

COMP5822M

Coursework 3

Contents

| | | |
|-----|----------------------------|---|
| 1 | Getting started | 2 |
| 2 | Tasks | 2 |
| 2.1 | Shared setup | 2 |
| 2.2 | Option: Pixelate | 3 |
| 2.3 | Option: Bloom | 3 |
| 2.4 | Option: Deferred Shading | 4 |
| 3 | Submission & Marking | 4 |
| A | Demo Receipt | 7 |



Coursework 3 revolves around render-to-texture methods. Render-to-texture is the foundation for many different techniques, including most post processing effects and deferred rendering. In CW 3, you will select one out of the following three options to implement:

- Post-processing: pixelate
- Post-processing: bloom
- Deferred shading

All three share a similar setup, but then diverge in slightly in implementation complexity. Study the three choices, and select the one that fits your circumstances. **Important:** *not all three choices give the same amount of marks! More complex choices will result in more marks.*

As always,, make sure to study this document in its entirety and plan your work before starting the coursework. Pay in particular attention to Section 3, which contains information about submission and marking.

*If you have not completed Exercises 1.1 through 1.4 and CW 1 & 2, it is heavily recommended that you do so before attacking CW 3. When requesting support for CW 3, it is assumed that you are familiar with the material demonstrated in the exercises! As noted in the module's introduction, you are allowed to re-use any code that **you have written yourself** for the exercises and previous courseworks in the CW submission. It is expected that you understand all code that you hand in, and are able to explain its purpose and function when asked.*

*While you are encouraged to use version control software/source code management software (such as git or subversion), you must **not** make your solutions publicly available. In particular, if you wish to use Github, you must use a private repository. You should be the only user with access to that repository.*

You are encouraged to build on the code provided with the coursework. If you have completed the exercises, you will already be familiar with the code and have some of the missing parts. If you wish to “roll your own”, carefully read the additional instructions that were presented in CW 1 (the same rules apply). *In particular, do pay attention to the section that states that you must get an OK from the module instructor before submitting/demo:ing!*

CW 3 uses the same project template as CW 1 and CW 2. See CW 1 and the exercises for details on this template.

1 Getting started

CW 3 focuses on post processing. You can continue from your CW 2 solution with either the Blinn-Phong or PBR shading model. *Important:* CW 3 uses a slightly modified `NewShip.obj` / `NewShip.mtl`, so make sure to use the models included with CW 3. The modified model adds emissive factors to a few more materials (this is mainly important if you choose to implement Section 2.3 - Bloom).

The `model.hpp` and `model.cpp` files have been updated again. However, these updates only matter if you wish to experiment with the Sponza model for e.g. deferred shading. The Sponza model relies on textures rather than material colors. If you want to render the Sponza, you will need to minimally load a diffuse texture. The Sponza model is downloaded separately. (Using the Sponza model is entirely optional, and you can get full marks without touching it.)

2 Tasks

The total achievable score for CW 3 is **20 marks**. CW 3 consists of a shared setup step (Section 2.1). After the shared setup, you should choose *one* of the three options defined in Sections 2.2 to 2.4. Mention which option you chose in your README.

As before, do not forget to check your application with the Vulkan validation enabled. Incorrect Vulkan usage -even if the application runs otherwise- may result in deductions. Do not forget to also check for synchronization errors via the Vulkan configurator tool discussed in the lectures and exercises.

General submission quality

3 marks

Up to **3 marks** are awarded based on general submission quality. Submission quality will be determined from several factors. Examples include overall readability/structure of your code (e.g., consistent style, naming and commenting), good coding practices, and how easy it is to build your code. Additionally, carefully read Section 3. Minor errors with respect to the instructions there may result in deductions of submission quality.

Your submission should solve the tasks presented in the coursework. Inclusion of large amounts of unnecessary code may result in deductions.

2.1 Shared setup

6 marks

So far, we've been drawing directly into the swap chain images, which have subsequently been presented to the user:

1. Acquire next swap chain image
2. Render pass: render 3D scene to swap chain image via associated framebuffer
3. Present swap chain image

In render-to-texture methods, we introduce additional steps into this process:

1. Render pass A: render 3D scene to intermediate texture image(s)
2. Acquire next swap chain image
3. Render pass B: perform post processing/deferred shading, using intermediate texture image as input and rendering results to swap chain image
4. Present swap chain image

For e.g., the bloom technique, there are multiple steps in the post processing, meaning that there are additional post-processing step. These would be similar to Step 3, but render to additional intermediate textures, and would likely take place between Steps 1 and 2. (It is also possible to swap the order of Step 1 and Step 2.)

This requires the following to be set up:

- Intermediate texture image(s) and associated framebuffer(s)
- Render passes A and B
- Full screen shader/pipeline that performs post processing/deferred shading.
- Descriptors, samplers, synchronization etc.



Figure 1: “Pixelate” effect from Section 2.2.

(You can probably repurpose the render pass from previous exercises/courseworks for use as Render pass A.)

It is recommended that you use the graphics pipeline for the post processing/deferred shading by drawing a full screen quad and performing the post processing/deferred shading computations in the fragment shader. The fragment shader will use the intermediate texture image drawn in the preceding steps as input. See Lecture 13’s slides for an efficient way of setting up and performing the full screen pass. To draw a single triangle without vertex buffers, refer to Exercise 1.2.)

Your choice of intermediate texture image format will depend on which option (Sections 2.2 to 2.4) you aim to implement.

Set up the infrastructure mentioned above (intermediate texture image(s), framebuffer(s), render pass(es), full screen shader etc). You can consider Section 2.1 to be successful if you can use the full screen shader to transfer data from the intermediate texture images to the swap chain image.

Pay attention to synchronization and consider what resources you need. In your README, briefly list your synchronization (e.g. what fences, semaphores, barriers and/or subpass dependencies did you need?).

2.2 Option: Pixelate

2 marks

Use the fragment shader to perform a “pixelation” effect (see Figure 1 for an example). Given a tile size of T , all pixels inside each $T \times T$ tile should receive the color of the pixel at the center of the tile. (The example uses $T = 8$ pixels.)

You must implement this as a post-process that uses an input texture that was rendered at the same resolution as the window. (That is, you are not allowed to just render the 3D scene at a lower resolution and then upscale the result.)

2.3 Option: Bloom

8 marks

Bloom is a very common technique that is used for a variety of effects. The most common effect is to strengthen the impression of very bright/glowing objects. You can see an example of this in the teaser image and in Figure 2. Here, the `NewShip.obj` has been slightly edited to add a strong emissive component to a few more materials. You can find additional examples in e.g. [Chapter 21](#) of the GPU Gems book and on the [Learn OpenGL](#) page on [Bloom](#).

The bloom technique that you shall implement consists of two steps:

- Filter out bright parts. The example uses a threshold of 1.0 and keeps pixels if any of the RGB components is over the threshold (i.e., pixels are kept if $\max(r, g, b) > 1$).
- Apply a Gaussian blur with a footprint of 44×44 pixels. (The example uses $\sigma = 9$, measured in pixels.)

You do not have to choose exactly these settings for your bloom - feel free to experiment a bit to find settings that look good.

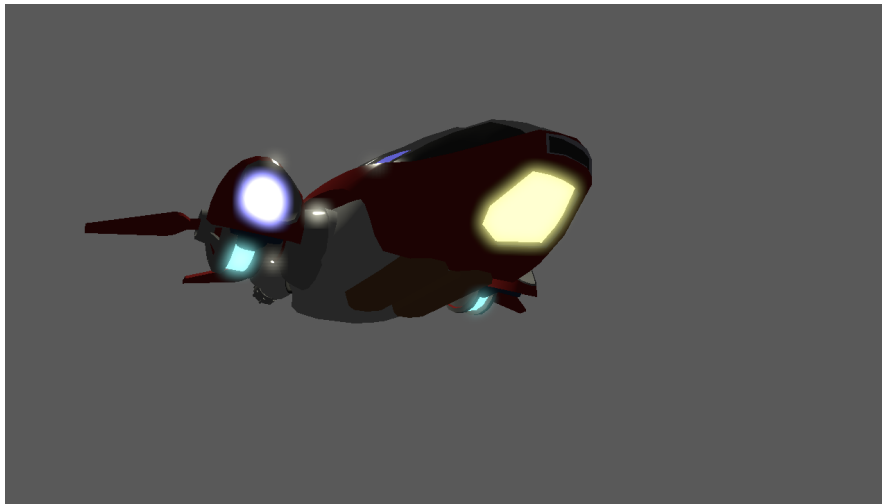


Figure 2: Bloom effect from Section 2.3. CW 3 contains a modified *NewShip.obj* that changes a few additional materials to be emissive.

For full marks, you must however do the following:

- For the intermediate texture, to which the 3D scene is rendered, you should choose a format that can store values ≥ 1 . Document your choice in the README and explain why you picked it.
- The Gaussian filter is separable, so it should be evaluated in two passes (a horizontal and a vertical). Figure 21-9 in [Chapter 21](#) (GPU Gems) illustrates the idea; *Learn OpenGL's* tutorial also discusses it.
- Use linear interpolation to reduce the number of taps (samples) that you need to take to evaluate. You can read about the optimization in [blog post](#) titled *Efficient Gaussian blur with linear sampling*.
- Derive your own weights for the filter and document the derivations in your README. (It is possible to perform the Gaussian blur in two passes with around 22 taps each.)

2.4 Option: Deferred Shading

11 marks


Implement deferred shading. See Lecture 13 for a quick overview on deferred shading.

Your G-Buffer must minimally store the albedo and normals. For full marks, you should come up with a reasonably efficient G-Buffer format, and you should compute each fragment's 3D position in the shading pass from the 2D fragment position (`gl_FragCoord.xy`) and the depth value from the depth buffer. You must render the G-Buffer in a single pass.

Document your G-Buffer format in your README. Explain why you picked that particular format.

Use three or more lights. You may use any appropriate shading model. You do not need to implement any optimizations that would reduce the number of lighting operations in deferred shading. (However, you might want to experiment with a fall-off for the light intensity; this was omitted in CW 2's shading models.)

3 Submission & Marking

Submission specifics are unchanged from previous courseworks, but are included here again to avoid any misunderstandings. If you studied the corresponding section in CW 1 or 2, you should already be familiar with the following. 

In order to receive marks for the coursework, follow the instructions listed herein carefully. Failure to do so may result in zero marks.

Your coursework will be marked once you have

1. Submitted project files as detailed below on Minerva. (Do *not* send your solutions by email!)
2. Successfully completed a one-on-one demo session for the coursework with one of the instructors. Details are below - in particular, during this demo session, you must be able to demonstrate your understanding of your code. For example, the instructor may ask you about (parts of) your submission and you must be able to identify and explain the relevant code.

3. If deemed necessary, participated in an extended interview with the instructor(s) where you explain your submission in detail.

You do *not* have to submit your code on Minerva ahead of the demo session.

Project Submission Your submission will consist of a single `.zip`, `.tar.gz` or `.tar.bz2` file. Other archive formats are not accepted. The submission must contain your solution, i.e.,

- Buildable source code (i.e., your solutions and any third party dependencies). Your code must at the minimum be buildable and runnable on the reference machines in the Visualization Teaching Lab (with the exception that you may use Vulkan 1.3 core features, which are not available on those machines).
- A README as a structured `.pdf` or as properly formatted Markdown (`.md`). The README should identify what code you have written (i.e., is part of your solution) and what code (if any) is from third parties. The README must list which tasks (Section 2) that you have attempted. If a task requires you to specify additional information, you include this in the README.
- A list of third party components that you have used. Each item must have a short description, a link to its source and a reason for using this third party code. (You do not have to list reasons for the third party code handed out with the coursework, and may indeed just keep the provided `third_party.md` file in your submission.)
- The `premake5.lua` project definitions with which the project can be built.
- Necessary assets / data files.

Your submission *must not* include any unnecessary files, such as temporary files, (e.g., build artefacts or other “garbage” generated by your OS, IDE or similar), or e.g. files resulting from source control programs. (The submission may optionally contain Makefiles or Visual Studio project files, however, the program must be buildable using the included `premake5.lua` file only.)

If you use non-standard data formats for assets/data, it must be possible to convert standard data formats to your formats. (This means, in particular, that you must not use proprietary data formats.)

Demo Session The demo session will take either place in person or online during the standard lab hours.

For in-person demos, bring a physical copy of the *Demo Receipt* page (Appendix A), pre-filled with your information. The instructor will sign both sides, and keep the second half (the first half is for you).

Demo sessions will take place on a FIFO (first-in, first-out) basis in the scheduled hours. You might not be able to demo your solution if the session’s time runs out and will have to return in the next session. *Do not wait for the last possible opportunity to demo your solution, as there might not be a next session in that case.*



Figure 3: Sponza with multiple light sources. You can find a modified Sponza model for download under the Assessments tab. Using the Sponza scene is entirely optional – you can complete Section 2.4 with the standard Ship model. If you do want to try to render the Sponza, you will need to load at least diffuse textures (Sponza does not define any particularly useful material colors). The scene is scaled by 0.03 (the original model is quite large, compared to the other models). The screenshots use six light sources with a quadratic fall-off. Light intensities are either [50, 50, 30] (middle corridor) or [60, 60, 10] (flanking one of the Lion heads).

A Demo Receipt

Please bring this page on an A4 paper if you are demo:ing your coursework in-person. Fill in the relevant fields (date, personal information) in both halves below. If the demo session is successful, an instructor will sign both halves. The top half is for your record. The instructor will take the bottom half and use it to record that you have successfully demo:ed your CW.

Please write legibly. :-)

Coursework 3 (Student copy)

Date.....

Name

UoL Username

Student ID

Instructor name

Instructor signature:

Coursework 3 (Instructor copy)

Date.....

Name

UoL Username

Student ID

Instructor name

Instructor signature:
