Windows Video Overlays

This Document is Public

For public docs, make them editable/commentable to all@chromium.org (and uncheck "notify" when doing so).

Status: Draft
Authors: jbauman@chromium.org
Last Updated: 2017-03-31

Objective

We're hoping to reduce power consumption in Chrome when playing video by using overlays.

Background

Currently for hardware-decoded video Chrome uses OpenGL commands to take the NV12 texture and put it on the backbuffer (in RGBA). These commands are translated to D3D11 in ANGLE. The texture sampling and drawing uses memory bandwidth, which corresponds to power. Approximately 10% of the power used when playing a fullscreen 1080p60 video goes into two copies - one from the NV12 texture to Chrome's RGBA backbuffer, and another from Chrome's RGBA backbuffer to the monitor's RGBA backbuffer.

Microsoft Edge and Windows 10 solve this problem by putting the video in an overlay. After Windows 10 Anniversary Update, Chrome's entire backbuffer can sometimes (depending on presentation history and window layout) be put in an overlay, which gets rid of a blit in DWM but forces the display to scan out from another buffer, which can hurt performance.

Overview

We're planning on using the overlay hardware to display the video. We can give the YUV texture to DirectComposition, and that can cause DWM to use an overlay plane to display it, if there are enough available overlay planes. If there aren't (Kaby Lake systems are limited to 2 overlay planes and 1 main plane) DWM will use the GPU to combine multiple textures into one backbuffer.

The browser compositor will analyze the entire tree of quads it receives from the renderer describing the scene. It'll then identify which quads are overlay candidates - for example they should be axis-aligned, and are disallowed if they're in a renderpass (e.g. because they're modified by CSS filters or going to be read back from). It'll then split them out for rendering. We'll initially use a *single-on-top* approach where only one overlay may be used at a time and must have no other content (e.g. controls or ads) above it. Eventually we may use an *underlay* approach, where the video content must be opaque and will be placed under the contents of the rest of the page. Everything below where the video would be would be will be cleared out and made transparent, so the page can render correctly.

Information on how to allocate overlay planes is transferred to the GPU process, which will create the DirectComposition tree and put the videos in it.

Detailed Design

Decoded video frames are placed in a new GLImageDXGI, which keeps track of the D3D11 texture, NV12 data, and color space. The GPU process will mark that the textures can be put in an overlay, and the renderer will transmit this information to the browser.

The DirectCompositionSurfaceWin [DCSWin] is a new type of GLSurface in the GPU process that manages the primary backbuffer. When overlays are enabled the contents of the page outside of an overlay will be put into an IDCompositionSurface. Currently we use an IDXGISwapChain instead for the rest of the contents, but the issue with that is that the present of the SwapChain isn't synchronized with DirectComposition visual updates, so scrolling or transitions would be weird. IDCompositionSurfaces are designed to allow that to work properly. However IDCompositionSurfaces can't be put into overlays, so in cases where there are now video overlays the content would be put into an IDXGISwapchain instead, similar to now. There must be a delay before switching from the IDCompositionSurface to the IDXGISwapChain to prevent a black flicker due to the lack of synchronization. A glEnableDCLayers command-buffer command will be used to switch between the modes.

IDCompositionSurfaces have a number of restrictions - they can't be read as a texture in a shader, every frame their contents are invalid until they're drawn into, the area to be drawn into must be specified before drawing and completely updated, and all draws to them must be offset by a certain amount. The DCSWin must have SetDrawRectangle called on it before drawing to it. BeginDraw will be called on the Surface, and the resulting texture will be passed to ANGLE using *EGL_ANGLE_d3d_texture_client_buffer.* Whenever the default framebuffer is bound, the current draw offset must be retrieved from the GLSurface and applied to the glScissor and glViewport bounds, including whenever those are changed.

For each layer, the overlay data will be sent through the command buffer as a DCRendererLayerParams (similar to a CARendererLayerParams).

In the browser a new DCLayerOverlayProcessor is responsible for determining what quads are eligible to be in overlays and will generate the DCRendererLayerParams. Initially it'll only allow single-on-top overlays with a simple transform.

In the GPU process side, the DCSWin owns a DCLayerTree that will aggregate a list of the GLImages and their positions, including those for the main backbuffer. It can create a new DirectComposition layer tree using IDCompositionDevice::CreateTargetForHwnd, and on swap will create a DirectComposition visual tree referencing all the surfaces.

Currently Intel on Windows doesn't support using NV12 textures as overlays. We'll need to create a YUY2 swapchain and use the D3D11 video processor in this case to convert the textures format - this is what Edge does currently. We may also want to detect when the overlay is actually composited by DWM and switch it to use BGRA, as that can reduce the number of unnecessary DWM conversions. We can even send feedback to the overlay candidate validator to get rid of the overlay in that case, though it'd be hard to determine when to bring it back.

If NV12 overlays are supported we can use one

CreateDecodeSwapChainForCompositionSurfaceHandle to create a decode swapchain on top of the existing NV12 textures. As new surfaces are presented the same swapchain will stay attached to the DirectComposition visual tree, but it will flip between them.

In cases where the texture isn't promoted to an overlay plane it will use the same path as before - the GLImage is bound to a texture and a draw will happen from it.

We'll decide whether to use the new codepath based on whether windows reports that any output supports overlays, and also based on a finch experiment. This will be reported to about:gpu and through UMA histograms. Eventually we'll want to get rid of our ChildWindowSurfaceWin once the new path is stable.

In the future we'll want more flexibility in how we divide elements into layers, so it'll be possible to have multiple DirectCompositionSurfaceWins each owned by a GpuMemoryBufferDirectComposition. Each one can be in one layer tree. A new GL command will be added to temporarily make a different GLSurface the main surface for a command buffer, so it can be rendered into.

Color spaces

A colorspace will be specified on the video texture using a new glSetColorSpaceForScanoutCHROMIUM gl command. On swap, if the swapchain is BGRA8 the video processor will convert from the video colorspace to SRGB. If the swapchain is YUY2 then it'll attempt to keep the same color-space and the display controller will convert color spaces when displaying on-screen.

The video processor and display controller have a limited set of supported color spaces, so the closest possible source and destination color spaces should be used. If the closest available conversion is too different from the ideal conversion, then the use of overlays should be disabled for the video.

HDR

Initially, HDR video will not be put into overlays. In the future, the HDR video will be put into P010 textures, and those will be processed into some (unknown as-yet) swapchain format to be displayed onscreen. If complex color-space conversion is needed, then the texture should fall-back to rendering using GL, as the D3D11 video processor only has a limited set of supported color-space conversions.

Project Information

- jbauman: engineering
- Johnpallett: PM
- *https://bugs.chromium.org/p/chromium/issues/detail?id=678800*
- *Code Location:*
    - gpu/
    - cc/output/
    - content/browser/compositor/
    - ui/gfx

Risks and alternatives

There's no way to meaningfully lower the amount of GPU power we use on hardware decoded video except to take advantage of this hardware.

One possible problem is that this may not save power. In particular, if we need to convert to YUY2 then we'll only save memory bandwidth for a small number of reasons

1) The backbuffer for the video can be the same size as the video instead of the on screen size. This matters the most for high-dpi, because videos are often 1280x720 or 1920x1080 and can be scaled up to 2560x1600 or 4k for display.
2) YUY2 is half the size of RGBA per pixel.
3) For VP9 we can cut out an unnecessary decoder NV12->NV12 texture blit that's always being done.

In fullscreen, if the overlay is fullscreen (like an entire chrome window) then anything under it can be removed. That doesn't help if only individual elements are fullscreen. Any additional overlay causes power usage in scanout.
We'll need to measure power in both circumstances to ensure we're actually helping.

Also it's possible an overlay plane won't be allocated, due to running out of all slots or the window configuration. The documentation suggests querying if an overlay plane was allocated and if not fall back to the non-overlay method. Periodically it could try using overlays again to see if they start working.

Some hardware may just be broken with overlay planes and could display them incorrectly or crash.

We need to ensure the behavior is consistent between having overlays enabled and disabled. Color management should be as similar as possible, video size should be close, and flickers due to bad synchronization shouldn't happen. This is can be particularly problematic when showing the video controls, which must use the old mode if underlays are disabled.

Content on the page may make overlays impossible to use - e.g. ads covering up the video

or other minor issues with the controls or areas outside the video.

Core Principle Considerations

Speed (and System Health)

Ideally we'll be running less code to recomposite a video frame. The only thing that needs to be done is process the video and call Present().

Stability

We have to worry about driver stability when using this new mechanism. We can use the GPU blacklist mechanism to avoid using overlays if they seem unstable.

Security

The GPU process needs to be secure against the client doing unexpected things with surfaces, or reading back things that it shouldn't be able to see, but this is all driven by the browser process, which makes some of the interactions simpler. It shouldn't be much different from what we do now in the GPU process.

Simplicity

This makes Chrome a bit more complicated, as we need to support a new path for rendering to the screen.

Privacy Considerations

None.

Testing Plan

Hardware support requires a windows 10 system with a Kaby Lake or Sky Lake GPU, which we have several of. We'll be able to test some performance and smoothness in the av analysis lab, though the limitation that overlays may not work if HDMI is connected will make it hard to thoroughly test smoothness or color management.

For the waterfall, we're adding additional pixel tests that use video. A flag is added to chrome to use the new direct composition layer path even when overlays aren't supported.

That should test much of the same code. For this to work, the screenshot code must be modified to read the real window contents rather than a CopyOutputRequest from Chrome's compositor.

The GPU try bots and waterfall bots should be upgraded to Win10 Anniversary update. Win 7 bots will be left on the fyi waterfall.

Unit tests and component pixel tests for the overlay candidate validator and DirectCompositionSurfaceWin.

We'll also have manual testing of power consumption and overall usability on a number of sites, including youtube and vimeo.

Work Estimates

- Add new GLImage type for IDCompositionSurface
- Add command buffer support for viewport offset and binding GLImage as texture
- Add GLImage support for picture buffers from DXVAVideoDecodeAccelerator
    - Also on-demand NV12->YUY2 conversion, with caching.
- Make compositor allocate GLImages for backbuffer
- Make/debug overlay candidate validator
- Add code for making a visual tree.
- Add code to detect when this should be used.