2/17/26[1]

Assignment 1
Real-time rendering using OpenGL
COMP 371 – Winter 2026
Dr. Tiberiu Popa

Introduction

In this assignment you are asked to render 3D models using OpenGL using a few custom-made shaders. **This assignment is individual and must be 100% original.**

What is given

This codebase is part of the course code repository available publicly in github. A link to this repository will be posted. We strongly advise to use *git clone* command rather than simply download it to pull updates if necessary. You should create your solution using the template available in `AssignmentTemplates/Assignment1/code/`

When you start building your solution, you will create a *src/* sub-folder where you can put as many C/C++ files as you wish. **This folder you will submit as your solution and all code in this folder must be completely original with no external resources.**

When you start building your solution you must follow the instructions below. These instructions will also be presented in the lab.

1. Enable in the CMakeLists.txt the definition of `STUDENT_SOLUTION`
2. The entry point to your code is a file *src/A1solution.h* that implements a *A1solution class.*
    a. The class must have a default constructor
    b. The class must have a *run(char*) or run(std::string)* member function that is called from the main function. The parameter is the name of the file to be opened. The format of this file is given below. Your run function should do the following:
        i. Open an OpenGL window (similarly with the capsules provided)
        ii. Loads and Renders a 3D model in an interactive loop
        iii. Listens to the keyboard and upon oppressing 's' it will toggle between 4 different rendering modes and pressing 'w' will toggle between the regular shading and a wireframe mode.

Your code must compile and run on the computers in the lab.

[1] This document might change throughout the term, please check moodle periodically and download the latest version.

Modes and Shaders

There are 4 rendering modes that correspond to 4 shaders: Phong shader, Flat shader, Circle shader and Voronoi shader.

- The Phong shader is the standard shader showed in class. You should use the following color written here as RGB triplets:
  - Diffuse: (1, 0.5, 0.5),
  - Ambient: (0.1, 0.05, 0.05),
  - Specular: (0.3,0.3,0.3),
  - specular exponent 5
  - Light should be white and maximum intensity (i.e. (1, 1, 1));
- The Flat shader is also the standard shader presented in class. Similar to the Phong shader with the difference that each vertex /face combination has a different normal, the normal of the face. Same color scheme as above.
- The Circle shader modulates the diffuse and specular color depending on where an individual fragment is located in relation to the inscribed circle in each triangle. The fragments outside the circle should have the same colors as above. The fragments inside the circle should have a diffuse color of (0.5, 0.5, 1), an ambient color of (0.05, 0.05, 0.1) and no specular highlights.
- The Voronoi shader modulates the diffuse and specular color depending on where an individual fragment is located in relation to the three vertices of its triangle. If it is closer the the first vertex the diffuse color should be (1, 0.5, 0.5), if it is closer to the second vertex the diffuse color should be (0.5, 1, 0.5) and if it is closer to the third one the color should be (0.5, 0.5, 1). The specular color should remain the same as in the Phong shader. Note that a simpler variation of the Voronoi shader is showcased in capsule2. You can use this as an example and starting point for both the Voronoi shader and for the Circle shader.

The *assets* folder provided in the codebase contains a number of examples of input files as well as snapshots taken from our solution for your reference. **Note that your code will be tested also on other examples that you have not seen before.**

File format

The command line takes as an argument a filename that contains the modelview matrix, perspective matrix, viewport and an array of vertices and triangles..
Your code should be able to read the information in the input file that follow the following simple format:

1) The first 16 numbers organized as 4 space separated numbers per line represents the modelview matrix,
2) The following 16 numbers organized as 4 space separated numbers per line represents the projection matrix,
3) The following 2 numbers organized as 2 space separated numbers represent the width and the hight of the window you need to create as well as the size of the viewport.
4) The following portion of the file contains the geometry to be rendered:
   a. Next number is the number of vertices in the file N.
   b. The following 3*N numbers are the position of the N vertices, where each vertex is represented as three floating point number corresponding to the x, y, and z coordinates
   c. Next number is the number of triangles in the file M.

d. The following 3*M numbers are the integer indices of the M triangle, where each integer is an index in the list of vertices starting at 0.

Personal Computers

The teaching team does not have the bandwidth to support any personal computers, **therefor we only support the setup in the lab**. However, this course uses open source, highly cross platform tools that should run easily on any of the 3 major platforms: Linux, Mac and Windows. For Linux and Mac using any reasonable package manager will easily install all the required dependencies. On Mac we recommend using Mac Ports[2].

For grading purposes, you must make sure that your solution builds and runs correctly on the lab machines.

Submission

You should develop your solution in the *src/* sub-folder. Your code will be called from the *main.cpp* and your implementation should match the interface provided.

You need to submit the entire *src* folder, which should contain your solution as a zip file: "<studentid>.zip". You should use the "zip" command on the lab machines (i.e. not tar, rar or other archiving software). For instance: "zip studentid.zip *src"*. This will archive the entire *src* folder into a file *studentid.zip*. When unzipped: *unzip raytracer.zip*, the *src* folder will be created and unpacked at the current location. A common mistake would be do type: "zip studentid.zip *src/"*. This will pack the contents of the folder and not the folder itself. This is incorrect.

You then submit the zip file using the EAS system as Assignment 1 before the respective deadline (see below). You can have as many submissions as you want, we will always consider the last one before the deadline. Therefore, make use of this feature and submit several times.

Following these instructions is very important as your code will be build and ran automatically on the lab machines. Failure to build and run according to the specifications may result in a mark of 0. To avoid such a situation you can submit early and we will do a fast sanity check 3 days and 5 days before each individual deadline. This sanity check will look if your code compiles and runs following the conventions mentioned in the assignment specifications.

The deadlines are firm and posted in the outline and on moodle.
No extensions possible except for medical reasons and with a medical note. Any late submission will receive a grade of 0.

Evaluation

The evaluation is done as a demo:

1. The code is automatically compiled and run on the lab machines using several unseen test files. The instructor will check the rendering of each of these files.
2. **The instructor may ask question about your code.**

[2] https://www.macports.org/

3. **You will be ask to make one or multiple change during the demo in 2-3 minutes each; All options of these changes are specified below. You should practice them ahead of the demo.**

Demo changes

As stated above, during the demo you will be asked to perform one or more changes to your program.
The changed that you will be asked to do are expected to be done in 2-3 minute so you should rehearse them ahead of time.
The list of possible changes are given here:

1) In the Flat shader, create a new vertex buffer where you can pass the index of the vertex. Modify the Phong shader to alternate the diffuse color between the default and (0.5, 0.5, 1) depending on the index of the triangle (mod 2). The 0 value should correspond to the default color.
2) Create a new uniform variable that can be passed to the Circle shader to have the radius of the circle a fraction of the inscribed radius. The number is a float and can have any non-negative value.
3) Create a new uniform variable that can be passed to the Voronoi shader that passes an integer that will modify the Voronoi shader to perform a cyclic permutation of the three colors based on the index mod three of the triangle.

Examples of these changes will be shown in the asset folder in the codebase.

Originality and Plagiarism

Your code needs to be 100% original. We run plagiarism tests and you are expected to explain your solution in detail to the instructor. You are ONLY allowed to use the code provided by the teaching team and located in the codebase repository. If you are in doubt, please consult the instructor.