULKAN IMAGE CREATION PARAMS, DRM, GL, ETC.**
A single desired application of a surface, such as rendering, on a single device

**CAPABILITY LIST - SINGLE DRM FORMAT MODIFIER**
A set of supported surface features for a given assertion and usage. Constraints not expressed

**USAGE TRANSITIONS - SWITCH FORMAT MODIFIERS IN PLACE**
Driver reports “compatible” modifiers. Driver deduces necessary transition from modifier pair
IMPLEMENTATION IN NOUVEAU
THE BLOCK LINEAR TILING FORMAT

Blocks made up of 1+ GOBs, or “Groups of Bytes”

Block dimensions can vary in width, height, and depth

Block width, height, and depth must be a power of 2

Byte layout in GOBs varies by “Page Kind”, and by GPU family
BETTER NVIDIA FORMAT MODIFIERS

Block-Linear is complicated, but finite

DRM_FORMAT_MOD_NVIDIA_BLOCK_LINEAR_2D(c, s, g, k, h)

Backwards-compatible with
DRM_FORMAT_MOD_NVIDIA_16BX2_BLOCK()

Uniquely defines surface layout down to bit level across all shipping NVIDIA GPUs

Reserves fields for more complex surfaces in the future

In practice, block width is always 1, depth 1 for 2D surfaces

Few “Page Kind” values used for single-sample 2D color surfaces

Max ~24 modifiers per GPU
Existing Mesa Nouveau format modifier support was limited
Only useful for sharing buffers between Mesa and TegraDRM
Added format modifier support to Nouveau DRM-KMS for Tesla through Turing
Added support for new format modifier layout to TegraDRM
Added support for setting EGLImage GEM buffer layout from format modifier on nvc0
Added support for compressed layout format modifiers in Mesa nvc0 driver
EXPOSED USAGE TRANSITIONS

[EGL, GL]_EXT_transition_format_modifier

// Query a list of format mods a given “source” mod can be transitioned to
eglQueryDmaBufCompatibleModifiersEXT(EGLDisplay dpy, EGLuint64KHR srcMod,
   EGLint *maxDstMods, EGLuint64KHR *dstMods,
   EGLint *numDstMods);

// Transition layout from one modifier to another
glTransitionFormatModifierEXT(GLbitfield mask, GLuint64 dstMod, GLuint64 srcMod);

"mask" is similar to glClear(), but limited to color buffer for now

Could use multi-buffer and/or DSA versions as well

When intersecting two format modifier lists, allow modifiers only in one list if a
compatible modifier is in the other list
IMPLEMENTED USAGE TRANSITIONS

NVIDIA 3D hardware supports internal lossless compression for most common formats. Enabled globally for a 3D engine instance, as well as per-mapping via “page kind”.

NVIDIA display hardware does not support this compression. “compressed” page kinds can have corresponding uncompressed page kind.

Newer hardware can transition between the two by decompressing in place:

- NV_A297_DECOMPRESS_SURFACE method

Also “invalidates” compression. Transition back to “compressed” layout is a no-op.
OTHER TRANSITIONS

Some NVIDIA Tegra engines also support a Color Decompression Engine (CDE) format.

Tegra GPUs can translate GPU’s compression data to CDE without decompressing.

CDE treated as read-only mapping. Transition back to GPU compression still a no-op.

Did not attempt to implement this in current patches.
RESULTS AND NEXT STEPS
PERFORMANCE GAINS DEPEND ON USAGE

Modified kmscube to use compressed layouts, decompress-in-place

Performance comparable to non-compressed case, as expected, when decompressing

Performance 50-300% better when not decompressing

A further-modified kmscube that behaves like a composited desktop steady-state presentation workflow gains ~80% overall
NEXT STEPS
Collect Feedback and Review, Productize

Get format modifier layout reviewed & committed
Clean up mesa patches adding format modifiers, get reviewed & committed
Write extension spec for EXT_transition_format_modifier - Collect Feedback
Integrate support into Wayland protocols/compositors and client libraries
Implement equivalent support in our binary drivers
Tackle Other Allocator Issues...
OPEN ISSUES

Constraints - Fix bugs like https://gitlab.freedesktop.org/xorg/xserver/issues/895

Capability Set - Need association between modifier and its constraints

Vendor-agnostic dma-buf factory - Is this needed?

API-agnostic capability set generator - Is this needed?

Comptags - 19-bit identifiers for compression store of a page. Where to stash them?

Persistent Transitions - Requires modifying mappings or separate mappings

Protected content - property of memory (GEM buffer) or mapping (Format modifier)?
QUESTIONS?
REFERENCES

Nouveau Kernel tree: https://gitlab.freedesktop.org/cubanismo/linux

Nouveau Mesa tree: https://gitlab.freedesktop.org/cubanismo/mesa

kmscube Testing tree: https://gitlab.freedesktop.org/cubanismo/kmscube

Original allocator prototype: https://gitlab.freedesktop.org/allocator/allocator

XDC 2017 Allocator Talk:

https://www.x.org/wiki/Events/XDC2017/Program/#james_jones

XDC 2016 Allocator Talk:

https://www.x.org/wiki/Events/XDC2016/Program/jones_unix_device_mem_alloc/