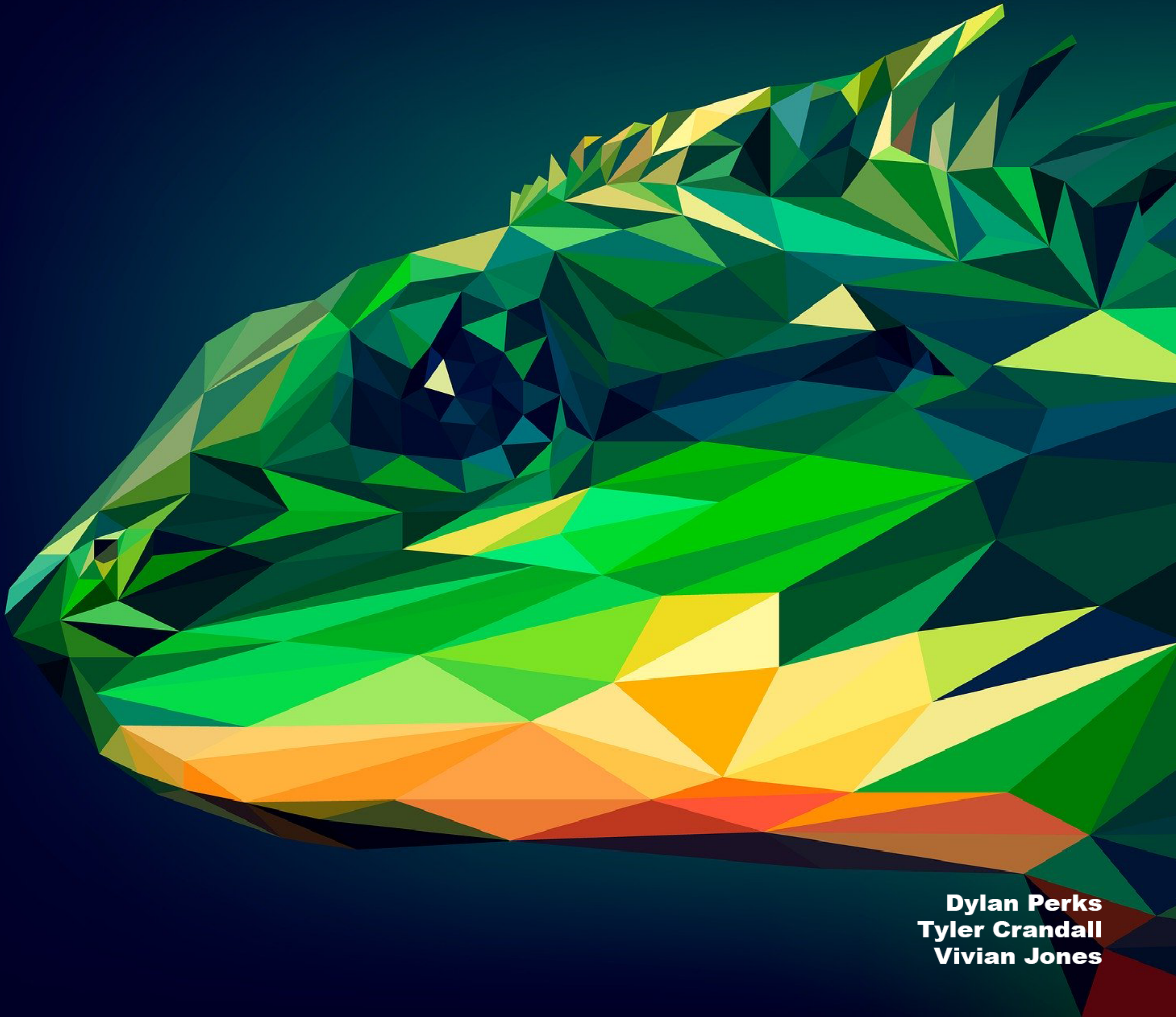
A Beginner's Guide to Vulkan



Dylan Perks
Tyler Crandall
Vivian Jones

Table of Contents

Chapter 1

Example

This chapter is specifically made for demonstration on how to add various elements to the book and this chapter **SHOULD NOT** be included in the final copy of the book.

```csharp
using System;

namespace Example
{
    public static class Program
    {
        // Entry Point
        static void Main()
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Figure 1.1: Example C# Snippet

Make sure to add figures.

```glsl
#version 450

#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable

layout (location = 0) in vec2 vsin_position;
layout (location = 1) in vec2 vsin_texCoord;
layout (location = 2) in vec4 vsin_color;

layout (binding = 0) uniform Projection
{
    mat4 projection;
};

layout (location = 0) out vec4 vsout_color;
layout (location = 1) out vec2 vsout_texCoord;

layout (constant_id = 0) const bool IsClipSpaceYInverted = true;
layout (constant_id = 1) const bool UseLegacyColorSpaceHandling = false;

out gl_PerVertex
{
    vec4 gl_Position;
};

vec3 SrgbToLinear(vec3 srgb)
{
    return srgb * (srgb * (srgb * 0.305306011 + 0.682171111) +
       0.012522878);
}

void main()
{
    gl_Position = projection * vec4(vsin_position, 0, 1);
    vsout_color = vsin_color;
    if (!UseLegacyColorSpaceHandling)
    {
        vsout_color.rgb = SrgbToLinear(vsin_color.rgb);
    }
    vsout_texCoord = vsin_texCoord;
    if (IsClipSpaceYInverted)
    {
        gl_Position.y = -gl_Position.y;
    }
}
```

Figure 1.2: Example GLSL Snippet

Chapter 2

Introduction

Chapter 3

Mathematics Principles

For almost any non-trivial graphical program, geometry will need to be moved across the screen. The simplest way to do that would be to update the geometry's vertices to the new position, but constantly sending new data to the graphics card is very slow, and not feasible for a large-scale program.

In addition, Normalized Device Coordinates are too abstract for most graphical work. For example, drawing an image at the proper size is nearly impossible when all you have are NDC. Wouldn't it be easier to use a different coordinates system?

The answer to both problems comes in the form of linear algebra. This chapter will be a **brief** introduction to the concept, and how it relates to graphics programming. If you find the topic interesting and want to know more, it is heavily recommended that you continue studying outside of this book. Several potentially-useful things (such as quaternions) are omitted here for brevity.

With that said, let's begin!

1 OpenGL Mathematics Library (GLM)

This tutorial is provided in C++, a language that does not include built-in linear algebra. Instead, we will use an external library: OpenGL Mathematics Library, or GLM for short. Despite the name, it will work for Vulkan just fine, as the mathematics required are the same. Download from here:

https://glm.g-truc.net/0.9.8/index.html

At the time of writing, version 0.9.9.8 is the current stable release. If you encounter problems, be certain that you use the same version as we do.

GLM is a header-only library, so no compiling or linking are necessary. Just download and copy the header folder into your includes folder.

To ensure that it works, try running the following code:

```cpp
#include <cstdio>
#include "glm/glm.hpp"

int main() {
 glm::vec4 vec(1.0f, 0.0f, 0.0f, 1.0f);
 glm::vec4 vec2(0.0f, 1.0f, 0.0f, 1.0f);

 vec += vec2;

 std::printf("{%f, %f, %f, %f}", vec.x, vec.y, vec.z, vec.w);
}
```

Figure 3.1: GLM test snippet

Chapter 4

Graphics Principles

Chapter 5

Initial Setup

Chapter 6

Shaders

Chapter 7

Pipelines

Chapter 8

Render Passes

Chapter 9

Drawing

Chapter 10

Project Guidance

Chapter 11

Textures

Chapter 12

Advanced Textures

Chapter 13

Mapping

Chapter 14

Instancing

Chapter 15

Tessellation