HDR Hackfest

The color-and-hdr (https://gitlab.freedesktop.org/pq/color-and-hdr/) repo contains links to introductions to digital color, a glossary of terms we use in wayland protocols, design documents for color management and HDR on linux.

How to learn about digital color

gnome-shell HDR

- Required gnome-shell/mutter version 44.0
- Display supporting HDR
    - EDID announcing support for PQ Colorimetry and HDR Metadata
    - edid-decode /sys/class/drm/card*/edid
        - Colorimetry Data Block: BT2020RGB
        - HDR Static Metadata Data Block: SMPTE ST2084
- DRM driver supporting Colorspace and HDR_OUTPUT_METADATA connector properties
    - drm_info
- Enable HDR mode via looking glass
    - Alt+f2, type “lg”, enter
    - global.backend.get_monitor_manager().experimental_hdr = 'on'
    - global.backend.get_monitor_manager().experimental_hdr = 'off'
- Play back HDR video (https://developers.google.com/media/vp9/hdr-encoding) in a “dumb” video player (totem, MPV, gst-launch pipeline, …)
    - Are command line or config options required to disable the players tone/gamut-mapping? (added by Harry Wentland)

For Intel an upstream kernel should work, for AMD a custom kernel is required (https://gitlab.freedesktop.org/swick/linux/-/tree/test/amd-colorimetry).

HDR KMS signaling (short term)

We need a reliable, generic and upstream way to enable HDR modes in displays as soon as possible. The sink to source communication via EDID/DisplayID works. The source to sink communication is in a worse state. There are two bits of information that need to be signaled to the sink for HDR:

- CTA-861 Colorimetry InfoFrame for HDMI or the MSA/VSC SDP Pixel Encoding/Colorimetry Format for DP
- CTA HDR Static Metadata InfoFrame on both HDMI and DP

When a source sets the Colorimetry, the sink expects content with specific primaries, white point, transfer characteristics, YCC to RGB conversion.

When a source sets the HDR Static Metadata the sink expects content according to the Colorimetry (see above) except with the transfer characteristics of the HDR Static Metadata. It also contains data which the sink can use to improve its own tone and gamut mapping.

KMS properties

HDR_OUTPUT_METADATA: Allows user space to set the HDR Static Metadata InfoFrame. No issue AFAIK.

Colorspace: Allows user space to set the Colorimetry InfoFrame or MSA/VSC SDP Pixel Encoding/Colorimetry. This one is problematic.

There are RGB and YCC Colorimetry variants. User space does not control the pixel format on the wire and has no way of knowing the pixel format that will be chosen for a commit. On AMD with HDMI for example one can experience YCC pixel formats being chosen. There is no way to choose the correct variant.

Possible solutions:

1. Let user space control pixel formats
2. Treat RGB and YCC variants as equal in the kernel and let it chose the correct InfoFrame
3. New property

Problems with solutions:

1. A new property to select the pixel format will make some modes which previously worked fail (Auto variant might be able to work around this?). Might be hard to implement.
2. Not a great API because we start lying but it could work. Probably the least amount of work though.
3. Yet another API which might or might not get supported.

HDR KMS (long term)

- Currently the KMS color pipeline is mostly opaque to user space: There might be a LUT and a CTM somewhere in there but all other conversions (input YCC to RGB, output RGB to YCC, input/output limited/full range conversions, ..) happen without user space control.
- The API is partially descriptive (is this plane RGB or YCC?) and partially mechanical (LUT, CTM).
- The order of operations is undefined
- Properties control both the implicit conversions of the color pipeline as well as the display signaling (Broadcast RGB, max bpc)
- Lots of color pipeline capabilities are not exposed

Two different approaches to fixing this:

1. Describe the input content and the output requirements and let the kernel come up with the right conversions and signaling
2. Split color pipeline and signaling. Expose color pipeline capabilities to user space.
3. Vendor specific pipeline description and a user space library providing a generic abstraction.

Last time most people favored the 2nd/3rd option mostly because compositors which do opportunistic KMS offloading require the KMS conversion to be visually identical to the shader version. Is this still consensus?

Starting point for 2: https://gitlab.freedesktop.org/pq/color-and-hdr/-/issues/11

Tl;dr: New DRM cap, remove all the existing properties which influence the color pipeline, add new properties which describe possible configurations of the color pipeline

Could we create a prototype with VKMS?

We discussed this approach and Simon Ser suggested introducing new drm objects for the drm_color_pipeline and possibly for drm_color_pipeline elements:
- drm_pipeline objects
- bitfield indicating what plane a pipeline can be used on
- driver can create many pipelines
- only one pipeline can be attached to a plane
- pipelines have a certain number and order of elements
- drm_pipeline would have array of pipeline elements
- drm_pipeline_element objects

how do we communicate scaling:
- pipeline_element_type: scaling

should we use multiple blobs or single blob for pipeline:

questions:
- how heavy is setting many blobs?

drm object idea vs old color pipeline blobs idea?
- everyone likes drm object idea
- explore the idea

element types we need:
- scaling
- 1D Curve
  - might be different types, such as traditional LUT, PWL, or enumerated TFs
- 3D LUT
- Matrix
  - might include enumerated matrix conversion, as well as custom

example pipe:
1. 1D curve
2. Matrix.
3. 1D curve

userspace might use this pipeline like:
1. program first 1D curve to linearize pixels
2. program the matrix for color space conversion
3. program the second 1D curve to go from linear pixels to blending space
4. program the GAMMA property on crtc to go from blending space to display space

Some vendors have very custom transformations, such as NVidia's LSM -> ICtCp conversion with tone mapping on "I" and saturation adjustment on CtCp. They could be exposed as custom element type with a well-documented mathematical description.

First goal is to tackle the pre-blending pipeline. We'll want to do something similar post-blending but it might not be needed as a first step.

Draft proposal with new DRM object type

Plane property: "color_pipeline" enum describing a list of possible COLOROP object IDs (head of the linked list, 0 to disable/bypass)
(Alternatively, blobs which contains a list of COLOROP object IDs)

DRM_MODE_OBJECT_COLOROP
Properties:
- "operation" enum (bypass, 1D curve, 3D LUT, matrix, nvidia special snowflake, etc)
- "1d_curve_type" enum (LUT, PQ, sRGB, PWL, ...)
- "lut_data" blob
- "lut_size" read-only integer
- "next" object ID COLOROP (next item in the linked list)

Wayland protocols

There are 3 wayland protocols involved in defining the content colors.

linux-dmabuf

unstable/linux-dmabuf · main · wayland / wayland-protocols · GitLab

- Specifies the pixel format
- Is used widely
- Clear scope and no issues for color management

color-representation

https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/183

- Specifies the alpha mode (straight, premult), the YCC to RGB conversion matrix, the chroma sampling location, and full vs limited range
- Not merged yet but is ready for implementation

color-management

https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/14

- Specifies the color space primaries, white point, transfer characteristics, and used color volume. It supports both ICC profiles as well as video system standards.
- We also want to support the remaining HDR static metadata fields as well as viewing environment and reference display description.
- Work in Progress, could use more feedback

Frame timing and VRR

For FRR there are generally 3 strategies to frame timings from a client POV:

1. Hot-loop rendering: the client produces frames as fast as it can
2. FIFO: the client produces a frame and then blocks until the next refresh cycle
3. Racing the deadline: the client knows when the deadline for a refresh cycle is and tries to finish its rendering just a bit before it

Vulkan WSI present mode for each case:

1. MAILBOX or IMMEDIATE
2. FIFO
3. MAILBOX or IMMEDIATE

The wayland protocol currently supports MAILBOX natively. The FIFO present mode is implemented by blocking on wayland frame callbacks. IMMEDIATE can be supported with the wayland tearing extension.

Wayland compositor implementations:

- Have their own refresh cycles which has a different phase than the display refresh cycle
- Modern compositors try to issue a single KMS commit shortly before the kernel deadline for the next display refresh cycle
- Might have to composite a new image which takes time
- Compositing starts some time before the kernel deadline and hopefully finishes in time
- Some compositors measure the time it takes to finish compositing and adjust the time at which they start compositing, some use a fixed time

Possible tearing implementations in wayland compositors:

- Only enable tearing in fullscreen direct scanout cases, issue a KMS commit as soon as a wayland commit becomes ready/available
- Same as in the non-tearing case but when the deadline is missed, instead of trying again for the next refresh cycle, let the KMS commit tear
- 1st approach in the direct scanout case otherwise the 2nd approach
- …

VRR can be implemented similar to tearing:

- Only enable VRR in fullscreen direct scanout cases, issue a KMS commit as soon as a wayland commit becomes ready/available
- Compositor targets the fastest refresh cycle of the VRR mode for compositing and issues a KMS commit when done. If the fastest refresh cycle is missed VRR will extend it. The frame clock needs to be adjusted each refresh cycle.

VRR can also be implemented in a way that involves the client more:

- The client can decide when a commit should be presented
    - e.g. a video player wanting to display 24hz content

    - e.g. to make sure the content that was produced for a specific time gets presented at that time even if it finished the work for the content earlier

KMS implementation:

- Normal commits makes the flip synced to vsync and commits cannot be replaced by a newer commit until they get presented
- Async commits make the content flip as soon as they become ready
- Commits with VRR active flip some time in vblank and commits cannot be replaced by a newer commit until they get presented. The exact window is display and driver specific.

VRR display and kernel implementation (take this with a grain of salt):

- The vblank period is not fixed but has minimum and maximum duration
- This is reported in a EDID/DisplayID block
- Changing vblank duration too much will cause flickering
- Kernel can provide a bigger range using LFC
- Kernel can limit the increase and decrease of the vblank duration from one refresh cycle to the next

Problems with VRR kernel support:

- No information about *actual* minimum and maximum vblank duration with LFC etc.
- No information about the increase and decrease limits each frame
- Do we even need the kernel to set those limits? Can't user space decide itself?

Problems with KMS async/amending commits:

- Async is not widely supported
- Amdending is not supported at all? I.e. being able to replace a KMS commit with a newer KMS commit.
- Maybe amending is not needed because compositors move towards committing right in front of the deadline anyway? But if we miss a deadline then the commit will be stuck until the next refresh cycle and we can't produce a newer commit for that cycle.
- Feedback needs to communicate what amendment made it in time for the deadline

Problems with wayland:

- The deadlines are not communicated
- VRR information is not communicated (other than want/dont-want with another protocol proposal)
- The frame callback mechanism for FIFO can block indefinitely
- Can not schedule wayland commits in the future
    - I.e. a commit always gets presented at the next possible time
    - Need to block in Vulkan queuePresent with FIFO because of this
    - Hard to reliably hit a certain refresh rate with VRR when you have to block in the client to hit the right timing

Related links:

- https://github.com/KhronosGroup/Vulkan-Docs/pull/1364
- https://gitlab.khronos.org/vulkan/vulkan/-/merge_requests/5223
- https://gitlab.freedesktop.org/wayland/wayland-protocols/-/merge_requests/199

Display control

- DisplayPort
    - We want a SBTM mode:
        - InfoFrame + DisplayID
        - Guarantee passthrough
- We want more control precise control over the Backlight/Brightness
    - DisplayID block which maps backlight control to absolute luminance
    - DisplayID block which shows availability of automatic brightness control
    - InfoFrame/DDCCI/whatever to turn automatic brightness on/off
- Want better certification for…
    - … all of the above plus
    - Subpixel layout (current QD-OLED have weird layouts but the nothing about this is machine readable)
    - DDCCI
        - Any user setting that affect the monitor colorimetry must be reflected properly in some DDCCI parameter
    - Monitor adjustment may only change the Default colorimetry, not any colorimetry set by an InfoFrame (e.g. bt2020)
- Require brightness control for HDR modes

HDR & Color testing/CI

- Want to test color management and HDR in CI
- Needs signaling support (i.e. full/limited range, colorimetry infoframe, hdr TF and metadata, …)
- VKMS
    - Currently doesn't support any kind of signaling
    - Currently doesn't support CTM, gamma, and other color pipeline properties we want to use
    - Could all be added
    - Readback + signaling can be used to compare colorimetry to expected colorimetry
    - Can't test real drivers
- Chamelium
    - Could do the same as VKMS but testing real drivers
    - Could maybe be added to fdo CI?
- Display Writeback (added by Harry Wentland)
    - Using VKMS (See above for missing bits)
    - Via actual HW
        - Amdgpu is working on enabling writeback support for testing
        - Some limitations might apply that won't exist with something like Chamelium or other external capture solutions

HDR / Color Use-cases

- It would be good to summarize the use-cases we all want to see for HDR / Color
- Main things:
    - Video playback
        - “Kind of easy” to do fullscreen HDR
        - Even “fullscreen” has to support overlays for settings menus, notifications, cursor, etc.
        - Blending of SDR subtitles with HDR video
    - Gaming
    - Color-critical work (e.g., image or video color correction work)
    - Traditional color-management; professional color management like print proofing
    - Measuring profiles, validating them (https://gitlab.freedesktop.org/pq/color-and-hdr/-/issues/27)
    - Screen recording of HDR content
    - HDR content display on both HDR and SDR displays
        - Ideally the input and output can be managed independently
    - Battery life on mobile devices
    - Performance/latency impact of HDR
    - Management of HDR brightness when scrolling across HDR content late night
        - See https://www.theverge.com/23529794/instagram-apple-iphone-hdr-video-brightness-samsung-annoying
        - Contradiction: do we want to give users control over limiting brightness and using night light, vs content creators wanting to show the exact colors, brightness
        - Content providers might not want the desktop to mangle (tone-map, etc.) their HDR content
        - SDR white vs HDR bright spots -> brightness slider?
        - See also https://prolost.com/blog/edr
    - Do we want to support direct metadata passthrough for content that wants to pass their metadata directly to the TV/display?
        - Passthrough is possible only with exactly one exclusive fullscreen surface on a monitor, which means no surfaces for things like notifications, cursors, sub-titles… which is extremely limiting, and therefore cannot be dictated by Wayland clients, it needs to be opportunistic but also not flickery during a switch.
    - Managing HDR content with Night Light features
    - What happens if you have 2 HDR-aware apps at the same time?
        - For example GIMP in one part of the screen and some video in another part
        - “The fundamental problem is that when you have to blend a BT.2020/PQ surface with another BT.2020/PQ surface, there is no single correct answer even in theory on what the blending formula must be.”

- Expressing a specific whitepoint for the desktop (via night light, etc?)
- Rendering-intent:
  - Allowing applications to communicate their rendering intent
- Display brightness adjustment in HDR mode
  - A lot of displays/TVs allow you to set brightness, but that doesn't make sense when dealing with HDR
  - How does that even scale with something like different viewing environments?
    - For example, my TV screen showing a scene at night during the day when the sun is shining through the windows
    - Something that the HDR spec doesn't deal with really
  - We might need to support SW adjustment for brightness for displays that don't allow brightness adjustments for HDR
- What about other platforms that have had HDR support for a while now?
  - E.g. Android phone, Windows 11, MacOS, iOS
  - For example when they show an SDR app with an HDR app simultaneously on the screen, how do they then deal with changes on the brightness slider
  - We should be wary though that this is an unsolved problem, and that a lot of the other implementations carry their own problems
  - Also: one person might have a different opinion on this than another (Blue/Grey dress discussion on Reddit :D)
- We should consider accessibility
- How do we test it all?

Now that we have all these topics, what do we take as topics?

Let's start with the short term stuff:

- Fullscreen HDR (without any titles/subtitles/other content)
  - What about the cursor plane?
    - With the current KMS API, the cursor is broken
- Getting HDR signalling working well enough so that userspace can start playing around with it

---

Notes topic 1: Frame timing / VRR

- Use case: gaming: games know when they will enter complex scenes, when not
- Problem: even a cursor change can drive up the refresh rate
    - Sudden changes in modes lead to flickering, making some games unusable
    - Screen gets brighter or darker
- We'd need a kernel API to make sure we're not going too much modesetting
    - Or a new timing API which sets a minimal timestamp?
- There's now a VESA certification for AdaptiveSync
    - Those monitors should be able to deal with sudden changes
- A game that wants to make use of VRR can't control stuttering, the compositor can
- The ideal goal is to turn on VRR by default for everyone and that it gets automatically used when needed
    - Problem is that we can't do this right now, because there are too many screens currently that are flickering when enabling VRR
- Problem with different use cases (and what if they show up on the screen at the same time)
    - Video with a fixed refresh rate
    - Games with VRR
    - Mostly static application
        - For example web browser: reading an article (static), if you then scroll it's very choppy
    - Contrast with power saving
    - Example: digital signage
        - 1 FPS for most of the time, but when it changes, it should ramp up immediately
        - Won't we run into the same issues as with GNOME Shell/Mutter, where the GPU is too slow to ramp up and we get stuttering?
- Possible solution: let the application tell what kind of refresh rate it would want?
    - Or even a flag to request VRR? Or to request "not VRR"?
        - If we enable by default: problem with regressions
        - If we don't enable it: not enough games will turn it on, which will defeat the purpose
    - Maybe have an allowlist/denylist somewhere that can be configured by the user?
- Part of the problem is that we're trying to guess what the user/app is doing
- More discussion needed
    - Do we solve in userspace? Or do we want kernel API to update the cursor without ramping up refresh rate?
    - Are we sure this is not going to break assumptions?
    - Another option is to have a uAPI for the atomic commit to indicate "no earlier than" target timestamp, and combine it with amend semantics. It would allow user space to manage cursor updates.

Notes Topic 2: short term KMS

- Sidetopic: One of the problem is that debugging KMS stuff is quite hard
    - Might be an idea to put stuff in debugfs
- We need to set the screen in HDR mode
- ColorSpace DRM property
    - Potential YUV on the wire should not block usage of this property
    - At this point, ColoSpace property doesn't care about YUV at least
    - Basic use case: userspace to enable BT2020, setting that through the info frame
    - Harry Wentland will adjust the patches to work with the agreed approach
- Issue with atomic commits and HDR metadata
    - Sending HDR metadata is … interesting in general
    - In NVIDIA drivers, setting that metadata should not necessarily need a modeset
    - Can we document which properties (don't) need a modeset? Maybe
        - Enabling/disabling infoframes
        - Changing transfer function in infoframes
    - For some properties, we can be pretty sure that we'll need a modeset
        - Maybe we could make a modeset mandatory there? Then userspace can use that expectation to build around that
- What about cursors
    - Well, since it's its own plane, that's probably better for the general plane discussion?
    - Userspace should treat the cursor plane the same as the primary plane, thus apply gamut mapping itself if it doesn't want cursors to be very bright when HDR is enabled
- Some properties that we're missing?
- Simon Ser to Improve kernel uapi documentation to define enum values etc.

Notes topic 3: Color (management)

- In one line: how to give the intended perception of colors
    - See also https://gitlab.freedesktop.org/pq/color-and-hdr/
- Color representation protocol: define whether surfaces use premultiplied or straight alpha, how to convert YUV to RGB
- Color management: gives meaning to (RGB) color values
- Difference between colorspace and colorimetry
    - Colorimetry is the science of measuring color.

    - Color space is a definition of what color values mean, how they relate to colorimetry.
- Harry proposes creating a lIbrary for color management to be used across compositors, toolkits, etc. due to the complexity of the topic. Sebastian agrees. Pekka wants to be cautious until we have more understanding of the different pieces.
- Short term, anything will be using shaders probably anyway
    - There is still plenty of room for a library to come up with the correct operations
- Pekka explains the use of littlecms in weston
    - Reason: support for ICC files
    - Problems
        - Littlecms has been designed to perform the color conversions in the CPU (not the GPU). Vitaly Prosyak has been working on making sure we can generate shaders instead.
        - HDR: littlecms is using the relative luminance like all traditional or ICC workflows, which doesn't work anymore with HDR
    - Question: is it even possible to deal with both the ICC "workflow" and HDR "workflow" with the same tool? Should we keep them separate
        - Pekka will attempt to unify both in Weston under LittleCMS.
- Difference between calibration and profiling
    - Calibration: changing the knobs to adapt the monitor behavior to display images the way you like
    - Profiling: measuring how the monitor behaves; for specialized use cases, when you want more of your colorimetry
- How much do we want to do the client vs the compositor?
    - For some clients, e.g. va-api it might be better that the compositor does the conversion, since they can make use of the kms api
    - Ideally, we don't need to turn on the GPU for certain color conversions if e.g. we can put something directly on a plane → would need KMS API
        - Of course, if there's no such plane available, then the compositor will need to do the conversion again

Color in KMS

- KMS (planes) is there for offloading, but you can't always rely on planes being available
    - Scenario where you have to continuously switch between composition in the compositor and the planes
        - At some point, users might start noticing
    - To do this well enough , you need to know about blending spaces and even your scaling spaces
        - Descriptive model
    - Prescriptive model: KMS exposes a set of basic operations (and their order)
        - Example operations: 1D LUTs, 3D LUTs, scale, …
            - For example AMD provides a HDR multiplier

        - Challenges:
            - Difference between equally spaced LUTs vs segmented LUTs
            - How to map hardware constraints (since we're now really close to the hardware)
                - For example in AMD we decided that 4096 LUTs are good enough, and to use maths for the other parts
    - After discussions with swick: color pipeline API
        - Color pipeline is the new kid on the block that obsoletes all color-related properties
        - Example step in the pipeline: setting a custom LUT
        - Userspace sends a configuration to the kernel driver
        - At this point just an idea, or a sketch
    - More recently, some people got impatient with all these discussion and tried to make something happen
        - Melissa posted patches on dri-devel that basically map to AMD color blocks
        - Harry proposes to discuss implementing driver-specific apis for the short term, until a long-term solution is in place
            - Possible problem: we can't ever get rid of those vendor-specific properties when some userspace depends on it
            - Hardware blocks also change between hardware generations so that API is also not necessarily going to match inter-generation
            - Maybe we can work around that by not defining this on the kernel uAPI but in a userspace library like Vulkan, or in a way similar to libcamera
                - Downside: doesn't encourage vendors to come to common solution
            - Would be different than what the KMS API has been trying to achieve for years
                - Then again, people haven't been able to do come up with something in the current API
        - Maybe we can generically describe the hardware though?
            - For example: query the precision of the LUT, what kind of LUT is available
        - Querying with atomic API might work, but problem is that you don't get feedback why something is not working
        - What if the hardware really requires e.g. a specific colorspace?
            - Maybe make mandatory operations, or that are read-only?
            - Can't always make it read-only though, since it needs to be programmed
                - But it still need to be programmed according to the content's color space
                - NVIDIA hardware has a specific fixed-function element that converts to LMS (and then internally to ICtCp)

            - So some pipeline that has an input colorspace to an output colorspace would not match the NVIDIA hardware well
            - Fear is that clients will not want to deal with that fixed stuff and fall back to shaders (being less perfmant, more power hungry) on NVIDIA which makes it worse wrt competitors
        - We _could_ maybe just expose it as a mathematical operation instead?
        - This could mean we'd end up with each vendor just providing the blocks matching their own hardware
            - Actually, this is already an imperovement compared to everybody doing shaders.
- **Proposed solution**: have a descriptive user library that translates to a prescriptive KMS API
    - Maybe extend libplacebo to do this?
    - If this then doesn't work out, then it's easier to provide a prescriptive API later on, than the other way around
- Discussion on Harry's drm_color_pipeline sketch
    - Harry posted his sketch in the "HDR KMS (long term)" section
- YUV to RGB / YUV on the wire
    - Wouldn't it be possible to describe that with a 3D LUT?

Day 3 Topics and Notes

- Testing
- Driver-specific color properties
- Drm_plane color pipeline
- Post-blending color pipeline
- KMS wishlist
- VRR

Driver-specific Color Properties

Discussion

Intro: AMD needed a 3D LUT and added a property for that. In the end, were adding a lot of properties that would be usable for them. Started noticing that some properties were really AMD-specific and specific for certain hardware generations/revisions. Started defining some vendor-specific properties, since that also allows them to check/test the usage of their hardware

blocks. This is already being used in Gamescope (due to urgency of the feature) even. In other words, AMD have been trying to get something done that works right now.

The discussion of yesterday seem to point that trying to define a too generic HDR pipeline, leads to not being able to map to hardware very well.

Whatever we come up with, we need to make sure that it will be tested extensively, in multiple places also (like IGT).

Another point is that finding a generic pipeline will take a long time before everything lands upstream (something like a year or 2). With vendor-specific properties, we can be quicker which means we can fill the short-term gap. Downside of the short-term gap is of course that vendors would need to support those properties forever.

We could hide the use of vendor-specific properties with that userspace library we talked about yesterday. Are there any takers to make that library though?

How can we make sure that there's still an incentive for vendors to come with a generic API? Maybe require a userspace test? Some test in IGT? This would mean that for example if a generic API is written by AMD, Intel would have to write their implementation. Uma agrees that's fine.

Can we somehow mark some API as experimental? We could hide behind a KConfig that's disabled by default, but might not be enough? Or module parameter? For userspace side, they need to be able to deal with missing KMS properties anyway. Should we print into dmesg that drm using experimental feature?

There are some concerns whether such a generic APi will ever happen, since past discussions on dri-devel never advanced really. Should we put a time limit? We can't special-case any specific vendor though. How do we make sure that both vendors and distros are compelled to improve the situation, as we want to move away from driver-specific properties. Can we maybe allow IHVs to have them experiment in generation N but require something generic by generation N+1 or N+2? Also clearly define that if the promise is not met, those properties might be removed or they can't add more properties

Action items

- Write down the proposal on vendor-specific properties (and on color management only) and post it on the mailing list

DRM plane color pipeline

Discussion on KMS uAPI

- See “Draft proposal with new DRM object type” by Simon Ser
- We need a guarantee that user-space can fallback to shaders if it needs to. Capability to guarantee that turning off the color pipeline doesn’t require a modeset? If user-space can enable the color pipeline without a modeset, it can expect the pipeline to be able to be disabled without a modeset? Set the LUTs to identity to emulate bypass?
- On AMD and Intel setting some LUTs might fail (AMD: ramp too steep, Intel: ramp goes down)
- On AMD and NVIDIA falling back to bypass doesn’t require a modeset. Make it a requirement for supporting the new uAPI. Seems like we have consensus on this.
- Precision? Fixed types for LUT data? Would be simpler, can always add new op types for higher precision. Still might want to advertise actual hw precision in some way? But there are many kinds of precision. 24 bit precision on Intel so might want to use something larger than u16. Downsampling silently in the driver might change the result but probably not user-visible, still might be a good reason to expose more hw details to user-space so that they can better replicate the output with shaders. Could add precision read-only props for each colorop in the future if we need to.
- Do we want to expose PWL/segmented LUT? For AMD, probably stick with LUTs. Intel interested in this (patch on ML). Feel free to send a patch for it but not a focus. https://patchwork.freedesktop.org/patch/452591/?series=90825&rev=2
-

Action items

- Send RFC to dri-devel for the new uAPI (Simon)
- Write the AMD impl (Harry, Melissa)
- Write the Intel impl (Uma)
- Write the NVIDIA impl (Alex)
- Write the gamescope impl (Simon)
- Write the IGT (Harry, Melissa)

Post-blending Color Pipeline

Discussion

~~We’ll probably (still a bit of a mystery :) ?) need post-blending for NVIDIA, since they don’t have any color conversion elements anymore after …. For normal usage, they do blending in linear space, then apply PQ.~~ Scratch all that, should be fine :D

Basically: Color blending on the CRTC.
Color pipelines like in the plane case but on the CRTC.

Would obsolete the properties :DEGAMMA, CTM, GAMMA -> client cap to enable the new world and disable the old one
Do we still want to allow some automatic colorspace in case the compositor doesn't care?
How do we want to deal with bandwidth constraints? Do we want compositors to care about stuff like MST (DSC) etc? Or do we want to give compositors some minimal stuff like min_bpc?
→ Don't we have all of these problems already with all the current APIs too?
→ maybe we can expose a single property for DSC min bpc, without exposing the whole DSC shebang
We could do that and let userspace test, so that they can decide to for example use a smaller resolution in favor of color fidelity
→ see also "KMS Wishlist" on getting feedback why commits are failing (e.g. bandwidth)

Action items

- Add COLOR_ENCODING CRTC/connector property (RGB, YUV420, …) for how data is sent over the wire → less important for now
- Add min_bpc CRTC/connector property

Testing

Discussion

Use cases

- Mechanical bits of the kms api
- Compositor mechanical bits (exercice API, no hardware dependent)
- Hardware

Problem is that we're using vkms in our CI, which doesn't expose all properties, and also doesn't really equal to what the actual hardware will be using.

What about using something like Camellium? We can reach out to Google and see their plans around future Camelliums and in parallel find an alternative.

We currently have igt test, to test crc? Stuff. If we need VKMS testing for hardware, then we need volunteers for teaching VKMS how to do stuff like 3D LUTs

    We'd need to implement the math for each of those properties

Adding more pixel formats to handle the variety of (color spaces???), such as floating point, might limit the hardware it can be run on, but that's already an expected cost.

https://lists.freedesktop.org/archives/wayland-devel/2022-July/042279.html

We agreed compositor/kernel developers should participate in VKMS discussions, reviews and general help. In addition, there is lack of mentorship for the amount of potential mentees, due to the low entry barrier for contributing to VKMS.

Hardware

Chromium testing with Chamelium - https://www.chromium.org/chromium-os/testing/chamelium/
XDC 2019 talk: https://www.youtube.com/watch?v=L7RLkMB3yFo
RTSP encoders where you can compare their output vs the expected outputb

Action items

- Teach VKMS the new colorspace properties
    - Ensure it doesn't impact performance when not enabled
- Add more pixel formats (floating point)
- Teach LLVMpipe to import VGEM dmabufs
- Write proposal on plan for mandating vkms and the need for an "experienced" internship to bootstrap it (Jonas)
    - Contact xorg foundation for coming up with an "experienced internship" to fund the bootstrapping (Carlos)
- For hardware testing, gather ideas from the community by sending an email to a public mailing list - which mailing list, and who is gonna do it?

KMS potential wishlist

- Feedback why commits fail
    - Bandwidth
- Mandatory VKMS for new apis!
- Non-changing commit with event results in page flip (which might already be required)
- Mailbox page flip - agreement seems to be too complex, and to make things work in userspace and only require kernel solutions if we hit hard limits not solvable in user space.

VRR

Discussion

- Start with summary about previous days discussions and conclusions
- How will we deal with the cursor plane
    - Using LFC (low frame rate compensation)
    - Scan from a different framebuffer with the exact same content so we can force the pageflip
- So: which proposal do we want to take forward? And what is at kernel side or compositor side?

- Google is working on enabling VRR in the compositor and experimenting with handling the cursor problem in userspace, and see if it's too much of a hack or not.
- Probably we need a way to disable LFC in the kernel to ensure there's no jumps in the refresh rate if the compositor doesn't make the deadline for the lowest refresh rate the display supports
- One challenge is detecting which monitor and client supports VRR, we might need a protocol to query that or a database of clients/games like Windows does.
    - Michel proposes that a good place to host a monitor database is libdisplay-info. "Short" term solution most likely.
    - Long-term, display manufacturers should provide data and provide it in EDID, for which we will need a VESA standard spec for that. (HDMI might have an extension with a max change rate, but HDMI is a bit problematic)

Action items

- To check if hardware drivers can do noop commit with event results in page flip. NVIDIA and Intel support it. AMD we assume it's doable, but not happening right now. Harry to look at it.
- Figure out if we can use HDMI max_vrr can be used for figuring out the rates (Simon). (Check section 7.6.3 Setting MVRR)

Yesterday notes:

- Experiment with Real Time low frame rate compensation and see if that helps with flickering
- Amend cursor, update lsc with real time to avoid flickering
- Look at what hardware can support page flips in the situations above
- Potential workaround is using a different cursor buffer with the same content. Simon thinks this is mostlt a hack and a would be better to find a proper alternative.