Vulkanised 2024

The 6th Vulkan Developer Conference
Sunnyvale, California | February 5-7, 2024

MoltenVK for Advanced Vulkan Renderers on macOS

Roman Kuznetsov, Meta

Author

Software Engineer @ Meta

Formerly: Mapbox, Organic Maps (ex MAPS.ME), Alawar

GitHub: @rokuz
X (Twitter): @rokuz7

Outline

- What's "advanced Vulkan renderer"
- MoltenVK initialization
- "Bindless" organization
- Compute shaders usage
- Specialization constants usage
- Q & A

Use “bindless” for textures and buffers

- **VK_KHR_buffer_device_address**
  https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_KHR_buffer_device_address.html

  “Support for the `bufferDeviceAddress` feature is mandatory in Vulkan 1.3, regardless of whether this extension is supported.”

  - **GL_EXT_buffer_reference**
  - **GL_EXT_buffer_reference_uvec2**

- **VK_EXT_descriptor_indexing**
  https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_EXT_descriptor_indexing.html

  “Functionality in this extension is included in core Vulkan 1.2, with the EXT suffix omitted. However, if Vulkan 1.2 is supported and this extension is not, the `descriptorIndexing` capability is optional.”

Dynamic rendering

- **VK_KHR_dynamic_rendering**
  https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_KHR_dynamic_rendering.html

  Rendering without `VkRenderPass` and `VkFramebuffer` objects where it makes sense.

  "Functionality in this extension is included in core Vulkan 1.3"

- **VK_EXT_extended_dynamic_state**
  https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_EXT_extended_dynamic_state.html

  Move out frequently changing states (cull mode, primitive topology, depth-stencil state) from `VkPipeline` object.

  "All dynamic state enumerants and entry points in this extension are included in core Vulkan 1.3, with the EXT suffix omitted. The feature structure is not promoted. Extension interfaces that were promoted remain available as aliases of the core functionality."

MoltenVK Compatibility

- **Metal Tier 2 Argument buffers must be enabled (Metal 3+)**
  MVK_CONFIG_USE_METAL_ARGUMENT_BUFFERS = 1 (or 2)

- **Vulkan 1.3 is almost supported in MoltenVK**
  https://github.com/KhronosGroup/MoltenVK/issues/1930

- **No VK_KHR_draw_indirect_count support**
  https://github.com/KhronosGroup/MoltenVK/issues/168

Extension	MoltenVK Support	Status Notes
VK_KHR_copy_commands2	✅	
VK_KHR_dynamic_rendering	✅	
VK_KHR_format_feature_flags2		
VK_KHR_maintenance4		
VK_KHR_shader_integer_dot_product		
VK_KHR_shader_non_semantic_info	✅	
VK_KHR_shader_terminate_invocation		
VK_KHR_synchronization2	✅	Added (#2021)
VK_KHR_zero_initialize_workgroup_memory		
VK_EXT_4444_formats	✅	
VK_EXT_extended_dynamic_state	✅	Added (#2036)
VK_EXT_extended_dynamic_state2	✅	Added (#2036)
VK_EXT_image_robustness	✅	
VK_EXT_inline_uniform_block	✅	
VK_EXT_pipeline_creation_cache_control	✅	
VK_EXT_pipeline_creation_feedback	✅	
VK_EXT_private_data	✅	
VK_EXT_shader_demote_to_helper_invocation	✅	
VK_EXT_subgroup_size_control	✅	
VK_EXT_texel_buffer_alignment	✅	
VK_EXT_texture_compression_astc_hdr	✅	
VK_EXT_tooling_info		
VK_EXT_ycbcr_2plane_444_formats		

MoltenVK Initialization

- **Enable Tier 2 argument buffers**

** After adding **VK_EXT_layer_settings** support*

```
#if defined(__APPLE__)
// https://github.com/KhronosGroup/MoltenVK/blob/main/Docs/MoltenVK_Configuration_Parameters.md
const int useMetalArgumentBuffers = 1;
const VkLayerSettingEXT settings[] = {
    {"MoltenVK", "MVK_CONFIG_USE_METAL_ARGUMENT_BUFFERS",
     VK_LAYER_SETTING_TYPE_INT32_EXT, 1, &useMetalArgumentBuffers}};

const VkLayerSettingsCreateInfoEXT layerSettingsCreateInfo = {
    .sType = VK_STRUCTURE_TYPE_LAYER_SETTINGS_CREATE_INFO_EXT,
    .pNext = config_.enableValidation ? &features : nullptr,
    .settingCount = (uint32_t)LVK_ARRAY_NUM_ELEMENTS(settings),
    .pSettings = settings};
#endif
```

More details: https://github.com/corporateshark/lightweightvk/pull/21

MoltenVK Initialization

- **Disable shader validation**

```
#if defined(__APPLE__)
// Shader validation doesn't work in MoltenVK for SPIR-V 1.6 under Vulkan 1.3:
// "Invalid SPIR-V binary version 1.6 for target environment SPIR-V 1.5 (under
// Vulkan 1.2 semantics)."
const VkValidationFeatureDisableEXT validationFeaturesDisabled[] = {
    VK_VALIDATION_FEATURE_DISABLE_SHADERS_EXT,
    VK_VALIDATION_FEATURE_DISABLE_SHADER_VALIDATION_CACHE_EXT,
};
#endif

const VkValidationFeaturesEXT features = {
    .sType = VK_STRUCTURE_TYPE_VALIDATION_FEATURES_EXT,
    .pNext = nullptr,
    .enabledValidationFeatureCount =
        config_.enableValidation
            ? (uint32_t)LVK_ARRAY_NUM_ELEMENTS(validationFeaturesEnabled)
            : 0u,
    .pEnabledValidationFeatures =
        config_.enableValidation ? validationFeaturesEnabled : nullptr,
#if defined(__APPLE__)
    .disabledValidationFeatureCount =
        config_.enableValidation
            ? (uint32_t)LVK_ARRAY_NUM_ELEMENTS(validationFeaturesDisabled)
            : 0u,
    .pDisabledValidationFeatures =
        config_.enableValidation ? validationFeaturesDisabled : nullptr,
#endif
};
```

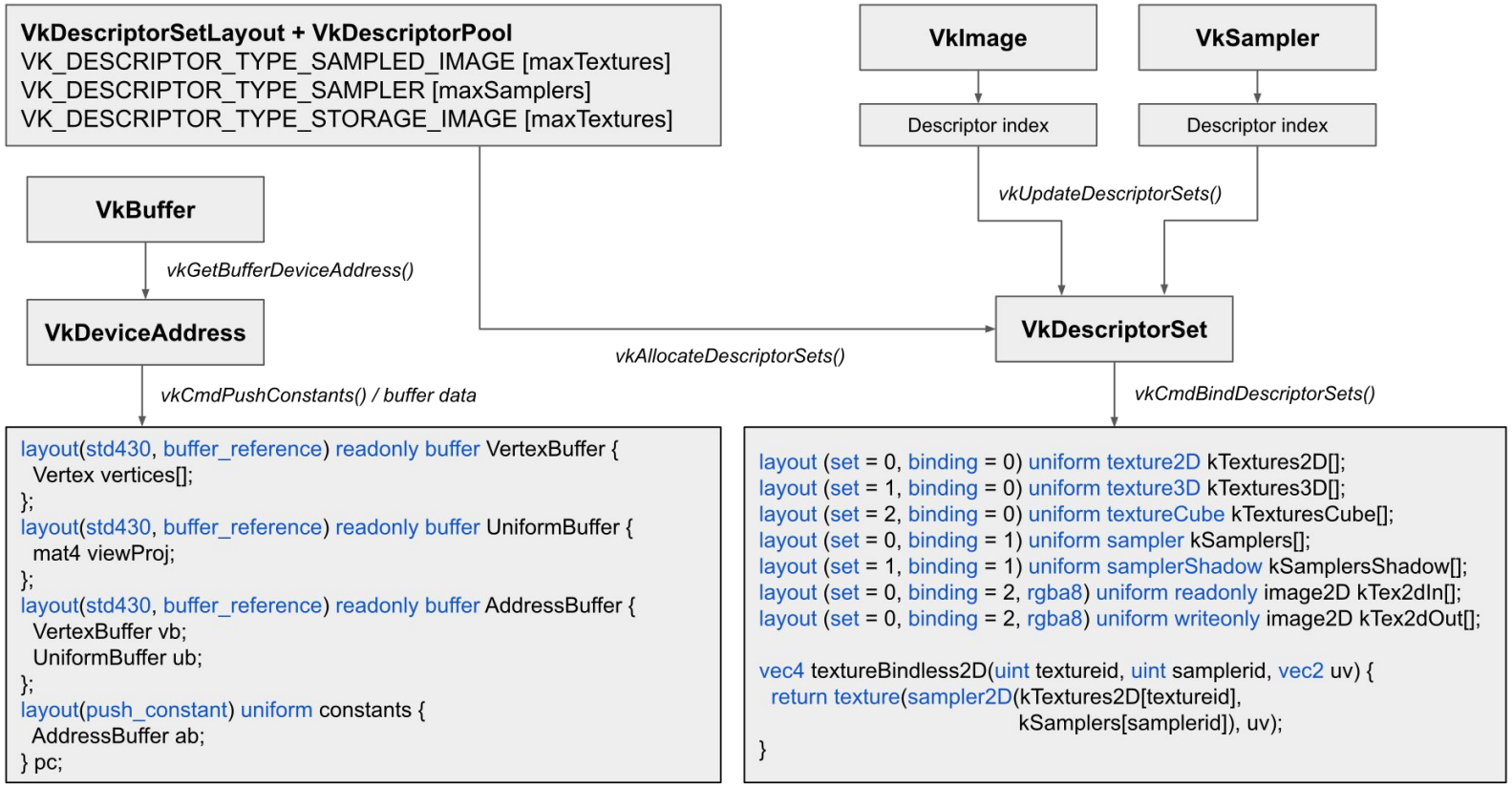
More details:
https://github.com/corporateshark/lightweightvk/blob/987f91e28bd9b4e783ece9db6ef0697b1ec58ccc/lvk/vulkan/VulkanClasses.cpp#L4214

MoltenVK Initialization

- **Hacking Vulkan-ValidationLayers (if you need them)**
  https://github.com/KhronosGroup/Vulkan-ValidationLayers

```
bool debug_printf::Validator::InstrumentShader(
    const vvl::span<const uint32_t> &input, std::vector<uint32_t> &new_pgm,
    uint32_t unique_shader_id, const Location &loc) {
  return false;
}

bool gpuav::Validator::InstrumentShader(const vvl::span<const uint32_t> &input,
                                        std::vector<uint32_t> &new_pgm,
                                        const uint32_t unique_shader_id,
                                        const Location &loc) {
  return false;
}
```

“Bindless” Organization



Bindings aliasing

- **Binding aliasing is supported in Vulkan**
  In this example, one texture and one sampler array will be added to a descriptor set.

```
layout(set = 0, binding = 0) uniform texture2D kTextures2D[];
layout(set = 0, binding = 0) uniform texture3D kTextures3D[];
layout(set = 0, binding = 0) uniform textureCube kTexturesCube[];
layout(set = 0, binding = 1) uniform sampler kSamplers[];
layout(set = 0, binding = 1) uniform samplerShadow kSamplersShadow[];
```

- **MoltenVK (Metal) does not support bindings aliasing :(**

```
struct spvDescriptorSetBuffer0 {
  array<texture2d<float>, 16> kTextures2D [[id(0)]];
  array<texture3d<float>, 1> kTextures3D [[id(0)]];
  // <...>
};

// [mvk-error] VK_ERROR_INITIALIZATION_FAILED: Shader library compile failed
// (Error code 3): program_source:24:55: error: cannot assign resource locations
// to 'spvDescriptorSet0' fragment void main0(constant spvDescriptorSetBuffer0&
// spvDescriptorSet0 [[buffer(0)]], constant spvDescriptorSetBuffer2&
// spvDescriptorSet2 [[buffer(2)]], constant spvDescriptorSetBuffer3&
// spvDescriptorSet3 [[buffer(3)]]) ^ program_source:9:32: note: attribute 'id'
// set location to 0, but minimum is 16
//    array<texture3d<float>, 1> kTextures3D [[id(0)]];
```

Bindings de-aliasing for MoltenVK

- **Option 1. Unique binding indices**

```
1  layout(set = 0, binding = 0) uniform texture2D kTextures2D[];
2  layout(set = 0, binding = 1) uniform texture3D kTextures3D[];
3  layout(set = 0, binding = 2) uniform textureCube kTexturesCube[];
4  layout(set = 0, binding = 3) uniform sampler kSamplers[];
5  layout(set = 0, binding = 4) uniform samplerShadow kSamplersShadow[];
```

  - Requires to assign several texture and sampler arrays to descriptor set -> increased size of descriptor pools

Bindings de-aliasing for MoltenVK

- **Option 2. Multiple set indices**

```
layout(set = 0, binding = 0) uniform texture2D kTextures2D[];
layout(set = 1, binding = 0) uniform texture3D kTextures3D[];
layout(set = 2, binding = 0) uniform textureCube kTexturesCube[];
layout(set = 0, binding = 1) uniform sampler kSamplers[];
layout(set = 1, binding = 1) uniform samplerShadow kSamplersShadow[];

// In C++
const VkDescriptorSet dsets[3] = {vkDSet_, vkDSet_, vkDSet_};
vkCmdBindDescriptorSets(cmdBuf, bindPoint, vkPipelineLayout_, 0,
                        (uint32_t)LVK_ARRAY_NUM_ELEMENTS(dsets), dsets, 0,
                        nullptr);
```

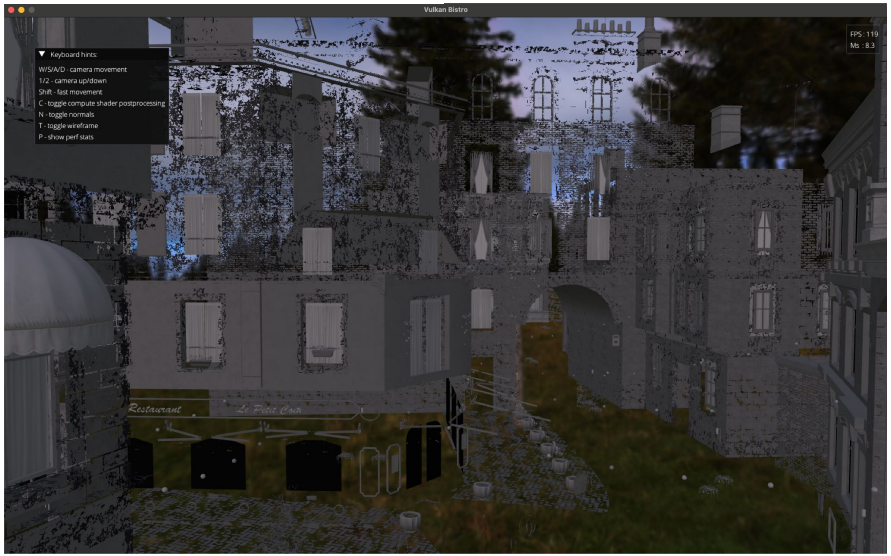
In MSL, we get:

```
struct spvDescriptorSetBuffer0 {
  array<texture2d<float>, 128> kTextures2D [[id(0)]];
  array<sampler, 16> kSamplers [[id(128)]];
};

struct spvDescriptorSetBuffer1 {
  array<texture3d<float>, 1> kTextures3D [[id(0)]];
  array<sampler, 16> kSamplersShadow [[id(128)]];
};

struct spvDescriptorSetBuffer2 {
  array<texturecube<float>, 128> kTexturesCube [[id(0)]];
};
```

Problem with Option 2: Multiple set indices

```
vec4 textureBindless2D(uint textureid, uint samplerid, vec2 uv) {
  return texture(sampler2D(kTextures2D[textureid], kSamplers[samplerid]), uv);
}

float textureBindless2DShadow(uint textureid, uint samplerid, vec3 uvw) {
  return texture(
      sampler2DShadow(kTextures2D[textureid], kSamplersShadow[samplerid]), uvw);
}
```

It worked until we used shadow maps ->

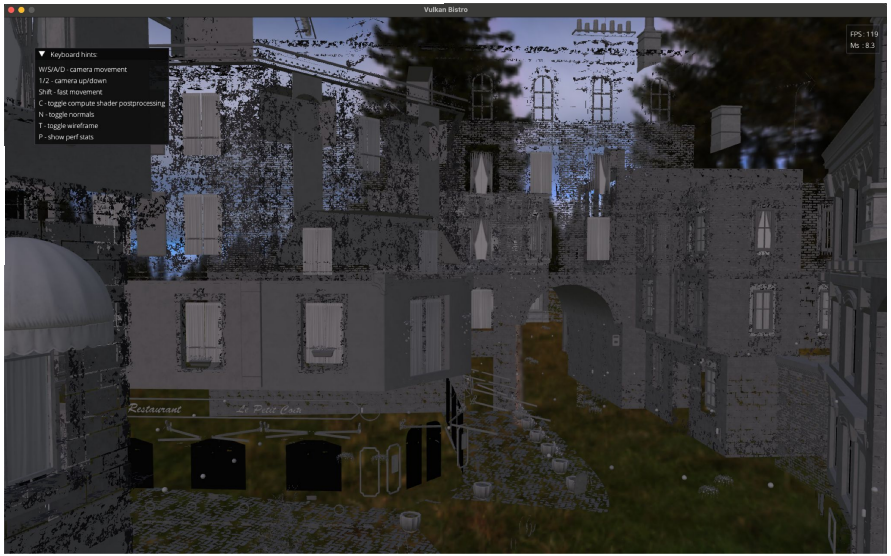


Problem with Option 2: Multiple set indices

```
vec4 textureBindless2D(uint textureid, uint samplerid, vec2 uv) {
  return texture(sampler2D(kTextures2D[textureid], kSamplers[samplerid]), uv);
}

float textureBindless2DShadow(uint textureid, uint samplerid, vec3 uvw) {
  return texture(
      sampler2DShadow(kTextures2D[textureid], kSamplersShadow[samplerid]), uvw);
}
```

In MSL:

```
struct spvDescriptorSetBuffer0 {
  array<depth2d<float>, 128> kTextures2D [[id(0)]];
  array<sampler, 16> kSamplers [[id(128)]];
};
```



Fixed Option 2: Multiple set indices

```
layout(set = 0, binding = 0) uniform texture2D kTextures2D[];
layout(set = 1, binding = 0) uniform texture3D kTextures3D[];
layout(set = 2, binding = 0) uniform textureCube kTexturesCube[];
layout(set = 3, binding = 0) uniform texture2D kTextures2DShadow[];
layout(set = 0, binding = 1) uniform sampler kSamplers[];
layout(set = 3, binding = 1) uniform samplerShadow kSamplersShadow[];

// In MSL
struct spvDescriptorSetBuffer0 {
  array<texture2d<float>, 16> kTextures2D [[id(0)]];
  array<sampler, 16> kSamplers [[id(16)]];
};

struct spvDescriptorSetBuffer1 {
  array<texture3d<float>, 1> kTextures3D [[id(0)]];
};

struct spvDescriptorSetBuffer2 {
  array<texturecube<float>, 16> kTexturesCube [[id(0)]];
};

struct spvDescriptorSetBuffer3 {
  array<texture2d<float>, 16> kTextures2DShadow [[id(0)]];
  array<sampler, 16> kSamplersShadow [[id(16)]];
};
```

“Bindless” in compute shaders

```
1  layout(local_size_x = 16, local_size_y = 16, local_size_z = 1) in;
2
3  layout(set = 0, binding = 2, rgba8) uniform image2D kTextures2Din[];
4  layout(set = 1, binding = 2, rgba8) uniform writeonly image2D kTextures2Dout[];
5
6  layout(push_constant) uniform constants {
7    uint tex;
8    uint width;
9    uint height;
10 }
11 pc;
12
13 void main() {
14   ivec2 pos = ivec2(gl_GlobalInvocationID.xy);
15
16   if (pos.x < pc.width && pos.y < pc.height) {
17     vec4 pixel = imageLoad(kTextures2Din[pc.tex], pos);
18     float luminance = dot(pixel, vec4(0.299, 0.587, 0.114, 0.0));
19     imageStore(kTextures2Dout[pc.tex], pos, vec4(vec3(luminance), 1.0));
20   }
21 }
```

https://github.com/corporateshark/lightweightvk/blob/main/samples/Tiny_MeshLarge.cpp

“Bindless” in compute shaders

```
1  layout(set = 0, binding = 2, rgba8) uniform readonly image2D kTextures2Din[];
2  layout(set = 1, binding = 2, rgba8) uniform writeonly image2D kTextures2Dout[];
3
4  // In MSL
5  struct spvDescriptorSetBuffer0 {
6    array<texture2d<float>, 256> _m0_pad [[id(0)]];
7    array<texture2d<float>, 256> _m256_pad [[id(256)]];
8    array<texture2d<float>, 256> kTextures2Din [[id(272)]];
9  };
10
11 struct spvDescriptorSetBuffer1 {
12   array<texture2d<float>, 256> _m0_pad [[id(0)]];
13   array<texture2d<float>, 256> _m256_pad [[id(256)]];
14   array<texture2d<float, access::write>, 256> kTextures2Dout [[id(272)]];
15 };
```

https://github.com/KhronosGroup/MoltenVK/issues/2106

“Bindless” in compute shaders

- **Not portable workaround**

  Binding 0 - VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL

  Binding 2 - VK_IMAGE_LAYOUT_GENERAL

```
layout(set = 0, binding = 0, rgba8) uniform image2D kTextures2Din[];
layout(set = 1, binding = 0, rgba8) uniform writeonly image2D kTextures2Dout[];

// In MSL
spvDescriptorSetBuffer0 {
  array<texture2d<float>, 64> kTextures2Din [[id(0)]];
};

struct spvDescriptorSetBuffer1 {
  array<texture2d<float, access::write>, 64> kTextures2Dout [[id(0)]];
};
```

or

```
layout(set = 0, binding = 0, rgba8) uniform image2D kTextures2DInOut[];

// In MSL
struct spvDescriptorSetBuffer0 {
  array<texture2d<float, access::read_write>, 256> kTextures2DInOut [[id(0)]];
};
```

VK_KHR_maintenance4 for compute shaders

"Add support for the SPIR-V 1.2 `LocalSizeId` execution mode, which can be used as an alternative to `LocalSize` to specify the local workgroup size with specialization constants."

```
1  layout(constant_id = 0) const uint kLocalSize = 16;
2
3  layout(local_size_x_id = 0, local_size_y_id = 0, local_size_z = 1) in;
4
5  // vkCreateComputePipelines(): pCreateInfos[0].stage SPIR-V OpExecutionMode
6  // LocalSizeId is used but maintenance4 extension is not enabled and used Vulkan
7  // api version is 1.2 or less. The Vulkan spec states: If Execution Mode
8  // LocalSizeId is used, maintenance4 must be enabled
9  // (https://www.khronos.org/registry/vulkan/specs/1.3-extensions/html/vkspec.html#VUID-RuntimeSpirv-LocalSizeId-06434)
```

Not supported in MoltenVK at the moment

https://github.com/KhronosGroup/MoltenVK/issues/1930

Atomics support in SPIR-V Cross

```
#version 460
#extension GL_EXT_buffer_reference : require
#extension GL_EXT_nonuniform_qualifier : require

layout(set = 0, binding = 0, r32ui) uniform uimage2D kTextures2D[16];

layout(push_constant) uniform PushConst { uint texture0; }
pc;

void main() {
  uint i = imageAtomicAdd(kTextures2D[pc.texture0], ivec2(0, 0), 1);
}
```

```
constant uint spvLinearTextureAlignmentOverride [[function_constant(65535)]];
constant uint spvLinearTextureAlignment =
    is_function_constant_defined(spvLinearTextureAlignmentOverride)
        ? spvLinearTextureAlignmentOverride
        : 4;
#define spvImage2DAtomicCoord(tc, tex)                        \
  (((((tex).get_width() + spvLinearTextureAlignment / 4 - 1) & \
     ~(spvLinearTextureAlignment / 4 - 1)) *                  \
    (tc).y) +                                                 \
   (tc).x)

struct PushConst {
  uint texture0;
};

fragment void main0(constant PushConst& pc [[buffer(16)]],
                    array<texture2d<uint>, 16> kTextures2D [[texture(0)]],
                    device atomic_uint* kTextures2D_atomic [[buffer(0)]]) {
  uint _30 = atomic_fetch_add_explicit(
      (device atomic_uint*)&kTextures2D[pc.texture0] _atomic // Syntax error.
          [spvImage2DAtomicCoord(int2(0), kTextures2D[pc.texture0])],
      1u, memory_order_relaxed);
  uint i = _30;
}
```

No proper atomics for images in SPIR-V Cross until
https://github.com/KhronosGroup/SPIRV-Cross/pull/2235

Requires Metal 3.1

Specialization constants

```
layout(constant_id = 0) const uint kColorConstantIndex = 0;
layout(constant_id = 1) const uint kColorConstantsSize = 1;

layout(local_size_x = 16, local_size_y = 16, local_size_z = 1) in;

layout(set = 0, binding = 0, rgba8) uniform image2D kTextures2DInOut[];

layout(push_constant) uniform constants {
  uint tex;
  uint width;
  uint height;
}
pc;

void main() {
  ivec2 pos = ivec2(gl_GlobalInvocationID.xy);

  const vec4[kColorConstantsSize] kColorConstants =
      vec4[kColorConstantsSize](vec4(0.299, 0.587, 0.114, 0.0));

  if (pos.x < pc.width && pos.y < pc.height) {
    vec4 pixel = imageLoad(kTextures2DInOut[pc.tex], pos);
    float luminance = dot(pixel, kColorConstants[kColorConstantIndex]);
    imageStore(kTextures2DInOut[pc.tex], pos, vec4(vec3(luminance), 1.0));
  }
}
```

Specialization constants

In Metal `function_constant(...)` can't be used as an array size.

```
#ifndef SPIRV_CROSS_CONSTANT_ID_1
#define SPIRV_CROSS_CONSTANT_ID_1 1u
#endif
constant uint kColorConstantsSize = SPIRV_CROSS_CONSTANT_ID_1;

constant uint kColorConstantIndex_tmp [[function_constant(0)]];
constant uint kColorConstantIndex =
    is_function_constant_defined(kColorConstantIndex_tmp)
        ? kColorConstantIndex_tmp
        : 0u;

```

Defining `SPIRV_CROSS_CONSTANT`s is currently not supported in MoltenVK

https://github.com/KhronosGroup/MoltenVK/issues/1423

LightweightVK as a sandbox

Deeply refactored fork of IGL* which is designed to run on top of Vulkan 1.3.

https://github.com/corporateshark/lightweightvk (MIT license)

* Intermediate Graphics Library (IGL) is a cross-platform library that commands the GPU. It encapsulates common GPU functionality with a low-level cross-platform interface.
https://github.com/facebook/igl

More details: "**Vulkanised 2023: Use "bindless" to quickly port "bindful" OpenGL apps to Vulkan**" by Sergey Kosarevsky
https://www.vulkan.org/user/pages/09.events/vulkanised-2023/vulkanised_2023_use_bindless_to_quickly_port_bindful_opengl_apps_to_vulkan.pdf

Summary

- Dynamic rendering just works;
- “Bindless” textures and buffers can be used in MoltenVK but it requires extra care to achieve cross-platform work;
- Compute shaders are possible, but atomics work well only with buffers at the moment. Atomics for textures are available in SPIR-V Cross now;
- Be careful with using specialization constants. MoltenVK has implicit behaviour for array sizes.
- Be careful with shader validation, possibly you need to turn it off.

Thanks

- **Sergey Kosarevsky** for LightweightVK and collaboration on it
- **Bill Hollings** and **Chip Davis** for making fixes and improvements in MoltenVK
- **Hans-Kristian Arntzen** for fixes and improvements in SPIV-Cross

Q & A