



**SOLUTION TO SHADER  
RECOMPILES IN RADEONSI**

SEPTEMBER 2015

# PROBLEM



- ▲ Shaders are compiled in draw calls
  - Emulating certain features in shaders
- ▲ Drivers keep shaders in some intermediate representation
- ▲ And insert additional code based on the states
- ▲ While compiling, everything stops
- ▲ Number of state combinations is exponential

## ▲ Fragment shader:

- Conversion to colorbuffer formats (RGBA32, RGBA FP16, ...)
- Alpha-test
- Selecting between front and back colors
- gl\_FragColor
- GL\_ALPHA\_TO\_ONE
- Polygon stippling
- Line & polygon smoothing
- Point smoothing
- Fragment color clamping

- ▲ Vertex shader:
  - Loading inputs from vertex buffers manually
  - Vertex color clamping

## ▲ Observation:

- All states can be applied at the beginning or end of shaders
- At link time, compile application shaders
- At draw time, append any shader bytecode needed

## ▲ 3 shader sections:

- Prolog section
- Main section (application shader)
- Epilog section

## ▲ Concatenate them

# FRAGMENT SHADER EPILOG



Color outputs are expected in r0, r1, ...

```
out0 = r0;  
out1 = r1;
```

# FRAGMENT SHADER EPILOG



If we need alpha-test:

```
if (!alphafunc(r0.w, alpharef)) discard;
```

```
out0 = r0;
```

```
out1 = r1;
```

# FRAGMENT SHADER EPILOG



If we need color clamping:

```
r0 = clamp(r0, 0, 1);  
r1 = clamp(r1, 0, 1);  
if (!alphafunc(r0.w, alpharef)) discard;
```

```
out0 = r0;  
out1 = r1;
```



If we need polygon stippling:

```
r0 = clamp(r0, 0, 1);  
r1 = clamp(r1, 0, 1);  
if (!alphafunc(r0.w, alpharef)) discard;  
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;  
out0 = r0;  
out1 = r1;
```

If we need smoothing:

```
r0 = clamp(r0, 0, 1);  
r1 = clamp(r1, 0, 1);  
if (!alphafunc(r0.w, alpharef)) discard;  
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;  
r0.w *= coverageMask; // popcount(gl_SampleMaskIn) / gl_NumSamples  
r1.w *= coverageMask;  
out0 = r0;  
out1 = r1;
```

If color conversion is required:

```
r0 = clamp(r0, 0, 1);  
r1 = clamp(r1, 0, 1);  
if (!alphafunc(r0.w, alpharef)) discard;  
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;  
r0.w *= coverageMask; // popcount(gl_SampleMaskIn) / gl_NumSamples  
r1.w *= coverageMask;  
r0.xy = vec2(packHalf2x16(r0.xy), packHalf2x16(r0.zw));  
r1.xy = vec2(packHalf2x16(r1.xy), packHalf2x16(r1.zw));  
out0 = r0;  
out1 = r1;
```

If GL\_ALPHA\_TO\_ONE is enabled:

```
r0 = clamp(r0, 0, 1);  
r1 = clamp(r1, 0, 1);  
if (!alphafunc(r0.w, alpharef)) discard;  
if (texture2D(stipple, gl_FragCoord.xy / 32).x < 0.5) discard;  
r0.w *= coverageMask; // popcount(gl_SampleMaskIn) / gl_NumSamples  
r1.w *= coverageMask;  
r0.w = 1;  
r0.xy = vec2(packHalf2x16(r0.xy), packHalf2x16(r0.zw));  
r1.xy = vec2(packHalf2x16(r1.xy), packHalf2x16(r1.zw));  
out0 = r0;  
out1 = r1;
```

# FRAGMENT SHADER PROLOG



- ▲ Only contains two-side color selection
- ▲ Decreases performance if done always
  
- ▲ 3 scenarios:
  - Two-side colors are enabled:
    - Select colors based on `gl_FrontFacing`
    - Store them into registers `r0`, `r1`
  - Two-side colors are disabled:
    - Just copy front colors into `r0`, `r1`
  - No color inputs => prolog is empty
- ▲ Application shader should read colors from `r0`, `r1`

# COMPILING PROLOGS/EPILOGS



- ▲ Still have to be compiled in draw calls
  - Can be slow
  
- ▲ Use an assembler instead of the compiler
  - Our LLVM backend has an assembler too

# VERTEX SHADER INPUTS

- ▲ R600 had fetch shader
- ▲ Removed since GCN
  
- ▲ Current implementation:
  - One buffer per input
  - Instance divisor == 0: Fetch  $\text{BaseVertex} + \text{VertexID}$
  - Instance divisor != 0: Fetch  $\text{StartInstance} + (\text{InstanceID} / \text{instance divisor})$

- ▲ Emulate fetch shader with prolog section
  - Drawback: can't move loads to hide latencies, register usage
- ▲ Instead, only calculate load addresses:
  - Prolog writes the addresses to r0,r1, ...
  - Main shader section executes the loads



# VERTEX SHADER EPILOG?

- ▲ Radeon has 3 ways to write VS outputs:
  - For rasterizer
  - For geometry shader
  - For tessellation control shader
- ▲ Don't use an epilog
- ▲ OpenGL sometimes knows which shader follows
- ▲ If not, compile all 3 variants with 3 threads in parallel
- ▲ Piglit only: Compile on demand in draw calls
- ▲ Vertex color clamping: use conditional assignment

- ▲ Middle-end, translates shaders from GLSL IR into TGSI
- ▲ Does that in draw calls
- ▲ State dependencies for draw calls:
  - Center vs sample interpolation
    - Instead, select coordinates with conditional assignment
  - Vertex and fragment color clamping
  - GL rendering context
- ▲ Any dependencies should be dealt with in drivers
- ▲ Other drivers will benefit too
  - GLSL->TGSI always done at link time

# IF GAMES COMPILE TOO LATE

- ▲ Compiling at link time doesn't help
- ▲ Use shader cache
- ▲ 1 shader variant => shader cache in core Mesa
  
- ▲ If games compile early => don't need it

# SKIP MESA OPTIMIZATIONS?

- ▲ Our LLVM backend can do most optimizations
  - No need to do them in Mesa
- ▲ Mesa/GLSL passes we do need:
  - Demoting inputs/outputs to local variables (dead code elimination?)
  - Function inlining
  - Breaking built-in input/output arrays into variables

# Questions?



**THANK YOU**

# DISCLAIMER & ATTRIBUTION



The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

## ATTRIBUTION

© 2015 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

AMD 