Setup tutorial and functionality.

<OpenGL Engine “PhySphere”>

Version 1.0

Prepared by Jakub Koterba

<UKSW>

<10.05.2022>

Table of Contents

[1. Introduction 1](#_Toc103639852)

[1.1 Project Overview 1](#_Toc103639853)

[1.2 Used Technology 1](#_Toc103639854)

[1.3 Intended Use 1](#_Toc103639855)

[1.4 Used Tutorials and sites 1](#_Toc103639856)

[2. Setup Manual 2](#_Toc103639857)

[2.1 Software Downloads 2](#_Toc103639858)

[2.2 Requirements 2](#_Toc103639859)

[2.3 Supported Operating Systems 3](#_Toc103639860)

[2.3.1 Windows 3](#_Toc103639861)

[2.3.2 Linux 3](#_Toc103639862)

[2.3.3 MacOS 3](#_Toc103639863)

[2.4 Windows setup 4](#_Toc103639864)

[2.4.1 General Assumptions 4](#_Toc103639865)

[2.4.2 Microsoft Visual Studio 2019 4](#_Toc103639866)

[2.4.3 Other IDE’s 5](#_Toc103639867)

[2.5 Linux Setup 5](#_Toc103639868)

[2.6 MacOS Setup 5](#_Toc103639869)

[3. Documentation 6](#_Toc103639870)

[3.1 Overall Functionality Description 6](#_Toc103639871)

[3.2 Overall Class Diagram 6](#_Toc103639872)

[3.3 Structure 6](#_Toc103639873)

[3.4 Class Description 7](#_Toc103639874)

[3.4.1 Logger 7](#_Toc103639875)

[3.4.2 Shader 7](#_Toc103639876)

[3.4.3 Block 7](#_Toc103639877)

[3.4.4 BlockType 7](#_Toc103639878)

[3.4.5 Chunk 7](#_Toc103639879)

[3.4.6 Vertex 7](#_Toc103639880)

[3.4.7 ChunkGenerator 7](#_Toc103639881)

[3.4.8 ChunkMesh 7](#_Toc103639882)

[3.4.9 BlockSide 7](#_Toc103639883)

[3.4.10 Frustum 7](#_Toc103639884)

[3.4.11 WorldRenderer 7](#_Toc103639885)

[3.4.12 DrawArraysIndirectCommand 7](#_Toc103639886)

[3.4.13 ChunkManager 7](#_Toc103639887)

[3.4.14 Direction 7](#_Toc103639888)

[3.4.15 Camera 8](#_Toc103639889)

[3.4.16 KeyboardInput 8](#_Toc103639890)

[3.4.17 MouseInput 8](#_Toc103639891)

[3.4.18 Player 8](#_Toc103639892)

[3.4.19 World 8](#_Toc103639893)

[3.5 Important and crucial methods 8](#_Toc103639894)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| First iteration | 28.04.2022 | The creation of the document | 1.0 |
| Full version | 12.05.2022 | Updating the document with needed data | 2.0 |
| Full Version | 16.05.2022 | Slowly finishing up the document | 2.1 |

# Introduction

## Project Overview

The purpose of creating this app was to obtain some basic knowledge about 3D programming using modern API’s and technology. I have implemented some optimization techniques to enable the creation of bigger worlds without much performance sacrifice. And you can check It via moving around the created world and by reading the log file to exactly see, what has been drawn to the screen in what kind of amount.

## Used Technology

This Framework is based on OpenGL and libraries that enabled calling GL functions, collecting the needed input from the keyboard and mouse and creating valid window contexts. Besides that I have implemented:

* Better that naïve meshing algorithm
* Frustum Culling
* AZDO Persistently mapped buffers
* Infinite and Random World Generation
* Depth Buffers

## Intended Use

The use case Is very simple. The app can be opened and the implemented technologies can be seen through moving around the map and checking out the drawn structures.

After that, all of the things that happened under the hood can be seen in the log file.

## Used Tutorials and sites

For getting the needed knowledge I have used a few of the most established and popular sources about OpenGL right now. I started from the tutorial from a YouTuber, who claims that he worked for EA in the game engine department, he is called “TheCherno”:

[The Cherno - YouTube](https://www.youtube.com/c/TheChernoProject). After that I have switched to a site called:

[Learn OpenGL, extensive tutorial resource for learning Modern OpenGL](https://learnopengl.com/) from where I have gathered most of my basic knowledge. Then I have decided to do my research about better meshing algorithms. I managed to find a nice site called: [0 FPS – Mostly geometry](https://0fps.net/). Then I wanted to search for inspirations in making the rendering pipeline better. So I have stumbled upon this: [Nick's Blog (nickmcd.me)](https://nickmcd.me/). I also reviewed some code from GitHub and tried to do some reverse engineering from It.

# Setup Manual

## Software Downloads

Here I am attaching the links to the download sites for the software that this framework is using for executing its functionality.

GLEW: [The OpenGL Extension Wrangler Library download | SourceForge.net](https://sourceforge.net/projects/glew/)

GLFW: [Download | GLFW](https://www.glfw.org/download.html)

GLM: [g-truc/glm: OpenGL Mathematics (GLM) (github.com)](https://github.com/g-truc/glm)

OpenSimplex: [SRombauts/SimplexNoise: A Perlin's Simplex Noise C++ Implementation (1D, 2D, 3D) (github.com)](https://github.com/SRombauts/SimplexNoise)

Here is the documentation to the given software(if there exists)

* GLEW: [GLEW: The OpenGL Extension Wrangler Library (sourceforge.net)](http://glew.sourceforge.net/basic.html)
* GLFW: [GLFW: Main Page](https://www.glfw.org/docs/latest/)
* GLM: [glm/manual.md at master · g-truc/glm (github.com)](https://github.com/g-truc/glm/blob/master/manual.md)
* OpenSimplex: [SimplexNoise/README.md at master · SRombauts/SimplexNoise (github.com)](https://github.com/SRombauts/SimplexNoise/blob/master/README.md)
* OpenGL: [docs.gl](https://docs.gl/)

## Requirements

The machine that will be running the code should have at least 4GB of RAM, a few processing cores based on any modern core architecture, an GPU that supports OpenGL 4.6 and that has ideally more or equal 1Gb of Video Memory. There should be at least 2GB of free storage to contain the needed binaries, libraries and the code. Overall, the framework isn’t very demanding to run. It can run on one thread and with the access to a very limited amount of memory. But each upgrade will improve the overall performance greatly. So It looks as follows

Minimum System Requirements:

CPU: Core i3 2nd Generation/AMD 4 core part newer than FX4100

RAM: 4 GB

GPU: Any gpu supporting OpenGL4.6 with 512 Mb of VRAM

STORAGE: At least 2 GB free for the Code and dependencies

OS: Windows 10/11, Linux, MacOS

Recommended System Requirements

CPU: Core i5 2nd Generation/AMD part faster than FX6300(4 or more threads) or equivalent

RAM: 8 GB

GPU:

Supporting OpenGL 4.6 with at least 2GB of VRAM

Integrated:

* Intel Iris 80EU
* AMD Vega(Ryzen 4000/5000) 6CU

Discrete:

* GT1030 4GB
* R7 250X 2GB

Or equivalent.

STORAGE: At least 2 GB free for the Code and dependencies

OS: Windows 10/11, Linux, MacOS

## Supported Operating Systems

### Windows

It supports all of the libraries natively. The setup difficulty depends on the IDE and software that We are using.

### Linux

Supports OpenGL through X11 and all other libraries.

### MacOS

Supports OpenGL non-natively through X11 and all other libraries.

## Windows setup

### General Assumptions

Setting up the environment on Windows and enabling it to run this framework on your machine isn’t as complicated as one could imagine. This framework demos some basic and some advanced 3D Functionality and that is a fact, but whole of the hard work is essentially done by our GPU manufacturers who deliver the correct drivers for our GPU’s with the needed functionality to execute and create new software implementations. Windows supports every GPU drivers natively, so Our task is to make our environment capable of detecting the OpenGL.context and being able to call the needed functions and not define them. More work is done when it comes to creating the window, handling the input and having the ability to edit OpenGL maths. Overall, the libraries and binaries that I use mostly come pre-compiled and We only have to supply the correct path and needed preproccesor commands for our IDE to run.

### Microsoft Visual Studio 2019

#### Overall Information

Here, the job comes down to creating a directory inside our project directory. It will contain all the needed files for our libraries to be executed. It would be good if the directory had some structure in it, so we would be able to change particular parts of the software in the future. After that, We should just inform Visual Studio about the file locations, give the needed paths and preprocessor commands, about whom We should learn from the used software’s documentation. And that is It. The Directory should contain: [GLEW, GLFW, GLM, OpenSimplex] directories inside.

#### GLEW

* Then extract the zip file into the folder with the Libraries that you will be using in your Visual Studio Project. I recommend to create a folder inside the project folder so as finding and checking the needed files will be easy and also, thanks to that we can do a little shortcut in the next step
* After that We have to proceed to setting up the linking and the macros in the Visual Studio project settings. The path looks as follows: Project name -> Settings -> C/C++ -> General -> Additional Include Directories -> „$(SolutionDir)Libraries\GLEW\include\GL”, where „$(SolutionDir) is a macro that gives out the path to the project folder and a „\” sign.
* There we give a path that is similar to the one that I have given.
* Then We have to set and Preprocessor command “GLEW\_STATIC”: Preprocessor ->Definitions->GLEW\_STATIC
* After that We have to correctly setup the Linker by adding the following paths. Linker->General->Additional Library Directories>$(SolutionDir)Libraries\GLEW\lib\Release\Win32

Linker->Input->Additional Dependencies->glew32s.lib; Shell32.lib; Gdi32.lib; User32.lib

#### GLFW

* Then We have to give the path to the Visual Studio and inform the Linker about the binaries: Project name -> Settings -> C/C++ -> General -> Additional Include Directories-> $(SolutionDir)Libraries\GLFW\include\GLFW
* After that We have to correctly setup the Linker by adding the following paths.

Linker->General->Additional Library Directories>$(SolutionDir)Libraries\GLFW\lib-vc2019

* Directories>$(SolutionDir)Libraries\GLEW\lib\Release\Win32
* Linker->Input->Additional Dependencies-> glfw3.lib; opengl32.lib;

#### GLM

* The we add the path: Project name -> Settings -> C/C++ -> General -> Additional Include Directories->„$(SolutionDir)Libraries\GLM

#### OpenSimplex

* Next We give the path to the files in Visual Studio: Project name -> Settings -> C/C++ -> General -> Additional Include Directories->$(SolutionDir)Libraries\SIMPLEX

### Other IDE’s

## Linux Setup

To be continued

## MacOS Setup

To be continued

# Documentation

## Overall Functionality Description

This framework, called “PhySphere OpenGL Engine” has the ability to create random generated world structure. It consists of many equally sized chunks that are stored in a chunk-map data structure. It then uses the chunk data to create the interleaved Mesh data, that means, the data of all the Vertices that the chunk contains. One vertex consists of the position in the world space, the colour of the vertex and the normal vector of that vertex. The chunk creation is decided by the ChunkManager, which contains the ChunkMap. It scans for the input and checks whether any new chunk has to be added. Furthermore, before reading, It has to meet all the procedures of creating buffers, parsing the shader code and compiling it, allocating the buffer memory and then specifying what to draw with the help of the Frustum Culling, a technique, that checks what is in our viewing frustum, or in other words, the field of view. That helps out by effectively cutting the drawn chunk amount by half.

## Overall Class Diagram

Here is the basic overview on all of the classes that the project contains.

## Structure

The framework source code consists of directories called:

* Engine
* Game
* Shaders
* Libraries

First three of them contain the source code, and the last one contains all of the needed dependencies.

First of all, inside the „Engine” there are directories called:

* BackEnd which is responsible for the Shaders and Logging
* Gameplay, which contains the information about the Player and the World
* Input, containing all of the input handling
* Rendering, which contains the whole rendering pipeline
* WorldCreation, contains the code responsible for the map of chunks, generating chunks and the meshing .

Inside the „Game” is the Main function that contains the execution of all of the functionality from the description.

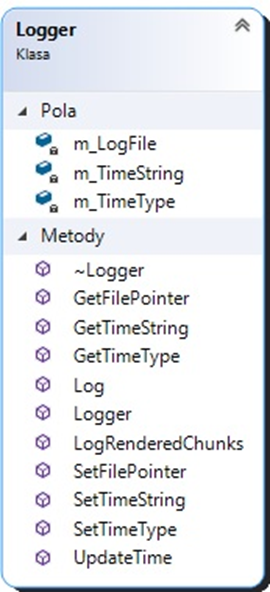
The „Shaders” directory contains the shaders code written in the GLSL, which is essentially a „Language”

„Libraries” contain the directories containing all of the Binaries and Libraries that We are using.

## Class Description

Here I will present the classes in the format as follows – „Class Name(Classes that are contained within): functionality, photo of the class structure.

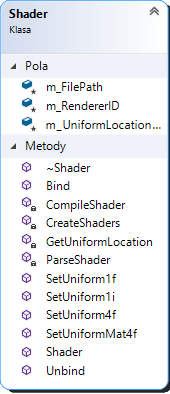
### Logger



The Logger class is meant to enable easy collection of the information to one Log file in the main directory of the project.

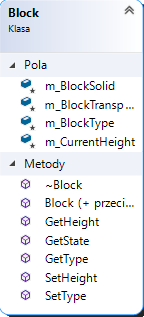
It has the variables to store the time struct in it. Also available are some getters/setters and 3 main methods for logging at runtime

### Shader

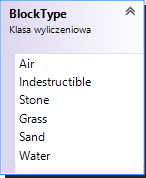


Shader class is created to easily read and process the GLSL shaders code. It reads it, and makes it available for the in-built compiler to compile and send to the gpu. The private variables contain the filepath to the shader file

### Block



### BlockType



### Chunk

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Vertex

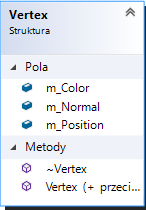
Obraz zawierający tekst

Opis wygenerowany automatycznie

### ChunkGenerator

Obraz zawierający tekst

Opis wygenerowany automatycznie



### ChunkMesh

Obraz zawierający tekst

Opis wygenerowany automatycznie

### BlockSide

Obraz zawierający tekst

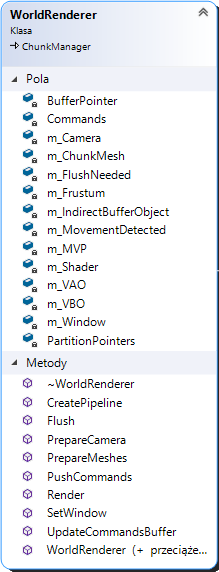
Opis wygenerowany automatycznie

### Frustum

Obraz zawierający tekst

Opis wygenerowany automatycznie

### WorldRenderer



### DrawArraysIndirectCommand

Obraz zawierający tekst

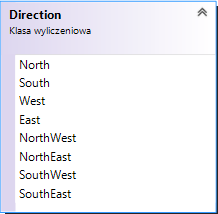
Opis wygenerowany automatycznie

### ChunkManager

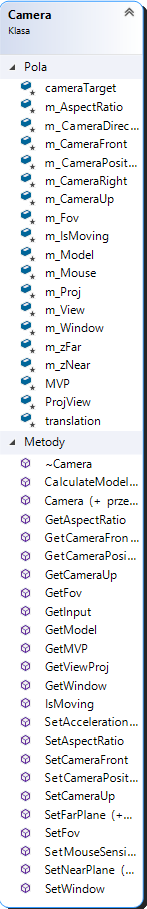
Obraz zawierający tekst

Opis wygenerowany automatycznie

### Direction



### Camera



### KeyboardInput

Obraz zawierający tekst

Opis wygenerowany automatycznie

### MouseInput

Obraz zawierający tekst

Opis wygenerowany automatycznie

### Player

Obraz zawierający tekst

Opis wygenerowany automatycznie

### World

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Important and crucial methods