**МИНОБРНАУКИ РОССИИ**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**

**(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кафедра |  |  |  |  |
|  |  | шифр |  | наименование кафедры, по которой выполняется работа |
| Дисциплина |  |  | | |
|  |  | наименование дисциплины | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УЧЕБНО-ПРАКТИЧЕСКАЯ РАБОТА | | | | |
|  |  |  |
|  | номер задания (при наличии) |  |
| ОБЩИЙ АЛГОРИТМ РАБОТЫ С OPENGL. СОЗДАНИЕ ОКОННОГО ПРИЛОЖЕНИЯ. СОЗДАНИЕ И ОТОБРАЖЕНИЕ ИЗОБРАЖЕНИЙ | | | |
|  | | | |

при наличии указать тему учебно-практической работы и (или) номер варианта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ОБУЧАЮЩИЙСЯ** | | | | | | |
| группы | | | |  | | о722б |
|  |  | Смыгалов Д. Д. | | | | |
| подпись |  | фамилия и инициалы | | | | |
| 24.10.2024 | | | | |
| дата сдачи | | | | |
| **ПРОВЕРИЛ** | | | | | | |
|  |  | Снижко Е. А. | | | | |
| подпись |  | фамилия и инициалы | | | | |
| Оценка / балльная оценка | | |  | | | |
|  | | | | |
| дата проверки | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| г. Санкт-Петербург | | | | | | | |
|  |  |  | 20 | 24 | г. |  |

**СОДЕРЖАНИЕ**

[**1 Задание** 3](#_Toc180638210)

# **1 Задание**

1. Создать проект в выбранной среде разработки и подключить к нему выбранные библиотеки для работы с API OpenGL.

2. Создать окно для вывода изображения.

3. С использованием шейдерных программ отобразить на окне три треугольника разных цветов. Для каждого треугольника следует создать свой фрагментный шейдер, определяющий его цвет.

# **2 Текст программы**

import glfw

import OpenGL.GL as gl

import numpy as np

from shaders import VERTEX\_SHADER, RED\_FRAGMENT\_SHADER, GREEN\_FRAGMENT\_SHADER, BLUE\_FRAGMENT\_SHADER

def compile\_shader(shader\_type, source):

    shader = gl.glCreateShader(shader\_type)

    gl.glShaderSource(shader, source)

    gl.glCompileShader(shader)

    if not gl.glGetShaderiv(shader, gl.GL\_COMPILE\_STATUS):

        raise RuntimeError(gl.glGetShaderInfoLog(shader))

    return shader

def create\_shader\_program(vertex\_shader\_source, fragment\_shader\_source):

    vertex\_shader = compile\_shader(gl.GL\_VERTEX\_SHADER, vertex\_shader\_source)

    fragment\_shader = compile\_shader(gl.GL\_FRAGMENT\_SHADER, fragment\_shader\_source)

    program = gl.glCreateProgram()

    gl.glAttachShader(program, vertex\_shader)

    gl.glAttachShader(program, fragment\_shader)

    gl.glLinkProgram(program)

    if not gl.glGetProgramiv(program, gl.GL\_LINK\_STATUS):

        raise RuntimeError(gl.glGetProgramInfoLog(program))

    gl.glDeleteShader(vertex\_shader)

    gl.glDeleteShader(fragment\_shader)

    return program

def main():

    # Initialize GLFW

    if not glfw.init():

        return

    # Configure GLFW

    glfw.window\_hint(glfw.CONTEXT\_VERSION\_MAJOR, 3)

    glfw.window\_hint(glfw.CONTEXT\_VERSION\_MINOR, 3)

    glfw.window\_hint(glfw.OPENGL\_PROFILE, glfw.OPENGL\_CORE\_PROFILE)

    # Create window

    window = glfw.create\_window(800, 600, "Three Triangles", None, None)

    if not window:

        glfw.terminate()

        return

    glfw.make\_context\_current(window)

    init\_opengl()

    # Create shader programs

    shader\_programs = [

        create\_shader\_program(VERTEX\_SHADER, RED\_FRAGMENT\_SHADER),

        create\_shader\_program(VERTEX\_SHADER, GREEN\_FRAGMENT\_SHADER),

        create\_shader\_program(VERTEX\_SHADER, BLUE\_FRAGMENT\_SHADER)

    ]

    # Triangle vertices

    triangles = [

        # Left triangle

        np.array([

            -0.8, -0.5, 0.0,

            -0.4, -0.5, 0.0,

            -0.6,  0.5, 0.0

        ], dtype=np.float32),

        # Middle triangle

        np.array([

            -0.2, -0.5, 0.0,

            0.2, -0.5, 0.0,

            0.0,  0.5, 0.0

        ], dtype=np.float32),

        # Right triangle

        np.array([

            0.4, -0.5, 0.0,

            0.8, -0.5, 0.0,

            0.6,  0.5, 0.0

        ], dtype=np.float32)

    ]

    # Create VAOs and VBOs

    vaos = gl.glGenVertexArrays(3)

    vbos = gl.glGenBuffers(3)

    for i in range(3):

        gl.glBindVertexArray(vaos[i])

        gl.glBindBuffer(gl.GL\_ARRAY\_BUFFER, vbos[i])

        gl.glBufferData(gl.GL\_ARRAY\_BUFFER, triangles[i].nbytes, triangles[i], gl.GL\_STATIC\_DRAW)

        gl.glVertexAttribPointer(0, 3, gl.GL\_FLOAT, gl.GL\_FALSE, 0, None)

        gl.glEnableVertexAttribArray(0)

    # Main loop

    while not glfw.window\_should\_close(window):

        # Clear the screen

        gl.glClear(gl.GL\_COLOR\_BUFFER\_BIT)

        # Draw each triangle with its respective shader

        for i in range(3):

            gl.glUseProgram(shader\_programs[i])

            gl.glBindVertexArray(vaos[i])

            gl.glDrawArrays(gl.GL\_TRIANGLES, 0, 3)

        # Swap buffers and poll events

        glfw.swap\_buffers(window)

        glfw.poll\_events()

    # Cleanup

    for vao in vaos:

        gl.glDeleteVertexArrays(1, [vao])

    for vbo in vbos:

        gl.glDeleteBuffers(1, [vbo])

    for program in shader\_programs:

        gl.glDeleteProgram(program)

    glfw.terminate()

def init\_opengl():

    # Get OpenGL version info

    version = gl.glGetString(gl.GL\_VERSION)

    print(f"OpenGL Version: {version.decode('utf-8')}")

    # Get GLSL version info

    glsl\_version = gl.glGetString(gl.GL\_SHADING\_LANGUAGE\_VERSION)

    print(f"GLSL Version: {glsl\_version.decode('utf-8')}")

    # Get GPU vendor and renderer

    vendor = gl.glGetString(gl.GL\_VENDOR)

    renderer = gl.glGetString(gl.GL\_RENDERER)

    print(f"Vendor: {vendor.decode('utf-8')}")

    print(f"Renderer: {renderer.decode('utf-8')}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

# **3 Результат работы программы**

Результат работы программы представлен на рисунке 1.

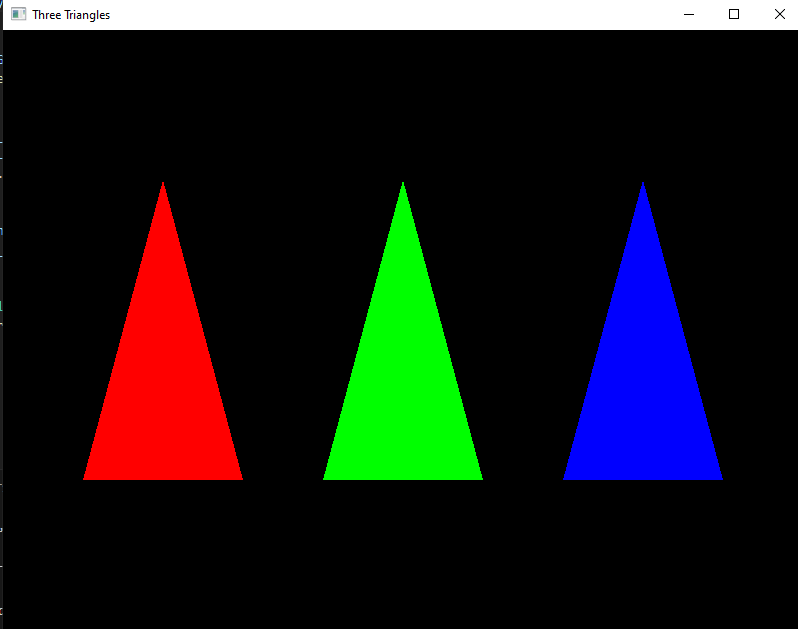


Рисунок 1