
GETTING STARTED
WITH VULKAN
VIKTOR MAKKI

WHO AM I?

- BME
- Graphisoft

SUMMARY

- What is Vulkan
- Lava Island Engine
- First triangle
- How to learn from LIE
- Configuration
- Error handling

WHAT IS VULKAN?

- Graphics API

WHY IS VULKAN?

Pros

- Object oriented
- More configuration
- Cross-platform

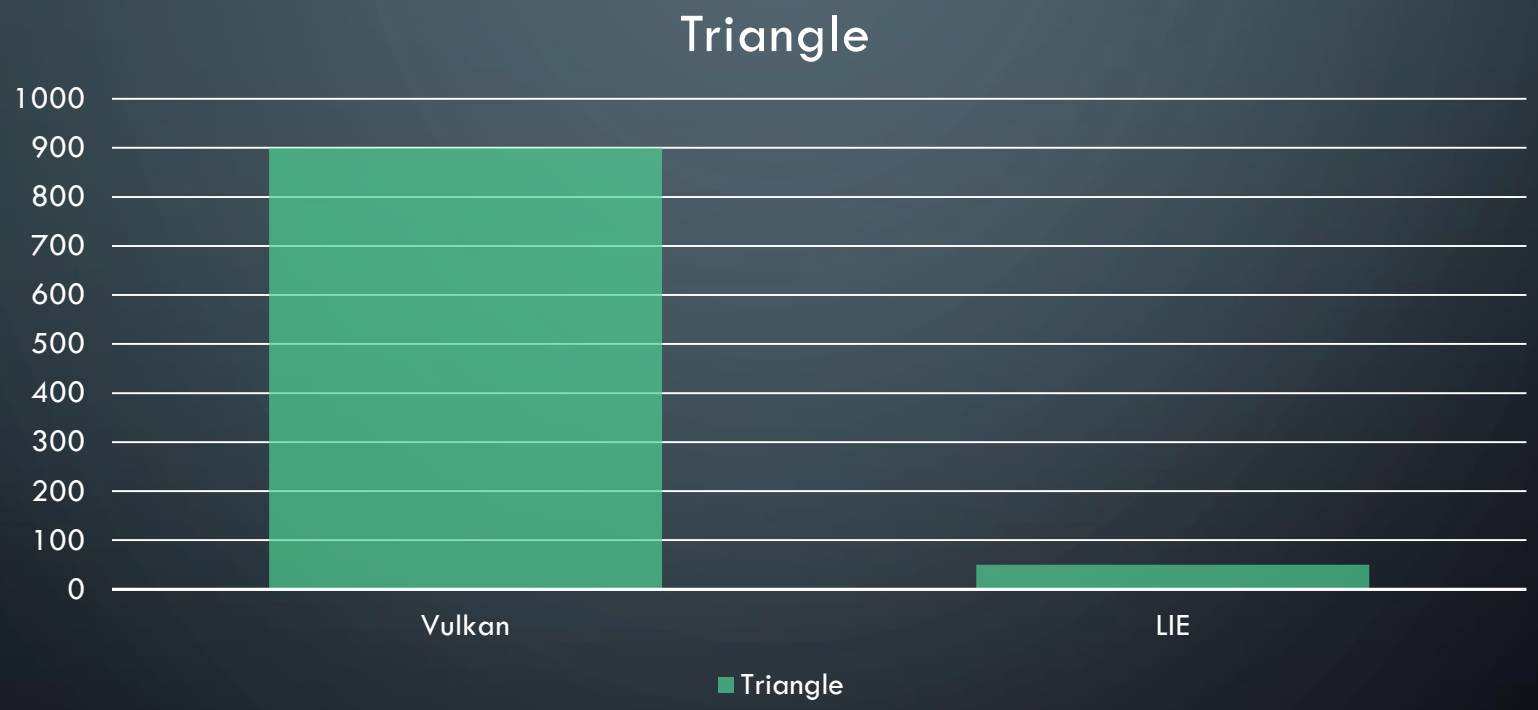
Cons

- Hard to learn
- Long initialization

LAVA ISLAND ENGINE (LIE)

- Motivation
- What does this library?

DRAWING A TRIANGLE



SHADER CODE!!!

DRAWING A TRIANGLE

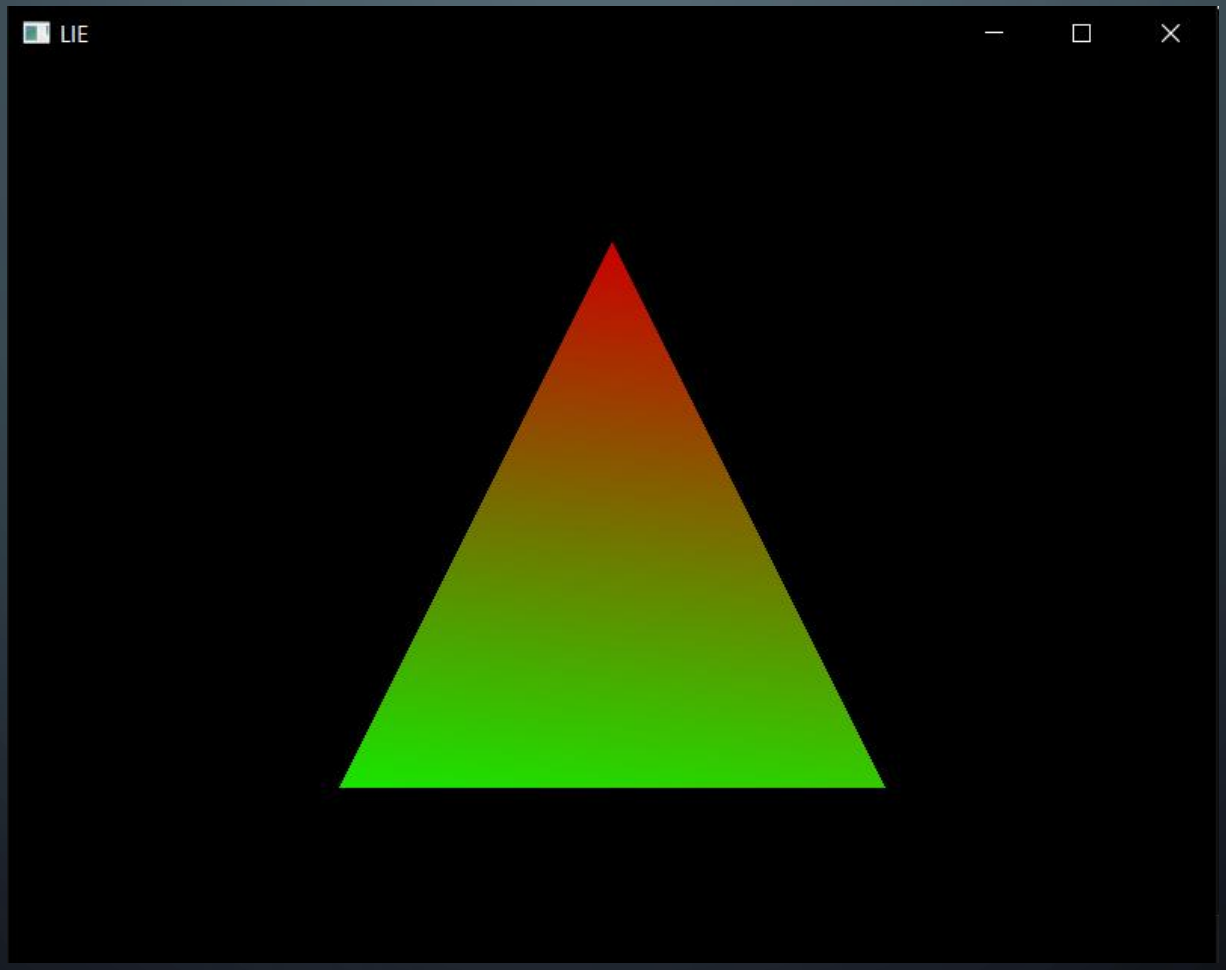
```
std::vector<VK::Vertex> triangleGeometry = {
{{-0.5f, -0.5f}, {0.1f, 0.9f, 0.0f}},
{{0.5f, -0.5f}, {0.2f, 0.8f, 0.0f}},
{{0.0f, 0.5f}, {0.8f, 0.0f, 0.0f}}};

std::vector<U32> triangleIndices = {0,1,2};

VK::Renderer renderer;
renderer.AddScene ({{{{"Shaders/vert.spv", VK_SHADER_STAGE_VERTEX_BIT},
                      {"Shaders/frag.spv", VK_SHADER_STAGE_FRAGMENT_BIT}},
                      triangleGeometry, triangleIndices}});

while(!renderer.IsWindowClosed()){
    renderer.Draw ();
}
```

OUR TRIANGLE



HOW DOES THE LIE WORK?

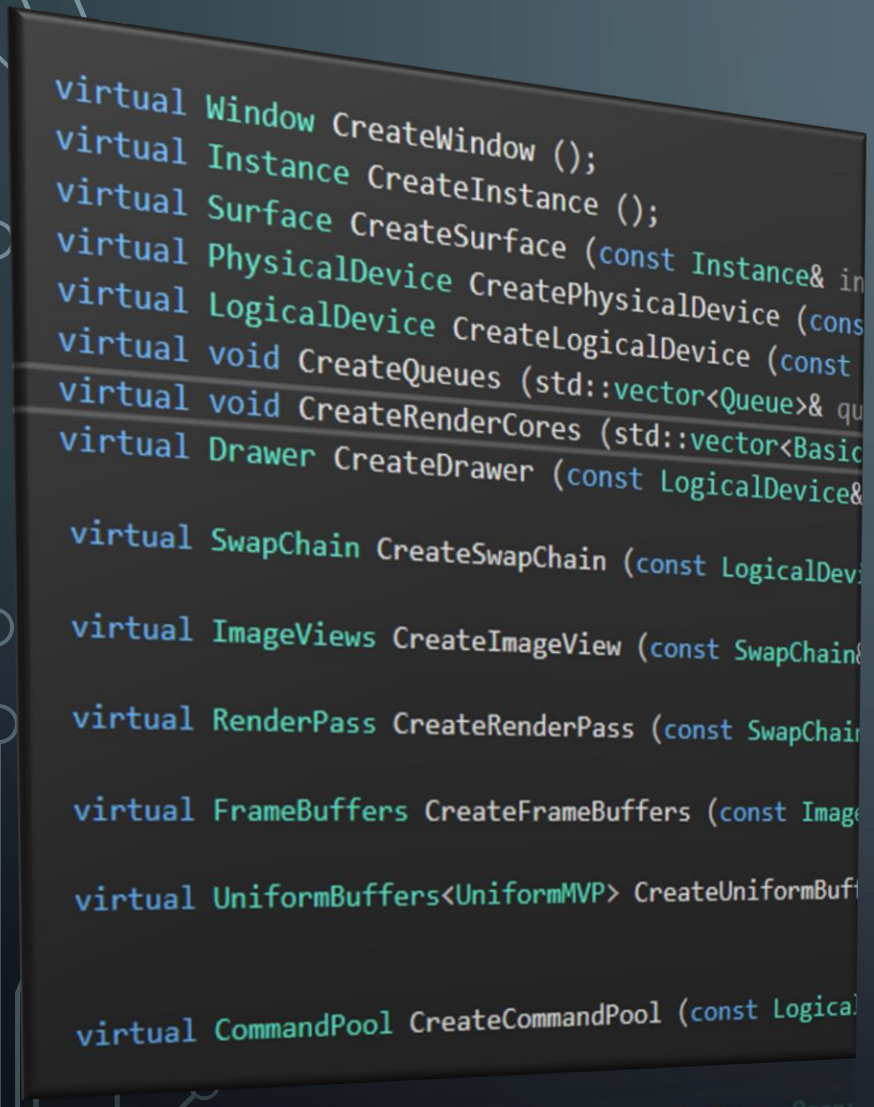
Scene
- Holds the actors

Renderer
- Handles the life of render cores

Render Cores
- Handles the life of LIE VK objects

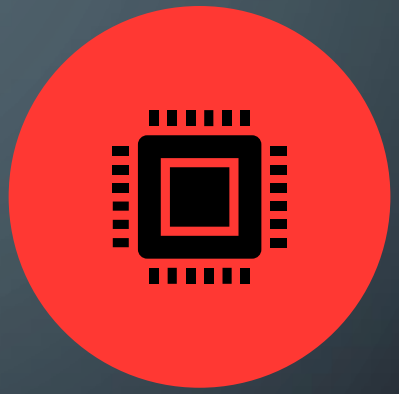
LIE VK objects
- Hides the Vulkan's dependencies

HOW TO LEARN FROM LIE



- Sequence
- Dependencies
- Vulkan specification

SEPARATED DEVELOPMENT



APPLICATION SIDE

VULKAN SIDE

APPLICATION SIDE

- New shaders
- New VertexData
- New UniformData

VERTEX DATA

```
enum ShaderLocation{
    position = 0,
    color = 1
};

struct Vertex{
    glm::vec2 position;
    glm::vec3 color;

    inline static std::vector<VkVertexInputBindingDescription> GetBindingDescriptions (){
        std::vector<VkVertexInputBindingDescription> bindingDescriptions (1);
        bindingDescriptions[0].binding = 0;
        bindingDescriptions[0].stride = sizeof (Vertex);
        bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
        return bindingDescriptions;
    }
    inline static std::vector<VkVertexInputAttributeDescription> GetAttributeDescriptions (){
        std::vector<VkVertexInputAttributeDescription> attributeDescriptions(2);

        attributeDescriptions[0].binding = 0;
        attributeDescriptions[0].location = ShaderLocation::position;
        attributeDescriptions[0].format = VK_FORMAT_R32G32_SFLOAT;
        attributeDescriptions[0].offset = offsetof (Vertex, position);

        attributeDescriptions[1].binding = 0;
        attributeDescriptions[1].location = ShaderLocation::color;
        attributeDescriptions[1].format = VK_FORMAT_R32G32B32_SFLOAT;
        attributeDescriptions[1].offset = offsetof (Vertex, color);
        return attributeDescriptions;
    }
```

- Implement static functions
- Static validation

VULKAN SIDE

- Configure Vulkan objects

CLASSES STRUCTURES

Public

- Create
- Destroy
- Getters
- ~~Setters~~

Private virtual

- FillCreateInfo

CONFIGURATION

1. Inherit from low level class
2. Override the FillCreateInfo
3. Inherit from Renderer and/or Render core
4. Override the matching CreateObject

INSTANCE EXAMPLE

```
class Instance{
private:
    VkInstance instance = VK_NULL_HANDLE;
    VkAllocationCallbacks* allocator;

    virtual void FillAppInfo (VkApplicationInfo& appInfo);
    virtual void FillCreateInfo (VkInstanceCreateInfo& createInfo,
                                 const VkApplicationInfo& appInfo,
                                 const Extension& extensions);
public:
    void Create (const Extension& extensions,
                 VkAllocationCallbacks* allocator = nullptr);
    void Destroy ();
    VkInstance GetInstance () const;
};
```

WHY IS THIS GOOD?

- New types for new configurations
- Less boilerplate code

ERROR HANDLING

- Only in debug mode
- Validation layers

FURTHER PLANS

- Clearly defined interface
- Optimization

THANK YOU?

- Viktor.makki.94@gmail.com