COLLABORA

Zink: OpenGL on Vulkan

Simplifying the future of the graphics stack

FOSDEM 19

Erik Faye-Lund
erik.faye-lund@collabora.com
02/02/2019

Existing solutions

- Think Silicon's GLOVE
  - Only implements OpenGL ES 2.0
  - CLA requires copyright assignment
- Google's ANGLE
  - Only implements OpenGL ES (2.0 and 3.x)
- VKGL
  - Only targets OpenGL 3 Core Profile
  - Doesn't really work yet…
- Nothing seems to fit!
  - But let's steal all the implementation details we can!

Why OpenGL on Vulkan

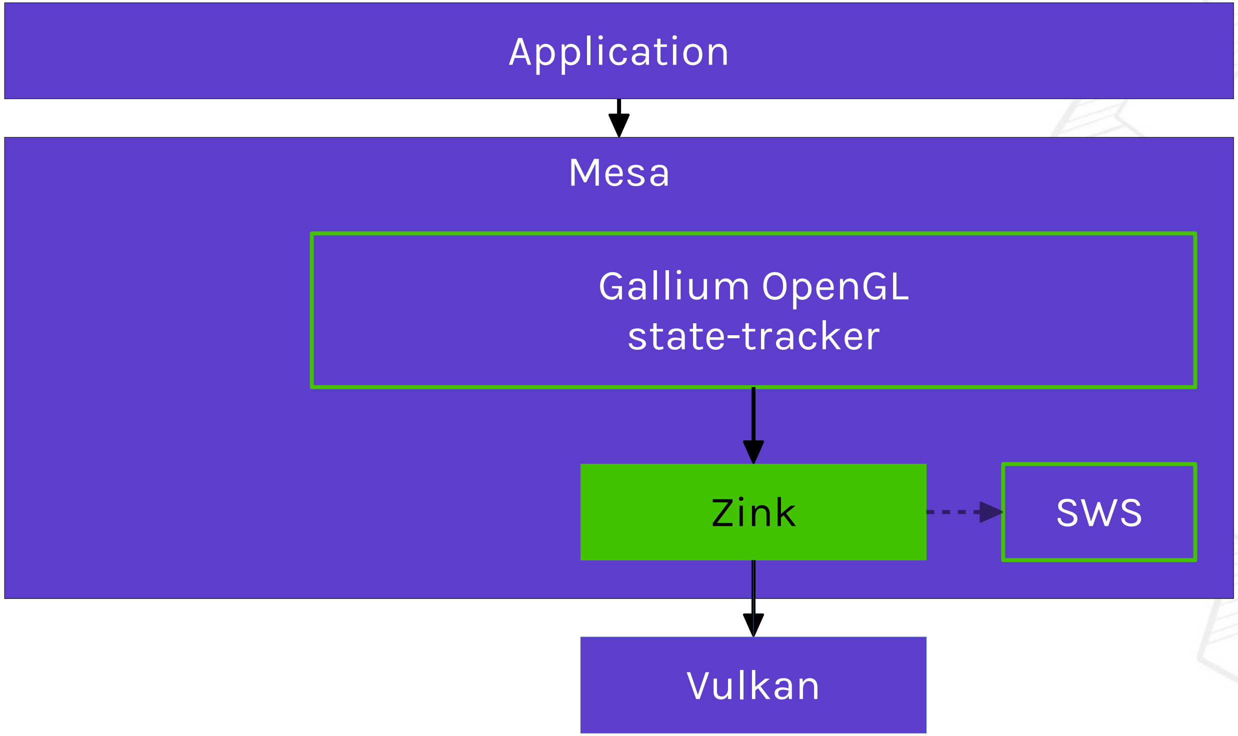
- OpenGL is a requirement for desktop
  - Some modern use-cases are outside of what OpenGL was designed for
    - GPU virtualization is the use-case that motivated my work
- Vulkan is here to stay
  - Likely to be the leading “high-end” API going forward
- It’s better for the community if we can focus on one API
  - Lots of existing software depends on OpenGL, so we need it for compatibility
- Support more use-cases?
  - Support **full** OpenGL applications to mobile?

Solution: Zink!

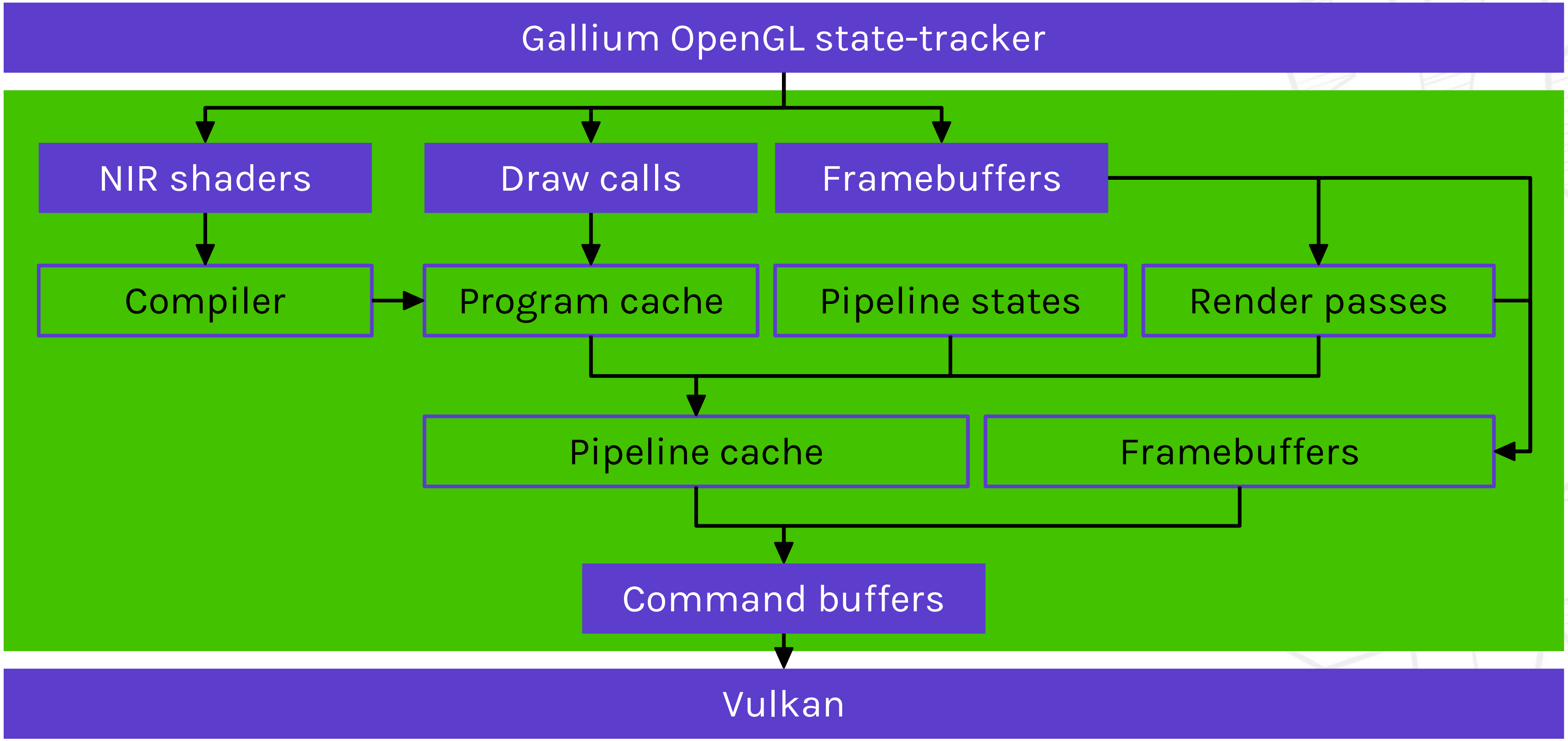
Proposal: Zink

- Translates Mesa's Gallium API calls to Vulkan
- An early out-of-tree "prototype" driver works
  - Supports OpenGL 3.0 on RADV and ANV
    - Not really tested on anything else either…
  - Lots of awesome contributions by Dave Airlie :)
- Happy-go-lucky approach
  - Works much better than I feared
  - I can run a lot of games and demos with usable performance
- Currently undergoing re-engineering phase to fix some early mistakes
  - Next step: clean up and upstream in Mesa

Zink: A Gallium Driver



Zink: How does it work (ish)



NIR → SPIR-V

- Choose NIR as source IR due to the SSA nature
  - Turns out this is a bit harder than I thought
    - More on this later
- Written as a reusable module
  - Can be reused as an in-tree GLSL → SPIR-V compiler?
- Doesn’t generate awesome code yet
  - Desktop Vulkan drivers seems to make up for it
  - Haven’t looked at mobile yet; probably need to do better.

COLLABORA

Difficulties

Control Flow

- No support for control-flow in the NIR → SPIR-V compiler yet
  - Prototype exists, but there's problems.
- Trickier than it sounds, because of some SSA-differences
  - In NIR, jumps can occur from inside basic blocks
    - nir_jump_instr: return, break, continue
    - nir_intrinsic_instr: discard, discard_if
  - In SPIR-V, all of these terminate the basic block
    - This leads to addressing inconsistencies with phi-nodes
  - Probably not **that** hard to solve, but I need to accept to do some lowering first…
    - I may have to give up on reusing the phi-nodes directly

Typeless SSA Values

- NIR SSA-values are untyped, SPIR-V values are typed
  - Currently use uint, and bitcast on every access
    - This creates a lot of needless instructions
  - Jason has been nice enough to code up an ALU-instruction scanning pass that can remove most of the casts
    - Also useful for OpenGL ES 2.0 GPUs
    - Untested so far AFAIK
  - Still awkward for constants
    - Delay constant-emitting to use?
      - This complicates SSA-traversal, but maybe not too bad
    - Extend Jason's scanning pass for this instead?

Shader Resources → Bindings / Descriptor Sets

- Currently just stuffs all UBOs and samplers in one giant descriptor set
  - Assigning binding-numbers based on shader stage and resource type
    - …Because we compile the different shaders independently
  - Approach shamelessly stolen from DXVK
- Vulkan spec suggest high binding numbers **might** give sub-par performance
- Probably better to use one descriptor set per stage
  - Should probably pack bindings
  - One problem: might not have enough descriptor-sets for all stages in the future
    - But should probably just not enable tessellation unless there's enough.
    - Alternatively: split descriptor-set for tess-stages in two and accept potentially lower than ideal performance.

Descriptor Set Management

- Naive approach cause **a lot** of VkDescriptorSet objects
  - Currently allocate a new one from a big pool for every draw
  - If allocation fails: wait for GPU to finish, and reset pool.
    - Causes a hitch when this happens, and gives a validation error.
- We can probably do something more clever here:
  - Reuse descriptor set if nothing changed
  - Several smaller pools rather than one big
    - Protected by fences
  - Keep track of when we need to flush and switch pool instead of waiting for failure

Pipeline Caching

- Pipeline objects are an encapsulation of pretty much all the drawing-state
  - Except for UBO/SSBO/texture-bindings
  - A **few** states can be marked as dynamic, and submitted separately
- Creating VkPipeline objects is relatively slow
  - We need to cache them to avoid re-creating the same objects over and over again.
  - Can generate non-optimized pipelines eagerly, and optimized pipelines on a background thread
    - VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
- This is similar to shader-variant caching for other drivers
  - Except the cache-key is bigger and changes more often?

Image Layouts

- Vulkan needs the client to manage image layout transitions
  - This is done by issuing vkCmdPipelineBarrier
- Minimal approach currently
  - Transitions to VK_IMAGE_LAYOUT_GENERAL early, and leaves it there
    - Not ideal from a performance point of view…
  - Racy: Resources can be shared between contexts
    - Needs some fencing to avoid two command buffers from racing to transitioning an image
- ANGLE is doing a frame-graph to optimize layout-transitions
  - We can consider something like that as well at some point

Uniforms / UBOs

- We currently lower uniforms into a “default” UBO
  - Consider using push-constants when possible instead?
- NIR doesn’t provide declarations of UBOs
  - SPIR-V need need to know size of each UBO
    - Use max-size + robustBufferAccess?
  - Perhaps UBO declarations can be added?
    - TGSI provides this…

Reliance on EXT extensions

- Will VK_EXT_transform_feedback stay forever?
  - My guess: No.
  - Probably already not supported on mobile GPUs.
- Emulation needed at some point?
  - Write to SSBO with vertex-id instead?
    - Only works if there's no geometry/tesselation shaders
  - Emulate GPU pipeline up to the fragment shader using compute?
    - Pass-through shader in the end for "original" draw?
- Similar concerns for other EXT extensions…

Missing features

Polygon Mode

- OpenGL allows different polygon mode for front and back-faces
- Vulkan only allow one mode for both
  - Currently print a warning and use the front-face state for both
  - Emulation?
    - Draw all back-faces, then all front-faces?
      - Not correct, but maybe **better**…
    - Write primitives out to a buffer and use geometry shader to construct triangles out of lines and points?
- Does conformance tests even exist for this?
  - Low priority either way

Texture Border Colors

- OpenGL allows arbitrary texture-border colors
- Vulkan only support three fixed colors:
  - Transparent black
  - Opaque black
  - Opaque white
- Currently hard-coded as transparent black
- Can emulate by injecting shader-code
  - Not as bad as it sounds, but not great either...
- Create Vulkan extension for hardware that supports this?
  - Low priority, unlikely to matter for real applications

Point Size

- Writing gl_PointSize from shaders works as expected, but…
- No automatic forwarding of glPointSize through shaders yet
  - Boring code to write, but should be easy
    - Famous last works, I know…
- Probably needed by some other mobile GPU drivers
  - Lima / Panfrost?

Alpha testing

- **In theory** just a matter of supporting the NIR discard intrinsics, but…
  - Requires control-flow, which we don't support yet
  - NIR and SPIR-V disagree if these count as control-flow or not
    - This throws a wrench into the shader-compiler…

COLLABORA

Current OpenGL Versions

OpenGL 2.1

- This is the “base version” version we can support
- Requires Vulkan 1.0
  - As well as these VkPhysicalDeviceFeatures:
    - logicOp
    - fillModeNonSolid
    - wideLines
    - largePoints
    - alphaToOne
    - shaderClipDistance
  - We don’t actually check for those yet, YMMV ;)

OpenGL 3.0

- Same HW requirements as OpenGL 2.1, plus
  - VK_EXT_transform_feedback
  - VK_EXT_conditional_rendering
- Enabled on RADV and ANV

COLLABORA

Wrapping Up

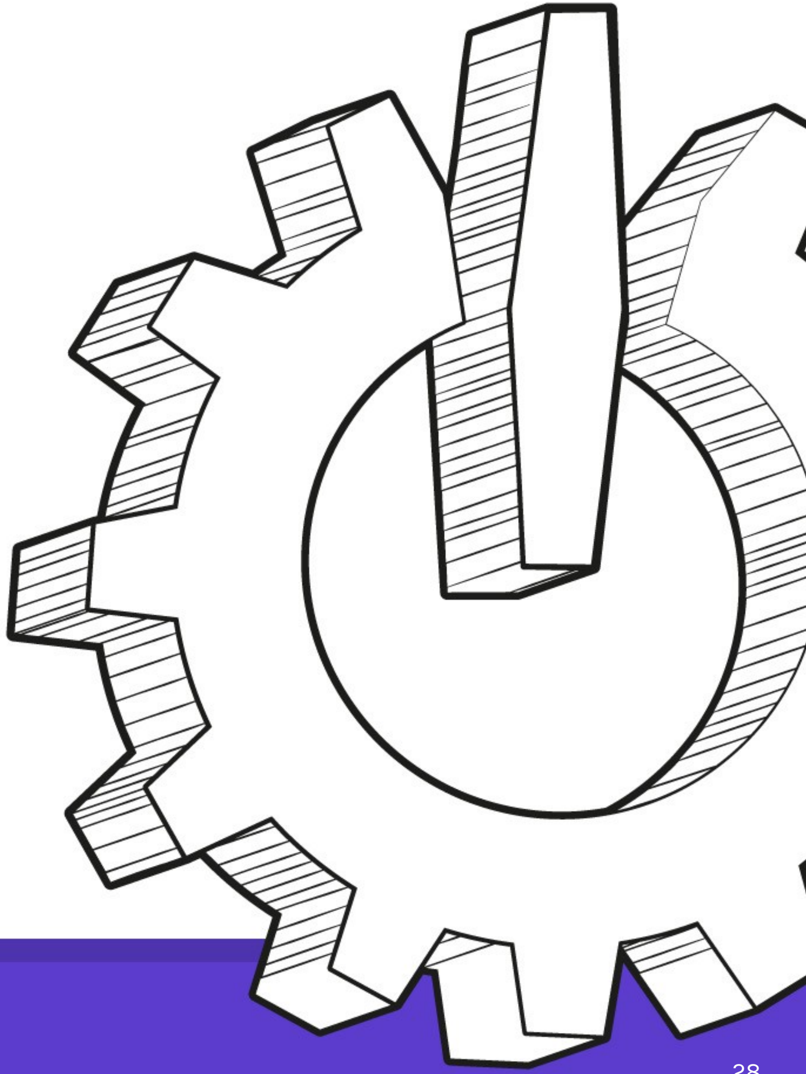
Future!

- Lots more work to be done, most importantly:
  - Making the compiler not suck!
    - Help here would be very much appreciated
  - Fixing rendering-issues in applications
  - Upstreaming in Mesa
    - Feature-set and performance are already usable
  - After that, implementing more modern OpenGL features
- I'm currently the bottle-neck here
  - Sorry about that :(
  - Open to suggestions on how to avoid this!

FOSDEM 19

Zink:
OpenGL on Vulkan

Questions?



Psst...
We're hiring!

COLLABORA

Backup Slides

Window System Integration

Two implementations so far:

- One exposed to Mesa as a software rasterizer
  - Slow, copies data with CPU on present
  - Portable, should work “everywhere”
- One using file descriptor based shared memory
  - Fast, share GPU-side buffers across processes
  - Requires VK_KHR_external_memory_fd extension
  - Currently also requires VK_KHR_maintenance1 to vertically flip rendering
    - not **strictly** speaking necessary, can flip in shader
  - Doesn’t work on the NVIDIA blob, due to lack of DRI2 support…

Why from an ECO-system prespective

- Ease support of legacy GPUs in the future
  - Fewer drivers required to maintain
  - Driver community can focus on making Vulkan drivers
- Ensure OpenGL applications won’t all of a sudden break
- Support **full** OpenGL on non-desktop platforms
  - Lots of code hasn’t been ported to OpenGL ES
  - Can ease application porting
    - Blender on Android some day?
- Possible to extend OpenGL features knowing hardware-specifics
  - As long as the features are exposed to Vulkan

Why from a Virgil 3D prespective

- Virgil 3D currently doesn’t support Vulkan
  - Vulkan support is in progress
- Virgil 3D on OpenGL has some open challenges
  - Doing most of the emulation in virglrenderer
    - Security issues; virglrenderer highly priviledged
      - …relatively easy to crash the host process?
  - Translating TGSI to GLSL and compiling that on the host compiler
    - Lots and lots of string manipulations
    - The future of TGSI doesn’t look too bright at the moment

ALU Swizzles

- In NIR, ALU instructions can have swizzles on their operands
- In SPIR-V ALU-instructions doesn’t have swizzles
  - Currently insert OpCompositeExtract, OpCompositeConstruct, and OpVectorShuffle while traversing operands
    - This code is, uh… awkward
  - Maybe lower away swizzles into dedicated imov instructions first or something?

Scalar vs Vector Types

- In NIR, scalars are one-component vectors
- SPIR-V disallows this
- Needs to “peel” away vec1 to float etc
  - Not really a big deal, but something worth noting

Flat shading

- Flat shading in Gallium is a bit… meh
- The standard approach is using shader-variants, but…
  - I would like to use specialization values for shader-variants if I can get away with it
  - But flat-decorations can't naively be added with a specialization value
  - Perhaps I can add two inputs and use a boolean specialization value to select which one I read from?
- Dave Airlie has written some patches to do this as part of the fixed-function emulation instead!
  - Works great!
  - Hope to land this separately first, some other drivers want it AFAIK.

Future OpenGL Versions

OpenGL 3.1

- Same HW requirements as OpenGL 3.0
- Missing features:
  - Uniform Buffer Objects
  - Primitive Restart
    - OpenGL supports arbitrary index-values for restart
    - Vulkan only supports the max-value
      - VK_INDEX_TYPE_UINT16: 0xFFFF
      - VK_INDEX_TYPE_UINT32: 0xFFFFFFFF
      - Pretty much the same as in OpenGL ES 3
    - Need to rewrite index-buffer and replace the values for correct behavior

OpenGL 3.2 – 3.3

- OpenGL 3.2:
  - Same HW requirements as OpenGL 3.1, plus these VkPhysicalDeviceFeatures:
    - depthClamp
    - geometryShader
    - shaderTessellationAndGeometryPointSize
- OpenGL 3.3:
  - Same HW requirements as OpenGL 3.2, plus:
    - occlusionQueryPrecise
    - VK_EXT_vertex_attribute_divisor
      - Supported on both RADV and ANV

OpenGL 4.0 – 4.6

- More VkPhysicalDeviceFeatures required
  - Too many to list here
- OpenGL 4.4 will require VK_KHR_sampler_mirror_clamp_to_edge
  - **Possible** to emulate in shader if needed?
- Apart from that, it seems mostly like implementation-work