

VkRunner

A simple shader script tester for Vulkan

Neil Roberts



igalia

Overview

- Introduction
- History
- Examples
- Current status
- Future
- Questions



Introduction



What is VkRunner?

- Tool to test shaders on your Vulkan driver
- Inspired by Piglit's shader_runner
- Minimal overhead to execute a script
- Just write the scripts and some simple commands to execute them
- Standalone tool, runs the script and reports status

Example

```
[vertex shader passthrough]
```

```
[fragment shader]
```

```
#version 450
```

```
layout(location = 0) out vec4 color_out;
```

```
void
```

```
main()
```

```
{
```

```
    color_out = vec4(0.0, 1.0, 0.0, 1.0);
```

```
}
```

```
[test]
```

```
# Fill the framebuffer with the output from the shader
```

```
draw rect -1 -1 2 2
```

```
# Check that we got the colour we wanted
```

```
probe all rgba 1.0 0.0 0.0 1.0
```

Example

```
$ vkrunner ./simple-example.shader_test
Command failed at line 18
Probe color at (0,0)
    Expected: 1.000000 0.000000 0.000000 1.000000
    Observed: 0.000000 1.000000 0.000000 1.000000
PIGLIT: {"result": "fail" }
```

Behind the scenes

- Compiles the shader to SPIR-V by invoking glslang as an external process.
- Creates pipelines for the state for each draw command.
- Creates an offscreen framebuffer (no window system support).
- Puts test commands into a command buffer and executes it.
- Probes result.



History



ARB_gl_spirv

- VkRunner was created during Igalia's work to add support for ARB_gl_spirv to the i965 driver in Mesa.
- ARB_gl_spirv uses the same compiler as Intel's Vulkan driver.
- We were testing this with an adaptation of Piglit's shader_runner.

- shader_runner is the same principle as VkRunner.
- Tested ARB_gl_spirv by automatically converting existing shader_runner tests to SPIR-V.
- Piglit has many many tests.
- This ended up testing more of the Intel SPIR-V compiler than was tested with existing Vulkan tests.
- We wanted a quick way to verify whether test failures were specific to SPIR-V on OpenGL or also happen with Vulkan.

- shader_runner tests can be converted to VkRunner with minimal changes.
- However there are differences because of how Vulkan works.
- For GL, shader_runner can use the API to query properties of the shader such as the uniform names.
- This isn't available in Vulkan.
- Instead we use explicit offsets to set uniforms and SSBOs.

shader_runner example

```
[require]
GL >= 4.3
GLSL >= 4.30

[vertex shader passthrough]

[fragment shader]
#version 430

uniform vec4 color;
uniform float multiplier;
layout(location = 0) out vec4 color_out;

void
main()
{
    color_out = color * multiplier;
}

[test]
uniform vec4 color 0.5 0.25 0.5 1.0
uniform float multiplier 0.5
draw rect -1 -1 2 2
probe all rgb 0.25 0.125 0.25
```

shader_runner example

```
[require]
```

```
GL >= 4.3
```

```
GLSL >= 4.30
```

```
[vertex shader passthrough]
```

```
[fragment shader]
```

```
#version 430
```

```
uniform vec4 color;
```

```
uniform float multiplier;
```

global named uniform

```
layout(location = 0) out vec4 color_out;
```

```
void
```

```
main()
```

```
{
```

```
    color_out = color * multiplier;
```

```
}
```

```
[test]
```

```
uniform vec4 color 0.5 0.25 0.5 1.0
```

```
uniform float multiplier 0.5
```

```
draw rect -1 -1 2 2
```

```
probe all rgb 0.25 0.125 0.25
```

shader_runner example

```
[require]
GL >= 4.3
GLSL >= 4.30

[vertex shader passthrough]

[fragment shader]
#version 430

uniform vec4 color;
uniform float multiplier;
layout(location = 0) out vec4 color_out;

void
main()
{
    color_out = color * multiplier;
}

[test]
uniform vec4 color 0.5 0.25 0.5 1.0
uniform float multiplier 0.5
draw rect -1 -1 2 2
probe all rgb 0.25 0.125 0.25
```

set uniform by name

VkRunner equivalent

```
[vertex shader passthrough]

[fragment shader]
#version 430

layout(push_constant) uniform block {
    vec4 color;
    float multiplier;
};
layout(location = 0) out vec4 color_out;

void
main()
{
    color_out = color * multiplier;
}

[test]
# Set color
uniform vec4 0    0.5 0.25 0.5 1.0
# Set multiplier
uniform float 16   0.5
draw rect -1 -1 2 2
probe all rgb 0.25 0.125 0.25
```

VkRunner equivalent

```
[vertex shader passthrough]
```

```
[fragment shader]
```

```
#version 430
```

```
layout(push_constant) uniform block {  
    vec4 color;  
    float multiplier;  
};
```

**no global uniforms
need to use something else
eg, push constants**

```
layout(location = 0) out vec4 color_out;
```

```
void
```

```
main()
```

```
{
```

```
    color_out = color * multiplier;
```

```
}
```

```
[test]
```

```
# Set color
```

```
uniform vec4 0    0.5 0.25 0.5 1.0
```

```
# Set multiplier
```

```
uniform float 16    0.5
```

```
draw rect -1 -1 2 2
```

```
probe all rgb 0.25 0.125 0.25
```


VkRunner equivalent

```
[vertex shader passthrough]

[fragment shader]
#version 430

layout(push_constant) uniform block {
    vec4 color;
    float multiplier;
};
layout(location = 0) out vec4 color_out;

void
main()
{
    color_out = color * multiplier;
}

[test]
# Set color
uniform vec4 0 0.5 0.25 0.5 1.0
# Set multiplier
uniform float 16 0.5
draw rect -1 -1 2 2
probe all rgb 0.25 0.125 0.25
```

— set uniform by byte offset

Some improvements over shader_runner

- shader_runner code grown organically over time. Lives in a single large C file.
- VkRunner code written from scratch with the benefit of hindsight.
- Tries to partially automatically generate commands.
 - Systematic method for setting pipeline properties.
 - Try to support all formats for vertex data and framebuffer.



Examples

Vertex data

[vertex data]

Position Colour
0/R32G32_SFLOAT 1/A8B8G8R8_UNORM_PACK32

0.4	-0.4	0xff00a0ff
0.7	-0.7	0xff00a0ff
0.4	0.4	0xff00a0ff
0.7	0.7	0xff00a0ff

0.4	0.4	0xff0000ff
0.7	0.7	0xff0000ff
-0.4	0.4	0xff0000ff
-0.7	0.7	0xff0000ff

-0.4	0.4	0xff00ff00
-0.7	0.7	0xff00ff00
-0.4	-0.4	0xff00ff00
-0.7	-0.7	0xff00ff00

0.4	-0.4	0xffff0000
0.7	-0.7	0xffff0000
-0.4	-0.4	0xffff0000
-0.7	-0.7	0xffff0000

Vertex data

location

[vertex data]
Position Colour
0/R32G32_SFLOAT 1/A8B8G8R8_UNORM_PACK32

0.4	-0.4	0xff00a0ff
0.7	-0.7	0xff00a0ff
0.4	0.4	0xff00a0ff
0.7	0.7	0xff00a0ff

0.4	0.4	0xff0000ff
0.7	0.7	0xff0000ff
-0.4	0.4	0xff0000ff
-0.7	0.7	0xff0000ff

-0.4	0.4	0xff00ff00
-0.7	0.7	0xff00ff00
-0.4	-0.4	0xff00ff00
-0.7	-0.7	0xff00ff00

0.4	-0.4	0xffff0000
0.7	-0.7	0xffff0000
-0.4	-0.4	0xffff0000
-0.7	-0.7	0xffff0000

Vertex data

```
[vertex data]
# Position      Colour
0/R32G32 SFLOAT 1/A8B8G8R8 UNORM PACK32
```

```
0.4  -0.4  0xff00a0ff
0.7  -0.7  0xff00a0ff
0.4   0.4  0xff00a0ff
0.7   0.7  0xff00a0ff
```

```
0.4   0.4  0xff0000ff
0.7   0.7  0xff0000ff
-0.4  0.4  0xff0000ff
-0.7  0.7  0xff0000ff
```

```
-0.4  0.4  0xff00ff00
-0.7  0.7  0xff00ff00
-0.4 -0.4  0xff00ff00
-0.7 -0.7  0xff00ff00
```

```
0.4  -0.4  0xffff0000
0.7  -0.7  0xffff0000
-0.4 -0.4  0xffff0000
-0.7 -0.7  0xffff0000
```

**format names
from Vulkan enums**

Vertex data

```
[vertex shader]  
#version 450
```

```
layout(location = 0) in vec2 position;  
layout(location = 1) in vec3 color_in;
```

**[vertex data]
specifies inputs
for these**

```
layout(location = 0) out vec3 color_out;
```

```
void  
main()  
{  
    gl_Position = vec4(position, 0.0, 1.0);  
    color_out = color_in;  
}
```

Indices

```
[indices]
0 1 2 3      65535
4 5 6 7      65535
8 9 10 11    65535
12 13 14 15  65535
```


Draw command

```
primitiveRestartEnable true
```

```
draw arrays indexed TRIANGLE_STRIP 0 20
```

Vertex data

```
[vertex shader]
#version 450

layout(location = 0) in vec2 position;
layout(location = 1) in vec3 color_in;
layout(location = 0) out vec3 color_out;

void
main()
{
    gl_Position = vec4(position, 0.0, 1.0);
    color_out = color_in;
}

[fragment shader]
#version 450

layout(location = 0) in vec3 color_in;
layout(location = 0) out vec4 color_out;

void
main()
{
    color_out = vec4(color_in, 1.0);
}

[vertex data]
# Position      Colour
0/R32G32_SFLOAT 1/A8B8G8R8_UNORM_PACK32

0.4 -0.4  0xff00a0ff
0.7 -0.7  0xff00a0ff
0.4  0.4  0xff00a0ff
0.7  0.7  0xff00a0ff

0.4  0.4  0xff0000ff
0.7  0.7  0xff0000ff
-0.4 0.4  0xff0000ff
-0.7 0.7  0xff0000ff

-0.4 0.4  0xff00ff00
-0.7 0.7  0xff00ff00
-0.4 -0.4 0xff00ff00
-0.7 -0.7 0xff00ff00

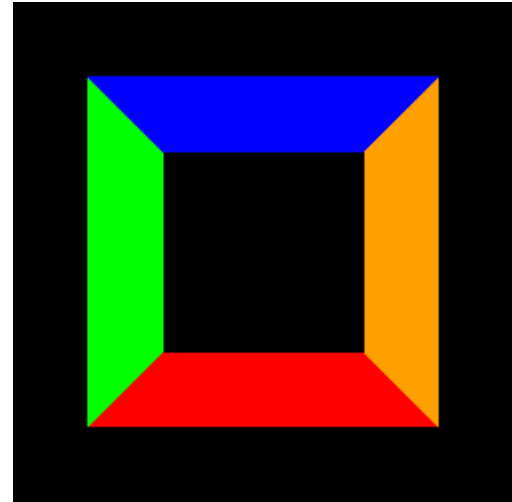
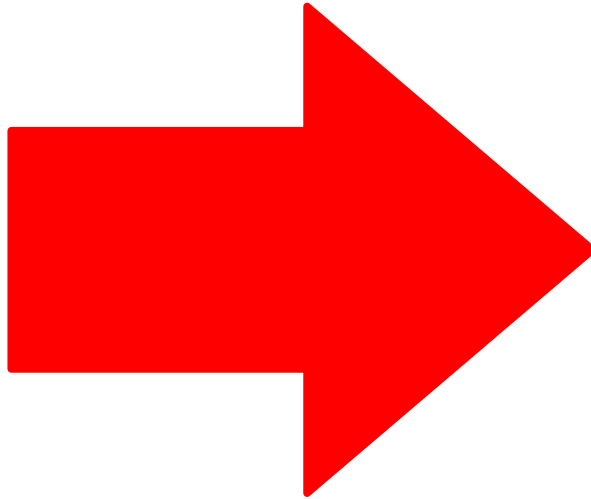
0.4 -0.4  0xffff0000
0.7 -0.7  0xffff0000
-0.4 -0.4 0xffff0000
-0.7 -0.7 0xffff0000

[indices]
0 1 2 3  65535
4 5 6 7  65535
8 9 10 11 65535
12 13 14 15 65535

[test]
clear

primitiveRestartEnable true

draw arrays indexed TRIANGLE_STRIP 0 20
```



Requires section

```
[require]
# Require an extension for the test to pass
VK_KHR_8bit_storage

# Change the framebuffer format
framebuffer R32_SFLOAT
fbsize 1024 768

# Require an optional Vulkan feature
shaderInt16
```

Compute shader

```
[compute shader]
#version 450

layout(binding = 0) buffer block {
    float values[];
};

void
main()
{
    // Calculate some square roots
    values[gl_WorkGroupID.x] = sqrt(gl_WorkGroupID.x);
}

[test]
ssbo 0 4096

# Run the compute shader
compute 1024 1 1

# Probe a few points in the buffer
probe ssbo float 0 0 ~= 0    1.0 1.4142 1.7320 2.0
probe ssbo float 0 2304 ~= 24.0
```

SPIR-V source

```
[fragment shader spirv]
    OpCapability Shader
    %1 = OpExtInstImport "GLSL.std.450"
    OpMemoryModel Logical GLSL450
    OpEntryPoint Fragment %main "main" %color
    OpExecutionMode %main OriginUpperLeft
    OpSource GLSL 450
    OpDecorate %color Location 0
    %void = OpTypeVoid
    %3 = OpTypeFunction %void
    %float = OpTypeFloat 32
    %_ptr_Output_float = OpTypePointer Output %float
    %color = OpVariable %_ptr_Output_float Output
    %float_1 = OpConstant %float 1
    %main = OpFunction %void None %3
    %5 = OpLabel
        OpStore %color %float_1
    OpReturn
    OpFunctionEnd
```

Binary source

- Script available to precompile scripts to binary format

```
./precompile-script.py -o precompiled *.shader_test  
vkrunner precompiled/*.shader_test
```

- Useful for running on devices where running the compiler isn't practical.

Binary source

```
[require]
framebuffer R32_SFLOAT
```

```
[vertex shader passthrough]
```

```
[fragment shader binary]
7230203 10000 70000 a 0 20011 1 6000b 1 4c534c47 6474732e
3035342e 0 3000e 0 1 6000f 4 2 6e69616d 0 3 30010 2 7 30003
2 1c2 40047 3 1e 0 20013 4 30021 5 4 30016 6 20 40020 7 3 6
4003b 7 3 3 4002b 6 8 3f800000 50036 4 2 0 5 200f8 9 3003e 3
8 100fd 10038
```

```
[test]
clear
```

```
draw rect -1 -1 2 2
```

```
probe all rgb 1 0 0
```



Current status



Features

- All shader stages
- UBOs/SSBOs
- Vertex data, simple drawing
- Probing the framebuffer or SSBOs

Library version

```
#include <stdio.h>

#include <vkrunner/vkrunner.h>

int
main(int argc, char **argv)
{
    struct vr_source *source =
        vr_source_from_file("simple-example.shader_test");

    struct vr_config *config = vr_config_new();
    struct vr_executor *executor = vr_executor_new(config);

    enum vr_result result = vr_executor_execute(executor, source);

    vr_executor_free(executor);
    vr_config_free(config);

    vr_source_free(source);

    return result == VR_RESULT_FAIL ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Integration

- Integrated into Khronos Vulkan CTS
 - Currently only experimental tests
 - Uses VkRunner's API
- Integrated into Piglit
 - Has real tests
 - Runs on Intel's CI

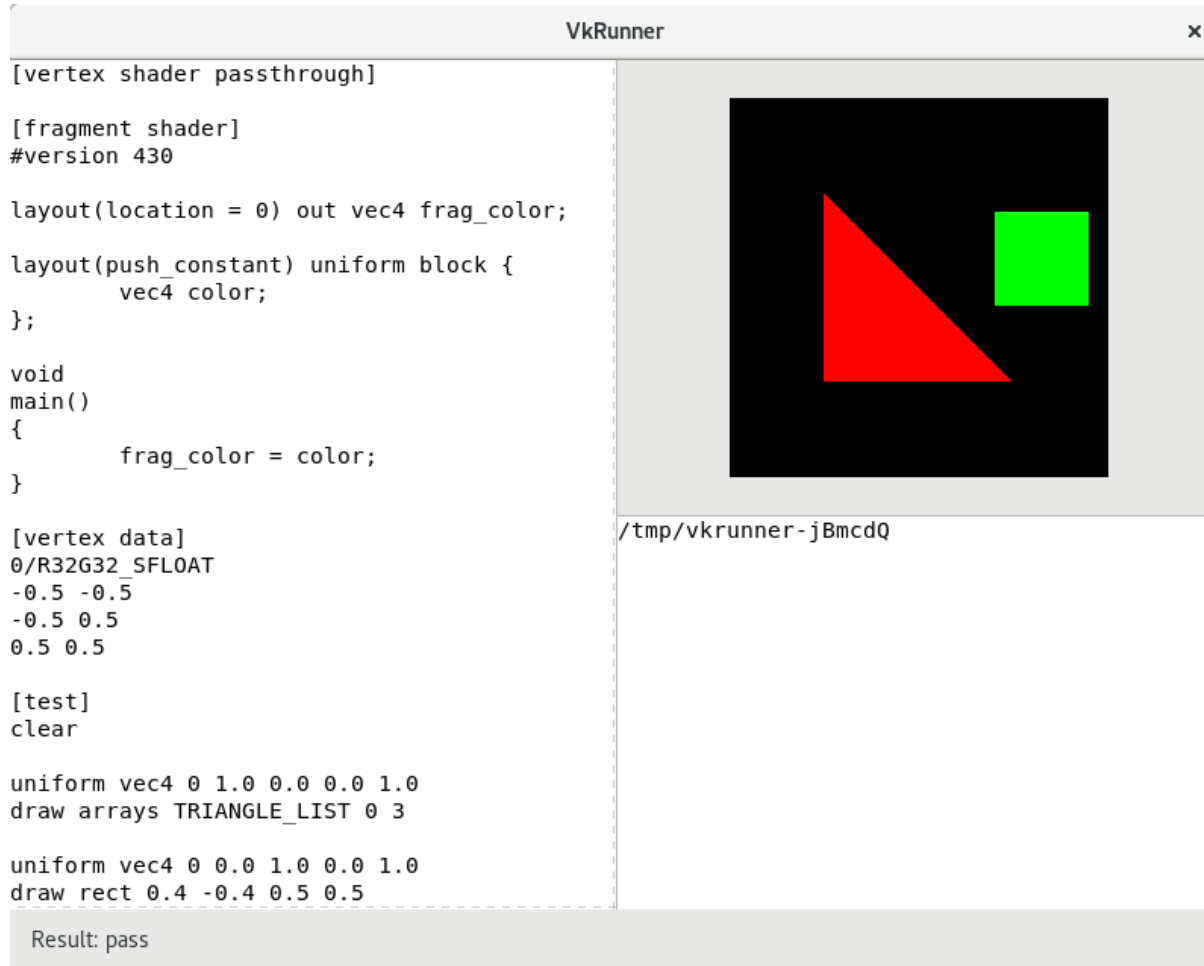


Future

Missing features

- Image / texture support
 - Although there is a pull request for this
- Arrays of buffer bindings
- Probably a lot of other things

User Interface



Video?

- There's a branch for making animations.
- Adds a magic uniform to specify the frame number.
- Can be used like an offline version of shadertoy

Amber

- Google are working on a similar tool.
- Can use the same scripting format as VkRunner.
- Yet to see which where it will lead.



[github.com/Igalia/
vkrunner](https://github.com/Igalia/vkrunner)



Questions?