📊 WGSL 2020-03-31

Chair : Dean Jackson
⌨️ **Scribe** : Mehmet Oguz Derin

- *Scribe's Preface: I am new to scribing, and I tend to write using telegraphic sentences. Please let me know or edit where needed.*

Location : Google Meet
Specification : https://webgpu.dev/wgsl
Open Issues : WGSL Issues
Meeting Issues : Marked Issues

To not be anonymous animals, the doc is shared for writing with the google accounts that are invited to the meeting. If you can't edit, let cwallez@google.com or dino@apple.com know. Also, be sure to be in "Edit" mode and not "Suggest" mode.

`break if ("too long");`

- Remove `break if` remains undecided and will be discussed again at the next meeting.
- Discussed implicit storage class on variables. Consensus to have an implicit <function> storage class for a `var` inside a function. Discussion around have an implicit <private> for `var` at module scope if not specified. General agreement during the meeting, but counter points raised by Dzmitry on the bug.
- FAQ discussion was delayed until a future meeting; Goals PR delayed on FAQ discussion.
- Agreement that implementations would determine the interfaces based on usage in the code. It would _not_ be explicitly specified in the WGSL file.

Tentative Agenda (click here to jump to the discussion section)

- 🔗 **Issue #562** - Add FAQ for direction on shader language
  - (click here to jump to section)
- 🔗 **Issue #588** - Update Goals section of spec
  - (click here to jump to section)
- 🔗 **Issue #643** - Remove `break if` in favor of `if() {break; }`
  - (click here to jump to section)
- ~~🔗 **Issue #642** - Add `break N` instead of `continue` and `continuing`~~
  - ~~(click here to jump to section)~~
- 🔗 **Issue #654** - var declaration without storage class: spell out rules, infer Function storage class inside a function
  - (click here to jump to section)
- 🔗 **Issue #644** - Interface matching rules
  - (click here to jump to section)

- 🔗 **Mail #public-gpu/2020Mar/0024** - Test for round-trippability
  - (click here to jump to section)
- 🔗 **Issue #640** - The set of functions which have to be imported seems arbitrary
  - (click here to jump to section)

📋 Attendance

WIP, the list of all the people invited to the meeting. **In bold, the people that have been seen in the meeting:**

- Apple
  - **Dean Jackson**
  - Fil Pizlo
  - **Myles C. Maxfield**
  - **Robin Morisset**
- Google
  - **Dan Sinclair**
  - **David Neto**
  - **Kai Ninomiya**
  - **Ryan Harrison**
  - Sarah Mashayekhi
- Intel
  - Yunchao He
  - Narifumi Iwamoto
- Microsoft
  - **Damyan Pepper**
  - **Rafael Cintron**
  - **Greg Roth**
  - Michael Dougherty
  - **Tex Riddell**
- Mozilla
  - **Dzmitry Malyshau**
  - **Jeff Gilbert**
- Joshua Groves
- **Mehmet Oguz Derin**
- **Timo de Kort**
- Lukasz Pasek
- Tyler Larson
- Lukasz Pasek
- **Pelle Johnsen**
- **Matijs Toonen**

📑 Prior Cheat Sheet [(click here to jump to the discussion section)]

Aggressively summarizes (strictly) only the updates from 2020 03 25 to 2020 03 31.

This cheat sheet was built by the scribe who would hugely appreciate and improve based on feedback, contact using mehmetoguzderin@mehmetoguzderin.com

#535

Author opens discussion about **ray tracing** with an example implementation made for **Dawn** with usage samples.

Mention of WGSL is subtle if not non-existent, only touching on the topic of specialized shaders for ray tracing extensions.

#651

Author converts definition of **vertex index accessor** to the Vulkan compatible **`gl_VertexIndex`** instead of `gl_VertexId`.

As non-controversial, merged as beneficial.

#658

Author shows tremendous and objective benefits of having **`FP16`** support. The **reduced battery usage** is mentioned in the author's post, which is quite

---

The discussion is positive for the support of **FP16** but comments mostly focus around having this as an extension. Building on that concern, method of activation of this feature in shaders that intend to use this extension is questioned.

#642

Author suggests usage of **`break N`** where N is an integer literal **instead of `continue` and `continuing`** constructs for control flow of **`loops`**.

Turns out that this proposal adds tracking burden to transpires at least when going from WGSL to SPIR-V. Request gets later closed by the author.

Reconsidering the approach and design of this section as it got too explicit and littered to speak to the eye, the sheet will start to cover all topics again soon. 😴😴😴

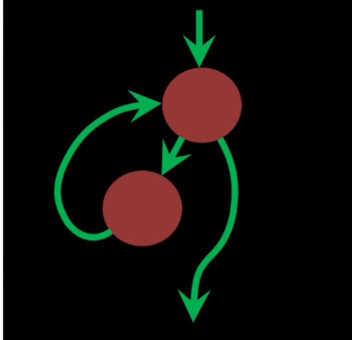
Meta (click here to jump to the discussion section)

-
-
-
-

⚖️ Discussion

🔗 Issue #643 - Remove `break if` in favor of `if() {break; }`

- RM: Got a question. About comment with snippet of code. Can DN clarify the meaning?

- DN: So statement A could only possibly execute on the last iteration. It's on the way out of the loop. When oyu look at the control flow graph, statement A not in the cycle for the loop. I have memories…
- RM: From a uniformatic point of view [. . .]
- DN: That's why I said this is a C example, single threaded. Yes, if we apply the uniformity goals then OK, otherwise I'd like to avoid.
-  John K's avatar, referenced in the issue.
- Overview: Robin and David discuss whether the C example in David's comment is actually part of the control flow, or whether it matters from a uniformity point of view.
- DN: I would like to guide people to patterns with less divergence. [...]. It will be idiomatic that way.
- MM: There is no way with `break if` for the A to be in the loop, so, confused.
- DN: There is complexity added by this. If the code can be outside the loop `break if` forces the code outside the loop
- RM: Should we have `break if` and `break`
- DN: would only have `break if` if had to choose, but can have both
- JG: THis is just about showing the difference between the two. So, a non-trivial compiler could do the work, but in this case the argument is that it's valuable to surface that it's different
- MM: How does `break if` solve the statement A issue
- DN: Difference between C and GPU languages where uniformity kicks in. Having the rule that you have to think about the correctness of where things appear. Want to steer folks away from having to code inside the breaking if condition. There will be ways to express the cond A case, but should be exceptional
- JG: The Smtt A is about uniformity. Outside the loop it's always in uniform control flow, but in the lopo could be non-uniform
- RM: The uniformity depends on conditional of the loop and the branche
- MM: So stmt A can't be a derivative
- JG: Correct
- RM: Sounds like 2 different arguments. Make it difficult to write code with uniformity issues. Write what a simple compiler would do. Do you agree these are 2 concerns
- DN: Yes. Tried to avoid translator augment. Tend to not mention those arguments but they are additive.

- RN: Basic blocks would be easy for the compiler to remove. As long as we have the ability to add `break` don't see much condition with keeping `break if` for uniformity argument. Do we agree `break if` is semantically equivalent to `if() break` just syntactic sugar
- DN: Yes, sugar for single threaded code
- JG: Somewhat worrying that this is opt-in. `break if` can be in best practice document but most people will just use `break` as `break if` is exotic. So we have to handle `if() break` properly
- DJ: Originally suggested to remove, now talking about leaving it in as optional. Robin suggesting removing as `break` is good enough
- JG: All just entertaining ideas. Is the extra data for the user of `breka if` useful to the user vs having the compiler handling `if() break`
- MM: Personally, `do x if y` is harder to understand. Don't want to say having an idiomatic way to write clear code is a bad idea.
- JG: Doesn't feel like break if adds that much. Especially if opt-in. On fence of having 2 semantically equivalent things in the language.
- DN: With `break unless` you have rule that `if condition break` has non-uniform control flow. But not case for `break fi` as uniformity resolves at exit. Having distinct construct means you don't have to think much
- JG: Don't agree
- RN: any case where `if () { A; B; break;}` is different in terms of uniformity?
- DN: Idea `if () break` without braces
- DS: Could we force break to be only statement
- RM: Some cases where you want statements in there, and the break if allows them avoid it so we want to allow it.
- PJ: This is a new construct so folks won't' be used to it so a lot of education of shader writers
- JG: Not that much requirement to do it the right way, would just write `if() {break}` and assume the compiler is doing the right thing.
- PJ: `break if()` would be unnatural to write
- DN: We are used to what we're used to and that's what we're taught. Turing language is different ordering,
- PJ: If we allow both most developers will use what they're used to
- KN: No semantic difference between `if (){break;}` and `break if()` so not much high mental overhead of checking what's inside that block. Default assumption is that the uniformity is dependent on the if. So, don't think it's that different. Extra syntax doesn't seem necessary to mark it as non-uniform.
- JG Give this some thought and work through other issues?
- MM: Straw poll?
- JG: Need to think about it
- DJ: Will come back to it next week.

~~🔗 Issue #642 - Add `break N` instead of `continue` and `continuing`~~

🔗 Issue #654 - var declaration without storage class: spell out rules, infer Function storage class inside a function

- DN: Artifact of draft being partial. Canonical way of writing variable is var <storage class > name : type . Because all variables inside a function are function storage class don't' have to write it. Outside functions default to Private storage class if not provided. Is that a sensible default and should we record it somewhere.
- JG Sounds reasonable to me
- MM: Reading about private vs function storage class, understand the words but not how they work together
- DN: Privat and function storage class, each work item gets its own copy. The private si for module scoped variables so visible to all functions called from that entry point. The function storage class is scoped to the function in which it is declared. Function storage class at top of function and visible throughout. Lifetime is for the duration of the function call. Same helper function called from 2 places in the call stack would have different Function scoped variables.
- MM: Like static on a variable in C
- DN: Yes. Private is static on the module scope. Function storage class is like alloca. If you have a function initializer it is re-initialized every time the function is entered.
- MM: But for Private they don't get re-initialized
- DN: Right, initialized on top level entry point
- MM: So, their initializer can't depend on state of function
- DN: Correct in shader can only depend on constants
- MM: Is there a rule they can only be declared outside functions if they can only depend on constants?
  DN: Yes if you are declared outside a function you _cannot_ be function storage class. Private variables must be declared outside any function. Private storage class can only be outside, Function can only be inside a function.
- MM: That sounds reasonable.
- DM: Not a fan of defaulting to Private. Don't expect it to be used. Global scope dominated by input/output. Would rather see all the qualifiers at global scope
- RN: Considering everything inside a function must be function scope, seems like a no-brainer. No opinion on private.
- DM: Is that true, can we have an input qualifier inside a function? Like a pointer to an input
- DN: WebGPU Shaders won't have pointers, and you can't have an input storage class inside a function, must be module scoped. The point of view is reasonable, haven't seen that many private variables
- MM: Just implementing on metal requires some amount of ??? software
- DN: Would be fine with Dzmitry's position, just need a rule

- MM: Dzmitry.. I don’t quite understand your argument. Having input/output more common, why should the developer have to type extra stuff
- DM: Why make an exception for Private when most things will have qualifiers? Will look ore uniform if all things have qualifiers
- JG: I intuitively assume that anything at global without qualifiers would be private, matches model in my head. This is true in GLSL. In favor of defaulting to private and folks can be explity if they want. Since were already inferring for Function we can infer for Private as well.
- DM: Different grammar rule.
- DS: No decorations inside functions for vars.
- JG: Private is what I’d expect
- DM: Private not used often, dominated by input/output and uniforms
- JG: Asking people to do more work just to do work seems unnecessary
- MM: Does HLSL have the concept of Private?
- TR: Not that I’m trying to understand. Private scope sounds like static but then in a function?
  DN: Private is like a c static variable file scope
- TR: But not in function?
  DN: Not in function
- TR: But has a copy?
- DN: No, for function variables
- TR: So, just like a module static?
- DN: Making clarification every invocation gets its own
- TR: HLSL has htat, static global
- MM: Is it writable?
- TR: Yes, jst like C but thread local
- MM: Are they common?
- TR: Yup, people use them. Sometimes used to store input values so don’t have to push down to functions. For translating GLSL for instance since they use globals
- MM: Makes sense, that maybe the tie breaker.
- DJ: Leaves consensus leaning towards not requiring Private on Module level variable. Implicitly private unless specified otherwise. Variables in Function are function scoped by default.
- DM: Still think it’s an unnecessary complication of the grammar for small benefit. Not a fan of this idea.
- DJ: Will to accept going against judgement?
- DM: Yes, it’s fine.
- DJ: Resolved.
- DM: Would like to ask group that all arguments are on the github issue. Folks looking at the discussion will have hard time to determine why the decision was made
- MM: Do you think linking minutes is not sufficient

- DM: minutes are the minimum. And they aren't necessarily complete. The Rust language process is that all positions are stated on the github issue and the conference call just discusses and does not include new ideas.
- JG: Think the minutes are generally sufficient.
- MM: How about arguments are put in the issue afterwards?
- DS: Could we tldr the notes into the issue?
- DJ: Resolution is still we came to agreement
- DN: Trying to summarize high level points
- DS: Asked for clarification on metal with Private scope
- MM: Metal doesn't have concept, would gather all those global variables and put into struct and entry point declare one instance of that struct and pass a pointer to the struct into each function then reads/writes are done through the struct.

🔗 Issue #562 - Add FAQ for direction on shader language

- RM: OK to delay the discussion until next time
- DJ: Next time
- RM: Link : https://github.com/RobinMorisset/gpuweb/tree/wgsl-faq3

🔗 Issue #588 - Update Goals section of spec

- DS: Can we merge it or not?
- MM: I'd like the discussion on FAQ solved first. We've been working hard on the FAQ. Goals should build on them.

🔗 Issue #644 - Interface matching rules

- MM: This needs to be in spec. We don't have any strong opinion but would like some spec.
- DM: I think that's how rules work for valid use in Vulkan. You can have more things producer than can be consumed, applies to input outputs and render targets too.
- MM: Is there a logic for not zeroing out things that are consumed by a subsequent stage but not produced.
- DN: This might require some fix up
- DJ: So if something gets consumed that's not produced that should produce error?
- JG: Yes
- DM: May have one vertex shader used with multiple fragment shaders, which would slow down pipeline creation
- MM: We can start with it being an error
- DN: Vulkan started with more strict rules but relaxed over time, it settled on something like this a couple of years ago. I am fine with this because it doesn't have any performance implications. When assigning locations . Can sweep away variables if they aren't used in the interface. Can have interactions with locations if they're determined automatically.

- MM: Shaders can get created as mismatch snippets thrown together. Shader author might not even know what's being i/o. Zeroing out could help these situations. But still, I am willing to be OK with the proposal.
- DS: ...Part in webgpu and part in wgsl. WebGPU should specify that in/out must match between the stages. In WGSL we need to specify that variables used in an entry point must be initialized to zero
- DJ: Someone want to write the section?
- MM: I can.
- DJ: Assigned to MM
- DM: So this is the text that will say it formally about being explicit or not about i/o variables
- MM: Does the driver fix this up?
- DN: Not sure where requirement comes from, WebGPU spec only requires some things to be initialized
- DS: That's fine, we can make text match requirements.
- DM: Makes sense for outputs. We don't want to completely inspect the control flow and prove it up.
- JG: WebGL initializes all the outputs at the beginning of the main function.
- DJ: I think ANGLE also does that. Action on Myles.

🔗 **Mail #public-gpu/2020Mar/0024** - Test for round-trippability

-
-
-
-

🔗 **Issue #640** - The set of functions which have to be imported seems arbitrary

-
-
-

🗓️ **Next Week**

-
-
-
-