CS562 Assignment 1: Deferred Shading

OBJECTIVE

In this assignment you are going to implement a Deferred Shading renderer. This assignment is divided into the following parts:

1. Loading the provided scene.
2. A simple one light Deferred Shading implementation.
3. Add support for multiple lights.
4. Post-processing: Bloom.

1. Loading scene

First, create the framework that allows you to load and show a flat colored mesh on screen. You can use libraries in order to create the window (SDL2, GLFW) and load the GLTF meshes and material data (tinygltfloader), any other library not listed below is not be permitted without prior instructor permission.

- Math: GLM
- Graphics: OpenGL (glew, glad)
- Model Loading: tinygltf
- Platform: SDL2, glfw
- GUI: ImGui
- Textures: stb, SDL_image2
- JSON: nlohmann

glTF

The mesh format used for this course will be glTF, a format designed to conveniently load meshes in OpenGL. Since the format is really powerful and therefore it can be complex to handle every case, some extra restrictions are guaranteed for the files that will be used throughout this course.

- Primitives will always contain the following attributes per vertex: `POSITION, TEXTURE COORD, NORMAL, TANGENT`
- Primitives will always have an index buffer
- Material will have the following properties:
  - Diffuse color: `material.pbrMetallicRoughness.baseColorFactor`
  - Diffuse texture: `material.pbrMetallicRoughness.baseColorTexture`
  - Specular texture: `material.pbrMetallicRoughness.metallicRoughnessTexture`
    - Specular value: `Blue channel`
    - Shininess: `Green channel (should not be 0, set 0.01 when 0)`
  - Normal texture: `material.normalTexture`
- Each glTF will be considered a model, but it can contain multiple primitives on a hierarchy. Refer to the glTF documentation to see how the scene is defined.

Scene setup

In order to have consistent scenes to test a scene file will be provided. The file will contain an object list with the mesh and their corresponding transform information (scale, rotation and translation) and the information of a camera (assume there is only one) formatted in JSON. You will need to load the file in order to have the scene and be able to load different scenes. Note: rotation order for Euler angles: Z first, Y after and X last.

```
{
    "camera":
    {
        "translate":
        {
            "x": -90.0,
            "y": 40.0,
            "z": -20.0
        },
        "rotate":
        {
            "x": 0.0,
            "y": 90.0,
            "z": 0.0
        },
        "near": 0.1,
        "far": 500.0,
        "FOVy": 60.0
    },
    "objects":
    [
        {
            "mesh":
"data/gltf/sponza/Sponza.gltf",
            "translate":
            {
                "x": 0.0,
                "y": 0.0,
                "z": 0.0
            },
            "rotate":
            {
                "x": 0.0,
                "y": 0.0,
                "z": 0.0
            },
            "scale":
            {
                "x": 0.3,
                "y": 0.3,
                "z": 0.3
            }
        }
    ],
    "lights":
    [
        {
            "position":
            {
                "x": 0.0,
                "y": 10.0,
                "z": 0.0
            },
            "color":
            {
                "x": 1.0,
                "y": 1.0,
                "z": 1.0
            },
            "radius": 30
        }
    ]
}
```

2. Single Deferred Shaded Light

In order to have smaller milestones, the first submitted part will contain the loaded scene and the base framework with the scene lit by a single light. This framework will apply the lighting using the two passes. The first pass will fill the GBuffer with the needed geometry information and the second pass will compute the lighting for the given light with no optimization.

Light

For this assignment light data will be simplified to the following structure:

```
struct Light
{
    glm::vec3 position;  // position of the light
    glm::vec3 color;     // color of the light (specular will always be white)
    float radius;        // radius of the reach of the light
}
```

3. Multiple Small Point Lights

In this part of the assignment you are to augment the renderer that you already have to support multiple small point lights. The fragment shader should implement radial attenuation, and there should be a maximum distance from the point light beyond which no intensity reaches any objects – this will become important in order to optimize your implementation to efficiently support large numbers of small lights. You need to do the following:

1. Implement a multiple-pass approach so you can go above the maximum number of lights of a single-pass implementation.
2. Optimize the multiple-pass implementation.

4. Post-Processing: Bloom

Implement bloom/glow with the algorithm explained in class.

1. Brightness detection: extract the pixels with brighter than a specified threshold to a texture.
2. Blurring: blur the texture with the bright spots using a two-pass approach.
3. Composite: add the blurred image to the original one.

INPUT

With the goal of showcasing your work the best possible way the following specifications need to be meet:

- A first-person camera, controllable by the user via the keyboard and/or mouse.
  - WS: move the camera forward/backward along its forward vector.
  - AD: move the camera left/right along its side vector.
  - QE: move the camera up/down along the **global** up vector (0,1,0).
  - Click + Drag: the mouse should tilt the camera around its yaw and pitch.
  - Up/Down: rotate the camera around the pitch.
  - Left/right: rotate the camera around the yaw.

- Lights should be animated, and the user should be able to control the pausing/resuming of these animations.
- Modifying the amount of lights is mandatory.
- The size of the volume affected by the lights should be editable, all lights on the scene should be modified with a single controller.
- The G-Buffer should contain all the information necessary to compute the ambient, diffuse and specular terms of Phong's shading equation. The depth should also be rendered to a texture for potential use by later post-processing stages.
- You should be able to display the content of any of the textures in the G-Buffer, and the user should be able to switch between the displays of any of these textures.
- Pressing the **Control + R** key should reload the scene from the file.
- Pressing **F5** should reload all the shaders.

COMMAND LINE ARGUMENTS

The program should allow for the following arguments passed to the main function through the command line arguments.

- `argv[1]`: JSON scene name to load
  - Can be omitted, a default scene (the one provided) should be loaded in that case
- `argv[2]`: Filename of the screenshot to save at the end of the first frame in PNG format
  - Can be omitted and no screenshot is saved in that case
  - Can only be used if the scene is also specified
  - When provided, the application should close right after the screenshot is saved

Example: `cs562_jon.sanchez.exe scene.json screenshot.png`

SUBMISSION

The project will be submitted as a Visual Studio 2019 that will compile and execute without any adjustment. The solution will **only** have the **64-bit platform**, so the 32-bit configuration should be removed. The framework will be created from scratch and will have the following folder hierarchy:

- data -> Folder with all the resources
    - gltf -> Meshes and textures to load **(NOT included on submission)**
    - scenes -> Scene json files **(NOT included on submission)**
    - shaders -> Implementation of the shaders **(INCLUDED in submission)**
- src -> Folder with your code
    - Any header only library to compile (ImGui, stb…)
- cs562_<login>.vcxproj
- cs562_<login>.vcxproj.filters
- cs562_<login>.vcxproj.user
- Required DLLs (only x64 versions)
- include (all include folders for the libraries used)
- lib -> All library files for the libraries used)
- obj -> Folder where the compilation intermediate files when solution is compiled **(NOT included on submission)**
    - Debug
    - Release
- bin -> Folder where the executable will be generated when solution is compiled **(NOT included on submission)**
    - Debug
    - Release
- Solution file (cs562_<login>.sln)
- README.txt -> Text file containing header (see below) as well as controls for the application.

Both submissions (part 1 and part 2) will be submitted to Moodle with the same filename, **cs562_<login>_1.zip**, each part to the corresponding link.

README FILE

Along with your Visual Studio 2019 solution you will attach a README.txt file that will contain the following contents:

1. **How to use your program:** Every control available when executing the program and hints on how to get use it.
2. **Important parts of the code:** Specify source code files and approximate function or lines that were most relevant for the assignment.
3. **Known issues and problems:** Any not completed part of the assignment or known bugs that can be encountered when running the program.

GRADING SCHEME

Section	Item	Points
Scene and controls	Correct Scene and interface	10
Deferred Shading	Lighting	25
	Show each GBuffer Texture	10
	Show depth buffer with contrast	5
Multiple lights	Multipass Lights	15
	Optimization	10
Post-processing	Bloom	25