Vulkanised 2025

The 7th Vulkan Developer Conference
Cambridge, UK | February 11-13, 2025

Update on HLSL

HLSL 202X, Clang, Vulkan & Inline SPIR-V

Nathan Gauër, Google

Agenda

01 **HLSL today**
Status & issues

02 **HLSL tomorrow**
Language, compiler & Vulkan support evolutions

03 **Current status**
Our progress so far

04 **Challenges targeting graphical SPIR-V from LLVM-IR**
Opaque pointers, address spaces & structured control flow.

05 **Timeline**
Clang release timeline.

Google

01 HLSL Today

Status & Issues

Issues with HLSL

- HLSL specification was a mix of:
  - hand-waving
  - looking at the compiler output
  - asking somebody at Microsoft if something is legal
- HLSL development was completely opaque:
  - SPIR-V had to deal with new features as they came.
  - Some features were simply badly designed (Yes, that’s you GetAttributeAtVertex)

The lack of proper specification and closed process didn’t help HLSL to grow.

Issues with DirectXShaderCompiler

- DirectXShaderCompiler (DXC) forked from Clang/LLVM 10 years ago!

- Very few upstream fixes were backported.
  - DXC's Clang/LLVM is essentially stuck in 2017.

- The team supporting the SPIR-V backend is small.
  - Developing a new feature takes time, we were the bottleneck.

Issues with the SPIR-V backend

- Emits SPIR-V directly from the AST
  - Takes a very different path from DXIL, leading to behavior divergences.

- Relies heavily on SPIRV-Tools
  - Diagnostic was very limited once in spirv-tools, leading to hard-to-understand validation errors.

- Every new SPIR-V extension requires patching DXC & SPIRV-Tools.
  - We had to develop & maintain those extensions, even niche vendor-specific extensions.

Google

02 HLSL Tomorrow

Language, compiler & Vulkan support evolutions

Open-sourcing the HLSL language

- HLSL specification is now public: https://github.com/microsoft/hlsl-specs/
  - Not only the content, but the process.
  - Anyone can contribute, but it's under the Microsoft CLA.

- Design of the HLSL implementation in Clang is also public: https://github.com/llvm/wg-hlsl

Open-sourcing the HLSL language

- The process is still new, and undergoing transformations.
  - vk::BufferPointer was the first real proposal.
    - The process took ~4 months (8 if we take the initial draft as start).
  - More recently, vk::khr::CooperativeMatrix proposal took 1.5 months to be accepted! 🥳
- Language specification currently under the Microsoft CLA
  - Raises IP questions.
- Specification of existing features is still in progress, but backlog is large!
  - They are specified as they they are built in Clang. Steady progress so far!

HLSL 202X

- FXC, then DXC was the language specification.
- Compiler bugs became features, and we had to deal with them.

- Moving to Clang is a good occasion to start “fresh”.
- HLSL 202X will disallow some existing patterns and irregularities to make HLSL more consistent.

- HLSL 202X will be the transition version from DXC to Clang.
  - This will be the last supported version on DXC.
  - Transitioning to HLSL202X will ease the move to Clang.
  - Saves us from inheriting issues dating back from FXC.
  - Document listing the changes: https://clang.llvm.org/docs/HLSL/ExpectedDifferences.html

Vulkan/SPIR-V support

- We aim to support Vulkan1.2 and later.
- For now, we need 2 extensions:
  - VK_KHR_shader_maximal_reconvergence to have a deterministic reconvergence.
  - VK_KHR_variable_pointers, allowing pointers to pointers (required for references).

- Vulkan extensions will be implemented as headers using inline SPIR-V
  - We want to allow vendors to write/ship their own HLSL headers.
  - This would allow them to move independently from Clang.
  - We will implement initial extensions using inline SPIR-V to verify usability.
  - VK_KHR_cooperative_matrix has already been implemented (https://github.com/microsoft/DirectXShaderCompiler/blob/main/tools/clang/lib/Headers/hlsl/vk/khr/cooperative_matrix.h)

Target profiles

- DXC required to select the ShaderModel version when targeting Vulkan.
  - The former controlled features at the HLSL level (like wave intrinsics).
  - The latter controlled implementation of said features in SPIR-V.
  - There is also the HV flag to control the HLSL version.
- DXC also assumed every Vulkan extension was available by default.
  - You could provide a list of allowed extensions, but you had to list them all.
  - Not convenient.
- Plan to solve this: Vulkan profiles!
  - Clang already dropped the ShaderModel from the Vulkan triple.
  - For extensions, a flag would allow you to select a vulkan profile as baseline, like VP_ANDROID_15_minimums.
  - Extensions missing from that profile would be considered forbidden.
  - We still plan to keep a flag to manually specify an allowed/forbidden extension list.

Google

03 Current status

Clang

- A trivial compute shader with RWBuffers can be compiled to DXIL.
- A trivial compute shader can be compiled to graphical SPIR-V.
  - RWBuffers support is almost ready.
- Other resources & shader types are not supported.

Google

04

Challenges targeting graphical SPIR-V from LLVM-IR

Opaque pointers, address spaces & structured control flow.

LLVM IR control flow

- LLVM IR allows irreducible control flow graphs.
- Graphical SPIR-V requires not only a reducible graph, but also a structured control flow:
    - no arbitrary goto
    - for each divergence, the reconvergence location must be known.
    - reconvergence has strict rules:
        - A single exit per loop, except for X and Y
        - Nested branch can exit one branch at a time, except when Z
        - etc…

We have a custom structurizer in LLVM to transform the control flow into something valid for SPIR-V.

HLSL address spaces vs SPIR-V storage classes

- HLSL local variables are in the same address space as static globals.
- Pointers to one are compatible with pointers to the other.
- LLVM IR default rules are similar.
- SPIR-V has 2 incompatible storage classes for those: Private & Function

All OpVariable are moved from the Function to the Private storage class.

- Brings a hard limit of 65k variables per module (vs 500k for local variables).
- Required because of the library target and `noinline` attribute.

LLVM has opaque pointers, SPIR-V types pointers.

- Since last year, LLVM has no typed pointers.
  - A pointer is just an address, the load result type defines the pointed type.
- Graphical SPIR-V only has typed pointers.

The backend has a “type scavenger”

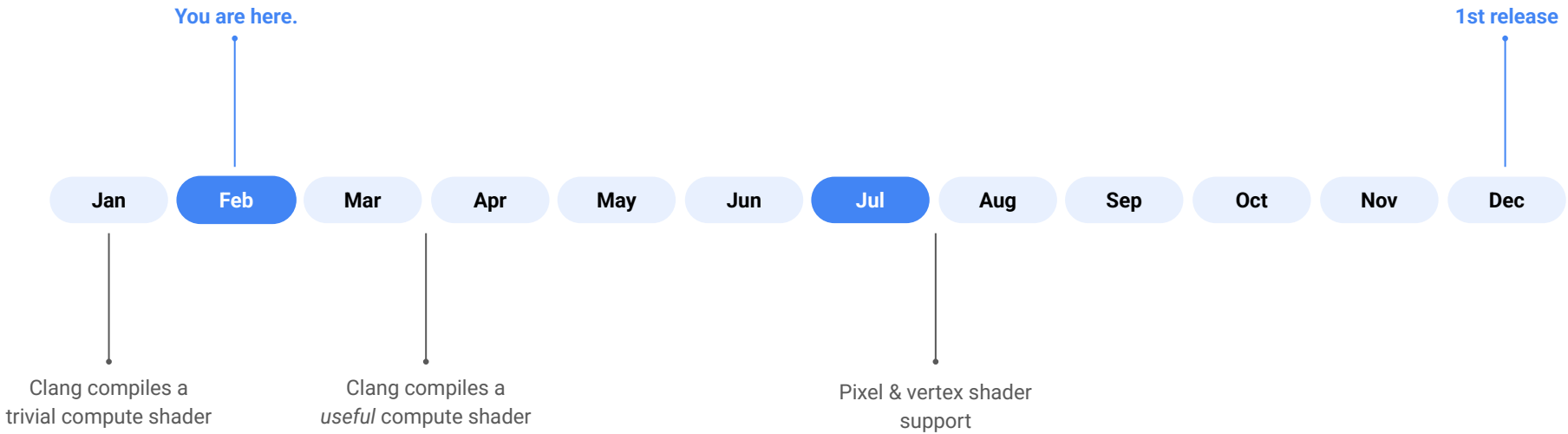
- finds the load, and link each pointer to the loaded type.
- expects the LLVM-IR to avoid using the same pointer to load 2 types.

05

Timeline

Timeline

Microsoft published a roadmap for HLSL to DXIL with Clang: https://github.com/orgs/llvm/projects/4/views/15
We try to follow the same rhythm for HLSL to SPIR-V.



Thank You!

Google