

Example of creation of multiplatform apps on basis of creating match-three game engine Senior Design Project One Report

Krzysztof Rudnicki, 307585

Supervisor: Tomasz Martyn, PhD, DSc

February 1, 2023

1 Introduction

For my thesis I am creating a game engine focused on match-three games (like Candy Crush), I create it in OpenGL with use of GLFW library. I focus on match three games because they are relatively easy to produce and can be played on all platforms

2 Accomplishments so far

2.1 Choices and Research

2.2 Choice of graphic rendering API

There are 3 main APIs for graphical rendering

- DirectX
- OpenGL
- Vulkan

DirectX developed by Microsoft focuses on Windows operating systems and Microsoft line of consoles Xbox, it is deemed as being harder with more low level programming and requiring better understanding of how underlying mechanisms work but in turn offers functionalities and better performance. It does not have free license.

OpenGL is developed by Khronos Group and offers good compatibility, especially if using OpenGL ES subset which works on Windows, Linux, Mac OS, Android, iOS and all major consoles. It is widely recognized as easiest of APIs and most popular choice for writing first game engine. On the other hand it lacks some of more advanced features which have to be written manually. It uses open source license similar to BSD

Vulkan is also developed by Khronos Group and as such is deemed as a spiritual successor of OpenGL with focus on using modern C++ features and fixing issues created by OpenGL 30 years old development time. Out of these three it is recognized as the hardest one as it is both complicated and newest. Similarly as OpenGL it uses open source license, namely Apache License 2.0. Considering all of those characteristics I decided to go with OpenGL API, specifically OpenGL ES subset with its focus on compatibility as making a multiplatform application is one of the focuses of this thesis. I decided that since this will be my first attempt at game engine development I need something that is relatively easy and has a lot of resources online. I would most likely not use advanced features of Vulkan and DirectX and therefore finish my thesis before approaching problems where OpenGL does not deliver more complicated architecture. From my own private preferences I also prefer software with open source license.

2.3 Choice of OpenGL Library

There are 4 basic OpenGL libraries that I considered:

- freeGLUT
- SDL
- SFML
- GLFW

freeGLUT was created as open source alternative to GLUT, is considered to be the worst out of all 4, written in archaic way, using C or very old C++,

which in turn results in unexpected "buggy" behaviour, it is also not really popular with lack of online guides

SDL - Simple DirectMedia Layer has big user base, it is not designed to be used as a standalone library and requires additional libraries to do networking or to create more complex applications.

SFML is the library with most features out of all 4, it supports networking, audio and has system features by default. It uses modern object oriented C++. Main problem with SFML is that it is not very popular API, therefore troubleshooting problems with SFML is quite hard and it has only few use guides online

GLFW is an library that is both the most popular and with fewest features by default. It forces users to use additional libraries for networking, sound, physic calculations and so on but in turn is also quite small and flexible. It has biggest community and a lot of guides, like one hosted at learnopengl.com I decided to use GLFW library. I wanted something that is relatively easy to troubleshoot and has abundance of learning materials online.

2.4 Initializing OpenGL

After choosing both the graphic rendering api and library used to implement it, next part of working on engine was setting up the environment with the GLFW library and library responsible for manufacturer specific drivers. This part consisted in 4 steps:

1. Downloading GLFW source code - from official GLFW Page
2. Building GLFW - Using CMake
3. Linking GLFW library with the project - Adding glfw build path to build option and including library in the source code
4. Setting up GLAD - Downloading correct source file for my device (HERE), copying and including it in engine source code.

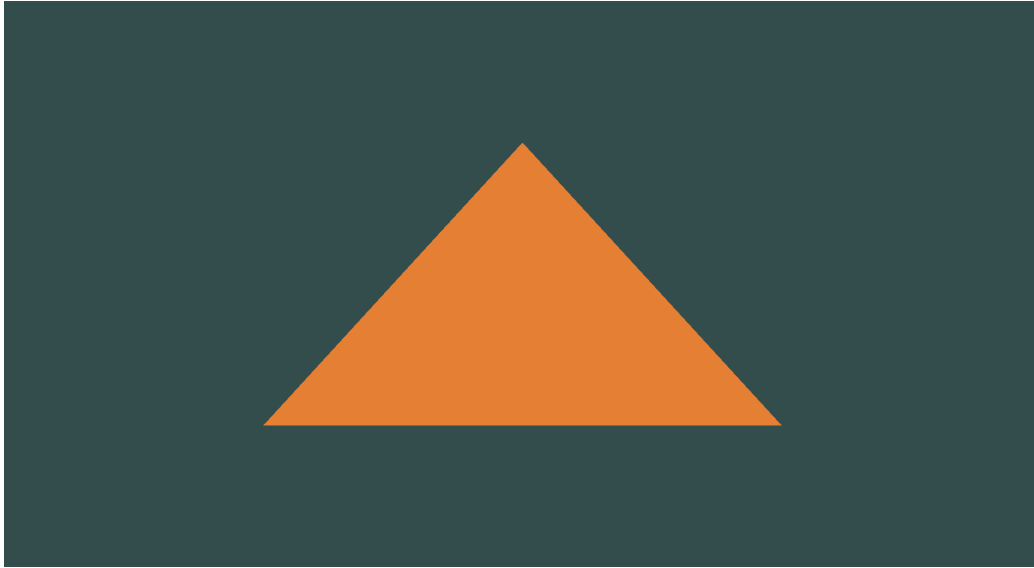


Figure 1: Program window after initializing OpenGL and drawing simple triangle

2.5 Shaders

After initializing opengl and creating first window I implemented dynamically, on-fly changing shaders using GLSL language and Shader C++ class. Shaders code is read every time the shader is used for an object so it is possible to change shader code on demand and see it immediately make change in the program itself.

Implemented shader functionality:

- Passing colors to shader from program code dynamically
- Passing data from vertex shader to fragment shader
- Offsetting rendered objects in shader
- Changing objects orientation in shader

Flowchart of shader class function can be seen below:

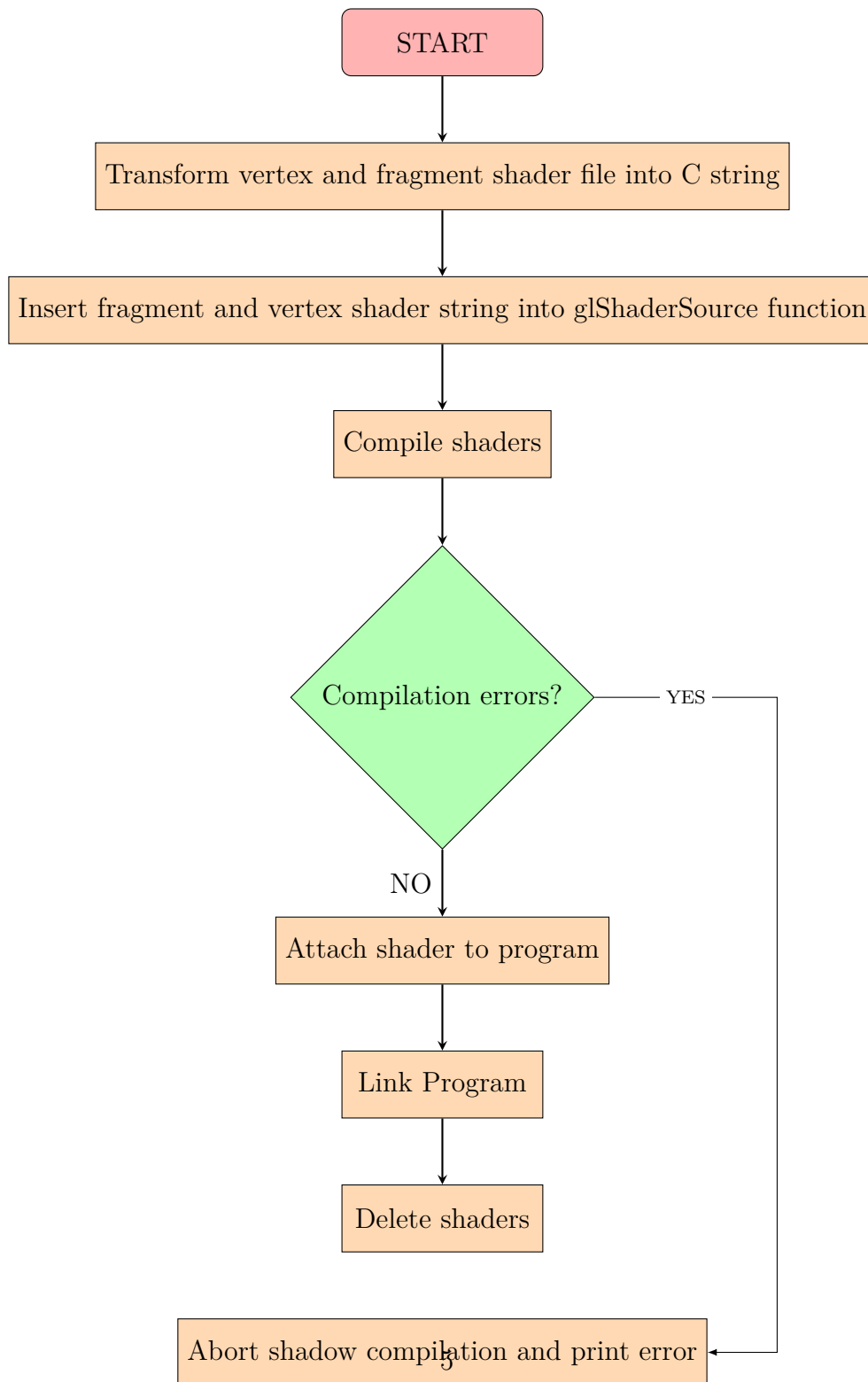


Figure 2: Shader usage example

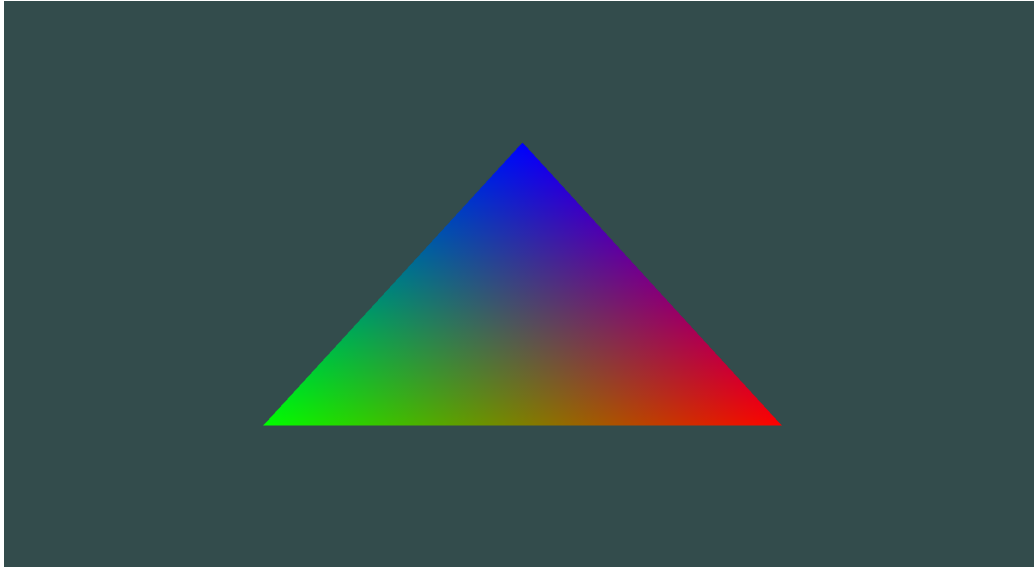


Figure 3: More complex shader coloring the triangle based on the distance from the vertice

3 Plans for future

These are main areas I have to implement in future semester:

3.1 Textures

Used for drawing objects with custom, user imported pictures on them, crucial for making tiles have an unique look without need to manually add them using shaders and vertices.

More specifically:

- Importing texture from a file to a program
- Wrapping texture around the object - For textures too small/big for an object
- Filtering texture - Choosing which pixel should be colored on a texture
- Combining textures with shaders - For coloring, adding effects to texture

3.2 Transformation

Using OpenGL Mathematics library - GLM, objects in the game need to be able to move, both on them own and on player/user command.

3.3 User Interface

Person using game engine needs to be able to:

- Add, delete and modify shaders for given object (tiles, game background)
- Add, delete and replace textures for given object (tiles, game background)
- Test the game before building it
- Build the game itself

Those elements need to be shown in the engine itself, using ImGui Menus

4 Overview

In total I feel confident in my ability to finish this project in next semester. Having more time to spend on it, with the scope that I chose and good coding practices used so far which make my code reusable and modular I should be able to finish the thesis as planned.