

AdamWilliams_GPUP_CW1

by Adam Williams

Submission date: 31-Mar-2023 11:06PM (UTC+0100)

Submission ID: 201244721

File name: 25476_Adam_Williams_AdamWilliams_GPUP_CW1_495096_1133019978.docx (880.96K)

Word count: 1951

Character count: 12514

GPU Programming Report

Title Page

Module Name:
GPU Programming (ACCB6014)

Assignment Title:
Shader Development

Student Name & Number:
Adam Williams (2008159)

Contents

GPU Programming Report	1
Title Page.....	1
Written Report.....	2
What is GPU Programming?	2
What does the GPU do in Graphics?	2
Real Time Ray Tracing	2
Phong Reflection Model.....	2
Code Report	3
Post Processing Effects Examples	3
Code Video Demonstration.....	6
Link	6
References	7
Appendix	8
"Source.cpp"	8
"fragShader1.glsl"	12
"fragShader2.glsl"	13

Written Report

What is GPU Programming?

"GPU Programming is a method of running highly parallel general-purpose computations on GPU accelerators" according to [1]. GPU programming is useful as in not just graphic scenarios as in recent years the GPU has been used more often to help solve problems that were previously only handled by the CPU. Some examples of scenarios where GPU's have been of use to accelerate video, digital image, and audio signal processing according to [2].

What does the GPU do in Graphics?

The GPU is responsible for the rendering of graphics such as windows, images, and videos. More powerful GPUs are needed for more GPU intensive commands such as rendering 3D models, playing video games, and video rendering. As stated by [3] a 20 years ago GPUs were only used for their typical uses such as accelerating processes related to real-time 3D graphics applications. They later go on to state that this has changed as of recently as GPU are "more programmable than ever before" and "accelerate a broad range of applications that go well beyond traditional graphics rendering." They later state these broader examples being use in video editing, content creation, and machine learning.

Real Time Ray Tracing

Ray Tracing is a rendering method used in computer graphics. This rendering method determines the colours of the pixels on your screen to display the current image. This method is gaining a lot of traction as it simulates the physical behaviour of light at a very high and realistic level. This has only gained traction in the past years as GPU's are becoming powerful enough to run intensive rendering methods like this in real time with the introduction of hardware such as the NVIDIA's RTX 30 series.

To better understand the strengths of real time ray-tracing its best to compare it to another method that it is deemed to supersede such as rasterization. [4] states that rasterization works by drawing the objects in the scene, and converts the 3D objects to the 2D plane with the use of transformation matrices. They later state that rasterization calculates the colour of a pixel by factoring in the colour, texture, normal, mesh of the model, with the lighting of the scene.

Ray Tracing differs from this as [4] explains it casts out rays from the point of view of the camera and traces the ray's path as it bounces around objects in the scene until a light source is reached. This method achieves higher quality results as the method follows the behaviour of light rays. Due to the nature of ray tracing it is a lot more intensive process which is why real time ray tracing was hard to achieve.

Real time ray tracing was made possible due to Microsoft's DXR framework, and NVIDIA's RTX Platform according to [4].

So as clearly stated ray tracing is higher quality, but its best to touch on in what aspects. Firstly, the shadows are a lot more realistic due to the ray tracing method used. The global illumination also appears to be much more realistic as ray tracing accurately calculates how light should interact with the colours of objects within the scene.

Phong Reflection Model

The Phong reflection model was made by Bui Tuong Phong during 1975. This is an older model which uses the the combination of the diffuse reflection and specular reflection to achieve its effect. Its common use is with Phong shading when used in computer graphics.

Code Report

Post Processing Effects Examples

The two post processing effects I have decided to code and make examples of is a Sepia filter, and a mask to achieve an average effect.

Sepia Effect

```
fragShader1.glsl  fragShader2.glsl*  Source.cpp
1  #version 430
2  //in vec3 col;
3  in vec2 texture_coordinates;
4  uniform sampler2D basic_texture;
5  out vec4 frag_colour;
6
7  void main(void) //sepia filter
8  {
9      frag_colour = texture(basic_texture, texture_coordinates);
10     //r = (.393, .769, .189) <- numbers used for calculation
11     //g = (.349, .686, .168) <- numbers used for calculation
12     //b = (.272, .534, .131) <- numbers used for calculation
13     frag_colour.r = 0.393*frag_colour.r + 0.769*(frag_colour.g) + 0.189*(frag_colour.b); //working out new value of the R value
14     frag_colour.g = 0.349*frag_colour.r + 0.686*(frag_colour.g) + 0.168*(frag_colour.b); //working out new value of the G value
15     frag_colour.b = 0.272*frag_colour.r + 0.534*(frag_colour.g) + 0.131*(frag_colour.b); //working out new value of the B value
16     frag_colour = vec4(frag_colour.r, (frag_colour.g), (frag_colour.b), 1.0);
17 }
18
19
```

The sepia effect is made within the fragShader2 file. I found the RGB values used in the calculation for a "Sepia Effect from [5].

How the sepia effect is applied is by making new RGB values for each pixel, this is done in lines 13 to 15, and applied on line 16. The way the new RGB values are calculated is by taking the current RGB the three number values from the pixels RGB (the red green and blue values) and multiplying each by a pre-set value and adding them together to get the new value. This is done three times (once for the red, green, and blue.

Below is the result achieved by this shader (original image on left, modified image taken from program on right).



Second Effect

```
fragShader1.glsl  fragShader2.glsl*  Source.cpp
1  #version 430
2  //in vec3 col;
3  in vec2 texture_coordinates;
4  uniform sampler2D basic_texture;
5  out vec4 frag_colour;
6
7  const float offset = 1.0 / 512.0;
8
9  void main()
10 {
11     vec2 offsets[9] = vec2[(
12         vec2(-offset, offset), // top-left
13         vec2( 0.0f, offset), // top-center
14         vec2( offset, offset), // top-right
15         vec2(-offset, 0.0f), // center-left
16         vec2( 0.0f, 0.0f), // center-center
17         vec2( offset, 0.0f), // center-right
18         vec2(-offset, -offset), // bottom-left
19         vec2( 0.0f, -offset), // bottom-center
20         vec2( offset, -offset) // bottom-right
21     )];
22
23     float kernel[9] = float[(
24         2.0/16.0, 3.0/16.0, 2.0/16.0,
25         3.0/16.0, 4.0/16.0, 3.0/16.0,
26         2.0/16.0, 3.0/16.0, 2.0/16.0
27     )];
28
29     vec3 sampleTex[9];
30     for(int i = 0; i < 9; i++)
31     {
32         sampleTex[i] = vec3(texture(basic_texture, texture_coordinates.st + offsets[i]));
33     }
34     vec3 col = vec3(0.0);
35     for(int i = 0; i < 9; i++)
36         col += sampleTex[i] * kernel[i];
37
38     frag_colour = vec4(col, 1.0);
39 }
40
```

The second effect uses the surrounding pixels to change the current pixel. The surrounding pixels are found during lines 11 to 21. It uses the variables "offset" and "-offset" to go +1 or -1 on the x or y axis respectively. The first value within the vec2 is the x axis, and the second value is the y axis. When all the surrounding. The values I had chosen in lines 24 to 26 were chosen to achieve an average effect. When the values were put higher the brightness of the image would become too bright losing the original image, whilst the values I have chosen still has weighting from the surrounding pixels to achieve an averaging effect, whilst not losing the original image's look.

Below is the result achieved by this shader (original image on left, modified image taken from program on right).



Code Video Demonstration

Link

https://youtu.be/k_WE02o4TbA

References

[1]

M. Levinas, “A Complete Introduction to GPU Programming With Practical Examples in CUDA and Python,” *Cherry Servers*, Sep. 30, 2021.

<https://www.cherryservers.com/blog/introduction-to-gpu-programming-with-cuda-and-python> (accessed Mar. 31, 2023).

[2]

D. Morelo, “GPU Programming Introduction,” *GPU Programming Introduction*, 2018.

<https://linuxhint.com/gpu-programming/> (accessed Mar. 31, 2023).

[3]

Intel, “What Is a GPU? Graphics Processing Units Defined,” *Intel*.

<https://www.intel.co.uk/content/www/uk/en/products/docs/processors/what-is-a-gpu.html> (accessed Mar. 31, 2023).

[4]

Unreal Engine, “What is real-time ray tracing?,” *Unreal Engine*.

<https://www.unrealengine.com/en-US/explainers/ray-tracing/what-is-real-time-ray-tracing> (accessed Mar. 31, 2023).

[5]

GeeksforGeeks, “Image Processing in Java - Colored Image to Sepia Image Conversion,” *GeeksforGeeks*, Nov. 19, 2016. <https://www.geeksforgeeks.org/image-processing-in-java-colored-image-to-sepia-image-conversion/> (accessed Mar. 31, 2023).

Appendix

"Source.cpp"

```
#include "GL/glew.h"
#include "GLFW/glfw3.h"
#include "glm/glm.hpp"
#include "../glm/ext.hpp" // perspective, translate, rotate
#include <iostream>
#include "utils.h"
#include "stb_image_loader.h"
using namespace std;

int direction = 0;
#define numVAOs 1
#define numVBos 3
// allocate variables used in display() function, so that they won't need to be
// allocated during rendering
GLuint mvLoc, projLoc;
int width, height;
float aspect;
glm::mat4 pMat(1.0), vMat(1.0), mMat(1.0), mvMat(1.0);
void setupVertices(void);
GLuint renderingProgram, renderingProgram1;
GLuint vao[numVAOs];
GLuint vbo[numVBos];

//Loading the image with the correct post processing effects
void init(GLFWwindow* window) {
    GLuint tex = 0;
    int x, y, n;
    int force_channels = 4; //RGB and A
    char filename[] = "frog.png"; //Getting the file name of the image and
    storing it into the variable
    unsigned char* image_data = stbi_load(filename, &x, &y, &n, force_channels);
    //x = x dimensions, y= ydims,
    if (!image_data)
    {
        cout << "error: could not load " << filename << endl;
    }
    // copy image into ogl buffer - stick to images power of 2 high and wide
    glGenTextures(1, &tex);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, tex);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, x, y, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image_data);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    renderingProgram = createShaderProgram("vertShader.glsl", "fragshader1.glsl");
    //load, compile, create, attach and linking to fragshader1's post processing effect
    (average mask effect)
    renderingProgram1 = createShaderProgram("vertShader.glsl", "fragshader2.glsl");
    //load, compile, create, attach and linking to fragshader2's post processing effect
    (sepia effect)

    glfwGetFramebufferSize(window, &width, &height); //Making the size of the
    window the size of the two images side to side
    aspect = (float)width / (float)height;
```

```

        setupVertices(); //initial vertices x,y,z values
        glEnable(GL_DEPTH_TEST);
    }

    //Displaying the image with the correct effects
    void display(GLFWwindow* window, double currentTime)
    //runs 60 times a second by default in glfw - see main loop
    {
        glClear(GL_DEPTH_BUFFER_BIT);
        glClearColor(0.8, 0.8, 0.9, 1.0);
        glClear(GL_COLOR_BUFFER_BIT);
        //First shader and draw left quad
        //First version of the image is loaded here and placed on the left, this is
        "renderingProgram1" which is using frashader2 which is the sepia effect
        glUseProgram(renderingProgram1); //specify linked program to use

        //get a reference to the shader variable Vcol
        //GLuint VcolLoc = glGetUniformLocation(renderingProgram, "Vcol");
        mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
        projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");

        // copy perspective and MV matrices to corresponding uniform variables
        glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
        glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
        // adjust OpenGL settings
        glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); //left quad bound to first VBO
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(0);

        glBindBuffer(GL_ARRAY_BUFFER, vbo[2]); //texture bound to third VBO
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(1);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glDrawArrays(GL_TRIANGLES, 0, 6); //draw left quad

        //*****Use Second shader to draw right quad
        //Second version of the image placed on the right, this is "renderingProgram"
        using fragsaahder1
        glUseProgram(renderingProgram); //specify linked program to use

        //get a reference to the shader variable Vcol
        //GLuint VcolLoc = glGetUniformLocation(renderingProgram, "Vcol");
        mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
        projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");

        // copy perspective and MV matrices to corresponding uniform variables
        glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
        glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
        // adjust OpenGL settings
        glBindBuffer(GL_ARRAY_BUFFER, vbo[1]); //right quad bound to first VBO
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(0);

        glBindBuffer(GL_ARRAY_BUFFER, vbo[2]); //texture bound to third VBO
        glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(1);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glDrawArrays(GL_TRIANGLES, 0, 6); //draw right quad
    }
}

```

```

int main(void) {
    if (!glfwInit()) { exit(EXIT_FAILURE); }
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    GLFWwindow* window = glfwCreateWindow(1200, 600, "glfw window", NULL, NULL);
    //glfwSetKeyCallback(window, key_callback);
    glfwMakeContextCurrent(window);
    if (glewInit() != GLEW_OK) { exit(EXIT_FAILURE); }
    glfwSwapInterval(1);
    init(window);
    while (!glfwWindowShouldClose(window)) {
        display(window, glfwGetTime());
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwDestroyWindow(window);
    glfwTerminate();
    exit(EXIT_SUCCESS);
}

void setupVertices(void) {

    float lquad[18] = {

        -1.0, 1.0, 0.0,
        0.0, -1.0, 0.0,
        -1.0, -1.0, 0.0,
        -1.0, 1.0, 0.0,
        0.0, -1.0, 0.0,
        0.0, 1.0, 0.0

    };
    float rquad[18] = {

        0.0, 1.0, 0.0,
        1.0, -1.0, 0.0,
        0.0, -1.0, 0.0,
        0.0, 1.0, 0.0,
        1.0, -1.0, 0.0,
        1.0, 1.0, 0.0

    };

    float quadTex[12] = {

        0.0, 1.0,
        1.0, 0.0,
        0.0, 0.0,
        0.0, 1.0,
        1.0, 0.0,
        1.0, 1.0 };

    //initialse VAO and VBOs
    glGenVertexArrays(numVAOs, vao); //we will use one VAO
    glBindVertexArray(vao[0]);
    glGenBuffers(numVBOs, vbo); //we have one VBO
    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]); //first VBO
    glBufferData(GL_ARRAY_BUFFER, sizeof(lquad), lquad, GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]); //first VBO
    glBufferData(GL_ARRAY_BUFFER, sizeof(rquad), rquad, GL_STATIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, vbo[2]); //2nd VBO
    glBufferData(GL_ARRAY_BUFFER, sizeof(quadTex), quadTex, GL_STATIC_DRAW);
}

```

}

```

"fragShader1.glsl"
#version 430
//in vec3 col;
in vec2 texture_coordinates;
uniform sampler2D basic_texture;
out vec4 frag_colour;

const float offset = 1.0 / 512.0;

void main()
{
    vec2 offsets[9] = vec2[](
        vec2(-offset, offset), // top-left
        vec2( 0.0f, offset), // top-center
        vec2( offset, offset), // top-right
        vec2(-offset, 0.0f), // center-left
        vec2( 0.0f, 0.0f), // center-center
        vec2( offset, 0.0f), // center-right
        vec2(-offset, -offset), // bottom-left
        vec2( 0.0f, -offset), // bottom-center
        vec2( offset, -offset) // bottom-right
    );

    float kernel[9] = float[](
        2.0/16.0, 3.0/16.0, 2.0/16.0,
        3.0/16.0, 4.0/16.0, 3.0/16.0,
        2.0/16.0, 3.0/16.0, 2.0/16.0
    );

    vec3 sampleTex[9];
    for(int i = 0; i < 9; i++)
    {
        sampleTex[i] = vec3(texture(basic_texture, texture_coordinates.st +
offsets[i]));
    }
    vec3 col = vec3(0.0);
    for(int i = 0; i < 9; i++)
        col += sampleTex[i] * kernel[i];

    frag_colour = vec4(col, 1.0);
}

```

```

"fragShader2.glsl"
#version 430
//in vec3 col;
in vec2 texture_coordinates;
uniform sampler2D basic_texture;
out vec4 frag_colour;

void main(void) //sepia filter
{
    frag_colour = texture(basic_texture,texture_coordinates);
    //r = (.393, .769, .189) <- numbers used for calculation
    //g = (.349, .686, .168) <- numbers used for calculation
    //b = (.272, .534, .131) <- numbers used for calculation
    frag_colour.r = 0.393*frag_colour.r + 0.769*(frag_colour.g) + 0.189*(frag_colour.b);
    //working out new value of the R value
    frag_colour.g = 0.349*frag_colour.r + 0.686*(frag_colour.g) + 0.168*(frag_colour.b);
    //working out new value of the G value
    frag_colour.b = 0.272*frag_colour.r + 0.534*(frag_colour.g) + 0.131*(frag_colour.b);
    //working out new value of the B value
    frag_colour = vec4(frag_colour.r, (frag_colour.g), (frag_colour.b), 1.0);
}

```

FINAL GRADE

60/100

GENERAL COMMENTS

Instructor

The similarity score is due to reuse of boiler plate framework code and generic library function calls. No issues here.

Code compiles and executes.

2 separate effects used. Good.

The colour effect is interesting and results in in little bloom/ over exposure. Did you consider a more complex/ localised effect perhaps?

Your review covers essential areas but lacks depth in places. Refs used to support discussions, assertions and justifications, but please try to use formal papers/texts - avoid too many electronic references.

Essential ideas and concepts, nevertheless identified. Some could be more detailed. Any omissions?

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

GRADING FORM: SMU GENERIC_GJ

ADAM WILLIAMS

0

RESEARCH-INFORMED LITERATURE

KNOWLEDGE AND UNDERSTANDING

ANALYSIS

PRACTICAL APPLICATION AND DEP.

SKILLS FOR PROFESSIONAL PRAC.