School of Computing: assessment brief

Module title	High Performance Graphics
Module code	COMP5822M
Assignment title	Coursework 4
Assignment type and description	Programming assignment: Shadowmaps
Rationale	Shadow maps are a standard technique for evaluating light visiblity during shading and thereby creating cast shadows. Cast shadows significantly increase realism in a scene and provide important cues on object's relative positions to each other.
Page limit and guidance	Report: 5 pages, 10pt font size. You are allowed to use a double-column layout. Code: no limit. **Please read the submission instructions carefully!**
Weighting	20%
Submission deadline	2022-05-29
Submission method	Gradescope: code and report
Feedback provision	Written notes
Learning outcomes assessed	Graphics techniques (shadow mapping)
Module lead	Markus Billeter

1. **Assignment guidance**

   In Coursework 4, you will implement shadow mapping using the Vulkan API.

   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 two tasks, accounting for 20% 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.

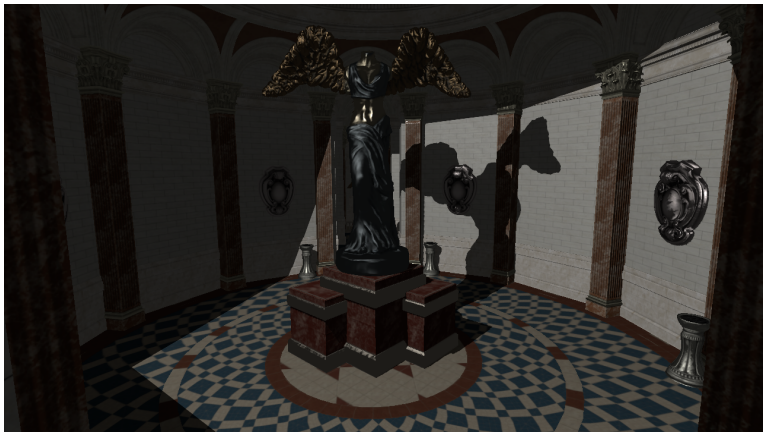
   By submitting this assignment you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

8. **Assessment/marking criteria grid**

   (See separate document.)

COMP5822M Coursework 4

Contents



In Coursework 4, you will implement shadow mapping.

If you have not completed Exercises 1.X and CW 1-3, it is highly recommended that you do so before attacking CW 4. When requesting support for CW 4, 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 4 is **20 marks**. CW 4 is split into the two tasks listed below.

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.

1.1 Shadow mapping

15 marks

The setup for shadow mapping is similar to the setup for post-processing (CW 3), in that you will draw the shadow maps into separate textures before drawing the main scene. Unlike post-processing, shadow mapping requires multiple geometry passes (i.e., passes where you rasterize the 3D scene). A possible outline follows:

1. Render pass A: render 3D scene from the light's point of view.
2. Acquire next swap chain image

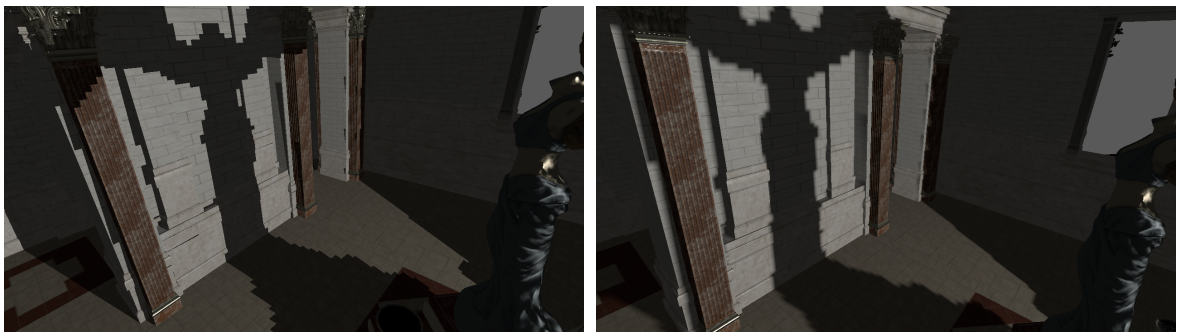


Figure 1: *Shadow map with and without PCF. The shadow map is low resolution on purpose, to make the effect of the PCF more visible. This uses just the simple built-in* 2×2 *filter.*

3. Render pass B: render 3D scene from the camera's point of view
4. Present swap chain image

Step 3 uses the shadow map from Step 1 to evaluate shadowing (light visibility) during the lighting computations. You may choose to combine shadow mapping with your Coursework 3 implementation, in which case there are additional steps between Step 1 and Step 2 (but this is not necessary). Generally, it is possible to render shadow maps early, before any other drawing steps.

> It is even possible to draw shadow maps outside of the per-frame processing paradigm that we have applied so far. For example, shadow maps may be reused over several frames if the scene or light position does not change significantly.

Start with a single 2D shadow map – this will result in a spot-light like light. One light source is sufficient. Determine a good shadow map resolution (note that 1024×1024 or larger might be necessary). Set up biases and related configuration to achieve good results and avoid e.g., surface acne and similar artifacts.

For full marks, you should consider the following:

- Shadow maps only use depth information. Ideally, you should create a framebuffer with only a depth attachment for Step 1. Consequently, the fragment shader for this pass will have no color outputs.
- Use built-in Vulkan and GLSL features where appropriate, as these might benefit from additional hardware acceleration. Relevant GLSL types/functions are: `sampler2DShadow` and `textureProj` (use the correct overload!); make sure to configure the Vulkan sampler correctly (see `VkSamplerCreateInfo`, specifically compare ops and border colors).
- Pre-compute the transformation into the light's projective space as an uniform value (ideally, the shadow lookup should be a single matrix multiplication and call to `textureProj`).
- Implement percentage closer filtering (PCF). Combine the built-in support for a 2×2 PCF filter with additional manual sampling to create a larger filter area. (You might want to test with lower shadow map resolutions to visualize the effects of this, see Figure 1 for an example.)
- Tweak the light's view frustum/projection to maximize the use of the available shadow map texels and the use of the depth-buffer resolution.

Renderdoc can be useful for debugging (e.g., looking at the shadow map contents).

In your report, describe your implementation (passes, pipelines/shaders, Vulkan resources, ...). Show your results. Include a comparison at different shadow map resolutions, and with/without PCF. Document your choices of parameters (e.g., shadow map resolution, projection settings, bias settings, ...), and motivate your choices. Discuss how these affect the results.

1.2 Extended shadow maps

5 marks

Add support for multiple light sources. There are several braziers in the scene (see Figure 2 for an example). Can you add a light source to/around each?

Experiment with different face culling modes when rendering the shadow map.

In your report, describe your implementation. Discuss the effects of the face culling. What culling mode would you recommend in this scene and why? Discuss different ways of rendering multiple shadow maps (e.g., multiple lights or multiple faces) can you find? What problems do they solve? What are the advantages?

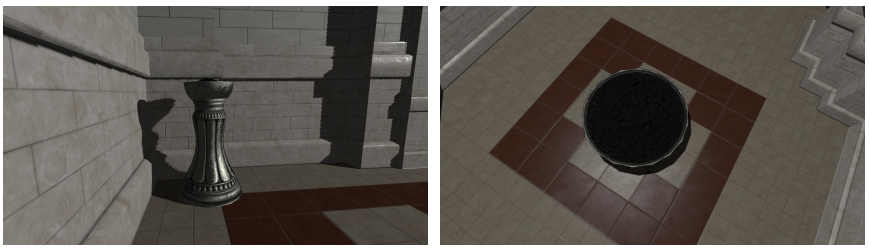


Figure 2: *Braziers in the Sun Temple scene. (Note that the lighting is just for illustration purposes, not how you should illuminate them.)*

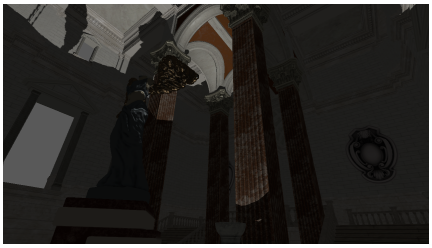


Figure 3: *Light "shining out" of a brazier (see bottom middle of the image). This requires a bit of a hack (modifying the scene); attempting to reproduce this is very very optional.*

What are the limitations? (You do not necessarily have to implement these variations. Refer to the Vulkan specification and online resources as necessary.)

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 4 (Student copy)

Date .....

Name .....

UoL Username .....

Student ID .....

The instructor(s) will fill in the following:

Instructor name .....

Instructor signature:

Coursework 4 (Instructor copy)

Date .....

Name .....

UoL Username .....

Student ID .....

The instructor(s) will fill in the following:

Shadow maps PCF Multiple lights

Instructor name .....

Instructor signature: