CS350
Basic framework

1 Objective

The objective of this assignment is to review the OpenGL graphics pipeline and have debug framework for the upcoming assignments. The student is required to be able to debug draw certain geometries. Requirements:

- Have a rendering framework
- Debug draw geometries
    - Points
    - Segments
    - Triangles
    - Planes
    - AABBs
    - Sphere discs
- Have an interactive demo.
    - Right-click activates free flight movement:
        - `WASD` to move.
        - Space to go up.
        - Shift to go faster.
        - Mouse movement to rotate.

Provided tests looks like the following, and will save a screenshot that will be compared with the original (roughly):

```
TEST(basic, draw_segment)
{
    window w(window_width, window_height, test_name(), false);
    camera c;
    setup_scene(c);
    debug_system debug(&c);

    // White quad
    debug.draw_segment({0, 0, 0}, {1, 0, 0}, {1, 1, 1, 1});
    debug.draw_segment({1, 0, 0}, {1, 1, 0}, {1, 1, 1, 1});
    debug.draw_segment({1, 1, 0}, {0, 1, 0}, {1, 1, 1, 1});
    debug.draw_segment({0, 1, 0}, {0, 0, 0}, {1, 1, 1, 1});

    // Random
    debug.draw_segment({0, 1, 0}, {1, 0, 0}, {1, 1, 0, 1});
    debug.draw_segment({0, 0, 1}, {5, 3, -1}, {1, 0, 1, 1});

    save_screenshoot(window_width, window_height, current_test_image_filename());
}
```

2 Provided

The student is provided a basic project framework as starting point. This framework has several projects. The tests will make screenshots of debug draws and save them. Your objective is to have similar screenshots in terms of functionalities.

The frame is divided in the following sections:

- `/demo`: Demo executable
  - `/demo_renderer.cpp`: **Some of your work should be done here**. This file is not meant to be reused, will only be used in this assignment. Its purpose is to showcase the features you made in this assignment. It's suggested that camera controls are coded here. You are allowed to make "messy" code in this file, you may use global variables, statics, ...
- `/src`: Core features (many will be shared with upcoming assignments)
  - `camera.cpp/camera.hpp`: This class should keep track of the camera state. Should not be used for actual controlling, but it's acceptable.
  - `debug_system.cpp/debug_system.hpp`: **Most of your work should be done here**. Here is where the functions for debug drawing will exists. This system is meant to be completely autonomous, resources are created in its constructor, and released in the destructor. Since you will need the VP matrix for debug drawing, it also contains a pointer to the camera to be used.
  - `math.cpp/math.hpp/opengl.cpp/opengl.hpp`: External libraries wrappers. If there is any functionality regarding any of those libraries, it should exist in these files. For example, taking a screenshot (provided)
  - `primitive.cpp/primitive.hpp`: Holds information about a primitive that can be rendered (VBO+[IBO]+VAO+...)
  - `shader.cpp/shader.hpp`: Shader class, do not use previous implementations. Note that for this assignment, you are not reading the shader from any file.
  - `window.cpp/window.hpp`: OpenGL and window controller. Note that you are given a parameter to decide whether the window is visible or not. (reference)
- `/test`: Testing framework. **THIS FOLDER WILL BE REPLACED BY THE INSTRUCTOR**
  - `common.hpp`: Common for all tests. Note the variable `WORKDIR`:

    ```
    // Modify this variable if working directory is giving issues
    #ifndef WORKDIR
    #define WORKDIR "./"
    #endif
    ```

    Modify this variable so if your IDE workdir does not match the expected
  - `debug_draw.cpp`: Holds some tests for this assignment. Each of the tests will write a screenshot.

3 Common mistakes

- Passing primitives/shaders by value: If your constructor/destructors are doing OpenGL work, passing them by value will invoke these operations, are you sure you want to pass a copy and not the original?
- Returning `vec3` by reference: Perfectly valid, if you are not returning a local variable!
- Putting the m2w matrix inside the primitive: A primitive could be rendered a thousand ways! So they should not have the m2w
- Do not use `sprintf_s` (not portable), instead use `sprintf` (portable)
- If including a header from a library, do it in its corresponding header (e.g. `math.hpp`)
- Do not create a primitive/buffer per render call.

4 Recommend approach

1. **Make it compile** with placeholders (or alternatively, comment all tests). This will serve as a good overview of the features that you need to implement.
2. Create a en empty window and have a `ClearColor` so you know that is working. (Make sure you enable OpenGL callbacks to see that you don't make errors)
3. Uncomment a test and make it work (AABBs and spheres are probably the easiest). Output doesn't need to be **exactly** the same.
4. Suggestion: You will notice that most debug draw calls have similar sections (they use the same shader, set the same uniforms ...), make a common function for them.

5 Notes

- You may include extra files if you see fit. You may not include other libraries.
- Comment your code: Files + Functions headers too!
- **Use GIT as a control version to track your progress**
- There is an extra test for the Frustum, no extra credit will be provided. It's a feature that will be handy in the future.

6 Delivery

- Files to submit:
  - All source code!
  - CMakeLists.txt (as you may have added new files)
  - `.git` folder
- DO NOT SUBMIT BINARY FILES NOR INTERMEDIATE FILES