TDT4195: Visual Computing Fundamentals

Computer Graphics - Assignment 1

August 22, 2020

Michael Gimle
Bart van Blokland

Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)

- **Delivery deadline: September 4th, 2020 by 23:59.**
- **This assignment counts towards 4% of your final grade.**
- You can work on your own or in groups of two people. Please note that we have significantly higher expectations in this case.
- Deliver your solution on *Blackboard* before the deadline.
- Use the Gloom-rs project as your starting point.
- Do not include any additional libraries apart from those provided with Gloom-rs.
- Upload your report as a single PDF file.
- Submit your code folder as a .zip, making sure to not include the target directory.
- All tasks must be completed using Rust.
- Use only functions present in OpenGL revision 4.0 Core or higher. If possible, version 4.3 or higher is recommended.
- The delivered code is taken into account with the evaluation. Ensure your code is documented and as readable as possible.
- Currently MacOS support for OpenGL is spotty at best. We will not support MacOS in any way, shape or form. We strongly advise you to use Windows or Linux instead, if at all possible.

Questions which should be answered in the report have been marked with a **[report]** tag.

Objective: Familiarise yourself with Rust and OpenGL. Basic rendering of primitives, trying out Shaders.

Computer Graphics Assignments Overview

In the computer graphics assignments of this course we'll be creating an animated, interactive scene. Since this task is quite large to put in a single assignment, it has been split into three separate assignments, which will guide you through its implementation.

These three assignments are worth 15% of your final grade. Each assignment is created in such a way that the tasks you complete contribute towards this final product, while at the same time giving you a practical introduction in rendering realtime images with OpenGL.

This also means that it may be very difficult to complete the later assignments if you only finished one of the earlier assignments either in part or not at all. We therefore highly recommend you start working on these assignments early, and seek out help whenever possible if you get stuck.

The first assignment (worth 4%) intends to show you the basics of working with OpenGL.

The second assignment (worth 5%) focuses on introducing and understanding affine transformations.

The third and final assignment (worth 6%) gives an introduction to scene creation and combinations of transformations.

Along with the assignments, a guide to OpenGL is made available on Blackboard. It is highly recommended you read through it in its entirety. Don't worry; we've attempted to make it as helpful and easy to read as possible. We hope reading it in full will enable you to complete this first assignment with ease.

Please do *not* include the target folder in your submission, as it can get massive!

Task 0: Preparation [0 points]

a) Ensure the following tools are installed:

- Git
- cargo
- rustc

Please see the simple rust cookbook for how to get started with Rust.

b) Clone the Gloom repository using the command:

```
git clone https://github.com/mgimle/gloom-rs
```

c) Compile and run the project.

This should be as simple as running `cargo run` from a terminal.

You should see an empty window if compilation is successful and your GPU supports the OpenGL version required for these labs (4.0 or higher). If not, computers have been made available in room ITS-015 for you to use.

Task 1: Drawing your first triangle [2.5 points]

We recommend trying to complete all subtasks of this task at once, instead of one subtask at a time.

a) **[1.5 points]** Create a function which sets up a Vertex Array Object (VAO) containing triangles. You can modify the function signature that is located just above the main function in `main.rs`.

The function must take a vector of three dimensional vertex coordinates (usually `&Vec<f32>`) and an array of indices (`&Vec<u32>`).

The function must return the integer ID of the created OpenGL VAO set up to draw the input arrays.

The contents of the buffer can be assumed to exclusively contain triangles, specified by floating point coordinates.

Refer to chapter 2 of the OpenGL guide for detailed information on how to accomplish the objectives of this task.

OpenGL can be extremely "picky" about values and parameters it is given. A single incorrect parameter or input value can cause nothing to be rendered, and no error to be reported. If you experience problems, check your code carefully line by line. It is also recommended to come to the designated lab hours for help.

The project is designed to crash if it recieves any error from OpenGL, so if you get a crash, it should be possible to figure something out from the stack trace.

b) **[0.2 points]** In the handout code, you'll find a pair of basic shaders in the folder "shaders".

Load the shader pair defined in the source files. It is highly recommended you use the code in shader.rs to load the shader pair from the source files. Basic usage is described in main.rs.

For detailed information on using shaders, refer to chapter 3 of the OpenGL guide.

c) **[0.8 points]** **[report]** Define and instantiate a VAO containing at least 5 distinct triangles using the function you defined in a). Use the shader pair you loaded in b) to render the VAO.

All triangles must be visible. Put a screenshot of the resulting scene in your report.

Hint: these coordinates will give a visible triangle when you perform the drawing correctly:

$$v_0 = \begin{bmatrix} -0.6 \\ -0.6 \\ 0 \end{bmatrix}, v_1 = \begin{bmatrix} 0.6 \\ -0.6 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 0.6 \\ 0 \end{bmatrix}$$

Task 2: Geometry and Theory [1.5 point]

For the theory questions in this task, when evaluating the your answers we are looking for whether you have understood the concepts being asked about. If you answer in a short sentence, we are not always able to deduce this. As such we recommend spending some words on your answers wherever applicable.

a) **[0.4 point]** **[report]** Draw a single triangle passing through the following vertices:

$$v_0 = \begin{bmatrix} 0.6 \\ -0.8 \\ -1.2 \end{bmatrix}, v_1 = \begin{bmatrix} 0 \\ 0.4 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} -0.8 \\ -0.2 \\ 1.2 \end{bmatrix}$$

Put a screenshot of the result in your report.

This shape does not entirely look like a triangle.

Explain in your own words:

i) What is the name of this phenomenon?
ii) When does it occur?
iii) What is its purpose?

Hint: try to move vertex coordinates around and see what happens.

b) **[0.4 point]** **[report]** While drawing one or more triangles, try to change the order in which the vertices of a single triangle are drawn by modifying the **index buffer**. A significant change in the appearance of the triangle(s) should occur. Show a screenshot in your report.

i) What happens?
ii) Why does it happen?
iii) What is the condition under which this effect occurs? Determine a rule.

For the sake of clarity, you don't need to submit a separate version of your code just for this task alone (as well as any subsequent tasks). Just make the modifications, show the screenshot in your report, and continue modifying your code in the questions ahead. In the end you submit the code you've ended up with to blackboard.

c) **[0.5 point]** **[report]** Explain the following in your own words:

i) Why does the depth buffer need to be reset each frame?

ii) In which situation can the Fragment Shader be executed multiple times for the same pixel?

iii) What are the two most commonly used types of Shaders? What are the responsibilities of each of them?

iv) Why is it common to use an index buffer to specify which vertices should be connected into triangles, as opposed to relying on the order in which vertices are specified in the vertex buffer(s)?

v) While the last input of `gl::VertexAttribPointer()` is a pointer, usually a null pointer is passed in. Describe a situation in which you would pass a non-zero value into this function.

d) **[0.2 point]** **[report]** Modify the source code of the shader pair to:

i) Mirror the whole scene horizontally and vertically at the same time.

ii) Change the colour of the drawn triangle(s) to a different colour.

Put a screenshot of each effect in your report.

Explain briefly how you accomplished each effect.

Before you submit your work: ensure that each OpenGL function call (those of the form `gl..()`) is present in the OpenGL 4.0 Core library. You can use the website `http://docs.gl/` to only show GL4 functions, and search that list. If you have exclusively used the guide, this should not be necessary.

We would also like to ask you to pay special attention to the way we're asking you to deliver your work. Please deliver a single *ZIP* file (please no other archive formats!) in the state your code is in after completing the final task of this assignment. And please make sure to omit the target folder! If it takes a long time to compress your submission, you're doing it wrong.

Task 3: Optional Bonus Challenges [up to 0.2 bonus points]

This task is optional. These questions are meant as further challenges or to highlight things you may find interesting.

However, if you successfully complete at least one of these, you will receive a bonus of 0.2 points (although the maximum score of the assignment is still capped at 4%).

Please show us some eye candy if you managed to answer some or all of them! We'll try our best to put them up in a gallery on blackboard for everyone to see :)

a) The Fragment Shader has access to a predefined variable called `gl_FragCoord`. It is a variable of type `vec4` which contains the x and y-coordinate of the pixel/fragment on the screen.

   Use this fragment coordinate to draw a checkerboard on triangles (or something that looks like something along those lines).

b) Draw a circle.

c) Draw a spiral.

d) Draw a shape and have its colour change slowly over time (hint: use a uniform variable).

e) Draw a square by combining two triangles. Abuse the fragment shader to rasterise a triangle manually.

f) Implement a function which can read a 3D object from a file (for example an STL or OBJ file), and draw its contents.

g) Plot a sine function.