Use "bindless" to quickly port "bindful" OpenGL apps to Vulkan

Sergey Kosarevsky
Software Engineer (Vulkan Tech Lead), Meta
Sergey Kosarevsky

Former Ubisoft rendering lead

Co-authored 3 books:

- Android NDK Game Development Cookbook (2013)
- Mastering Android NDK (2015)
- 3D Graphics Rendering Cookbook (2021)
Sergey Kosarevsky

Old "bindful" API - **IGL** (Intermediate Graphics Library)

- Cross-platform, graphics API agnostic rendering library
- Used in 20+ Meta’s projects (internal and external)
- Windows, macOS, iOS, Android, Linux
- OpenGL ES 2, OpenGL ES 3, OpenGL 4.1, Metal
- Add (quickly) **Vulkan 1.3**

*With a possibility to backport to Vulkan 1.1 with some extensions*
Modelled after the Metal API and has a very strong OpenGL ES 2 influence:

```cpp
class IRenderCommandEncoder {
    public:
        ...
        virtual void bindRenderPipelineState(IRenderPipelineState& pipelineState) = 0;
        virtual void bindBuffer(int index, IBuffer& buffer, size_t bufferOffset) = 0;
        virtual void bindSamplerState(int index, ISamplerState& samplerState) = 0;
        virtual void bindTexture(int index, ITexture& texture) = 0;
        virtual void drawIndexed(PrimitiveType primitiveType, uint32_t indexCount,
                                  IndexFormat indexFormat, IBuffer& indexBuffer, size_t indexBufferOffset) = 0;
        ...
};
```
Vulkan 1.3 features to the rescue!

Features that were optional in earlier versions of Vulkan are now required:

- VK_KHR_buffer_device_address
- VK_EXT_descriptor_indexing
- Sanity checks (omitted in my slides)
- Handle special types of buffers
- Delegate to the `ResourcesBinder` class

```cpp
void RenderCommandEncoder::bindBuffer(int index, IBuffer& buffer, size_t bufferOffset) {
    auto* buf = static_cast<igl::vulkan::Buffer*>(buffer.get());

    VkBuffer vkBuf = buf->getVkBuffer();

    if (buf->getBufferType() & BufferTypeBits::Vertex) {
        const VkDeviceSize offset = bufferOffset;
        vkCmdBindVertexBuffers(cmdBuffer_, index, 1, &vkBuf, &offset);
    } else {
        binder_.bindBuffer(index, buffer, bufferOffset);
    }
}
```
void RenderCommandEncoder::bindSamplerState(int index, ISamplerState &samplerState) {
    binder_.bindSamplerState(index, samplerState);
}

void RenderCommandEncoder::bindTexture(int index, ITexture &texture) {
    binder_.bindTexture(index, texture);
}

Upload the bindings to the GPU before a draw call:

void RenderCommandEncoder::drawIndexed(PrimitiveType primitiveType, uint32_t indexCount,
                                        IndexFormat indexFormat, IBuffer &indexBuffer,
                                        size_t indexBufferOffset) {
    binder_.updateDynamicUniforms(
        cmdBuffer_, VK_PIPELINE_BIND_POINT_GRAPHICS, pipelineLayoutGraphics_);
    dynamicState_.setTopology(primitiveType);
    dynamicState_.bindPipeline();
    auto *buf = static_cast<igl::vulkan::Buffer*>(indexBuffer.get());
    vkCmdBindIndexBuffer(cmdBuffer_, buf->getVkBuffer(), indexBufferOffset, indexFormat);
    vkCmdDrawIndexed(cmdBuffer_, indexCount, 1, 0, 0, 0);
}
Under the hood – IGL/Vulkan – Low-level C++

**ResourcesBinder** prepares the binding data for shaders:

```cpp
class ResourcesBinder {
    struct Slot {
        uint32_t texture = 0;
        uint32_t sampler = 0;
        VkDeviceAddress buffer = 0;
    };
    struct Bindings {
        Slot slots[kMaxBindingSlots] = {};
    } bindings_;

    public:
    void bindBuffer(int index, IBuffer& buffer, size_t bufferOffset) {
        auto* buf = static_cast<igl::vulkan::Buffer*>(buffer.get());
        bindings_.slots[index].buffer = buf->getVkDeviceAddress() + bufferOffset;
    }
    void bindSamplerState(int index, ISamplerState& samplerState) {
        auto* sampler = static_cast<igl::vulkan::SamplerState*>(samplerState.get());
        bindings_.slots[index].sampler = sampler ? sampler->getSamplerId() : 0;
    }
    void bindTexture(int index, ITexture& texture) {
        auto* tex = static_cast<igl::vulkan::Texture*>(texture.get());
        bindings_.slots[index].texture = tex ? tex->getTexturized() : 0;
    }
    void updateDynamicUniforms(VkCommandBuffer, VkPipelineBindPoint, VkPipelineLayout);
    ...}
```
Under the hood – IGL/Vulkan – Low-level C++

Manage 2 descriptor set layouts:

```cpp
// all images/samplers
{
  constexpr uint32_t kNumBindings = 3;
  const VkDescriptorSetLayoutBinding[kNumBindings] bindings = {
    ivkGetDSLBinding(kBinding_Texture, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, config_.maxTextures),
    ivkGetDSLBinding(kBinding_Sampler, VK_DESCRIPTOR_TYPE_SAMPLER, config_.maxSamplers),
    ivkGetDSLBinding(kBinding_StorageImage, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, config_.maxTextures),
  };
  const uint32_t flags = VK_DESCRIPTOR_BINDING_UPDATE_AFTER_BIND_BIT |
                         VK_DESCRIPTOR_BINDING_UPDATE_UNUSED_WHILE_PENDING_BIT |
                         VK_DESCRIPTOR_BINDING_PARTIALLY_BOUND_BIT;
  const VkDescriptorBindingFlags[kNumBindings] bindingFlags = { flags, flags, flags };
  dslBindless_ = vulkan::VulkanDescriptorSetLayout(device, kNumBindings, bindings, bindingFlags);
}

// a dynamic uniform buffer
{
  const VkDescriptorSetLayoutBinding bindings =
    ivkGetDSLBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, 1);
  const VkDescriptorBindingFlags bindingFlags = 0;
  dslDynamicUniformBuffer_ = vulkan::VulkanDescriptorSetLayout(device, 1, &bindings, &bindingFlags);
}
```
Dynamic Uniform Buffer (DUB):
Store bindings in a single uniform buffer object and offsetting into multiple dynamic uniform buffers.

DUB 0  64Kb    DUB 1  64Kb    DUB 2  64Kb
Update and bind dynamic uniform buffer:

```cpp
void ResourceBinder::updateDynamicUniforms(VkCommandBuffer cmdBuf,
                                          VkPipelineBindPoint bindPoint,
                                          VkPipelineLayout pipelineLayout)
{
    if (DUBs_.empty() || !DUBs_[currentDUB_].canFit(sizeof(bindings_))) {
        currentDUB_++;
    }
    if (currentDUB_ > DUBs_.size()) {
        allocateDynamicUniformsBuffer();
    }
    auto& buf = DUBs_[currentDUB_];
    memcpy(buf.buffer_->getMappedPtr() + buf.offset_, &bindings_, sizeof(bindings_));
    buf.buffer_->flushMappedMemory(buf.offset_, sizeof(bindings_));
    vkCmdBindDescriptorSets(cmdBuf,
                             bindPoint,
                             pipelineLayout,
                             kBindPoint_DynamicUniforms,
                             1, &buf.ds_,
                             1, &buf.offset_);
    buf.offset_ += (sizeof(bindings_) + kMinAlignment - 1) & ~kMinAlignment - 1;
}
```
Under the hood – IGL/Vulkan – Shaders

- SPIR-V: custom shading language compiler SparkSL
- GLSL: injecting helper functions before compilation

```cpp
#extension GL_EXT_nonuniform_qualifier : require
#extension GL_EXT_buffer_reference : require
#extension GL_EXT_buffer_reference_uvec2 : require

layout (set = 0, binding = 0) uniform texture1D ktextures1D[];
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[];

layout (set = 1, binding = 0) uniform Bindings {
    uvec4 slots [ kMaxBindingSlots ]; // x : textureId; y : samplerId; zw : uvec2 buffer_reference
}

vec2 getBuffer ( uint slot ) {
    return bindings . slots [ slot ] . zw;
}

vec4 textureSample2D ( uint slotTexture , uint slotSampler , vec2 uv ) {
    uint idxTex = bindings . slots [ slotTexture ] . x;
    uint idxSmp = bindings . slots [ slotTexture ] . y;
    return texture ( sampler2D ( kTextures2D [ nonuniformEXT ( idxTex ) ,
                                kSamplers [ nonuniformEXT ( idxSmp ) ] ), uv );
}
```
Bonus – MoltenVK compatibility

Metal Shading Language can not support aliased resources in our GLSL:

```glsl
layout (set = 0, binding = 0) uniform texture1D ktextures1D [];
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 [];
```

Some extra C++ fiddling is required do de-alias resources.
Questions?

Thank you!

Sergey Kosarevsky

@corporateshark
sk@linderdaum.com