CS4515 - 3D Computer Graphics and Animation Elmar Eisemann, Ricardo Marroqium

Basic Graphics Techniques Assignment 1 - Part 1

In this deliverable you will combine parts of the weekly exercises into a single unified application. If you have completed the exercises you should be mostly done, except for the Transparency feature describe at the end of this document. We have provided a stripped down version of exercise 3 which you can extend to make it your own. Your application should be a single executable and should support the following features **without recompiling**:

The required tasks are marked at the end of each section by the header “To be implemented”.

Scene Loading from `toml` file

Your application should support loading the scene state, by specifying the path to a `toml` file (similar to the one used in practical 3.2). A couple of example scene config files are included in the resources directory.

The code to parse the `toml` file is already included. Please make sure the parser is working as you will be asked to read specific config files during the check-out moment. These files will be made available in Brightspace before the deadline. The name of the `toml` should be the first argument when calling the application,

e.g., `./Master_Assignment1_P1 resources/default_scene.toml`

For completeness we provide more explanation about the parser below, but you should not have to change this part of the code except to integrate with a UI and the rest of code to be implemented.

```
// utility function for parsing vec3s
std::optional<glm::vec3> tomlArrayToVec3(const toml::array* array) {

    auto len = array->size();

    if (len != 3) {
        return std::nullopt;
    }

    glm::vec3 output {};

    if (array) {
        int i = 0;
        array->for_each([&](auto&& elem) {
            if (elem.is_number()) {
                if (i > 2)
                    return;
                output[i] = static_cast<float>
        (elem.as_floating_point()->get());
                i += 1;
            } else {
                return;
            }
        });
    }

    return output;
}

// Program entry point. Everything starts here.
int main(int argc, char** argv) {

    // parse initial scene config
    auto config_path = "resources/default_scene.toml";
    toml::table config;
    try {
        config = toml::parse_file(config_path);
    } catch (const toml::parse_error& err) {
        std::cerr << "Failed to parse " << config_path << std::endl;
        return EXIT_FAILURE;
```

```
    }


    // read material data, if available
    shadingData.shininess = config["material"]
        ["shininess"].value<float>
        ().value_or(shadingData.shininess);
    shadingData.toonDiscretize = config["material"]
        ["toonDiscretize"].value<int>
        ().value_or(shadingData.toonDiscretize);
    shadingData.toonSpecularThreshold = config["material"]
        ["toonSpecularThreshold"].value<float>
        ().value_or(shadingData.toonSpecularThreshold);
    shadingData.kd = tomlArrayToVec3(config["material"]
        ["kd"].as_array()).value_or(shadingData.kd);
    shadingData.ks = tomlArrayToVec3(config["material"]
        ["ks"].as_array()).value_or(shadingData.ks);


    // read lights
    if (!config.contains("lights")) {
        std::cerr << "Lights not provided, using default light" <<
        std::endl;
        lights = std::vector<Light> {
            Light { glm::vec3(0, 0, 3), glm::vec3(1) },
        };
    }


    if (!config["lights"]["positions"].is_array()) {
        std::cerr << "Light positions must be array of vectors" <<
        std::endl;
        exit(EXIT_FAILURE);
    }


    if (!config["lights"]["colors"].is_array()) {
        std::cerr << "Light colors must be array of vectors" <<
        std::endl;
        exit(EXIT_FAILURE);
    }


    size_t num_lights = config["lights"]["positions"].as_array()-
        >size();
    for (size_t i = 0; i < num_lights; ++i) {
        auto pos = tomlArrayToVec3(config["lights"]["positions"]
        [i].as_array()).value();
        auto color = tomlArrayToVec3(config["lights"]["colors"]
        [i].as_array()).value();
        lights.emplace_back(Light { pos, color });
```

```
    }


    // read camera settings
    auto look_at = tomlArrayToVec3(config["camera"]
        ["lookAt"].as_array()).value();
    auto rotations = tomlArrayToVec3(config["camera"]
        ["rotations"].as_array()).value();
    float fovY = config["camera"]["fovy"].value_or(50.0f);
    float dist = config["camera"]["dist"].value_or(1.0f);


    // read render settings
    std::string illumination_model = config["render_settings"]
        ["illumination_model"].value_or("phong");
    bool shadows = config["render_settings"]
        ["shadows"].value_or(false);
    bool pcf = config["render_settings"]["pcf"].value_or(false);


    std::cout << "Illumination model: " << illumination_model <<
        std::endl;


    // read mesh
    bool animated = config["mesh"]["animated"].value_or(false);
    auto mesh_path = std::string(RESOURCE_ROOT) + config["mesh"]
        ["path"].value_or("resources/dragon.obj");


    // YOUR CODE HERE
}
```

Meshes

Here you specify the geometry to render.

```
[meshes]
path = "resources/meshes"
animated = true # or false
```

To be implemented

If `animated` is set to true, then `path` should be to a folder containing a list of obj files exported from blender. Your application should display the mesh as an animation. Alternatively if `animated` is set to false then `path` should be the path to a single `.obj` file and it should be loaded as a single, static frame.

Camera

Here you can specify the camera intrinsic (fov) and extrinsic (position/rotation) parameters. This is done by providing a `lookAt` vector, `rotations` and `distance`, as well as a `fovY` parameter.

```
[camera]
lookAt = [0.0, 0.0, -1.0]
rotations = [0.0, 0.0, 0.0]
dist = 2.0
fovy = 50.0
```

To actually set the camera position the `Trackball` class you can use the following snippet.

```
// fovY, look_at, rotations and dist all parsed from file

Trackball trackball { &window, glm::radians(fovY) };
trackball.setCamera(look_at, rotations, dist);
```

To be implemented

For the Camera you do not have to implement anything new but make sure that the correct camera parameters are being loaded (try for example changing the `toml` file and observing the effect).

Material

Here you can specify the material property of the mesh. For simplicity we will have a single material for the whole scene, and you can specify its params here:

```
[material]
kd = [1.0, 0.0, 0.0]
ks = [0.0, 1.0, 0.0, 1.0]
shininess = 3
toonDiscretize = 4
toonSpecularThreshold = 0.49
```

Note that `kd` and `ks` are RGB values loaded as `glm::vec3` arrays (see parsing code above).

To be implemented

Your application should additionally support modifying the material parameters via the UI and loading them into shaders.

Lights

Here you specify the data about the lights in the scene. `N` lights are specified by providing arrays of length `N` describing the properties of each light. Each light should have each property, even if a light does not have a texture a string needs to be provided (it can be empty however, since it is not used). In this example we only have a single light:

```
[lights]
positions = [[0.4, 1.2, 0.2]]
colors = [[0.9, 0.9, 0.9]]
is_spotlight = [true]
direction = [[0.707, 0.0, 0.707]]
has_texture = [false]
texture_path = ["resources/smiley.png"]
```

To be implemented

Additionally your application should support adding/removing lights and editing the color and position of lights via the UI. In the given code, the variable `selectedLightIndex` indicates which one is in use and should be passed to the shaders.

Render Settings

You should be able to specify the rendering settings used in the scene file. An example config is:

```
[render_settings]
diffuse_model = "debug" # or "lambert" or "toon" or "x-toon"
specular_model = "none" # or "phong" or "blinn-phong" or "toon"
shadows = true # or false
pcf = true # or false
```

- `diffuse_model` should be a string describing the diffuse material model to use is always one of `debug`, `lambert`, `toon` or `x-toon`.
- `specular_model` should be a string describing the specular material model to use is always one of `none`, `phong`, `blinn-phong` or `toon`
- When `shadows` is set to true you should render shadows using shadow mapping
- When `pcf` is set to true the shadows should be filtered with PCF like in the shadows practical.

To be implemented

All material models should be implemented with modern OpenGL, as described in practical 3.2. The debug material should show the normals and is already implemented. Your application should also include shadows via Shadow Mapping and PCF.

Your code should support switching between all the options above (material models, shadows and PCG) via the UI, but some test scenes might have them already set, so make sure you can load the settings directly from the file.

Transparency effect

This is the only new feature as it was only partially covered by the exercises.

To be implemented

Lastly your application should have a separate mode for a transparency effect implemented with depth peeling. It should display a series of semitransparent planes (or quads) behind each other. You should handle at least two layers of transparency. You should implement this feature using depth peeling to extract the semi-transparent planes and then compose them using alpha blending. Moreover, it should support editing the color and opacity of the planes via the UI. Don't forget to include an option to turn transparency on/off.

Submission

You should submit your code to Brightspace before the assigned deadline. Please make sure to **not** submit your build folder or any executables.

During the practical session on Friday October 4th a TA or lecturer will check you out. Before this day some extra scene files (`toml`) will be made available in Brightspace via an announcement. You should download these files and place them in your `resources` folder.

Check-out

During the check-out you will run your application from your computer. Make sure you have it compiled and ready to run. You will then be asked to follow some instructions. Some examples of instructions are:

- run the application with this `toml` file
- turn shadow map off/on
- turn shadow map on and PCG off/on
- create a new light source and move it around
- turn on Lambert mode for diffuse and Blinn-Phong for specular

Grading

Your grade for this deliverable is determined by the check-out procedure and a follow-up inspection of the code.

The grade of Assignment 1 is composed of three parts with equal weights. This current deliverable is Part 1 and is worth 1/3 of the Assignment 1 grade, which is equivalent to 20% of the Final Grade of the course.

As a reminder: to pass the course you need to achieve at least 5.0 in each Assignment 1 and Assignment 2 grades, and your final rounded grade should be at least 6.0.