HDR stack enablement in Gnome Mutter

Authors

Naveen Kumar <naveen1.kumar@intel.com>

Uma Shankar <uma.shankar@intel.com>

Agenda

- HDR: Introduction
- Types of HDR – HDR10 vs HDR10+
- HDR Metadata & Transfer functions
- Color Transformation & Tone mapping
- Functional Flow of HDR in Gnome Mutter
- Key design consideration
- Implementation plan
- Future of HDR on Gnome Mutter
- References

HDR Basics

- It is a technique to reproduce greater dynamic range of luminosity
- Bright's are brighter; darks are darker.
- It increases the range and granularity of luminance i.e., from very dark values (0.00005 nits) to very bright values (10000 nits).
- Standard Dynamic Range (SDR) (pre-HDR displays): max ~100 nits.
- HDR defines up to max 10000 nits.
- 2 primary HDR standards are:
  - HDR10 : Static HDR
  - HDR10+: Dynamic HDR

Static HDR vs Dynamic HDR

- Static HDR uses a single image descriptor in metadata that is a compromise that applies to every scene and every frame of the whole movie.
- Dynamic HDR enables a noticeable progression in overall video image quality from SDR to static, and now static HDR to dynamic HDR.
- Dynamic HDR ensures every moment of a video is displayed at its ideal value for depth, detail, brightness, contrast and wider color gamut's – on a scene by scene or even a frame-by-frame basis.

HDR Metadata & Transfer Functions

- HDR metadata
  - x,y chromaticity coordinates for color primaries and white point (i.e., color gamut).
  - Maximum luminance (in cd/m^2).
  - Minimum luminance (in cd/m^2).
- Transfer functions
- Electro-Optical Transfer Function (EOTF):
  - Defines how display should convert non-linear digital signal to linear light values.
  - sRGB is the defacto EOTF for SDR.
  - Two common HDR EOTFs:
    - SMPTE 2084: Perceptual Quantizer (PQ).
    - Hybrid-Log (HLG).
  - To create digital signal, GPU needs to do inverse of EOTF (aka “OETF”).
- OETF is used by image sources (camera, graphics processor) where as EOTF is used by sink devices such as a display panel. The following equations describe OETF and EOTF mathematically. The figures compare OETF and EOTF with gamma encoding and decoding.

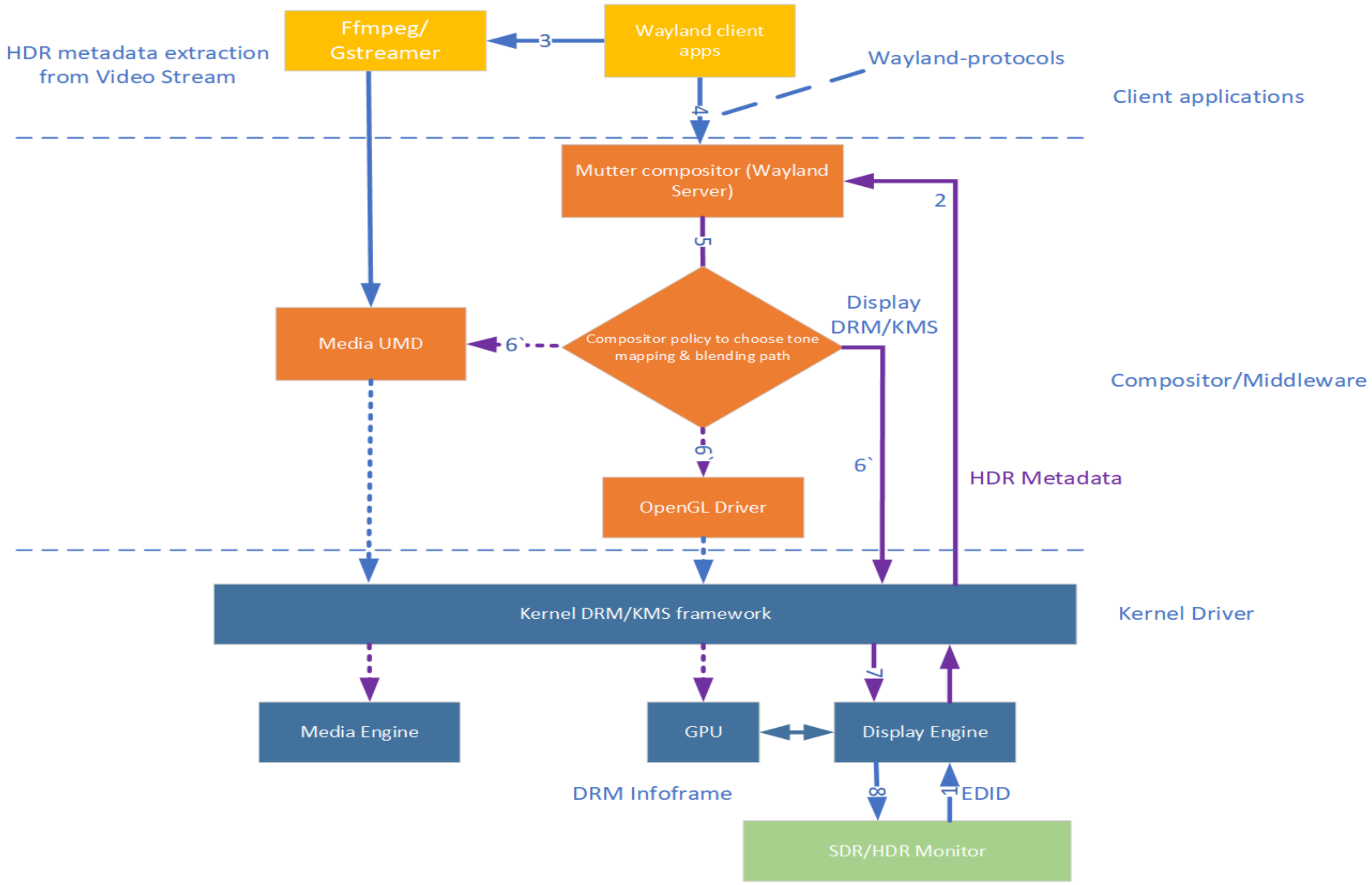
```
double m1 = 0.1593017578125;
double m2 = 78.84375;
double c1 = 0.8359375;
double c2 = 18.8515625;
double c3 = 18.6875;
```

- double oetf_output = pow(((c1 + (c2 * pow(input, m1))) / (1 + (c3 * pow(input, m1)))), m2); // input within [0.0, 1.0] range.
- double eotf_output = pow(((max((pow(input, (1.0 / m2)) - c1), 0)) / (c2 - (c3 * pow(input, (1.0 / m2))))), (1.0 / m1)); // input within [0.0, 1.0] range.

Color Transformation & Tone mapping

- Color transformation:
  - It is to convert respective layers from one colorspace/format to a common colorspace/format for blending
- Tone Mapping:
  - It is a mechanism to map content luminance to panel luminance or, to one common luminance level for blending.
- Compositor needs to define a blending policy.
- Based on the hardware capabilities, either use
  - Display Engine
  - Media Engine
  - GPU
- Tone mapping decisions will be based on below usecases:
  - HDR to HDR
  - SDR to HDR
  - HDR to SDR

Functional Flow of HDR in Gnome Mutter



Key design consideration

- How client parses the HDR metadata from content
  - Client can directly create the metadata based on the input rendering content
  - It can parse the HDR metadata with the help of FFMPEG/Gstreamer media frame
- Wayland-protocols for HDR
  - To pass the hdr metadata to compositor
- Wayland Server side implementation for HDR in Gnome Mutter
  - This will receive the metadata from client and will help in making blending decisions
- Mutter/KMS Interface
  - Based on the blending policies, the final metadata is converted into DRM blob and passed to driver through KMS interface
- Media engine/Vaapi interface - HDR vaapi/libva interface
  - Tone mapping can also be done using Media engine. Compositor can choose based on hardware capability & related policies
- 3D GL/Vulkan shaders implementations
  - Tone mapping can also be done using GPU. Compositor can choose based on hardware capability & related policies

Client HDR metadata parsing

- FFMPEG/Gstreamer Media framework is used for content demuxing|decoding|metadata-parsing
- Reference implementation :

[clients/simple-hdr-video-gbm.c · test-hdr · Naveen Kumar / weston · GitLab]()

```c
frame = demux_and_decode(&app->video);
if (!frame) {
    fprintf(stderr, "no more frames?\n");
    return;
}


sd = av_frame_get_side_data(frame, AV_FRAME_DATA_CONTENT_LIGHT_LEVEL);
if (sd) {
    ll_metadata = (AVContentLightMetadata *)sd->data;
    max_cll = ll_metadata->MaxCLL;
    max_fall = ll_metadata->MaxFALL;
}
sd = av_frame_get_side_data(frame, AV_FRAME_DATA_MASTERING_DISPLAY_METADATA);
if (sd) {
    hdr_metadata = (AVMasteringDisplayMetadata *)sd->data;
    if (hdr_metadata->has_luminance && hdr_metadata->has_primaries) {
        ensure_hdr_surface(app);
        rx = wl_fixed_from_double(av_q2d(hdr_metadata->display_primaries[0][0]));
        ry = wl_fixed_from_double(av_q2d(hdr_metadata->display_primaries[0][1]));
        gx = wl_fixed_from_double(av_q2d(hdr_metadata->display_primaries[1][0]));
        gy = wl_fixed_from_double(av_q2d(hdr_metadata->display_primaries[1][1]));
        bx = wl_fixed_from_double(av_q2d(hdr_metadata->display_primaries[2][0]));
        by = wl_fixed_from_double(av_q2d(hdr_metadata->display_primaries[2][1]));
        wx = wl_fixed_from_double(av_q2d(hdr_metadata->white_point[0]));
        wy = wl_fixed_from_double(av_q2d(hdr_metadata->white_point[1]));
        max_luma = wl_fixed_from_double(av_q2d(hdr_metadata->max_luminance));
        min_luma = wl_fixed_from_double(av_q2d(hdr_metadata->min_luminance));
        zwp_hdr_surface_v1_set(
                        app->hdr_surface,
                        rx,
                        ry,
                        gx,
                        gy,
                        bx,
                        by,
                        wx,
                        wy,
                        max_luma,
                        min_luma,
                        max_cll == -1 ? 0 : max_cll,
                        max_fall == -1 ? 0 : max_fall);

        zwp_hdr_surface_v1_set_eotf(
                        app->hdr_surface,
                        ZWP_HDR_SURFACE_V1_EOTF_ST_2084_PQ);

    }
} else {
    // No metadata for this frame.
    destroy_hdr_surface(app);
}

```

Wayland-protocols Interface

- Wayland-protocols specifies the communication between a display server and its clients
- Reference implementation:

[unstable/hdr-metadata/hdr-metadata-unstable-v1.xml · test-hdr · Naveen Kumar / wayland-protocols · GitLab]

unstable/hdr-metadata/hdr-metadata-unstable-v1.xml 0 → 100644 +115 -0

```xml
+
+ <interface name="zwp_hdr_surface_v1" version="1">
+
+   <description summary="hdr surface">
+     An extension interface to the wl_surface object to set the HDR metadata
+     associated with the surface.
+   </description>
+
+   <enum name="eotf">
+     <entry name="ST_2084_PQ" value="0"
+            summary="SMPTE ST 2084:2014, HDR EOTF of Mastering Reference Displays"/>
+     <entry name="HLG" value="1"
+            summary="Hybrid Log-Gamma (HLG) based on ITU-R"/>
+   </enum>
+
+   <request name="set">
+     <description summary="set the HDR metadata for a surface">
+       Set the HDR metadata for the associated wl_surface. The HDR metadata
+       state is double buffered and will be applied on the next
+       wl_surface::commit.
+     </description>
+     <arg name="display_primary_r_x" type="fixed" summary="Red primary X"/>
+     <arg name="display_primary_r_y" type="fixed" summary="Red primary Y"/>
+     <arg name="display_primary_g_x" type="fixed" summary="Green primary X"/>
+     <arg name="display_primary_g_y" type="fixed" summary="Green primary Y"/>
+     <arg name="display_primary_b_x" type="fixed" summary="Blue primary X"/>
+     <arg name="display_primary_b_y" type="fixed" summary="Blue primary Y"/>
+     <arg name="white_point_x" type="fixed" summary="White point X"/>
+     <arg name="white_point_y" type="fixed" summary="White point Y"/>
+     <arg name="min_luminance" type="fixed" summary="Minimum luminance"/>
+     <arg name="max_luminance" type="fixed" summary="Maximum luminance"/>
+     <arg name="max_cll" type="uint" summary="Maximum content light level"/>
+     <arg name="max_fall" type="uint" summary="Maximum frame-average light level"/>
+   </request>
+
+   <request name="set_eotf">
+     <description summary="set the HDR metadata for a surface">
+       Set the eotf curve for the associated wl_surface. The eotf of the
+       surface state is double buffered and will be applied on the next
+       wl_surface::commit.
+     </description>
+     <arg name="eotf" type="uint" enum="eotf"
+          summary="Electro optical transfer functions for non linear encoding"/>
+   </request>
+
+   <request name="destroy" type="destructor">
+     <description summary="destroy the hdr_surface_v1 object">
```

Wayland Server Metadata handling

- Wayland Server side implementation for HDR in Gnome Mutter
- Reference implementation:

src/wayland/meta-wayland-hdr.c

src/wayland/meta-wayland-hdr.c 0 → 100644 +213 -0

```
+
+ /* Implements the protocol function set metadata */
+ static void
+ hdr_surface_set_metadata (struct wl_client *client,
+                           struct wl_resource *surface_resource,
+                           uint32_t primary_r_x, uint32_t primary_r_y,
+                           uint32_t primary_g_x, uint32_t primary_g_y,
+                           uint32_t primary_b_x, uint32_t primary_b_y,
+                           uint32_t white_point_x, uint32_t white_point_y,
+                           uint32_t min_luminance, uint32_t max_luminance,
+                           uint32_t max_cll, uint32_t max_fall)
+ {
+   MetaWaylandSurface *surface = wl_resource_get_user_data (surface_resource);
+
+   meta_verbose ("Implement set hdr metadata on the surface");
+
+   MetaHdrMetadata *data = surface->pending_state->hdr_metadata;
+   data->metadata_type = HDR_METADATA_TYPE1;
+   STATIC_METADATA(primaries.r.x) = wl_fixed_to_double(primary_r_x);
+   STATIC_METADATA(primaries.r.y) = wl_fixed_to_double(primary_r_y);
+   STATIC_METADATA(primaries.g.x) = wl_fixed_to_double(primary_g_x);
+   STATIC_METADATA(primaries.g.y) = wl_fixed_to_double(primary_g_y);
+   STATIC_METADATA(primaries.b.x) = wl_fixed_to_double(primary_b_x);
+   STATIC_METADATA(primaries.b.y) = wl_fixed_to_double(primary_b_y);
+   STATIC_METADATA(primaries.white_point.x) = wl_fixed_to_double(white_point_x);
+   STATIC_METADATA(primaries.white_point.y) = wl_fixed_to_double(white_point_y);
+   STATIC_METADATA(max_luminance) = wl_fixed_to_double(max_luminance);
+   STATIC_METADATA(min_luminance) = wl_fixed_to_double(min_luminance);
+   STATIC_METADATA(max_cll) = max_cll;
+   STATIC_METADATA(max_fall) = max_fall;
+ }
+
+ /* Implements the protocol function set_eotf */
+ static void
+ hdr_surface_set_eotf (struct wl_client *client,
+                       struct wl_resource *surface_resource,
+                       uint32_t eotf)
+ {
+   MetaWaylandSurface *surface = wl_resource_get_user_data (surface_resource);
+
+   meta_verbose ("Implement set hdr transfer function (eotf) on the surface");
+
+   MetaHdrMetadataEotf internal_eotf = META_EOTF_TRADITIONAL_GAMMA_SDR;
+   MetaHdrMetadata *data = surface->pending_state->hdr_metadata;
+
+   switch (eotf) {
+     case ZWP_HDR_SURFACE_V1_EOTF_ST_2084_PQ:
+       internal_eotf = META_EOTF_ST2084;
+       break;
+     case ZWP_HDR_SURFACE_V1_EOTF_HLG:
+       internal_eotf = META_EOTF_HLG;
+       break;
+   }
+
+   data->metadata_type = HDR_METADATA_TYPE1;
+   STATIC_METADATA(eotf) = internal_eotf;
+ }
+
```

```
+
+ /* Destroy the hdr surface */
+ static void
+ hdr_surface_destroy (struct wl_client *client,
+                      struct wl_resource *resource)
+
+ {
+   wl_resource_destroy (resource);
+ }
+
+ /* Destroy the zwp_hdr_surface_v1_interface Wayland object */
+ static void
+ destroy_hdr_surface (struct wl_resource *resource)
+ {
+   wl_list_remove (wl_resource_get_link (resource));
+ }
+
+ static const struct zwp_hdr_surface_v1_interface
+ hdr_surface_implementation = {
+   hdr_surface_set_metadata,
+   hdr_surface_set_eotf,
+   hdr_surface_destroy
+ };
+
```

Mutter-KMS interface

- Metadata handling & property interface with KMS
- Reference implementation:

src/backends/native/meta-kms-impl-device-atomic.c

```c
  if (connector_update->hdr10.has_update)
    {
      struct hdr_output_metadata hdr10_metadata  ={
        .metadata_type = 0, // HDR_METADATA_TYPE1
        .hdmi_metadata_type1 = {
          .eotf = 2, // EOTF_ST_2084_PQ
          .metadata_type = 0, // HDR_METADATA_TYPE1
        },
      };

      meta_topic (META_DEBUG_KMS,
                  "[atomic] Toggling hdr10 to %d on connector %u (%s)",
                  connector_update->hdr10.is_enabled,
                  meta_kms_connector_get_id (connector),
                  meta_kms_impl_device_get_path (impl_device));

      if (!add_connector_property (impl_device,
                                   connector, req,
                                   META_KMS_CONNECTOR_PROP_COLORSPACE,
                                   connector_update->hdr10.is_enabled ? 5 : 0, // FIXME BT2020RGB
                                   error))
        return FALSE;

      /* TODO: fill content hdr metadata for now */
      hdr10_metadata.hdmi_metadata_type1.display_primaries[0].x =
              PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.r.x);
      hdr10_metadata.hdmi_metadata_type1.display_primaries[0].y =
              PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.r.y);

      hdr10_metadata.hdmi_metadata_type1.display_primaries[1].x =
              PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.g.x);
      hdr10_metadata.hdmi_metadata_type1.display_primaries[1].y =
              PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.g.y);

      hdr10_metadata.hdmi_metadata_type1.display_primaries[2].x =
              PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.b.x);
      hdr10_metadata.hdmi_metadata_type1.display_primaries[2].y =
              PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.b.y);

      hdr10_metadata.hdmi_metadata_type1.white_point.x =
              WP_PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.white_point.x);
      hdr10_metadata.hdmi_metadata_type1.white_point.y =
              WP_PRIMARY(connector_update->hdr10.md->metadata.static_metadata.primaries.white_point.y);

      hdr10_metadata.hdmi_metadata_type1.max_display_mastering_luminance =
              connector_update->hdr10.md->metadata.static_metadata.max_luminance;

      hdr10_metadata.hdmi_metadata_type1.min_display_mastering_luminance =
              WP_PRIMARY(connector_update->hdr10.md->metadata.static_metadata.min_luminance);

      hdr10_metadata.hdmi_metadata_type1.max_cll =
              connector_update->hdr10.md->metadata.static_metadata.max_cll;
      hdr10_metadata.hdmi_metadata_type1.max_fall =
              connector_update->hdr10.md->metadata.static_metadata.max_fall;

```

```c
      blob_id = 0;
      if (connector_update->hdr10.is_enabled)
        {
          blob_id = store_new_blob (impl_device,
                                    blob_ids,
                                    &hdr10_metadata,
                                    sizeof hdr10_metadata,
                                    error);

          if (!blob_id)
            return FALSE;
        }

      if (!add_connector_property (impl_device,
                                   connector, req,
                                   META_KMS_CONNECTOR_PROP_HDR_OUTPUT_METADATA,
                                   blob_id,
                                   error))
        return FALSE;
    }
```

Implementation plan

- KMS HDR support – Latest upstream kernel supports HDR
  - It basically defines HDR metadata structures, property to pass content (after blending) metadata from user space compositors to driver.
  - DRM HDR Metadata property (hdr_metadata_property) - This is used to pass HDR metadata information from user space to driver. Driver will use this and create the DRM Infoframe packet and send to HDMI panel.
  - For details - HDR support in DRM
- Userspace/Compositors HDR support - Blending policies and metadata blob creation and passing to driver. [WIP] The implementation of HDR userspace development is planned in following steps:
  - clutter: Attach color state information to actors (!2443) · Merge requests · GNOME / mutter · GitLab
  - cogl/texture: Add higher bit depth offscreen framebuffers support (!2461) · Merge requests · GNOME / mutter · GitLab - Add 10bit (higher bit depth) pixel formats support
  - Draft: [WIP] Attach shaders to cogl pipelines when painting actors with color state (!2643) · Merge requests · GNOME / mutter · GitLab - Attach shaders to clutter actor using cogl pipeline
  - Colorspace conversion using shaders
  - Send HDR metadata using KMS properties
  - Tone mapping
- Wayland-protocols for HDR - Commits · test-hdr · Naveen Kumar / wayland-protocols · GitLab
- Wayland HDR client application - Commits · test-hdr · Naveen Kumar / weston · GitLab

HDR POC Implementation

A POC to implement HDR on Gnome Mutter is available. Details below

- This implements a full screen direct scanout without tone mapping or color conversion.
- It creates an E2E HDR pipeline and exercises DRM KMS interfaces
- It implements a reference wayland-protocols for handling HDR metadata across client & mutter
- It also parses EDID to get the sink's HDR capability
- **Steps To Test HDR Fullscreen Direct Scanout**
  - Compile Weston for HDR video client app Weston HDR video client app
  - Compile Wayland-protocols for HDR, CM Wayland-protocols for HDR, CM
  - Compile Mutter with HDR metadata passthrough support: this MR !2356
  - Test HDR video playback with following command line:
  - $ ./weston-simple-hdr-video-gbm -i <path_video_file> -f 1 -w 3840 -h 2160 -p P010

Future of HDR on Gnome Mutter

- HDR10+ (Dynamic HDR)
- Multiplane overlay support can be enabled
- Plane level color properties are proposed for compositors to display operations per surface. So the composition can directly happen on display hardware
- YUV support

References

- Mutter
  - Basic support for presenting HDR content from Wayland clients (#2134) · Issues · GNOME / mutter · GitLab
  - Draft: HDR support in Mutter (!2356) · Merge requests · GNOME / mutter · GitLab
- Wayland-protocols for HDR reference only - Commits · test-hdr · Naveen Kumar / wayland-protocols · GitLab
- HDR client application - Commits · test-hdr · Naveen Kumar / weston · GitLab
- KMS – Latest upstream kernel supports HDR
- http://files.spectracal.com/Documents/White%20Papers/HDR_Demystified.pdf

Thank You