COLLABORA

Zink: reducing stutters with uber shaders

Antonino Maniscalco

Collabora

October 14, 2023

Open First

Hi

- Linux, Vulkan, Rust etc. tinkerer
- Zink/Mesa contributor since Jan 2023
- Some of things I've done on Zink:
  - fixed a bunch of things
  - added emulation for GL_POINT, edge flags, pv mode and other features

The problem

- OpenGL has features controlled by state
- Zink may do emulation in shaders
  - some of them don't exist in vulkan, you would use shaders instead
  - each state might require a shader variant
  - state only known at draw time
- $\Longrightarrow$ compilation stutters

Precompilation

Compile variants ahead of time?

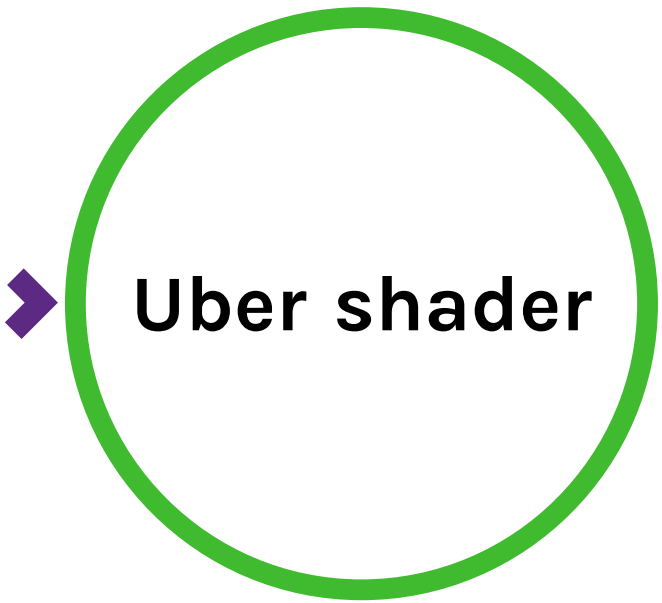
- Explosive number of combinations
- that $*$ user shaders

Current state

- Zink precompiles base variants
  - works great when no emulation is needed
  - does nothing otherwsie

COLLABORA

Uber shaders



- Big shader that can do all emulation
- Dynamically controlled

- Advantages:
  - no need for variants $\Longrightarrow$ can be precompiled
- Disadvantages:
  - potentially slower (bad branching and register pressure)
  - takes longer to process and compile

So ...

- Kick uber shader compilation ASAP
  - done asynchronously with util_queue
- When drawing
  - use variant when ready
  - bind uber shader if no variant is ready
  - kick variant compilation
- Best of both worlds

Presentation Outline

What does it look like in practice

Implementing in Zink

Introducing uber shaders

Current state of the patch

Some numbers

NIR passes

- We don't have the luxury of just creating shaders
- The user (gallium frontend) provides them
- Emulation done with NIR passes most of the time

Sysvals

- Sysvals are great
- NIR passes might use sysvals for parameters
  - lower to push constant loads for uber
  - lower to inlined constants for variants

Changes to passes

- Making sure sysvals are used for all parameters
- All passes need a way to be dynamically disabled
- Sometimes no changes are necessary (nir_lower_alpha_test)
- Sometimes not enough
  - nir_lower_flatshade changes variable attributes
  - can't be changed dynamically
  - every var needs to be duplicated and bcsel-ed from fragment
  - interface needs to match between multiple shader combinations

COLLABORA

Hilbert's shader slots

- Each variable slot becomes slot $* 2$
- Each duplicated variable goes in slot $* 2 + 1$

but..

- We don't have infinite rooms ...

Alternative solutions

- Vulkan extension to expose all attributes and barycentric?
  - VK_KHR_fragment_shader_barycentric exists
  - stable vertex order?
  - not widely available

nir_passthrough_gs

- Geometry shader used to emulate some features
- Interface needs to match with vs or tes
- Not the only emulation GS
- Created on demand
- Causes precompiled GPLs to be discarded and disabled

COLLABORA

Presentation Outline

What does it look like in practice

Implementing in Zink

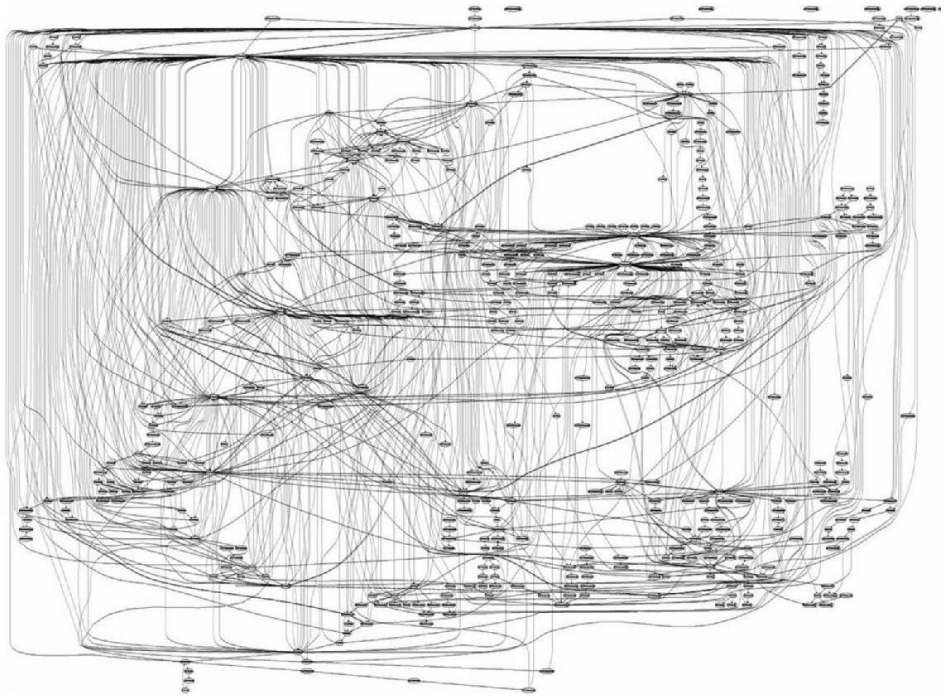
Introducing uber shaders

Current state of the patch

Some numbers

Shaders in Zink

- Shader caching in Zink is very simple, here is a diagram:



COLLABORA

Shaders in Zink

Open First

Shaders in zink

- Shader state gets created and bound
- At draw time:
  - a pipeline is searched for state hash
    - if found it might get replaced with an optimized linked pipeline
  - if not found a pipeline is created from current program + some state

Shader in zink

- Programs are fetched from a cache
  - (the key is a hash of user shaders and some state)
- To handle variants the shader modules are updated
- Shader modules cached by shader keys
- Pipeline libraries for variants are stored in a cache (owned by program)
- Caches might be filled asynchronously

Asynchronous precompilation

- During shader state or program creation compilation is kicked
- Entry added to cache
- Entry contains a fence
  - on cache hit wait on fence

Cached cache

- Some games use a separate context to compile programs asynchronously
  - DOOM 2016 does this
- To support this zink will share the pipeline lib caches across contexts
  - this is effectively a cache for the cache
  - same key as program caches

Presentation Outline

What does it look like in practice

Implementing in Zink

Introducing uber shaders

Current state of the patch

Some numbers

Variants handling

- When separate shaders are used zink replaces the program
  - whatever was precompiled gets lost
- we need to keep the uber shader around

Variants handling

- We now have a program per variant
  - a program only holds one GPL
  - a cross context cache is used for uber shaders GPLs
- Programs now hold a cache of variants
- Fast path for base variant
- When we need a variant:
  - if cache hit check fence
  - if the fence is signaled use variant
  - if fence not signaled or cache miss use uber
  - on cache miss we also start the kick compilation

Compiling variants

- Compiling a variant requires some steps
  - program created from separate shaders must have been replaced
  - run compilation pipeline to get the shader modules
  - create gpl
- For each stage the corresponding caches and fences are checked
- If any not ready use uber and kick job for the next stage
- If all stages are done use variant prog

COLLABORA

Presentation Outline

What does it look like in practice

Implementing in Zink

Introducing uber shaders

Current state of the patch

Some numbers

Supported legacy features

- PIPE_CAP_GL_CLAMP
- PIPE_CAP_CLIP_PLANES
- PIPE_CAP_FRAGMENT_COLOR_CLAMPED
- PIPE_CAP_ALPHA_TEST
- PIPE_CAP_FLATSHADE (wip)

Current state of the patch

- Two branches
  - dirty branch about 80 commits
  - clean branch about 60 commits
- Plan is to land what has been cleaned first
- Some features have not been tackled at all yet

Requirements

- All requirements for Zink's optimal path
  - GPL, dynamic state and others
- 256 bytes of push contants

COLLABORA

Presentation Outline

What does it look like in practice

Implementing in Zink

Introducing uber shaders

Current state of the patch

Some numbers

A pathological example

- OpenMW trace
- Uses ucp and GL_CLAMP
- At one point it sends a whole bunch of shaders
  - this will always stutter on any driver
  - variants need to be compiled for each shader

The numbers

- Without patches:
  - cold cache: 893ms
  - hot cache: 262ms
- With patches:
  - hot cache: 276ms
- ???

What is going on?

A check is performed that might disable the uber shaders path

```
bool can_use_uber = zink_can_use_uber(&ctx->gfx_pipeline_state);
```

- single feature not supported by the uber shader
  $\Longrightarrow$ stutter
- added overhead (TODO improve)

More numbers

let's hack it to always use the uber shader path

```
bool can_use_uber = true || zink_can_use_uber(&ctx->gfx_pipeline_state);
```

- Rendering breaks a bit
- Without patches:
  - no disk cache 388ms
  - with disk 262ms
- With patches:
  - no disk cache 317ms
  - with disk 221ms
- Improvement!

More recent numbers

Those are numbers after rebasing on more recent zink

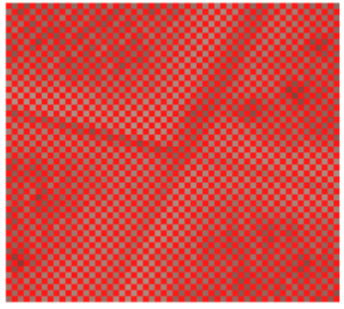
- Without patches:
  - no disk cache 504ms
  - with disk cache 383ms
- With patches:
  - no disk cache 453ms
  - with disk cache 360ms
- With patches and hack:
  - no disk cache 382ms
  - with disk cache 310ms
- improvement!

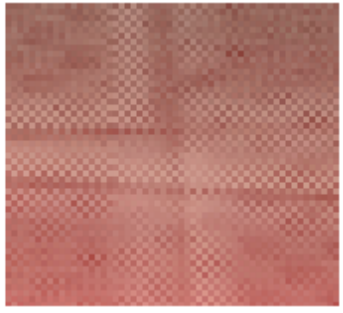
Currently working on...

- Only start compiling uber shaders once they are needed once
  - otherwise pre compile base variants
  - cuts down precompile time for well behaved applications
  - requires annoying logic

Visualizing uber shaders

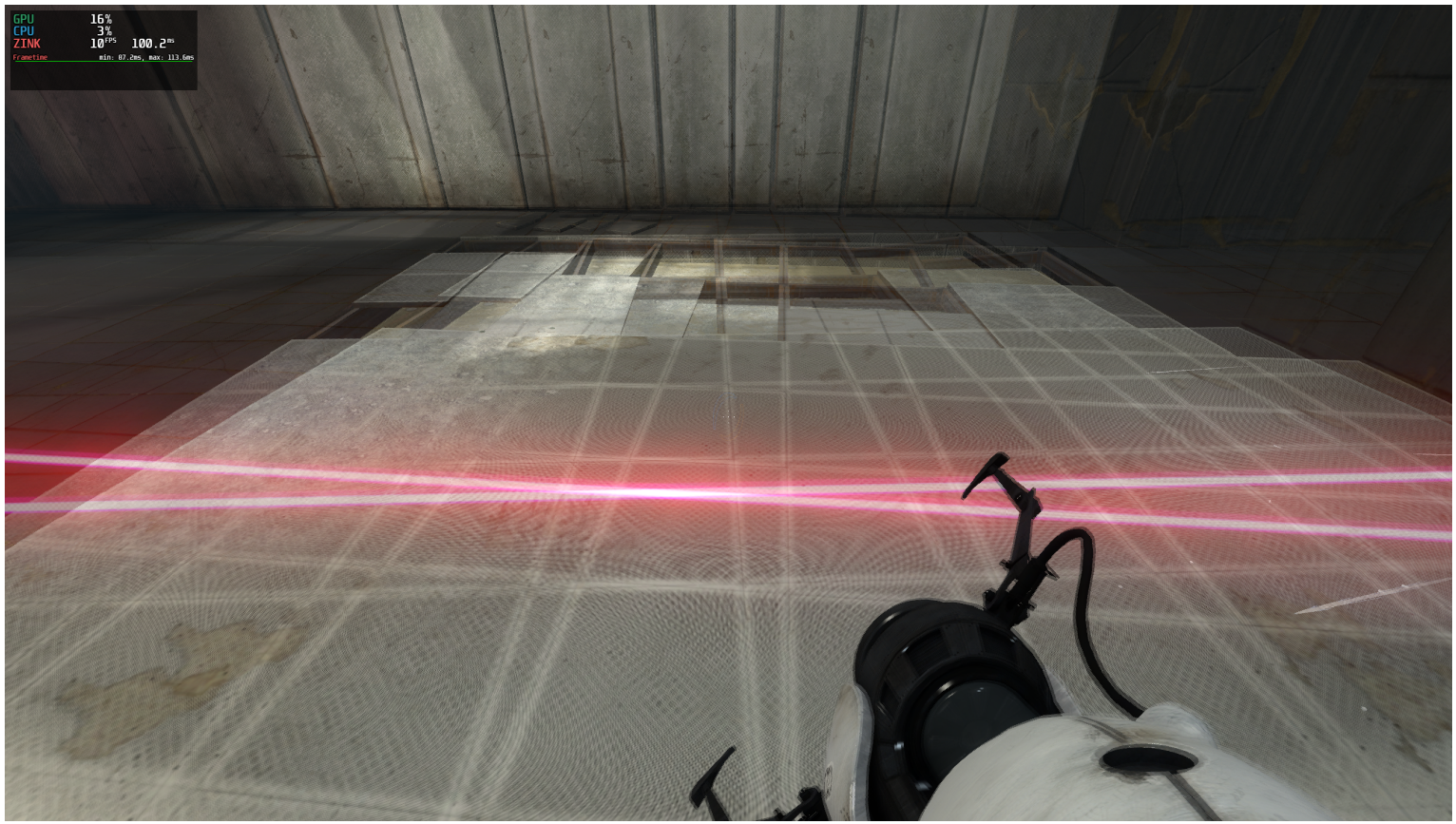
- Just output red from uber fragment shaders
  - objects often covered or offscreen
- Use discard in a checkerboard pattern for non uber
  - uber still writes all pixels so always visible
  - those pixels would never get cleared so alternate pattern
  - previous frames will remain in the non drawn pixels





COLLABORA

Visualizing uber shaders



Open First

Visualizing uber shaders

Demo!

COLLABORA

Visualizing uber shaders

Open First

COLLABORA

Visualizing uber shaders

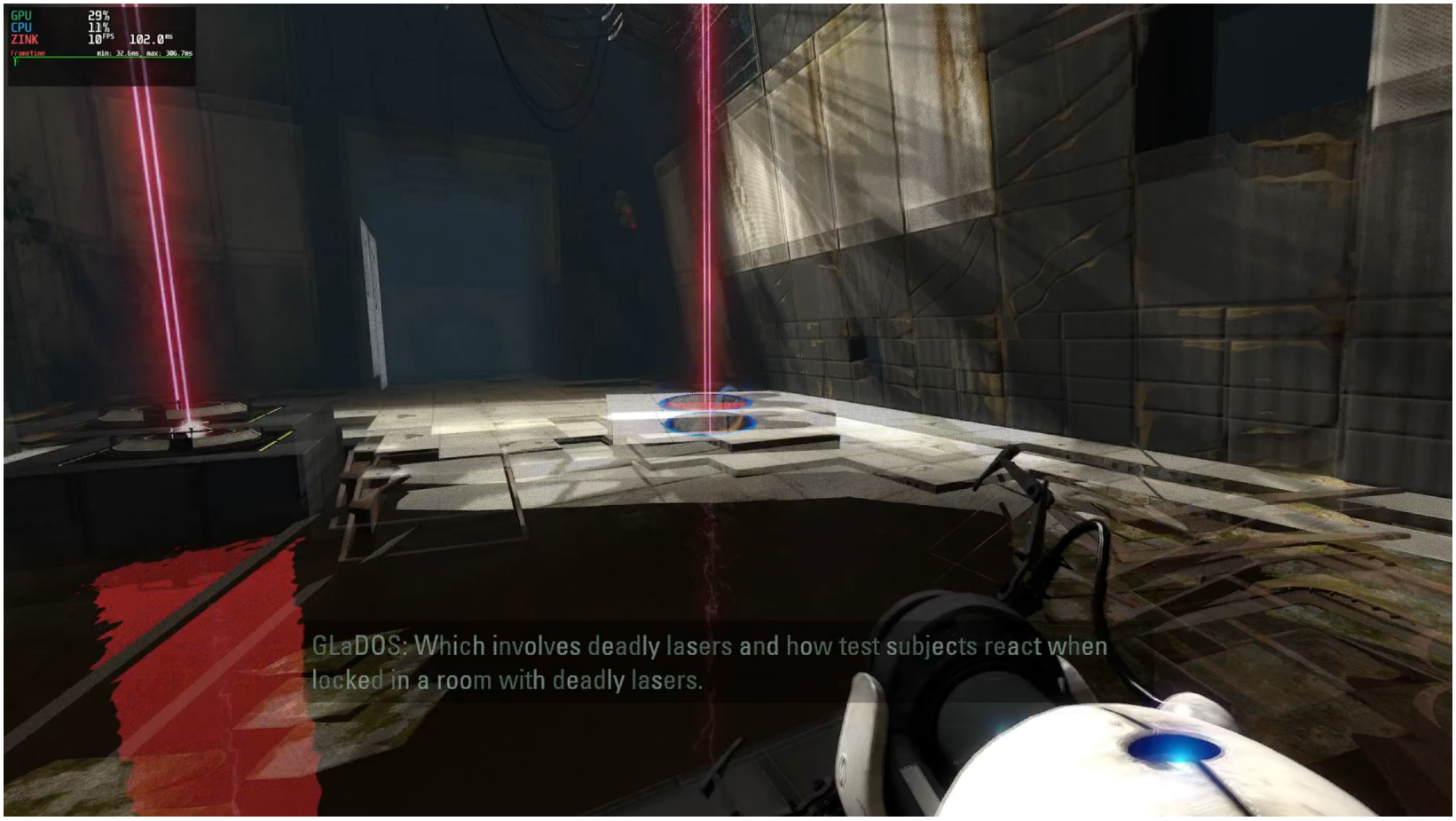
Open First

COLLABORA

Visualizing uber shaders

COLLABORA

Visualizing uber shaders



Open First

Thanks

- Collabora
  - for allowing me to work on this
- Erik Faye-Lund @kusma
  - orignal author of Zink
  - helped me getting started with Zink
  - helped discussing spec details
- Mike Blumenkrantz @zmike
  - originally proposed I'd work on this
  - tons of suggetions
- other people that have or will help review
  - @zmike
  - @alyssa

Thanks!

Q & A

COLLABORA