

Nathan Willis FINAL GPU programming

by Nathan Willis

Submission date: 28-Mar-2025 02:47PM (UTC+0000)

Submission ID: 254102651

File name: 41436_Nathan_Willis_Nathan_Willis_FINAL_GPU_programming_650609_1438792908.docx (54.44K)

Word count: 2305

Character count: 14216



GPU Programmin g

Assignment 1

Nathan Willis
(2103532@student.uwtsd.ac.uk)

Contents

Edge Detection	2
Edge detection methods.....	2
How does Sobel edge detection work	2
Colour Inversion.....	3
Code Writeup	3
Main.....	3
Test Class.....	4
.h.....	4
.cpp.....	4
Shader	4
References.....	4
Appendix	5
CMakeHelloWorld.cpp.....	5
testFinal.cpp	7
testFinal.h.....	8
Final.shader.....	9

Edge Detection

Edge detection is a technique used in image processing to identify the boundaries of an image or more accurately in this case to identify the sudden changes in pixel colour values. The edge detection used in my code is a version of Sobel edge detection.[1] The changes in colour value can be used to identify the boundaries of an objects, shadows and textures, the end result of an image going through this edge detection process is a map of high contrast areas of the image in this case coloured in white.

Edge detection methods

Sobel edge detection is a type of gradient based edge detection as opposed to Laplacian based methods or the Canny method which combines the two approaches. Gradient based edge detection works by determining the gradient of colours in a grid of pixels and highlighting high contrast areas, Sobel is not the only gradient based edge detection method there are also Prewit and Roberts which use variations on grid size and weighting of the importance of neighbouring pixels this results in simpler and more computationally efficient methods of edge detection at the cost of more accurately detecting edges at complex angles.

Laplacian based methods involve smoothing the image using a gaussian filter to reduce noise and use something called the Laplacian Operator to find areas of high contrast this results in an edge detection method that while much more effective at ignoring noise and detecting edges at more convoluted angles is highly computationally inefficient. The Canny edge detection method that was mentioned previously uses the gaussian filter but relies on gradient based edge detection afterwards resulting in crisper edges with good awareness at any angle but still less computationally efficient than solely gradient based methods.[1]

Machine learning can be used to enhance edge detection by learning patterns from labelled image datasets. Instead of relying solely on predefined grids, models like convolutional neural networks learn features that represent edges more accurately, adapting to various image conditions. This improves precision and handles complex textures and noisy environments effectively.[2]

How does Sobel edge detection work

Sobel edge detection as stated previously is a type of gradient based edge detection it uses a grid of 3x3 pixels in the x and y directions and weights them thusly

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$
$$G = \sqrt{G_x^2 + G_y^2}$$

Where G_x is the change in the x direction and G_y is the change in the y direction.[3]

Colour Inversion

Colour inversion is a process by which the colours in an image are inverted to be their opposites, creating a negative of the image. It is a fairly straight forward image processing technique that is implemented on each individual pixel, in the implementation used in my code the colour value of every pixel in a given image is replaced by its complement by taking the rgb values of the pixel and subtracting them all from 1. For instance, if a pixels rgb values are (0.2,0.3,0.5) after processing the pixels new colour values would be (0.8,0.7,0.5) this is a very efficient very simple way of doing this.

Code Writeup

The code I used is a version of the code worked on in class with our tutor, it has been cleaned up and retooled for purpose. See appendix for all code discussed in this section.

Main

This program initializes a graphics application using OpenGL and GLFW for rendering and window management. It combines a custom rendering pipeline, mathematical utilities, and GUI capabilities via Dear ImGui for interactive control. The purpose is to set up a framework for testing visual effects and rendering techniques, with clean resource management and a dynamic main loop.

The program begins by initializing GLFW to create a window and manage the OpenGL context. It configures the OpenGL version and profile using GLFW window hints, targeting OpenGL version 3.3 core profile. After creating the window and making it the current OpenGL context, GLEW is initialized to load OpenGL extensions and manage function pointers. Once GLEW is ready, the program prints the OpenGL version for confirmation and enables V-Sync for smoother rendering.

Then ImGui is integrated for GUI functionality. The ImGui context is created, and the GLFW and OpenGL backends are initialized. A dark theme is applied to the GUI interface. The program uses a test system, allowing for dynamic registration and execution of different rendering tests. In this case, the `test` is registered, which handles effects like edge detection or colour inversion, as demonstrated in other code snippets.

The main rendering loop manages frame timing and rendering tasks. It calculates time between frames for updates, clears the screen, and handles ImGui frame rendering. If a test is active, it updates, renders the test content, and displays GUI controls specific to the test. Users can navigate back to the main menu from active tests or adjust parameters such as opacity interactively through GUI sliders though there is only one test given here.

Within the loop, ImGui processes user input, renders its interface, and integrates seamlessly with the application's rendering flow. The program swaps buffers at the end of each frame and polls events like keyboard and mouse input.

Finally, the program cleans up resources by deleting the current test, shutting down ImGui, destroying the window, and terminating GLFW. This ensures proper cleanup and memory management.

Test Class

.h

This code defines a class within the namespace for rendering tests. It includes methods for updating, rendering, and providing a GUI. The class manages key objects like a window, a full screen quad, a texture, and a shader. It also includes variables to control the selected effect and opacity. This structure allows for flexible rendering and user interaction.

.cpp

This code implements a graphical rendering test class using modern OpenGL and the ImGui library. It initializes resources, handles rendering, and provides a GUI for user interaction.

In the constructor, several objects are created and initialized. A shader program is loaded from a file and bound. A texture is also loaded and assigned to texture unit 0 for use in the shader. Uniform variables are set in the shader, including the texture unit, texel size, the effect type and opacity. These settings define how the texture will be rendered. The shader and texture are unbound after setup to prepare for rendering.

The function is defined but currently does not perform any operations. This is a placeholder for potential logic that might run every frame, such as animation updates or dynamic changes.

The function clears the screen and then binds the shader and texture. It updates the opacity uniform based on the current value of, then draws a full screen quad using these settings. After rendering, the shader and texture are unbound.

Finally, the function provides a slider in the ImGui interface, allowing users to adjust the opacity interactively between 0 and 1.

Shader

The shader file includes the vertex and fragment shaders. The vertex shader calculates the position of each vertex and sends texture coordinates to the fragment shader for further processing. The fragment shader processes the texture. On the left half, it highlights edges using gradients, edge detection, and on the right half, it inverts colours.

The fragment shader is doing the heavy lifting here so I will focus my explanation there. It contains 2 functions the edge detection function and the colour inversion function. The edge detection function uses Sobel kernels to sample neighbouring pixels, calculate gradients from their values, and combines the gradient magnitudes to highlight edges and the colour inversion function super simply takes the existing colour of a given pixel and subtracts the r g and b values from 1 giving the inverted colour of every pixel on the right side of the window.

The main function in the shader simply takes the x coordinate of every pixel determines if it is on the left side of the screen and if so, applies the edge detection function if not applies the colour inversion function.

References

References

- [1] "Comprehensive Guide to Edge Detection Algorithms."
<https://www.geeksforgeeks.org/comprehensive-guide-to-edge-detection-algorithms/>
(accessed Mar 28).
- [2] B. Tian and W. Wei, "Research Overview on Edge Detection Algorithms Based on Deep Learning and Image Fusion," *Security and Communication Networks*, vol. 2022, no. 1, pp.1155814, 2022. doi: 10.1155/2022/1155814. Accessed: Mar 28, 2025, [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/1155814>
- [3] G. M. H. Amer and A. M. Abushaala, "Edge detection methods," March 1, 2015. doi: 10.1109/WSWAN.2015.7210349. [Online]. Available: https://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7210349

Appendix

Working code video - <https://youtu.be/6hzoQrPuWY4>

CMakeHelloWorld.cpp

```
#include <GL/glew.h> // GLEW library for managing OpenGL
extensions
#include <GLFW/glfw3.h> // GLFW for window management and input
handling
#include <iostream> // For console output
#include <string>

#include "glm/glm.hpp" // GLM for mathematical operations and
matrix manipulation
#include "glm/gtc/matrix_transform.hpp" // For GLM transformations, like
translating matrices

#include "VertexBufferLayout.h" // Custom layout wrapper for vertex
attributes
#include "Renderer.h" // Custom renderer wrapper
#include "Shader.h" // Custom shader wrapper
#include "Mesh/Cube.h"
#include "tests/testFinal.h"
#include "tests/Tests.h"

#include "vendor/imgui/imgui.h" // Dear ImGui library for GUI elements
#include "vendor/imgui/imgui_impl_glfw.h" // ImGui GLFW backend
#include "vendor/imgui/imgui_impl_opengl3.h" // ImGui OpenGL backend

#define IMGUI_IMPL_OPENGL_LOADER_GLEW // ImGui macro to specify GLEW as the
OpenGL loader

int main() {
    // Initialize GLFW to create a window and manage the OpenGL context
    if (!glfwInit()) {
        std::cerr << "Failed to initialize GLFW \n";
        return -1;
    }

    // Set GLFW window hints for OpenGL version and profile
```

```

glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3); // Major version 3
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3); // Minor version 3
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE); // Core
profile

// Create a windowed mode window and OpenGL context
GLFWwindow* window = glfwCreateWindow(960, 540, "GLFW Window", NULL, NULL);
if (!window) {
    std::cerr << "Failed to create GLFW window" << std::endl;
    glfwTerminate();
    return -1;
}
glfwMakeContextCurrent(window);

// Initialize GLEW to manage OpenGL function pointers and extensions
glewExperimental = GL_TRUE; // Enable modern OpenGL functionality
if (glewInit() != GLEW_OK) {
    std::cerr << "Failed to initialize GLEW" << std::endl;
    glfwTerminate();
    return -1;
}
std::cout << "GLEW VERSION:" << glGetString(GL_VERSION) << std::endl;

glfwSwapInterval(1); // Enable V-Sync for smoother rendering

// Scoped block for managing OpenGL resources
//Resource Management (RAII): Using curly braces { ... } creates a block
scope in C++.
// When you define variables within this scope, they automatically go out
of scope (and are destroyed) when execution leaves that block.
// This helps with automatic resource management for objects like
VertexArray, VertexBuffer, IndexBuffer,
// and others that might be handling OpenGL resources (like buffers or
shaders).
// By structuring your code this way, you ensure that certain objects are
destroyed at the right time and their resources are released without needing
manual cleanup code.

Renderer renderer; // Renderer object for handling draw calls

// Setup Dear ImGui context for GUI rendering
ImGui::CreateContext();
ImGui_ImplGlfw_InitForOpenGL(window, true); // Initialize ImGui for
OpenGL + GLFW
ImGui::StyleColorsDark(); // Set default dark theme

if (!ImGui_ImplOpenGL3_Init()) {
    std::cerr << "ImGui OpenGL3 initialization failed" << std::endl;
    return -1;
}
test::Tests* currentTest = nullptr;
test::TestMenu* TestMenu = new test::TestMenu(currentTest);

currentTest = TestMenu;

TestMenu->RegisterTest<test::testFinal>("Edge Detection/Colour
Inverse", window);

float lastTimeFrame = 0.0f;
float deltaTime = 0.0f;

```

```

// Main rendering loop
while (!glfwWindowShouldClose(window)) {

    float currentFrameTime = static_cast<float>(glfwGetTime());

    deltaTime = currentFrameTime - lastTimeFrame;

    lastTimeFrame = currentFrameTime;

    renderer.Clear(); // Clear the screen to prepare for a new frame
    //renderer.ClearColour_White();

    // Start ImGui frame
    ImGui_ImplOpenGL3_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();
    if (currentTest)
    {
        currentTest->Update(deltaTime);
        currentTest->Render();
        ImGui::Begin("Test control panel");
        if (currentTest != TestMenu && ImGui::Button("<-"))
        {
            delete currentTest;
            currentTest = TestMenu;
        }
        currentTest->RenderGUI();
        ImGui::End();
    }
    ImGui::Render(); // Render ImGui frame
    ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData()); // Draw
ImGui to the screen

    // Swap front and back buffers
    glfwSwapBuffers(window);
    // Poll events (keyboard, mouse, etc.)
    glfwPollEvents();
}

delete currentTest;
if(currentTest != TestMenu)
    delete TestMenu;

// Shutdown ImGui and GLFW
ImGui_ImplGlfw_Shutdown();
// Cleanup resources and destroy window
glfwDestroyWindow(window);
glfwTerminate();

return 0;
}

```

testFinal.cpp

```

#include "testFinal.h"
#include "imgui.h"

test::testFinal::testFinal(GLFWwindow* window)
{
    m_Shader = std::make_unique<Shader>("res/shaders/Final.shader");
    m_Quad = std::make_unique<FullscreenQuad>();
    m_Texture = std::make_unique<Texture>("res/Textures/2.png");
}

```

```

        m_Shader->Bind();
        m_Shader->setUniformli("u_Texture", 0); //texture unit 0
        m_Shader->setUniform2f("u_Texel", m_Texture->getTexelSize().x,
m_Texture->getTexelSize().y);
        m_Shader->setUniformli("u_Effect", m_Effect);
        m_Shader->setUniformlf("u_Opacity", m_Opacity);
        m_Shader->Unbind();
        m_Texture->Unbind();
    }

    void test::testFinal::Update(float deltaTime)
    {
    }

    void test::testFinal::Render()
    {
        Renderer renderer;
        renderer.Clear();
        m_Shader->Bind();
        m_Texture->Bind();
        m_Shader->setUniformlf("u_Opacity", m_Opacity);
        m_Quad->Draw();
        m_Shader->Unbind();
        m_Texture->Unbind();
    }

    void test::testFinal::RenderGUI()
    {
        ImGui::SliderFloat("Opacity", &m_Opacity, 0.0, 1.0);
    }

```

testFinal.h

```

#pragma once
#include <memory>
#include "GL/glew.h"
#include <GLFW/glfw3.h>

#include "Tests.h"
#include "../Mesh/FullscreenQuad.h"
#include "../Texture.h"

namespace test
{
    class testFinal : public Tests
    {
    public:
        testFinal(GLFWwindow* window);

        void Update(float deltaTime);
        void Render();
        void RenderGUI();

    private:
        GLFWwindow* m_Window;

        std::unique_ptr<FullscreenQuad> m_Quad;
        std::unique_ptr<Texture> m_Texture;
        std::unique_ptr<Shader> m_Shader;

        int m_Effect = 0; //selected effect
        float m_Opacity = 1.0f;
    }
}

```

```

    };
}

```

Final.shader

```

#shader vertex
#version 330 core

```

```

layout(location = 0) in vec3 aPosition; //vertex position
layout(location = 1) in vec3 aNormal;   //Vertex Normal
layout(location = 2) in vec2 aTextureCoords; //Texture Coordinates

```

```

out vec2 v_TexCoords; //output to the fragment shader

```

```

void main()
{
    v_TexCoords = aTextureCoords;
    gl_Position = vec4(aPosition, 1.0);
};

```

```

#shader fragment
#version 330 core

```

```

in vec2 v_TexCoords;
out vec4 FragColor;

```

```

uniform sampler2D u_Texture;
uniform vec2      u_Texel;
uniform float     u_Opacity;

```

```

vec4 applyEdgeDetection()
{
    vec3 kernelX[3] = vec3[](vec3(-1,0,1),
                               vec3(-2,0, 2),
                               vec3(-1,0,1));

    vec3 kernelY[3] = vec3[](vec3(-1,-2,-1),
                               vec3( 0, 0, 0),
                               vec3( 1, 2, 1));

    //gradient in the x directions
    vec3 colourX = vec3(0.0);
    //gradient in the y directions
    vec3 colourY = vec3(0.0);

    for(int y = -1; y <= 1; y++)
    {
        for(int x = -1; x <= 1; x++)
        {
            vec3 sample = texture(u_Texture, v_TexCoords + u_Texel * vec2(x,
y)).rgb;

            colourX += kernelX[y + 1][x + 1] * sample;

```

```

        colourY += kernelY[y + 1][x + 1] * sample;
    }
}

//the maginitude of the gradient that is used to highlight edges
float edgeStrength = length(colourX) + length(colourY);
return vec4(vec3(edgeStrength), 1.0);
}

vec4 applyColourInversion()
{
    vec4 color = texture(u_Texture, v_TexCoords);
    return vec4(1.0 - color.rgb, 1.0);
}

void main()
{
    vec4 baseColour = texture(u_Texture, v_TexCoords);
    vec4 result = baseColour;
    if(v_TexCoords.x <0.5)
    {
        result = applyEdgeDetection();
    }else{
        result = applyColourInversion();
    }
    FragColor = mix(baseColour, result, u_Opacity);
}

```

Nathan Willis FINAL GPU programming

GRADEMARK REPORT

FINAL GRADE

45/100

GENERAL COMMENTS

The submitted code closely mirrors that provided during class exercises, with minimal adaptation or development beyond the original material. While it is acceptable to build on in-class examples, this submission does not demonstrate the student's ability to abstract the key concepts and apply them in an original context. As such, there is insufficient evidence of independent code development or of designing and implementing their own method for displaying textures and applying the intended visual effects.

Additionally, the written report showed limited engagement with relevant literature and lacked depth in analysis and critical evaluation.

===== MARKING GRID =====	Engagement with Literature Skills [20] - Review of GPUs/shaders
5	
Knowledge and Understanding Skills [10] - Understanding shader/GPU architectures - Evident in review + code	6
Cognitive and Intellectual Skills [25] - Critical review of literature - Problem analysis & solution design	10
Practical Application Skills [40] - Solution/testing in C++ OGL + shaders	20
Transferable/Professional Skills [5] - Report writing & referencing	4 =====

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11
