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](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](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](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](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+)**
  
  \[\text{MVK\_CONFIG\_USE\_METAL\_ARGUMENT\_BUFFERS} = 1\ (or\ 2)\]

- **Vulkan 1.3 is almost supported in MoltenVK**
  
  [link](https://github.com/KhronosGroup/MoltenVK/issues/1930)

- **No VK\_KHR\_draw\_indirect\_count support**
  
  [link](https://github.com/KhronosGroup/MoltenVK/issues/168)
MoltenVK Initialization

- Enable Tier 2 argument buffers

*After adding VK_EXT_layer_settings support*

```c
#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](https://github.com/corporateshark/lightweightvk/pull/21)
MoltenVK Initialization

- Disable shader validation

```c
#include (__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_SHADER_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/987f91e28bd9b4e783ecee9db66f0697b1ec58ccc/lvk/vulkan/VulkanClasses.cpp#L4214
MoltenVK Initialization

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

```cpp
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

**VkDescriptorSetLayout + VkDescriptorPool**
- VK_DESCRIPTOR_TYPE_SAMPED_IMAGE [maxTextures]
- VK_DESCRIPTOR_TYPE_SAMPLER [maxSamplers]
- VK_DESCRIPTOR_TYPE_STORAGE_IMAGE [maxTextures]

- **VkBuffer**
  - vkGetBufferDeviceAddress()

- **VkDeviceAddress**
  - vkCmdPushConstants() / buffer data

- **VkImage**
  - VkSampler
  - VkDescriptorSet
    - vkUpdateDescriptorSets()
    - vkAllocateDescriptorSets()
    - vkCmdBindDescriptorSets()

```c
layout(std430, buffer_reference) readonly buffer VertexBuffer {
    Vertex vertices[];
};
layout(std430, buffer_reference) readonly buffer UniformBuffer {
    mat4 viewPro;
};
layout(std430, buffer_reference) readonly buffer AddressBuffer {
    VertexBuffer vb;
    UniformBuffer ub;
};
layout(push_constant) uniform constants {
    AddressBuffer ab;
} pc;

type Bindless2D(uint textuereid, uint samplerid, vec2 uv) {
    return texture(sampler2D(kTextures2D[textuereid],
                          kSamplers[samplerid]), uv);
}
Bindings aliasing

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

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

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

```cpp
1  struct spvDescriptorSetBuffer0 {
2      array<texture2d<float>, 16> kTextures2D [[id(0)]];
3      array<texture3d<float>, 1> kTextures3D [[id(0)]];
4      // <...>
5  };
6
7  // [mvk-error] VK_ERROR_INITIALIZATION_FAILED: Shader library compile failed
9  // to 'spvDescriptorSet0' fragment void main0(const spvDescriptorSetBuffer0&
10  // spvDescriptorSet0 [[buffer(0)]], constant spvDescriptorSetBuffer2&
11  // spvDescriptorSet2 [[buffer(2)]], constant spvDescriptorSetBuffer3&
12  // spvDescriptorSet3 [[buffer(3)]]) ^ program_source:9:32: note: attribute 'id'
13  // set location to 0, but minimum is 16
14  // 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**

```cpp
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[];
```

```cpp
// 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:

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

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

```cpp
struct spvDescriptorSetBuffer2 {
  array<texturecube<float>, 128> kTexturesCube [[id(0)]];  
};
```
Problem with Option 2: Multiple set indices

```cpp
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

In MSL:

```c
struct spvDescriptorSetBuffer0 {
  array<depth2d<float>, 128> kTextures2D [[id(0)]];  // Changed from 16
  array<sampler, 16> kSamplers [[id(128)]];         // Changed from 8
};
```
Fixed Option 2: Multiple set indices

```cpp
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

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

layout(set = 0, binding = 2, rgba8) uniform image2D kTextures2DIn;
layout(set = 1, binding = 2, rgba8) uniform writeonly image2D kTextures2DOut;

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

pc;

void main() {
    ivec2 pos = ivec2(gl_GlobalInvocationID.xy);
    if (pos.x < pc.width && pos.y < pc.height) {
        vec4 pixel = imageLoad(kTextures2DIn[pc.tex], pos);
        float luminance = dot(pixel, vec4(0.299, 0.587, 0.114, 0.0));
        imageStore(kTextures2DOut[pc.tex], pos, vec4(vec3(luminance), 1.0));
    }
}

https://github.com/corporateshark/lightweightvk/blob/main/samples/Tiny_MeshLarge.cpp
```
“Bindless” in compute shaders

```cpp
layout(set = 0, binding = 2, rgba8) uniform readonly image2D kTextures2Din[];
layout(set = 1, binding = 2, rgba8) uniform writeonly image2D kTextures2Dout[];

// In MSL

struct spvDescriptorSetBuffer0 {
    array<texture2d<float>, 256> _m0_pad [[id(0)]]; // Example array
    array<texture2d<float>, 256> _m256_pad [[id(256)]]; // Example array
    array<texture2d<float>, 256> kTextures2Din [[id(272)]]; // Example array
};

struct spvDescriptorSetBuffer1 {
    array<texture2d<float>, 256> _m0_pad [[id(0)]]; // Example array
    array<texture2d<float>, 256> _m256_pad [[id(256)]]; // Example array
    array<texture2d<float>, access::write, 256> kTextures2Dout [[id(272)]]; // Example array
};

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

```cpp
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)]];  
};

// In MSL
spvDescriptorSetBuffer1 {
  array<texture2d<float, access::write>, 64> kTextures2DOut [[id(0)]];  
};

or

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

// In MSL
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.”

```cpp
1    layout(constant_id = 0) const uint kLocalSize = 16;
2    layout(local_size_x_id = 0, local_size_y_id = 0, local_size_z = 1) in;

// vkCreateComputePipelines(): pCreateInfos[0].stage SPIR-V OpExecutionMode
// LocalSizeId is used but maintenance4 extension is not enabled and used Vulkan
// api version is 1.2 or less. The Vulkan spec states: If Execution Mode
// LocalSizeId is used, maintenance4 must be enabled
// (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
#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 mainO {
  uint i = imageAtomicAdd(kTextures2D[pc.texture0], ivec2(0, 0), i);
}
```

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

Requires Metal 3.1
Specialization constants

```c
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.

```c
#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_CONSTANTs` 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

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