Adoption of ANGLE in WPE/WebKitGTK

Eleni Maria Stea <estea@igalia.com>

WebKit Codecamp

Outline

Outline

A few words about ANGLE

ANGLE is an EGL/GLESv2 implementation

EGL/GLESv2:

- GLESv2 is used to render graphics using the GPU (graphics API).
- EGL is used to create a GLESv2 context.

ANGLE EGL/GLESv2:

- GLESv2 is implemented **on top of other APIs** (OpenGL, Vulkan, GLESv2). EGL too (EGL, GLX, gbm, headless display).
- Users can select the most convenient backend using some **ANGLE EGL extensions** (`eglext_angle.h`) that provide extra attributes to some standard EGL API stucts.
- Primary **purpose** of ANGLE is to provide EGL/GLESv2 to systems lacking it.

Using ANGLE in WebKit

Using ANGLE (EGL/GLESv2 backend) in WebGL2

Reasons:

- better performance in some cases (Žan Doberšek)
- it'll be mostly an optimized wrapper around the native driver (libGLES* is available on Linux desktop!)

Problem:

- **ANGLE** renders on a **GLESv2 texture** created by ANGLE context
- WebKit **graphics pipeline** components use **OpenGL textures** that are composited by the WebKit compositor
- We are currently ***copying*** the ANGLE texture data to an OpenGL texture to assemble the final image and this is ***slow***!

We need to replace this copy with something better!

Experiments on Linux

1. **Similar to WebKit: I've used ANGLE and EGL in the same program (created 2 X11 windows, and rendered 2 images from 2 different contexts by 2 different drivers on them.**

   (In several cases ANGLE behaves different from EGL: e.g. when `eglMakeCurrent` doesn't call `eglMakeCurrent`!!).

2. **Shared context: Filled a shared texture with ANGLE, displayed it with the native driver. (REJECTED)**

   (required modifications in ANGLE OpenGL driver, writing an ANGLE extension, forcing the EGL/OpenGL backend, and it wouldn't work with multiple processes)

3. **DMA-buffers: Filled two textures from two drivers simultaneously by using a shared Linux kernel dma-buf buffer. (ADOPTED)**

   (drivers should support some EGL/GL extensions: both mesa and ANGLE support them)

4. **Investigated the multiple processes case. (FUTURE)**

   (we need some sort of IPC to exchange the dma-buf FD)

Outline

Step 1: Setting up to debug ANGLE with GDB

I've ran my experiments using test programs and ANGLE,
and I had to modify the default set up to step into ANGLE calls with GDB:

Things that need attention when building ANGLE for debugging:

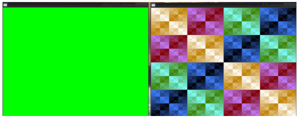
- Default **gn** configuration redirects debugging symbols into separate files where GDB can't find them.
- **GDB** doesn't know where to find the ANGLE installation directories.
- **dwarf5** is not fully implemented on GDB, and so it's impossible to step without errors when it's enabled.
- **Debugging symbols** aren't enabled by default.

- **Blog post** on how to set up ANGLE and GDB for debugging: `https://eleni.mutantstargoat.com/hikiko/debug-angle/`
- **Gist** with the GN args I've used: `https://gistof.com/gnargs`

Step 2: Contexts from two drivers in the same program 1/2

First Test

I've first written a program where both ANGLE and the native system driver render images on X11 windows:



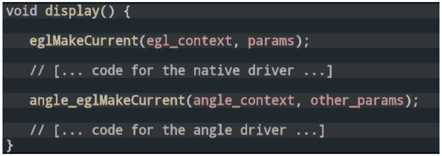
Some parts that needed attention:

- Set ANGLE **library paths** in Makefile and link with native system EGL.
- **Dynamically open** ANGLE EGL and load its functions prefixed with `angle_` to distinguish them from native EGL ones.
- Do the same for GLESv2.
- **Invalidate the ANGLE context** at every display call.
  (*When ANGLE is not the only implementation available* `MakeCurrent` *is not working as expected!!!*)

Step 2: Contexts from two drivers in the same program 2/2

Why invalidate the ANGLE context before MakeCurrent?

Context is cached in ANGLE!



```
void display() {
    eglMakeCurrent(egl_context, params);
    // [... code for the native driver ...]
    angle_eglMakeCurrent(angle_context, other_params);
    // [... code for the angle driver ...]
}
```

Read more about invalidation and dynamic loading:

`Blog post:` Sharing texture data between ANGLE and the native system driver:
`https://eleni.mutantstargoat.com/hikiko/angle-dma/`

- Step 1: Using both libraries in the same program.
- About ANGLE MakeCurrent.

`Code:` `https://github.com/hikiko/shctx/tree/wip/system_egl_dynamic_angle`

Outline

Shared Context

A texture can be accessed by multiple OpenGL or GLESv2 contexts when these contexts are *shared*:

```
new_ctx = eglCreateContext(new_ctx_dpy, new_ctx_config, shared_ctx, new_ctx_attrib);
```

Each texture created by the shared_ctx can be bound and used by the new_ctx while it stays in the GPU!

```
glBindTexture(GL_TEXTURE_2D, gl_shared_tex);
/* ... gl operations ... */

angle_glBindTexture(GL_TEXTURE_2D, gl_shared_tex);
/* ... angle_gl operations ... */
```

Shared context restrictions:

- Contexts must be created by the **same API** (both OpenGL, or both GLESv2, same driver).
- Contexts must be created by the **same process**.(**=> not suitable for every project!**)

Could we use shared context in WebKit?

Could we use shared context in WebKit?

Short answer: Currently yes (with some workaround) but not in the future.

What would we need to change in WebKit for shared context to work?

- Force the **ANGLE EGL/OpenGL backend in WebGL2** to match the main Graphics Pipeline API/driver (both contexts should be OpenGL or GLESv2 and from the same driver!)
- We'd need an **ANGLE extension** to allow passing native shared context to ANGLE's eglCreateContext instead of ANGLE shared context.
- We should never split WebGL2 and Graphics Pipeline in different processes. Shared contexts should be created by the same process! (**Reason we've rejected this method**).

NEW ANGLE extension:
`EGL_ANGLE_native_shared_context`

NEW `EGLAttrib` attribute can be passed to `eglCreateContext`:
`EGL_NATIVE_SHARED_CONTEXT_ANGLE`

This attribute indicates that the shared context in `eglCreateContext` is **not ANGLE** and should be used as native in the internal implementation.

EGL_ANGLE_native_shared_context

This is how we create shared OpenGL context using EGL (system or ANGLE):

```
ctx_A = eglCreateContext(..., 0, atts_A);

ctx_B = eglCreateContext(..., ctx_A, atts_B);
```

This is how we'd create shared OpenGL context between native EGL and ANGLE EGL using `EGL_ANGLE_native_shared_context`:

```
native_ctx = eglCreateContext(...,
                              0, atts_native)

angle_ctx = angle_eglCreateContext(...,
                                   native_ctx, atts_angle)

                { ... /* ANGLE context attributes */,
                 EGL_NATIVE_SHARED_CONTEXT_ANGLE, EGL_TRUE,
                 EGL_NULL
                };
```

Some work on the shared context approach

What happens inside ANGLE (with EGL_NATIVE_SHARED_CONTEXT_ANGLE):

```
angle_eglCreateContext(...,
                       EGL_context shared,
                       EGLattrib atts) {
  /* ... */
  if (atts:EGL_NATIVE_SHARED_CONTEXT_ANGLE is EGL_TRUE) {
     /* Use shared as native context and pass it to the
        system driver eglCreateContext call */
     eglCreateContext(..., shared, ...);
  } else {
     /* handle shared as ANGLE shared context */
     [... ANGLE code ...]
  }
}
```

Experimental programs (to test the extension and familiarize with the shared context concept):

- A test program that uses EGL ctxA to create a texture and EGL ctxB to display it on an X11 window.
- The same program written in ANGLE (there were differences!)
- A test program that uses EGL ctxA to create a texture and ANGLE EGL ctxB to display it on an X11 window.
- Variations of the above: `https://github.com/hikiko/shctx/branches`

Abandoned!

Why not shared context?

- Shared context would be a nice and clean approach if we wanted to use it with our current WebKit code.
- But there is a plan to use **separate processes** for WebGL2 and the main graphics pipeline!! (Žan Doberšek)

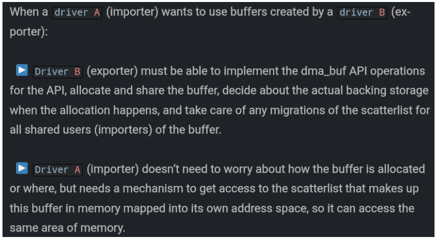
But there was a *better* approach!

Outline

DMA buffers for the win!

Fortunately we are on Linux!

There is a Linux kernel framework that can be used for content sharing across different Linux drivers!


When a `driver A` (importer) wants to use buffers created by a `driver B` (exporter):

- `Driver B` (exporter) must be able to implement the dma_buf API operations for the API, allocate and share the buffer, decide about the actual backing storage when the allocation happens, and take care of any migrations of the scatterlist for all shared users (importers) of the buffer.

- `Driver A` (importer) doesn't need to worry about how the buffer is allocated or where, but needs a mechanism to get access to the scatterlist that makes up this buffer in memory mapped into its own address space, so it can access the same area of memory.

A few interesting things about content sharing with dma buffers:

- There are EGL and OpenGL/GLESv2 extensions to make it easier!
- It's a driver independent method!
- It works with multiple processes!
- As long as ANGLE can expose the required extensions to import a dma_buf file descriptor this method is also "ANGLE backend independent"!
- DMA buffers are a Linux-only thing... but we won't need to support other systems! (Žan Doberšek)

Extensions and new approach

EGL and GL extensions to share content across drivers using dma_buf buffers:

- EGL_MESA_image_dma_buf_export:
  This extension allows creating one or multiple Linux dma_buf file descriptors from the EGLImage that corresponds to a texture.
- EGL_EXT_image_dma_buf_import:
  This extension allows creating an EGLImage (that will be used to create a texture) from one or multiple Linux dma_buf file descriptors.
- EGL_EXT_image_dma_buf_import_modifiers:
  This extension builds on EGL_EXT_image_dma_buf_import, in order to support format modifiers used for tiling, compression, and additional non-linear modes.
- OES_EGL_image_external:
  This extension provides a mechanism for creating EGLImage texture targets from EGLImages.

Example Programs:

- Associating two textures with the contents of the same buffer without copy taking place.
  Blog post: `https://eleni.mutantstargoat.com/hikiko/egl-dma-1/`
  Code: `https://gistof.com/dma-egl-version`
- Sharing texture data between ANGLE and the native system driver using DMA buffers and EGL.
  Blog post: `https://eleni.mutantstargoat.com/hikiko/angle-dma/`
  Code: `https://gistof.com/dmaangleeglversion`

Example:

Snippet from the exporter:

Exporting a dma buffer from a texture `texA`:

```
EGLImage imgA = eglCreateImage(ctxA.dpy, ctxA.ctx, EGL_GL_TEXTURE_2D, (EGLClientBuffer)(uint64_t)texA, 0);
assert(imgA != EGL_NO_IMAGE);

PFNEGLEXPORTDMABUFIMAGEQUERYMESAPROC eglExportDMABUFImageQueryMESA =
    (PFNEGLEXPORTDMABUFIMAGEQUERYMESAPROC)eglGetProcAddress("eglExportDMABUFImageQueryMESA");
PFNEGLEXPORTDMABUFIMAGEMESAPROC eglExportDMABUFImageMESA =
    (PFNEGLEXPORTDMABUFIMAGEMESAPROC)eglGetProcAddress("eglExportDMABUFImageMESA");

EGLBoolean ret;
ret = eglExportDMABUFImageQueryMESA(ctxA.dpy,
                                    imgA,
                                    &gl_dma_info.fourcc,
                                    &gl_dma_info.num_planes,
                                    &gl_dma_info.modifiers);
if (!ret) {
    fprintf(stderr, "eglExportDMABUFImageQueryMESA failed.\n");
    return false;
}
ret = eglExportDMABUFImageMESA(ctxA.dpy,
                               imgA,
                               &dmabuf_fd,
                               &gl_dma_info.stride,
                               &gl_dma_info.offset);
if (!ret) {
    fprintf(stderr, "eglExportDMABUFImageMESA failed.\n");
    return false;
```

```
struct tex_storage_info {
    EGLint fourcc;
    EGLint num_planes;
    EGLuint64KHR modifiers;
    EGLint offset;
    EGLint stride;
};
```

Example continued

Snippets from the importer:

Creating an EGLImage from the dma buffer using the exported fd and the exported modifiers:

```
EGLAttrib atts[] = {
    // W, H used in TexImage2D above!
    EGL_WIDTH, 256,
    EGL_HEIGHT, 256,
    EGL_LINUX_DRM_FOURCC_EXT, gl_dma_info.fourcc,
    EGL_DMA_BUF_PLANE0_FD_EXT, dmabuf_fd,
    EGL_DMA_BUF_PLANE0_OFFSET_EXT, gl_dma_info.offset,
    EGL_DMA_BUF_PLANE0_PITCH_EXT, gl_dma_info.stride,
    EGL_NONE,
};
EGLImageKHR imgB = eglCreateImage(ctxB.dpy, EGL_NO_CONTEXT, EGL_LINUX_DMA_BUF_EXT, (EGLClientBuffer)(uint64_t)0, atts);
assert(imgB != EGL_NO_IMAGE);
```

Creating a texture using that external EGLImage:

```
PFNGLEGLIMAGETARGETTEXTURE2DOESPROC glEGLImageTargetTexture2DOES =
    (PFNGLEGLIMAGETARGETTEXTURE2DOESPROC)eglGetProcAddress("glEGLImageTargetTexture2DOES");
assert(glEGLImageTargetTexture2DOES);

glGenTextures(1, &texB);
glBindTexture(GL_TEXTURE_2D, texB);
glEGLImageTargetTexture2DOES(GL_TEXTURE_2D, imgB);
```

Final test program (WORKS!)

An exporter-importer that uses ANGLE and native EGL

- First context is EGL/OpenGL like the one in main graphics pipeline.
- Second is ANGLE with EGL/GLESv2 backend like the one in WebGL2.
- EGL/OpenGL context creates an empty texture and exports the dma_buf fd and all other information about the buffer.
- ANGLE context creates another empty texture using the same dma_buf and the import mechanism.
- ANGLE context fills the emty ANGLE texture.
- EGL/OpenGL context displays the previously empty OpenGL/EGL texture.
- EGL/OpenGL texture contains what ANGLE texture had.
- **We shared the ANGLE data without copying them!**

Check the blog posts for more details!

Outline

What if WebGL and Graphics pipeline were separate processes?

There is a plan to split the main graphics pipeline and the WebGL2 pipeline in two processes. (Žan Doberšek)

Can we still use shared DMA buffers?

- **YES!** DMA buffers can be shared across multiple processes.
- But we need some sort of ***interprocess communication*** to exchange the file descriptor.
- This is a client-server ***example*** that uses unix sockets to pass the dma-buf FD from one process to the other: `https://gitlab.com/blaztinn/dma-buf-texture-sharing`.

Outline

WIP WebKit:

Ongoing work on WebKit

- The extensions to import dma-buf buffers in ANGLE are implemented and exposed to the user => in WebGL we can easily import the main pipeline DMA buf when we create the render target!
- The extension to export dma-buf buffers from EGL is supported on mesa => we could run a check before creating the shared dma buffer and use either use it or fallback to something else (libgbm? copying?)

DONE/WIP/TODO

- Use the right CMake options (one can't simply enable `USE_ANGLE_WEBGL`!): **FIXED**
- There were compile errors when ANGLE was used: **FIXED/Pending to send the patches**
- Link errors when ANGLE is used: **WIP/Partially FIXED**
- Copy replacement: **WIP/TODO**

Outline

Links I

- *Blog posts with more details.* https://eleni.mutantstargoat.com/hikiko/tag/angle.
- *Example code (hikiko@github).* https://github.com/hikiko/shctx.
- *EXT_image_dma_buf_import.* https://www.khronos.org/registry/EGL/extensions/EXT/EGL_EXT_image_dma_buf_import.txt.
- *EXT_image_dma_buf_import_modifiers.* https://www.khronos.org/registry/EGL/extensions/EXT/EGL_EXT_image_dma_buf_import_modifiers.txt.
- *MESA_image_dma_buf_export.* https://www.khronos.org/registry/EGL/extensions/MESA/EGL_MESA_image_dma_buf_export.txt.
- *GL_OES_EGL_image_external.* https://www.khronos.org/registry/OpenGL/extensions/OES/OES_EGL_image_external.txt.

Links II

- *Buffer Sharing and Synchronization.* https://01.org/linuxgraphics/gfx-docs/drm/driver-api/dma-buf.html.
- *DRM fourcc.h (about modifiers).* https://raw.githubusercontent.com/torvalds/linux/master/include/uapi/drm/drm_fourcc.h.
- *Blaztinn's client server example.* https://gitlab.com/blaztinn/dma-buf-texture-sharing.

Outline

Thank you!