School of Computing: assessment brief

Module title	High Performance Graphics
Module code	COMP5822M
Assignment title	Coursework 3
Assignment type and description	Programming assignment: Render-To-Texture and Post-Processing
Rationale	Render-to-texture is a fundamental technique in real-time rendering pipelines that underpins many other methods. One example is post-processing. Modern engines commonly apply a number of post-processes to rendered images.
Page limit and guidance	Report: 7 pages, 10pt font size. You are allowed to use a double-column layout. Code: no limit. **Please read the submission instructions carefully!**
Weighting	25%
Submission deadline	2022-05-09
Submission method	Gradescope: code and report
Feedback provision	Written notes
Learning outcomes assessed	Graphics techniques; Shader programming
Module lead	Markus Billeter

1. **Assignment guidance**

   In Coursework 3, you will implement a Vulkan-based render-to-texture pipeline and use it to implement tone mapping and bloom.

   Before starting your work, please study the coursework document in its entirety. Pay special attention to the requirements and submission information. Plan your work. It might be better to focus on a subset of tasks and commit to these fully than to attempt everything in a superficial way.

2. **Assessment tasks**

   Please see detailed instructions in the document following the standardized assessment brief (pages i-iv). The work is split into four tasks, accounting for 25% of the total grade.

3. **General guidance and study support**

   Support will be provided during scheduled lab hours. Further support may be provided throught the module's "Teams" channel (but do not expect answers outside of scheduled hours).

4. **Assessment criteria and marking process**

   Submissions take place through Gradescope. Valid submissions will be marked primarily based on the report and secondarily based on the submitted code. See following sections for details on submission requirements and on requirements on the report. Marks and feedback will be provided through Minerva (and not through Gradescope - Gradescope is only used for submissions!).

5. **Submission requirements**

   Your coursework will be graded once you have

   (a) submitted project files as detailed below on Gradescope.

   (b) successfully completed a one-on-one demo session for the coursework with one of the instructors.

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

   Details can be found in the main document.

   Your submission will consist of source code and a report. *The report is the basis for assessment. The source code is supporting evidence for assertions made in the report.*

   Submissions are made through Gradescope (do *not* send your solutions by email!). You can use any of Gradescope's mechanisms for uploading the complete solution and report. In particular, Gradescope accepts `.zip` archives (you should see the contents of them when uploading to Gradescope). Do not use other archive formats

(Gradescope must be able to unpack them!). Gradescope will run preliminary checks on your submission and indicate whether it is considered a valid submission.

The source code must compile and run as submitted on the standard SoC machines found in the 24h teaching lab (2.15 in Bragg). Your code must compile cleanly, i.e., it should not produce any warnings. If there are singular warnings that you cannot resolve or believe are in error, you must list these in your report and provide an explanation of what the warning means and why it is acceptable in your case. This is not applicable for bulk warnings (for example, type conversions) – you are always expected to correct the underlying issues for such. *Do not change the warning level defined in the handed-out code. Disabling individual warnings through various means will still require documenting the warning in the report.*

Your submission must not include any "extra" files that are not required to build or run your submission (aside from the report). In particular, you must *not* include build artifacts (e.g. final binaries, `.o` files, ...), temporary files generated by your IDE or other tools (e.g. `.vs` directory and contents) or files used by version control (e.g. `.git` directory and related files). Note that some of these files may be hidden by default, but they are almost always visible when inspecting the archive with various tools. Do not submit unused code (e.g. created for testing). Submitting unnecessary files may result in a deduction of marks.

*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.*

The demo sessions will take place in person during the standard lab hours. You must bring a physical copy of the *Demo Receipt* page, pre-filled with your information. During the demo session, the instructor may ask questions to test your knowledge of the code. If satisfactorily answered, the instructor will sign both portions of the receipt, and keep the second half (the first half is for you).

6. **Presentation and referencing**

   Your report must be a single PDF file called `report.pdf`. In the report, you must list all tasks that you have attempted and describe your solutions for each task. *Include screenshots for each task unless otherwise noted in the task description!* You may refer to your code in the descriptions, but descriptions that just say "see source code" are not sufficient. Do **not** reproduce bulk code in your report. If you wish to highlight a particularly clever method, a short snippet of code is acceptable. Never show screenshots/images of code - if you wish to include code, make sure it is rendered as text in the PDF using appropriate formatting and layout.

   Apply good report writing practices. Structure your report appropriately. Use whole English sentences. Use appropriate grammar, punctuation and spelling. Provide figure captions to figures/screenshots, explaining what the figure/screenshot is showing and what the reader should pay attention to. Refer to figures from your main text. Cite external references appropriately.

Furthermore, the new UoL standard practices apply:

The quality of written English will be assessed in this work. As a minimum, you must ensure:

- Paragraphs are used
- There are links between and within paragraphs although these may be ineffective at times
- There are (at least) attempts at referencing
- Word choice and grammar do not seriously undermine the meaning and comprehensibility of the argument
- Word choice and grammar are generally appropriate to an academic text

These are pass/ fail criteria. So irrespective of marks awarded elsewhere, if you do not meet these criteria you will fail overall.

7. **Academic misconduct and plagiarism**

If you use any external resources to solve tasks, you must cite their source *both* in your report and in your code (as a comment). This applies both to code as well as to general strategies (e.g. papers, books or even StackOverflow answers).

Furthermore, the new UoL standard practices apply:

Academic integrity means engaging in good academic practice. This involves essential academic skills, such as keeping track of where you find ideas and information and referencing these accurately in your work.

By submitting this assignment you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

8. **Assessment/marking criteria grid**

(See separate document.)

COMP5822M Coursework 3

Contents

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 implement a render-to-texture setup and use it to implement a simple tone mapping operator and bloom.

If you have not completed Exercises 1.X and CW 1 and 2, it is highly 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 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 must build on the code provided with the coursework. In particular, the project must be buildable through the standard premake steps (see exercises). Carefully study the submission requirements for details.

1 Tasks

The total achievable score for CW 3 is **25 marks**. CW 3 is split into the tasks described in Sections 1.1 to 1.4. Each section lists the maximum marks for the corresponding task.

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.

> CW 3 uses the `NewShip.obj` object. It does not have any textures, but rather uses per-material parameters. The baking tool has been updated to also store the per-material parameter values. These include a base color, an emissive color, and a metalness and roughness (see relevant source files for details). The baking tool adds a fallback texture to each non-textured material. You can pass the per-material parameters as uniform values and multiply them with the value from the texture. You should otherwise reuse your CW 2 PBR shader (but make sure you add the emissive component).

1.1 Render-To-Texture Setup

10 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 (RTT) 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 some techniques, there are multiple steps in the post processing. These would function similarly to Step 3, but render to additional intermediate textures, and would likely take place between Steps 1 and 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.

You can probably repurpose the render pass from previous exercises/courseworks for use as Render pass A. (Also study the following tasks and consider the requirements of task that you want to solve before committing to this task.)

Use the graphics pipeline for the post processing by drawing a full screen quad/triangle and performing any post processing computations in the fragment shader. The fragment shader will use the intermediate texture image drawn in the preceding steps as input. See lecture 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.

Set up the infrastructure mentioned above (intermediate texture image(s), framebuffer(s), render pass(es), full screen shader etc). You can consider Section 1.1 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, consider what resources (Vulkan images, framebuffers, ...) you need, and how many of each resource are required.

In your report, briefly list your synchronization (e.g. what fences, semaphores, barriers and/or subpass dependencies did you need?). List the resources that you've created, mention how many of each your implementation requires, and explain why these are needed. Document your choices of intermediate texture formats.

1.2 Tone Mapping

3 marks

We have so far been trying to keep our color values in the $[0, 1]$ range. However, you can probably see that this is increasingly difficult with more complex shading models and with more complex lighting environments. Just the accumulated lighting from multiple light sources can easily results in color values that exceed 1.0. By default, such values are simply clamped to 1.0. This is not always ideal (color differences in bright regions disappear).

A solution to this is to render to a "HDR" (high dynamic range) framebuffer. A tone mapping operator then reduces the values back into the 0.0 to 1.0 range that we can show on a normal ("SDR") screen.

> With a "true" HDR display, i.e., in a system with an HDR monitor and an HDR swap chain, this is no longer strictly necessary. For example, with the HDR10(+) standard, color values indicate brightness between 0 nits and 10000 nits (physical quantities!). However, these values are still encoded in a buffer that ranges from 0.0 to 1.0 (e.g. using the `R10G10B10A2_UNORM` format). The mapping uses a specific curve to translate between the two values.

Implement the simple Reinhard tone mapping operator *in a post processing pass*:

$$c_{out} = \frac{c_{in}}{1 + c_{in}}$$

Here, c_{in} is an input color component in the range $[0, \infty)$ and c_{out} is the output color value in the range $[0, 1]$.

> More advanced tone mapping operators exist. These may emulate the exposure of a (physical) camera or take additional whole-scene properties into account. Unlike the simple Reinhard operator, such operators can only by implemented as post processes where the whole rendered image is available. However, for now, stick to the simple tone-mapping operator in the post process.

In your report include a pair of screenshots illustrating the rendering results before and after the tone mapping. You may, if necessary, adjust the parameters such as light intensity/number of light sources to highlight the effect. Describe what adjustments you have made.

1.3 Bloom

7 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. Here, the `NewShip.obj` has been slightly edited to add a strong emissive component to a some of the materials. (Make sure you update your shading from CW 2 to account for per-material emissive values.) You can also find a few examples of Bloom in action 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 conceptually includes three steps:

- Filter out bright parts.
- Apply a blur to the bright parts.
- Combine the original image with the blurred results.

The example uses a threshold of 1.0 in the first step and keeps pixels if any of the RGB components are over the threshold (i.e., pixels are kept if $\max(r, g, b) > 1$). It uses a Gaussian blur with a 44×44 pixel footprint (with $\sigma = 9$, measured in pixels).

For full marks, you must do the following (partial marks possible):

- Select appropriate texture formats for intermediate textures (see Section 1.1). Do not create unnecessary intermediate textures.
- Implement a 44×44 pixel footprint Gaussian blur filter.
- Derive your own weights for the filter. Use $\sigma = 9$ (in pixels). It is possible to perform the 44^2 Gaussian blur in just two passes with around 22 taps each.
- 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*.

In your report, show your results (screenshots). Outline your implementation, including what resources/intermediate textures are necessary (and why). Describe how you have calculated the weights. Describe how you make available the weights to the shader and why you have chosen this approach.

1.4 Performance

5 marks

Section 1.3 asks you to implement a very specific blur kernel. In this task, you will verify whether this is a good choice (or, indeed, disprove it!).

Measure the performance of the blur using appropriate techniques (see e.g. `vkCmdWriteTimestamp` for one possible choice). Be as selective as possible. Perform the measurement on at least two different types of graphics cards. (For example, compare the machines in the 24 teaching lab with their integrated GPUs against the machines in the Visualization Teaching Lab with their dedicated GPUs.) Document the machines' specs in your report.

Vary the number of passes/samples (e.g., more passes with fewer samples), while keeping the blur size constant. How does this affect performance? Are there any differences in behaviour between the two different types of machines?

Vary the size of the filter. How does the performance change? How far can you go?

In your report, describe how you have measured the performance. Detail which operations are included in your measurement. Document and analyze your observations from the measurements. What filter implementation would you recommend? Describe both the optimal filter for each machine you have tested and an overall compromise filter that is efficient on both machines.

2 Submission & Marking

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 Gradescope. (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 Gradescope ahead of the demo session.

Project Submission You will submit your solutions through Gradescope. You can upload a `.zip` file or submit through Github/Bitbucket. If successful, Gradescope will list the individual files of your submission. Additionally, automated tests will check your submission for completeness. Only solutions that pass these tests will be considered for marking.

The submission must contain your solution, i.e.,

- A report, named `report.pdf`. Only PDF files are accepted.
- Buildable source code (i.e., your solutions and any third party dependencies). Your code mustbe buildable and runnable on the reference machines in the Visualization Teaching Lab or in the 24h teaching lab.
- 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 code must be buildable in both debug and release configurations. It should compile without any warnings. Ask for assistance if you have trouble resolving a certain type of warning.

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 `permake5.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 sessions will take place in person during the standard lab hours.

Bring a physical copy of the *Demo Receipt* page (Appendix A), pre-filled with your information. If successful, the instructor will sign both portions of the receipt, 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.*

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 . . . .

The instructor(s) will fill in the following:

Instructor name . . . .

Instructor signature:

Coursework 3 (Instructor copy)

Date . . . .

Name . . . .

UoL Username . . . .

Student ID . . . .

The instructor(s) will fill in the following:

RTT Setup Tone mapping Bloom Performance

Instructor name . . . .

Instructor signature: