Báo cáo thực tập

Tóm tắt chung

Công việc tuần vừa rồi bao gồm :

- cài đặt các module core cho một ứng dụng bao gồm một Object Singleton quản lý cửa sổ GLFW_Window, Object Singleton quản lý việc xử lý event thay đổi kích thước màn hình từ người dùng.

WindowManager:

```h
#pragma once
#include <glad/glad.h>
#include <GLFW/glfw3.h>

#include "../Singleton.h"

#define WINDOW_MANAGER WindowManager::getInstance()

class WindowManager : public Singleton<WindowManager>
{
	friend class Singleton<WindowManager>;
public:
	WindowManager();
	~WindowManager();
	void render();

	void setWindow(GLFWwindow* window);
	GLFWwindow* getWindow();
private:
	GLFWwindow* m_window;
};
```

```cpp
#include "WindowManager.h"

WindowManager::WindowManager()
{
	m_window = nullptr;
}

WindowManager::~WindowManager()
{
}

void WindowManager::render()
{
	glfwSwapBuffers(m_window);
}

void WindowManager::setWindow(GLFWwindow* window)
{
	m_window = window;
	glfwMakeContextCurrent(m_window);
}

GLFWwindow* WindowManager::getWindow()
{
	return m_window;
}
```

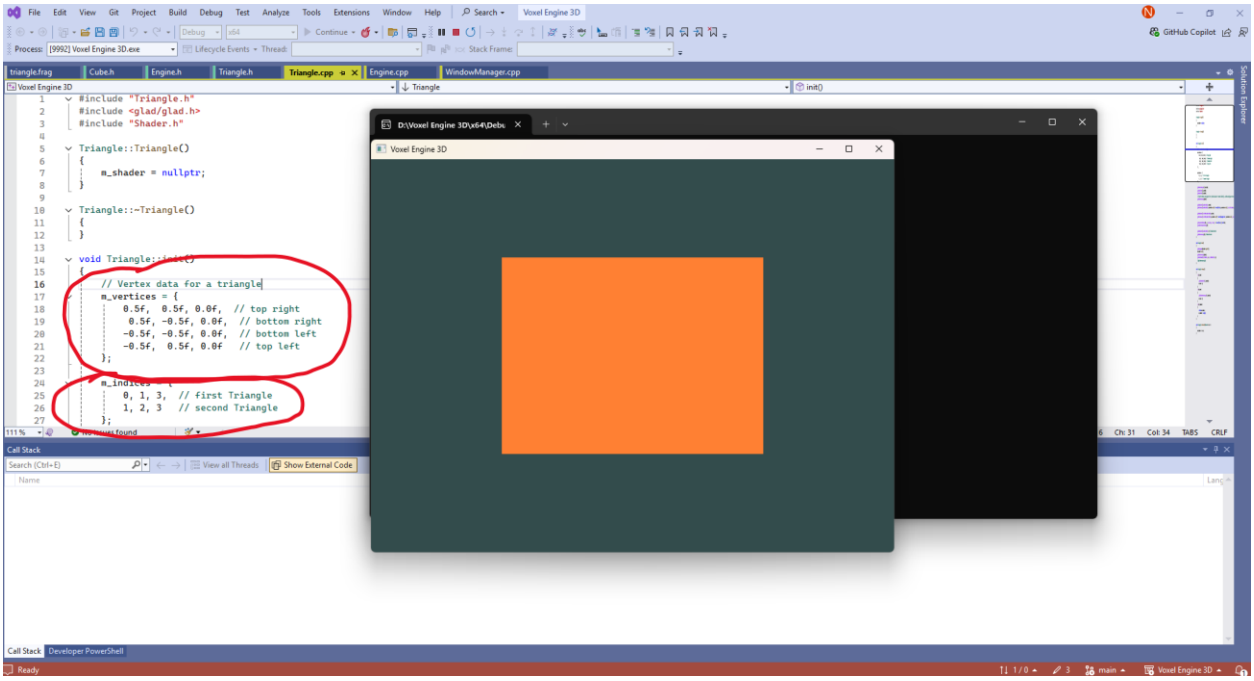
InputManager:

```cpp
#pragma once
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include "../Singleton.h"

#define INPUT_MANAGER InputManager::getInstance()

class InputManager : public Singleton<InputManager>
{
friend class Singleton<InputManager>;
public:
        InputManager();
        ~InputManager();

        void handleEvent();
        void processInput(GLFWwindow* window);
};
```

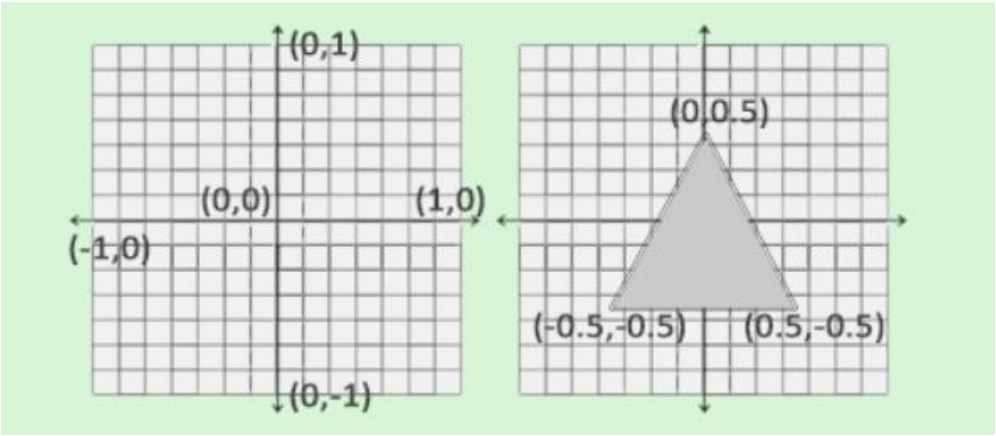
Đánh giá: 2 module hoạt động tốt, có khả năng quản lý và tối ưu cho chương trình, giúp source code clean.

- Áp dụng kiến thức đã học từ tuần trước, xây dựng một class Triangle kiểm tra việc tải các chương trình GLSL đơn giản.

Kết quả:



Giải thích: Hình chữ nhật được vẽ lên màn hình GLFW_Window được tạo bởi 2 tam giác. Để vẽ được 2 tam giác, chúng ta cần có tọa độ 4 đỉnh. Các tọa độ này được lấy từ tọa độ sau khi đã chuẩn hóa. Tọa độ chuẩn hóa là tọa độ sau khi đã được GPU tính toán bằng cách nhân các ma trận model, world , view và projection(đã giải thích từ báo cáo tuần trước) để tính toán tọa độ điểm sẽ được vẽ ở đâu trên màn hình GLFW_Window.



Theo quy ước chung của GLFW_Window, kích thước của màn hình được sinh ra bởi GLFW_Window sẽ được chuẩn hóa về tỉ lệ với trục x(-1, 1) và trục y(-1, 1).

Vertex Shader Program và Fragment Shader Progam tải xuống GPU để vẽ:

```
#version 330 core
layout (location = 0) in vec3 aPos;

void main()
{
    gl_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);
}
```

Vertex shader

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);
}
```

Fragment shader

Giải thích: Vertex shader nhận 1 tham số đầu vào, chính là tọa độ 4 điểm của 2 tam giác sau khi đã được tính toán chuẩn hóa theo kích thước màn hình từ trước. Vậy nên sẽ không cần phải thực hiện bất kì phép biến đổi nào khác.

Fragment shader không nhận tham số nào làm tham số đầu vào và trả về một giá trị là màu của điểm được xử lí. Ta gán giá trị cho tất cả các điểm giá trị vec4(1.0f, 0.5f, 0.2f, 1.0f). Giá trị này tương đương với màu cam trong bảng giá trị RGB.

Đánh giá kết quả: Việc tải được một mảng data vào GPU và vẽ là bước đầu để có thể tiến hành tải các mảng dữ liệu kích thước lớn hơn để có thể vẽ các đối tượng phức tạp hơn. Ngoài ra cũng kiểm chứng là các module và class shader hoạt động hiệu quả.

Công việc dự kiến tuần sau

Tiến hành vẽ các block lập phương trong không gian 3 chiều. Cài đặt lớp Camera để có thể tương tác di chuyển trong thế giới 3D. Thêm các function cho Object InputManager để xử lý sự kiện nhấn bàn phím từ người dùng. Xây dựng các chunk(là các khối được xây dựng từ các block lập phương – thường có kích thước 16x16x16).