Vulkanised 2026
The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

vkDuck - Taking the Pain Out of Vulkan: A GUI for Building Renderers

Lamies Abbas, University of Vienna

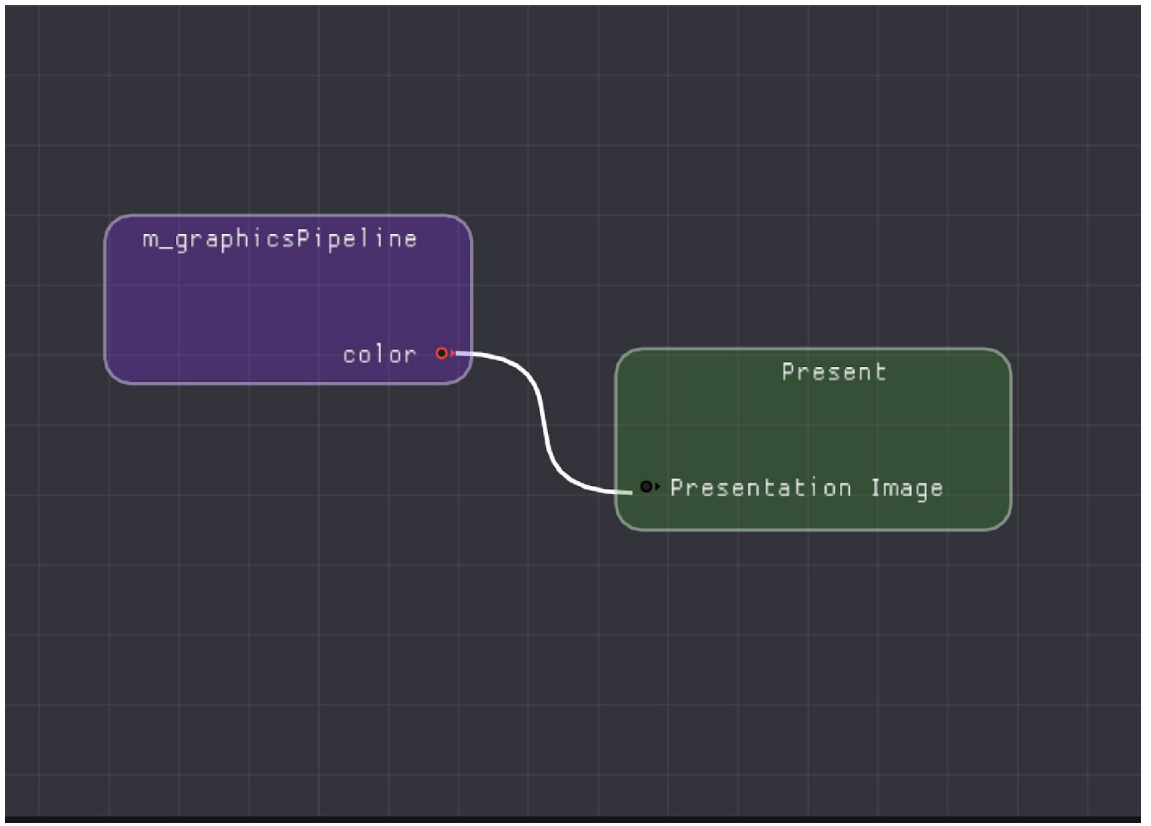
I Just Want to Render a Triangle

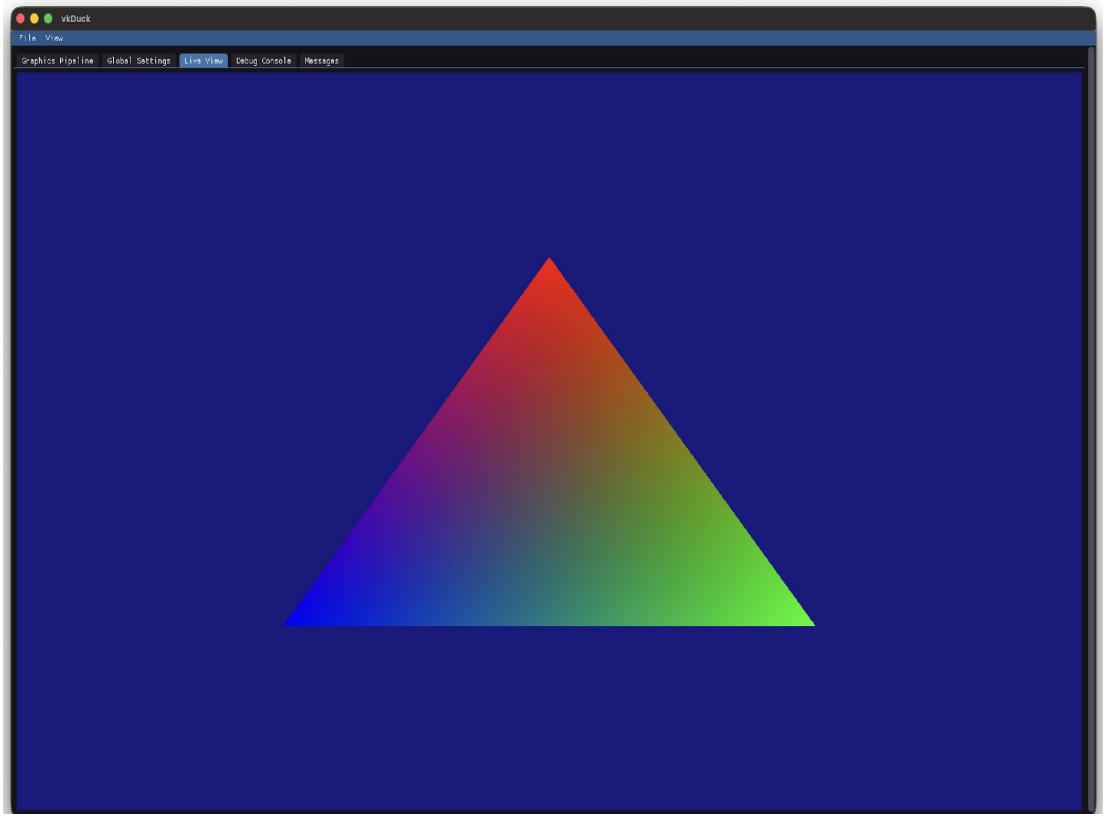
```
struct VSOut {
    float4 position : SV_Position;
    float4 color    : COLOR;
};

struct FSOut {
    float4 color : SV_Target0;
};

[shader("vertex")]
VSOut main(uint vertexId : SV_VertexID) {
    VSOut output;
    // Hardcoded triangle vertices: Top, Right, Left
    float2 positions[3] = { { 0, -0.5 }, { 0.5, 0.5 }, { -0.5, 0.5 } };
    float3 colors[3] = { { 1, 0, 0 }, { 0, 1, 0 }, { 0, 0, 1 } };

    output.position = float4(positions[vertexId], 0.0, 1.0);
    output.color = float4(colors[vertexId], 1.0);
    return output;
}

[shader("fragment")]
FSOut main(float4 color : COLOR) {
    FSOut output;
    output.color = color;
    return output;
}
```

- What Vulkan Needs
  - VkInstance
  - VkPhysicalDevice/VkDevice
  - VkSwapchain
  - VkRenderpass
  - VkPipeline & VkPipelineLayout
  - VkFramebuffer
  - VkCommandPool/VkCommandBuffer
  - …and more

The Alternative

What if you could just… build it visually?





🦆 Who am I?

- Hi, I'm Lamies Abbas 👋🏻
- TA in a computer graphics course
- Writing master's thesis in the EDEN research group with professor Helmut Hlavacs
- Loves gaming and creating games with Vulkan:
  - -¸` ¸´- QuackBlast: https://github.com/fini03/QuackBlast
  - ⋆.°⋆.° ¸´- Quackie-in-Space: https://github.com/fini03/Quackie-in-Space
- Check out my page: https://quackie.at/

Where It Started

- Real-Time Graphics Course
  - Task: Build a game from scratch with Vulkan
- Rendering triangle takes 1k lines of code
- Even for a simple game, the amount of Vulkan setup was overwhelming

By the time you're done with the setup, your motivation might already be gone…

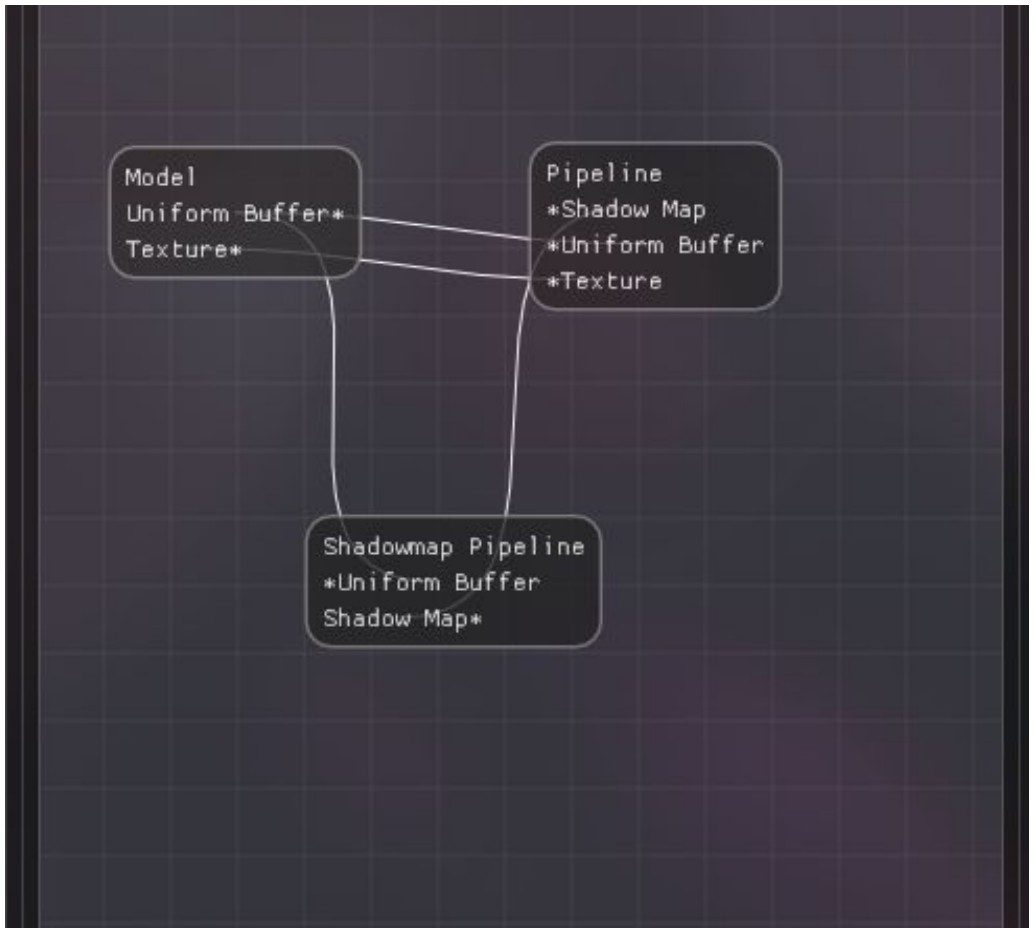
Idea

- What if I could build my game using a visual editor instead?
- I ended up doing everything manually in the course because the editor was a lot of work…
- Inspiration: https://github.com/electronicarts/gigi

So why am I working on this?

Because I got tired of writing the same boilerplate every time I wanted to try something new 😔

First Attempt: Pipeline Settings



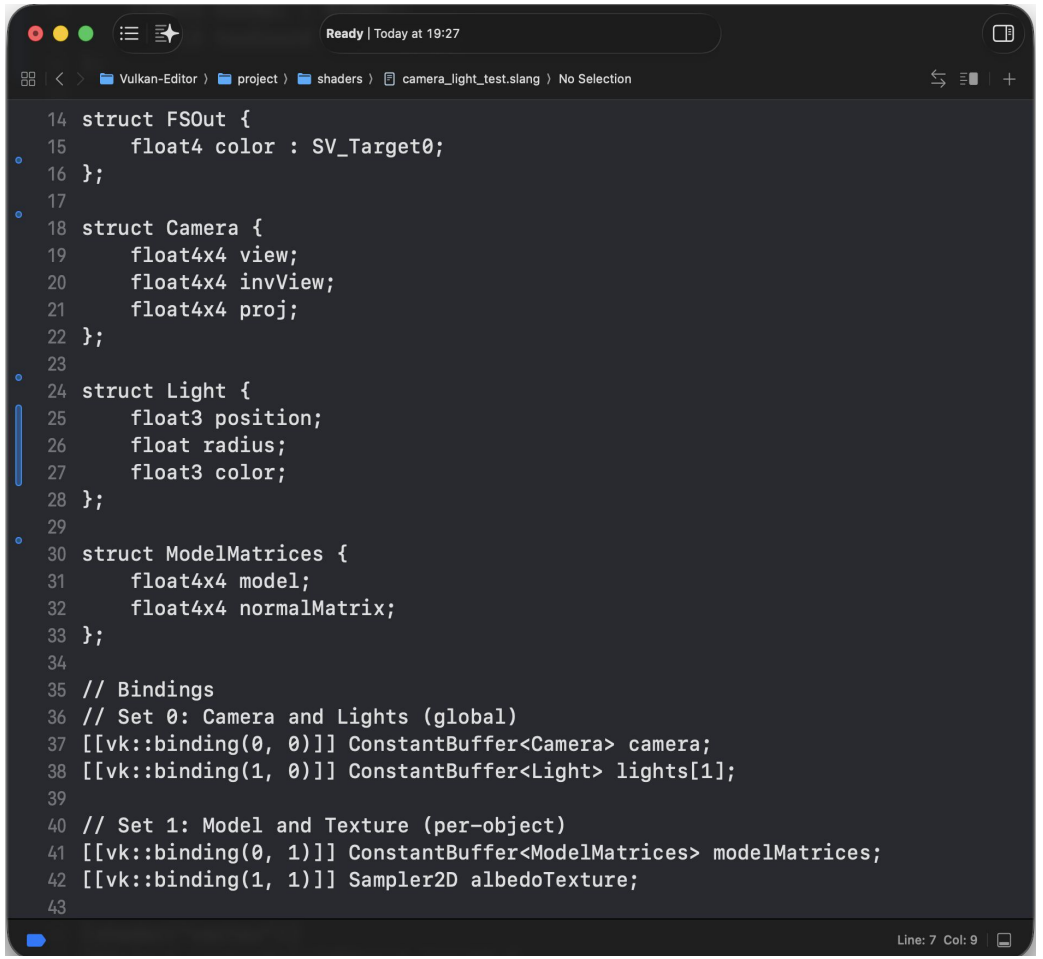
But this wasn't enough…

- Each node type had fixed input/outputs
- New shader —> change editor code
- No flexibility
- Couldn't adapt to user shaders

shader-slang

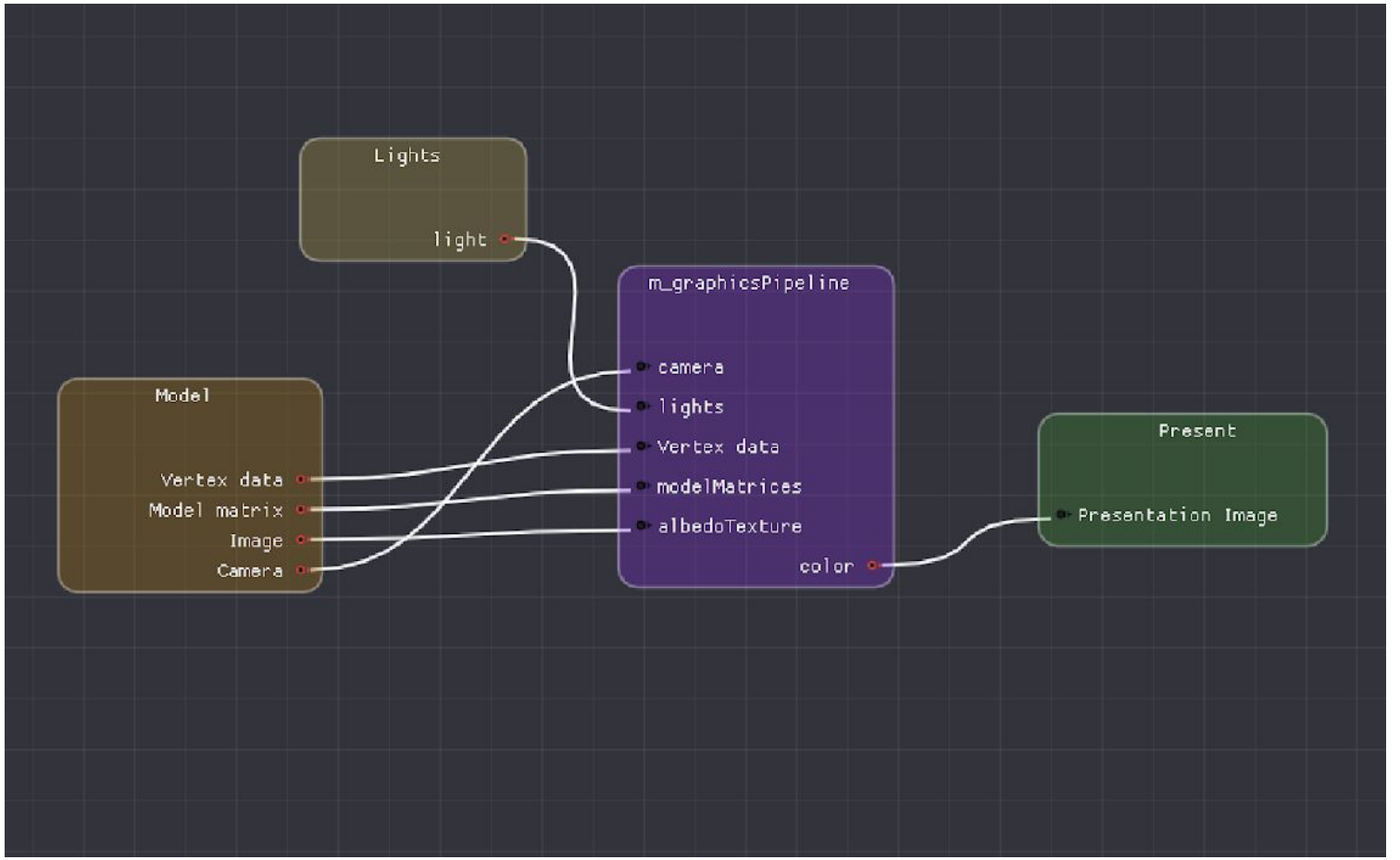
- Shader language with reflection capabilities
- Can parse shader code and extract metadata
- Query inputs, outputs, uniforms, bindings, and more…
- Get type information (float, vec3, mat4, etc.)
- Shader becomes the main source for the editor

Slang Reflection

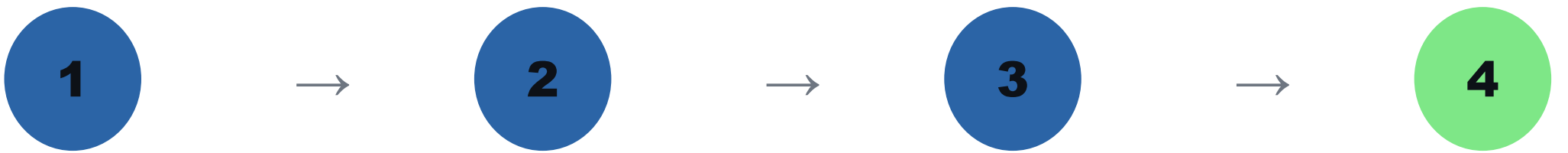



```
+============================================================+
|                  SHADER REFLECTION RESULTS                 |
+============================================================+
| Entry Point: fragmentMain
+------------------------------------------------------------+

+-- SHADER OUTPUTS ---------------------------------------+
| Name             | Semantic         | Type               |
+------------------+------------------+--------------------+
| color            | SV_TARGET        | vector             |
+------------------+------------------+--------------------+

+-- RESOURCE BINDINGS ------------------------------------+
| Resource             | Set | Bind | Kind             | Type               |
+----------------------+-----+------+------------------+--------------------+
| camera               | 0   | 0    | ConstantBuffer   | Camera             |
|   Members:
|     +-- view : float4x4  (offset: 0)
|     +-- invView : float4x4  (offset: 64)
|     +-- proj : float4x4  (offset: 128)
| lights               | 0   | 1    | ConstantBuffer   | Light              |
|   Members:
|     +-- position : float3  (offset: 0)
|     +-- radius : float  (offset: 12)
|     +-- color : float3  (offset: 16)
| modelMatrices        | 1   | 0    | ConstantBuffer   | ModelMatrices      |
|   Members:
|     +-- model : float4x4  (offset: 0)
|     +-- normalMatrix : float4x4  (offset: 64)
| albedoTexture        | 1   | 1    | Resource         | _Texture           |
+----------------------+-----+------+------------------+--------------------+

+-- DETECTED LIGHT STRUCTS -----------------------------------+
| lights : Light[1]
|   +-- position : float3
|   +-- radius : float
|   +-- color : float3
+------------------------------------------------------------+

+-- DETECTED CAMERA STRUCTS ----------------------------------+
| camera : Camera
|   +-- view : float4x4
|   +-- invView : float4x4
|   +-- proj : float4x4
+------------------------------------------------------------+
```

Draw Graph from Slang Reflection



The Workflow



Write Shaders

Write Slang shaders

Build the Graph

Define passes and connections

Preview

See your renderer running live

Generate

Export project

From Graph to Code: Primitives

- Every Vulkan concept is a "primitive" (node)
- Each primitive has:
  - create()
  - destroy()
  - recordCommands()
  - generateCreate()
- Can be used for live view & code generation

Live View

- Primitives call Vulkan API directly
- create() allocates GPU resources
- recordCommands() builds command buffer
- Result rendered to ImGui texture

Code generation

Remember all that Vulkan setup?

```
std::vector<VkDescriptorSetLayoutBinding> descriptorSet_2_layoutBindings = {{
    {
        .binding = 0,
        .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
    },
    {
        .binding = 1,
        .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
    },
    {
        .binding = 2,
        .descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
    },
    {
        .binding = 3,
        .descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
    },
    {
        .binding = 4,
        .descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER,
        .descriptorCount = 1,
        .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
    }
}};

VkDescriptorSetLayoutCreateInfo descriptorSet_2_layoutInfo{
    .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,
    .bindingCount = static_cast<uint32_t>(descriptorSet_2_layoutBindings.size()),
    .pBindings = descriptorSet_2_layoutBindings.data()
};
```

- vkDuck writes it for you
- Initialization, descriptor sets, render loop and more…
- No more writing boilerplate code
- Output is a compilable C++ project

Primitive Types

VertexData Model geometry	**Image** Textures & attachments	**Camera** View/projection matrices	**Light** Light parameters
UniformBuffer Shader uniforms	**DescriptorSet** Resource bindings	**Shader** Vertex/fragment code	**Pipeline** Full graphics pipeline
RenderPass Render target config	**Attachment** Color/depth targets	**Present** Output to live view	**Array** Group multiple items

Node Types

Model

Model geometry

Camera

View/projection matrices

Light

Light parameters

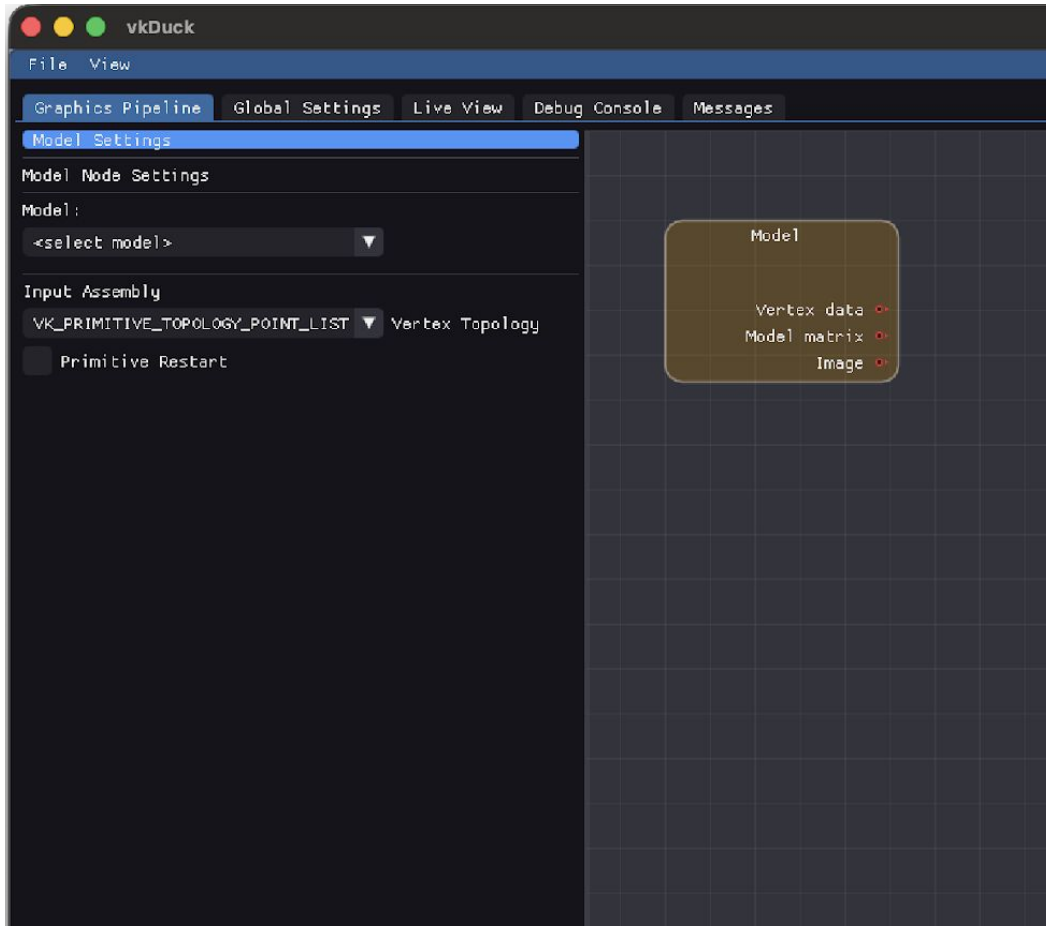
Pipeline

Full graphics pipeline

Present

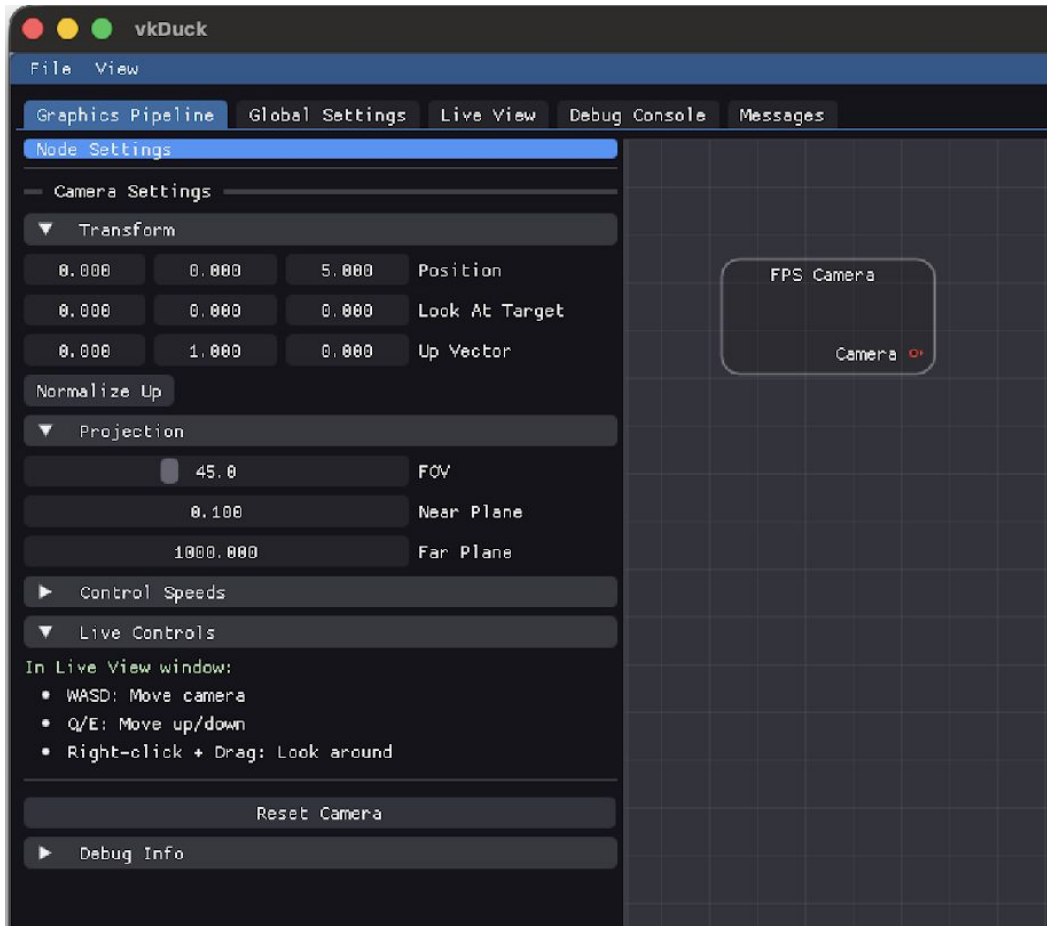
Output to live view

Node: Model



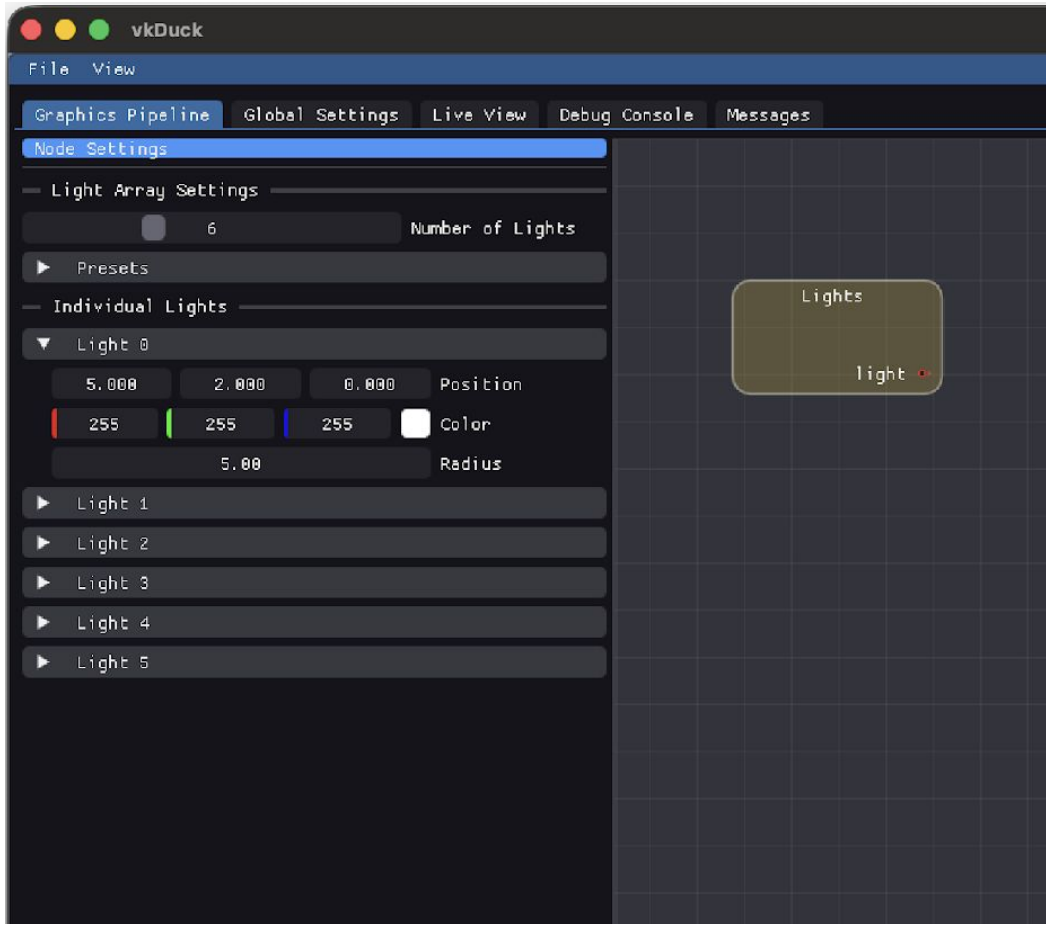
- Loads and manages 3D geometry
- Parses glTF and extracts vertex data (and cameras)
- Creates VkBuffer for vertices and indices
- Stages data to GPU memory
- ~ 1500-2000 LoC

Node: Camera



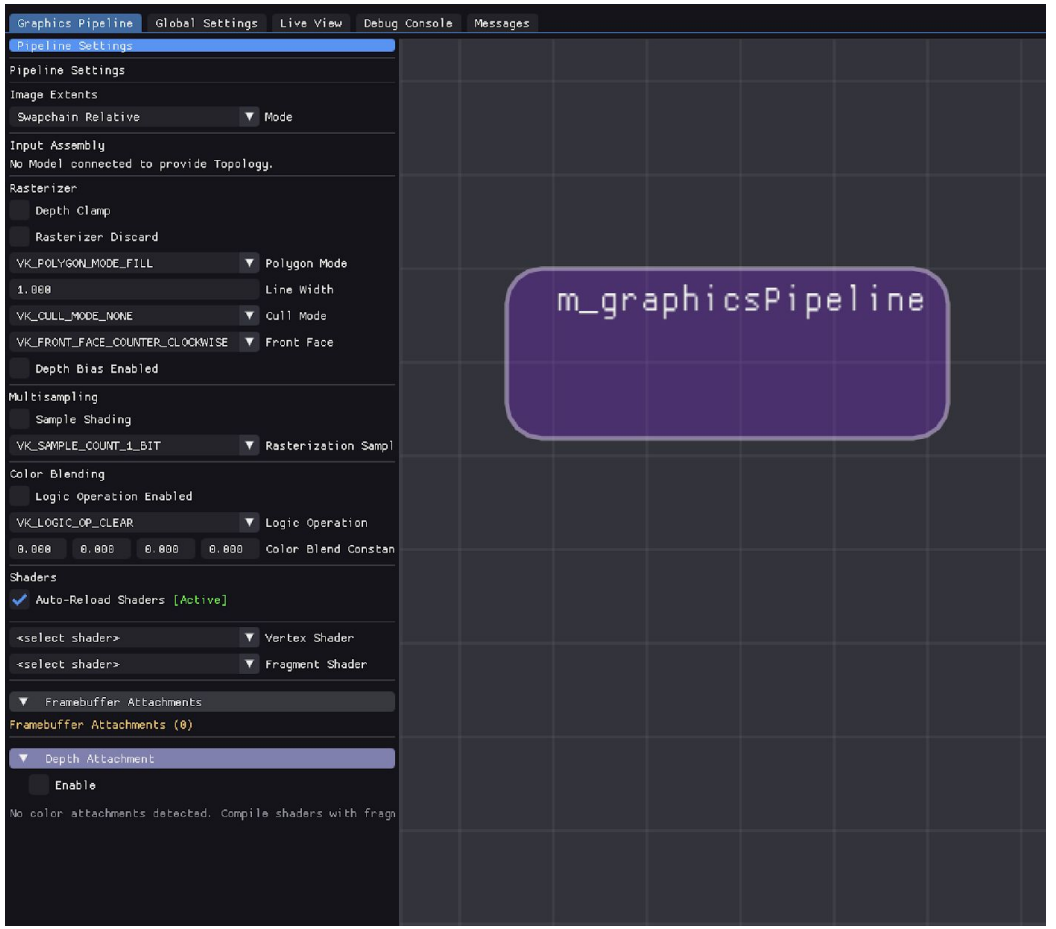
- Types: Fixed, FPS, Orbital
- Outputs view & projection matrices
- Interactive in live view
- ~ 800-1000 LoC

Node: Light



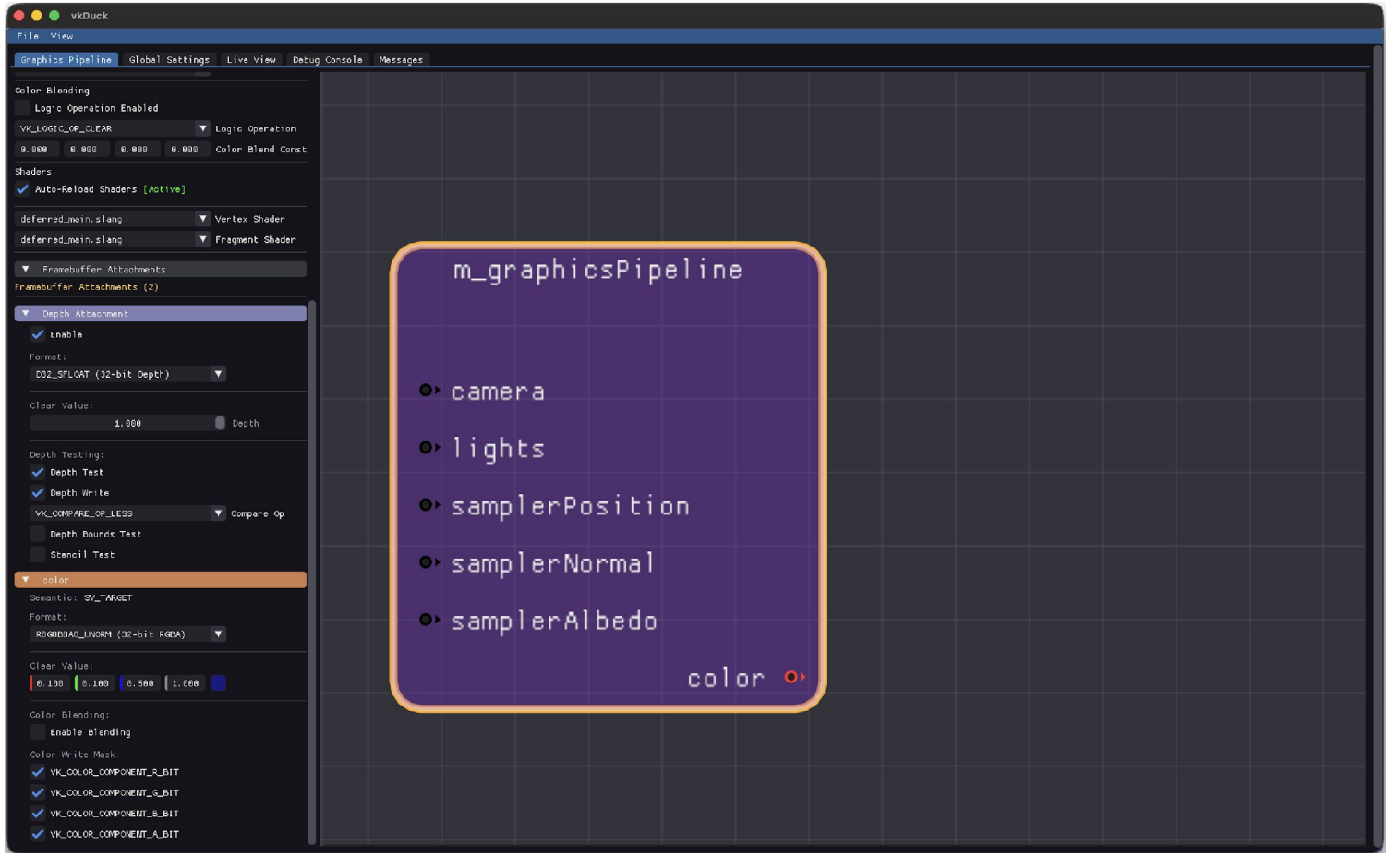
- Position in world space
- Color (RGB)
- Radius / intensity
- Can have arrays of lights
- ~ 500-600 LoC

Node: Pipeline

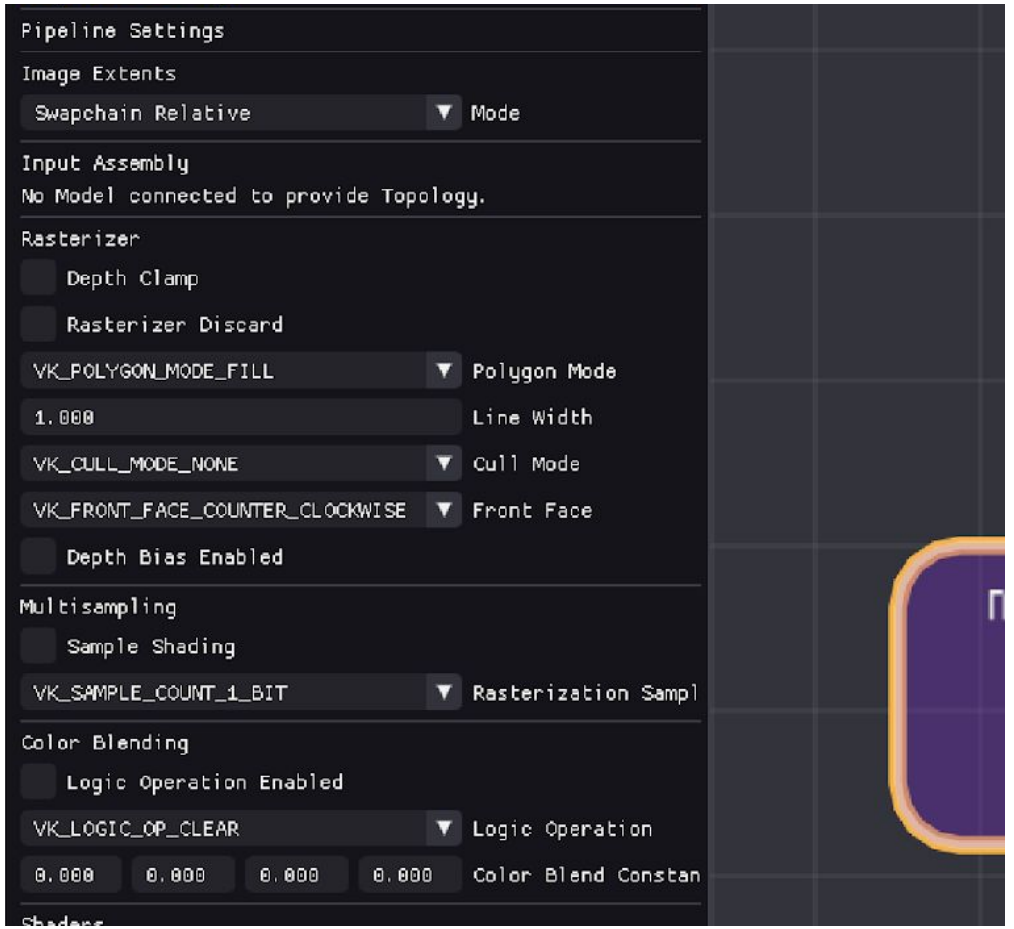


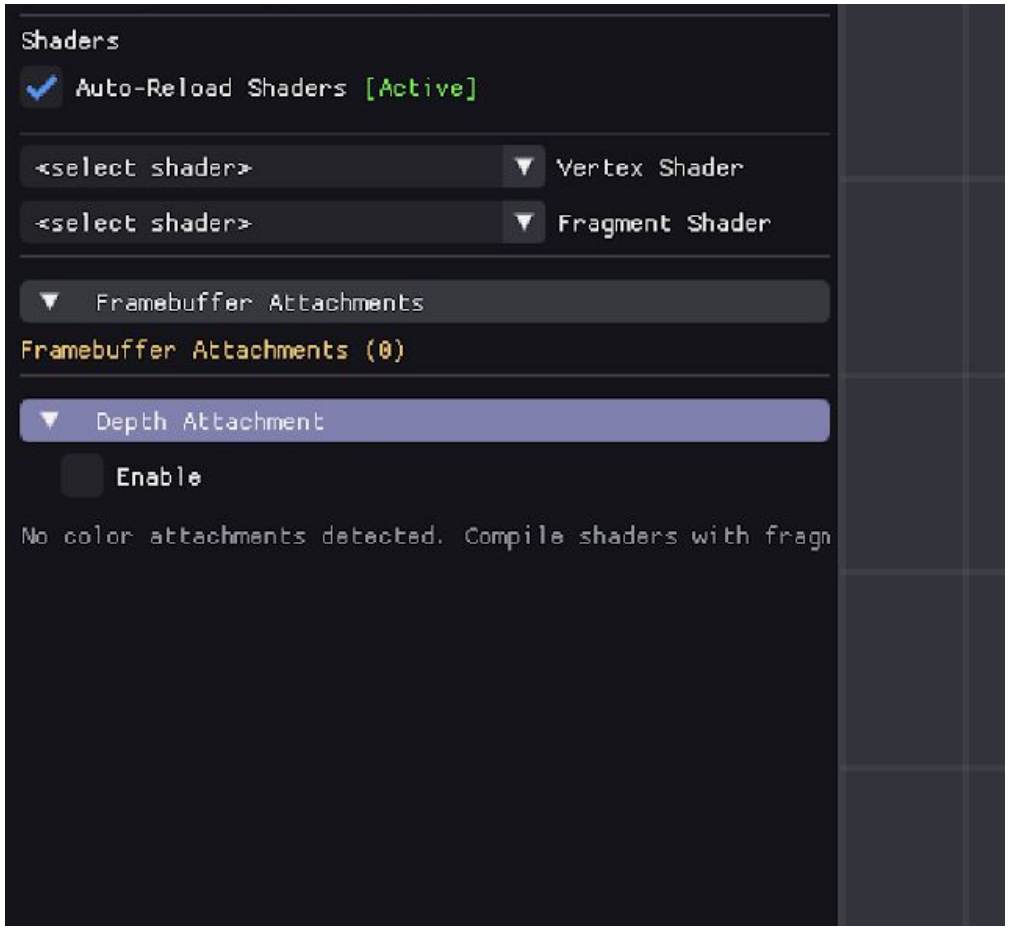
- All Vulkan pipeline state exposed as UI controls
- ~ 500-700 LoC

Node: Pipeline

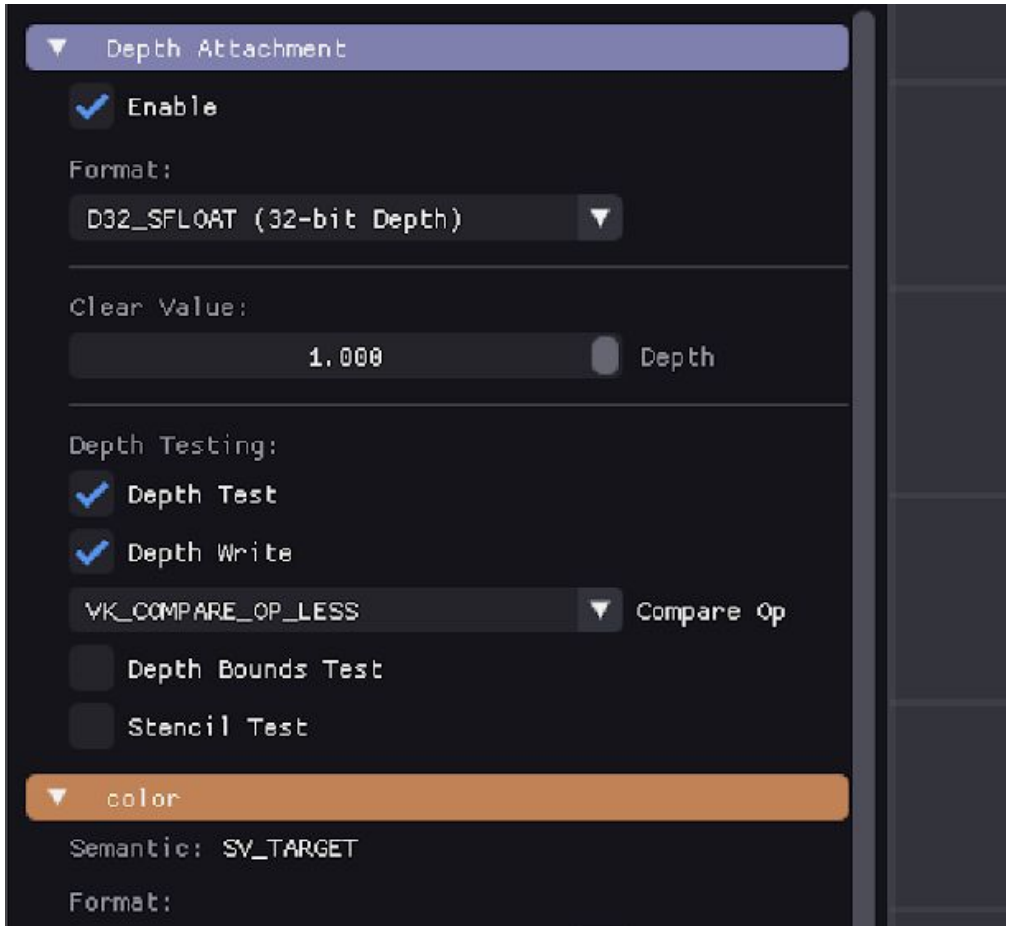


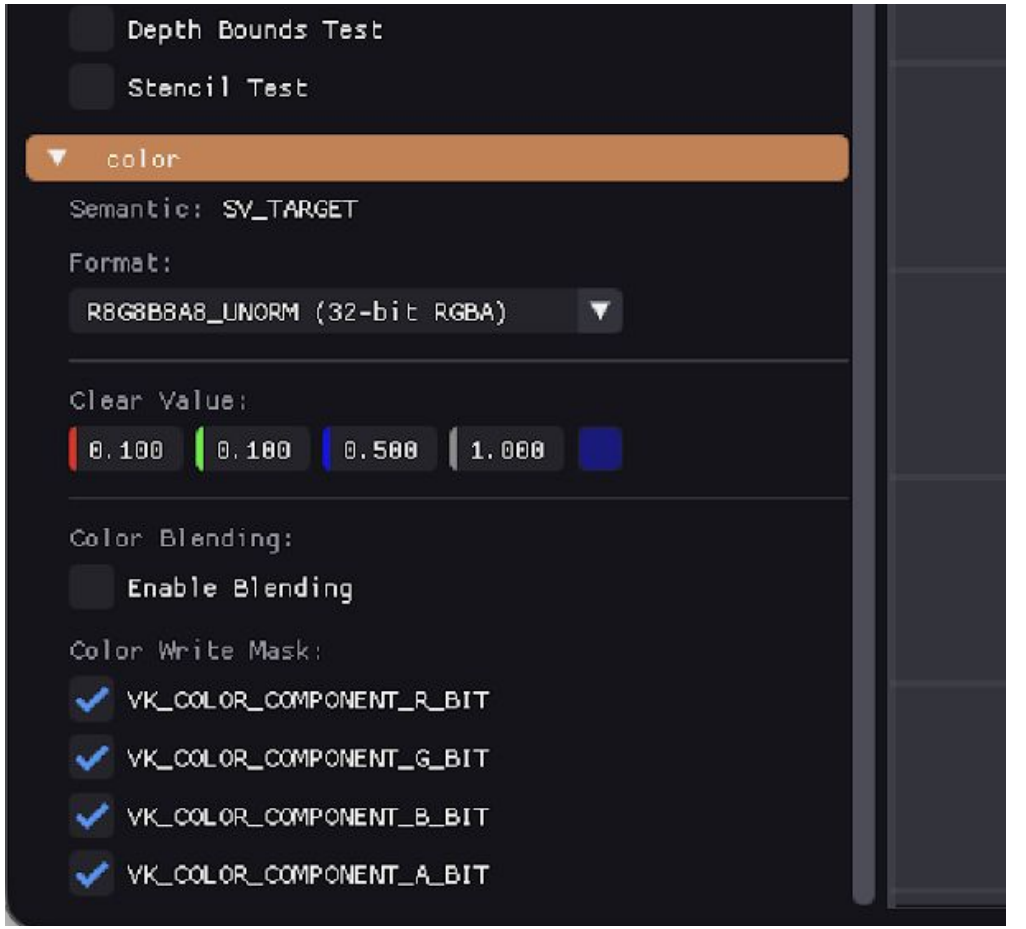
Node: Pipeline



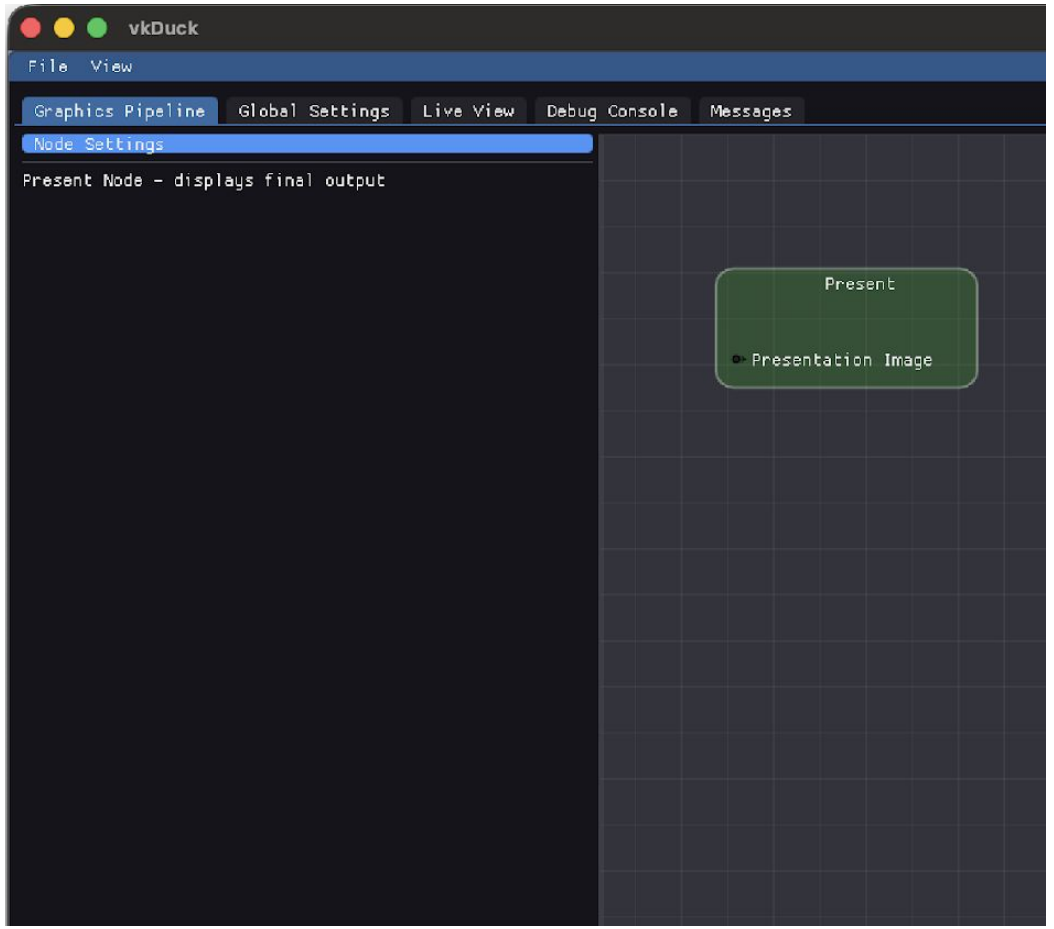


Node: Pipeline





Node: Present



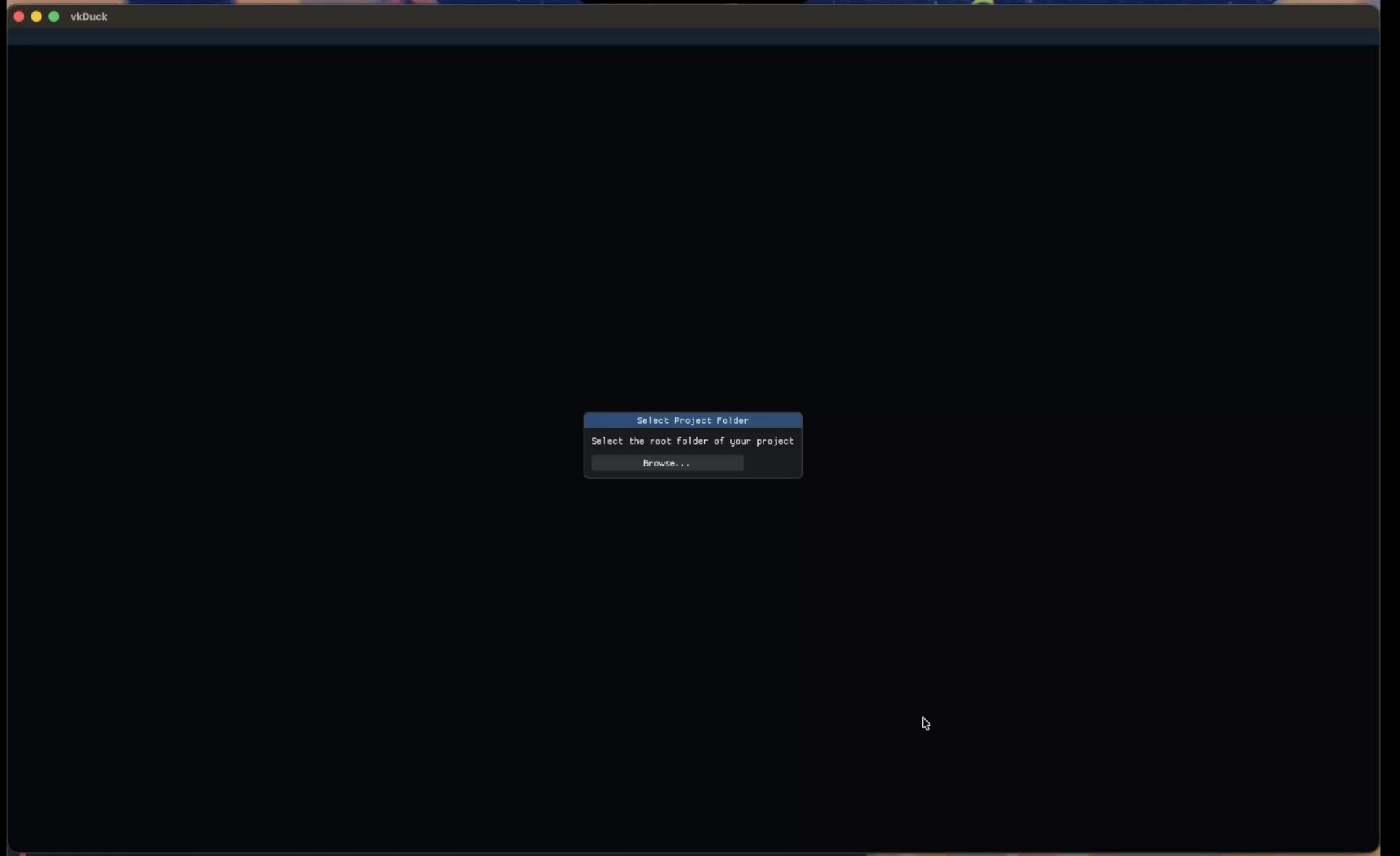
- Connect any Image to the Present node
- Creates ImGui-compatible descriptor set
- Renders to the Live View panel
- Only used in editor — not in generated code
- ~ 500-800 LoC

vkDuck

- A visual editor that generates Vulkan renderers
- Built with C++, ImGui, ImGui-Node-Editor, Vulkan, Slang
- Cross-platform
- Designed for learning, prototyping, and experimentation
- Actively developed as part of my master's thesis
- Open Source - try it yourself :)
- github.com/fini03/vkDuck

Live Demo

Let's build a simple renderer together.


main Window
1°C
64 %
US
Tue Feb 3 23:18
vkDuck
Select Project Folder
Select the root folder of your project
Browse...

The Shader

Ready | Today at 19:27

Vulkan-Editor ⟩ project ⟩ shaders ⟩ camera_light_test.slang ⟩ No Selection

```
struct VSOutput {
    float4 position : SV_Position;
    float3 worldPos : WORLD_POS;
    float3 worldNormal : WORLD_NORMAL;
    float2 uv : UV;
};

struct VSInput {
    float3 position : POSITION;
    float3 normal : NORMAL;
    float2 texCoord : TEXCOORD0;
};

struct FSOut {
    float4 color : SV_Target0;
};

struct Camera {
    float4x4 view;
    float4x4 invView;
    float4x4 proj;
};

struct Light {
    float3 position;
    float radius;
    float3 color;
};

struct ModelMatrices {
    float4x4 model;
    float4x4 normalMatrix;
};

// Bindings
// Set 0: Camera and Lights (global)
[[vk::binding(0, 0)]] ConstantBuffer<Camera> camera;
[[vk::binding(1, 0)]] ConstantBuffer<Light> lights[1];

// Set 1: Model and Texture (per-object)
[[vk::binding(0, 1)]] ConstantBuffer<ModelMatrices> modelMatrices;
[[vk::binding(1, 1)]] Sampler2D albedoTexture;
```

Ready | Today at 19:27

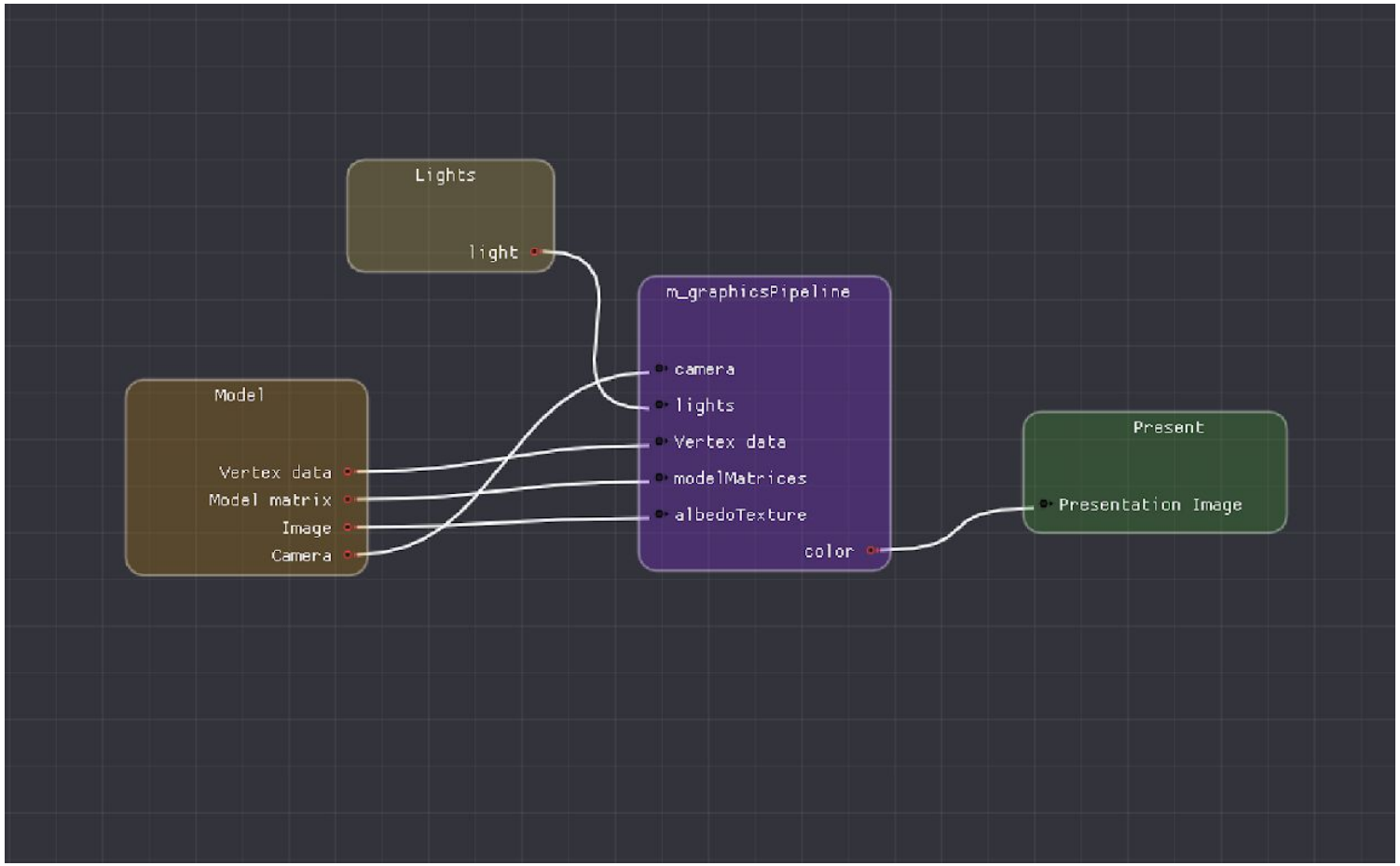
Vulkan-Editor ⟩ project ⟩ shaders ⟩ camera_light_test.slang ⟩ No Selection

```
[shader("vertex")]
VSOutput vertexMain(VSInput input) {
    VSOutput output;

    // Transform to world space
    float4 worldPos = mul(modelMatrices.model, float4(input.position, 1.0));

    // Transform normal using normal matrix
    float3 worldNormal = normalize(mul(float3x3(modelMatrices.normalMatrix), input.normal));

    // Transform to clip space
    float4 viewPos = mul(camera.view, worldPos);

    output.position = mul(camera.proj, viewPos);
    output.worldPos = worldPos.xyz;
    output.worldNormal = worldNormal;
    output.uv = input.texCoord;

    return output;
}

[shader("fragment")]
FSOut fragmentMain(VSOutput input) {
    FSOut output;
    float3 texColor = albedoTexture.Sample(input.uv).rgb;
    float3 cameraPos = float3(camera.invView[0][3], camera.invView[1][3], camera.invView[2][3]);

    float3 N = normalize(input.worldNormal);
    float3 V = normalize(cameraPos - input.worldPos);
    float3 L = normalize(lights[0].position - input.worldPos);
    float3 H = normalize(L + V);

    // Phong lighting
    float diffuseIntensity = max(dot(N, L), 0.0);
    float specularIntensity = pow(max(dot(N, H), 0.0), 32.0);

    float3 ambient  = texColor * 0.1 * lights[0].color;
    float3 diffuse  = texColor * diffuseIntensity * 0.8 * lights[0].color;
    float3 specular = 0.2 * specularIntensity * lights[0].color;

    output.color = float4(ambient + diffuse + specular, 1.0);
    return output;
}
```

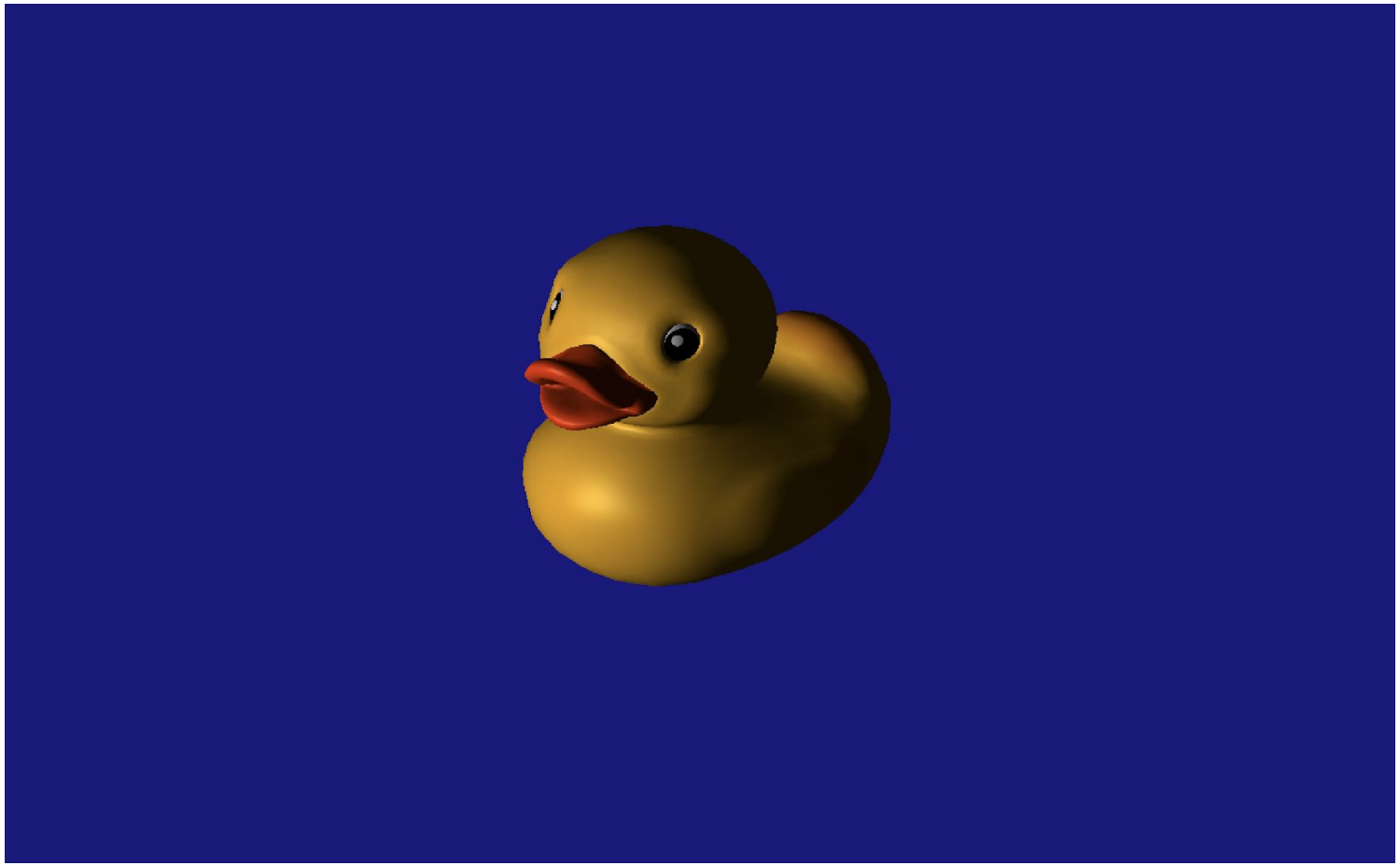
The Shader

```
1  struct VSOutput {
2      float4 position : SV_Position;
3      float3 worldPos : WORLD_POS;
4      float3 worldNormal : WORLD_NORMAL;
5      float2 uv : UV;
6  };
7
8  struct VSInput {
9      float3 position : POSITION;
10     float3 normal : NORMAL;
11     float2 texCoord : TEXCOORD0;
12 };
13
14 struct FSOut {
15     float4 color : SV_Target0;
16 };
17
```

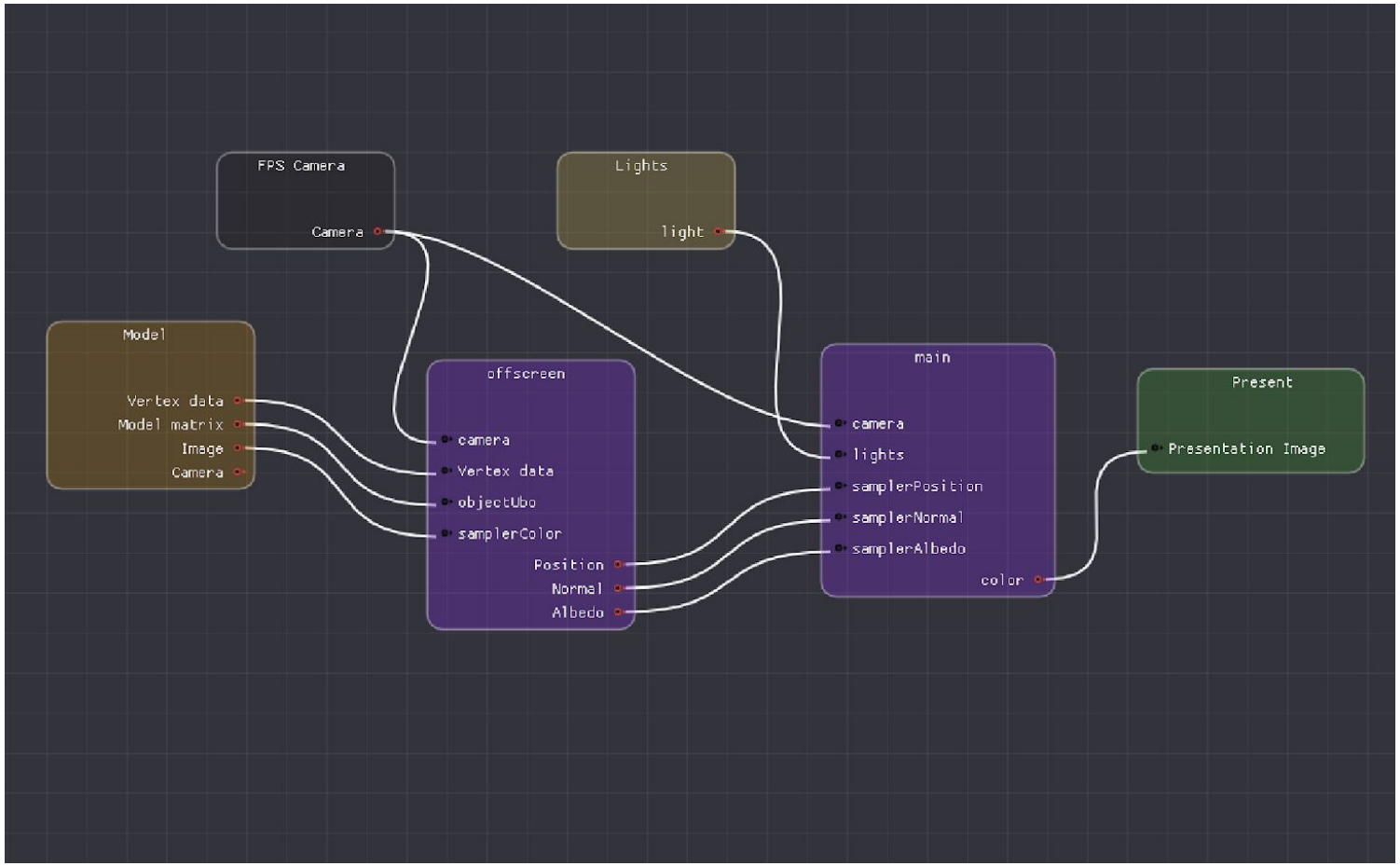
The Render Graph



Live Preview



The Render Graph (more complex)



Live Preview (more complex)

Generated Output

```
// Descriptor Set: descriptorSet_0
{
    std::vector<VkDescriptorSetLayoutBinding> descriptorSet_0_layoutBindings = {{
        {
            .binding = 0,
            .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
            .descriptorCount = 1,
            .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
        },
        {
            .binding = 1,
            .descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
            .descriptorCount = 1,
            .stageFlags = VK_SHADER_STAGE_VERTEX_BIT|VK_SHADER_STAGE_FRAGMENT_BIT
        }
    }};

    VkDescriptorSetLayoutCreateInfo descriptorSet_0_layoutInfo{
        .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO,
        .bindingCount = static_cast<uint32_t>(descriptorSet_0_layoutBindings.size()),
        .pBindings = descriptorSet_0_layoutBindings.data()
    };

    vkchk(vkCreateDescriptorSetLayout(device, &descriptorSet_0_layoutInfo, nullptr, &descriptorSet_0_layout));

    uint32_t descriptorSet_0_numSets = 1;
    std::vector<VkDescriptorSetLayout> descriptorSet_0_layouts(descriptorSet_0_numSets, descriptorSet_0_layout);
    VkDescriptorSetAllocateInfo descriptorSet_0_allocInfo{
        .sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO,
        .descriptorPool = descriptorPool_0,
        .descriptorSetCount = descriptorSet_0_numSets,
        .pSetLayouts = descriptorSet_0_layouts.data()
    };
    descriptorSet_0_sets.resize(descriptorSet_0_numSets);
    vkchk(vkAllocateDescriptorSets(device, &descriptorSet_0_allocInfo, descriptorSet_0_sets.data()));
```

Whats next?

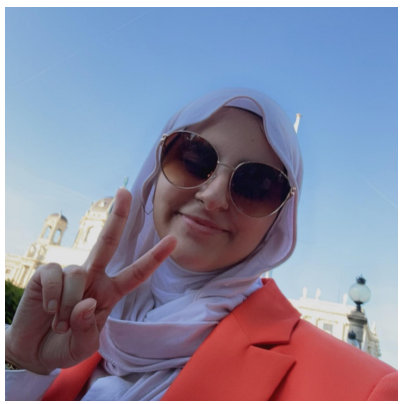
- Maintaining project, cleaning code & fixing bugs
- Adding more Vulkan features
- Adding more model types
- Adding additional shader types
- Adding subpasses (currently every pipeline is in it's own renderpass)
- Letting user choose what code to generate
- Currently macOS support via MoltenVK - experimenting with KosmicKrisp?
- Adding template shaders & features
- Enhancing UI/UX

Questions & Discussion

- Contact me: lamies.abbas@outlook.com
- Meet me: Available for discussion after this session and the next days :)
- Links:
  - Source Code: https://github.com/fini03/vkDuck
  - Demo Video: https://www.youtube.com/watch?v=otcTfEzbs04
  - My own page: https://quackie.at/

Thank you!

Lamies Abbas



Helmut Hlavacs