COMP5822M Coursework 2

Contents

Coursework 2 focuses on shading, specifically lighting models. You will mainly write shaders in GLSL (although some setup in Vulkan is still required to get the necessary data into place). CW 2 utilizes the material covered by the first four exercises, the lectures, and (optionally) the previous coursework. In CW 2, you will:

- Implement modified Blinn-Phong shading for reference
- Implement a physically-based shading model
- Animate the light source, and potentially shade with multiple light sources
- Implement a custom PBR model with some of your own choices

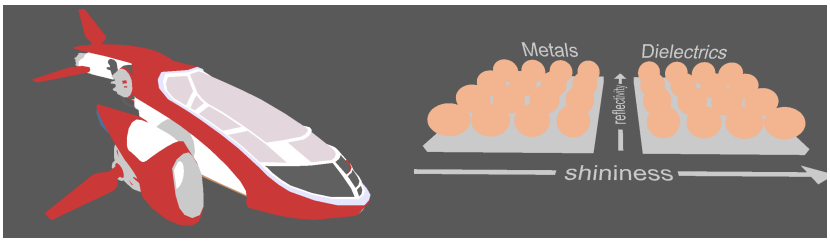
Before starting work on the CW, make sure to study this document in its entirety and plan your work. Pay in particular attention to Section 3, which contains information about submission and marking.

If you have not completed Exercises 1.1 through 1.4 and CW 1, it is heavily recommended that you do so before attacking CW 2. When requesting support for CW 2, 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 2 uses the same project template as CW 1. See CW 1 and the exercises for details on this template.



(a) `NewShip.obj` **(b)** `materialtest.obj`

Figure 1: *Starting point, prior to implementing any shading.*

1 Getting started

CW 2 focuses on shading, and to keep thing simple, the sample models use only solid per-material colors (no textures). Nevertheless, you have two choices for working with CW 2. If you have completed CW 1 successfully, you can continue to work on on your code base from it (but make sure to keep the CW 1 solution around, and submit it separately!). In this case, download the `cw2.zip` and transfer your CW 1 solution there. Make sure to update your code to load one of the CW 2 models instead (see Figure 1; models are located in the `assets/cw2` directory).

Warning: CW 2 uses a slightly different set of `model.{hpp,cpp}`, where the `MaterialInfo` struct has been updated to match the CW 2 requirements. This will likely require minor updates to your CW 1 code.

Alternatively, you can start with the `cw2-alt.zip` project. This project contains a pre-built DLL (Windows) / shared object (Linux) that implements a basic Vulkan rendering infrastructure to help you get going with CW 2 regardless of your status in CW 1.

Regardless of your choice, you make sure you see something akin to Figure 1 when you run your program (you may have to move the camera a bit, depending on where you set the initial location – the models are centered around the origin).

If you go digging into the material definitions of the OBJ files, you may find a few non-standard things. The provided OBJ files use a PBR extension, which defines a few new PBR-related material properties. You can find the details here. The OBJ loader that CW 2 uses, tinyobj, supports this extension.
Additionally, if you go digging into the source code (and later, into the solutions), you will find some *more* weirdness. In particular, the `Pr` (=roughness) material parameter's value is used as *shininess* instead. Further, the authors of the models intended to use the standard specular color, `Ks`, as a "reflectivity" parameter. However, we will not be interpreting it that way.

2 Tasks

The total achievable score for CW 2 is **30 marks**. CW 2 is split into the tasks described in Sections 2.1 to 2.6. 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.

General submission quality

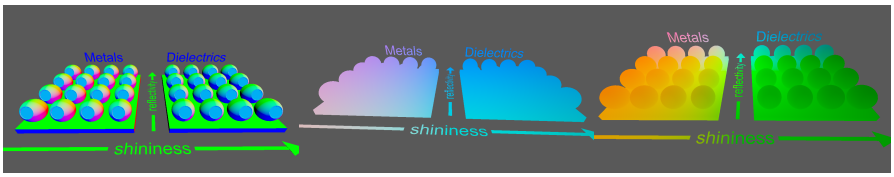
4 marks

Up to **4 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 Prep and Debug

4 marks



(a) *Normal* (**n**) **(b)** *View direction* (**v**) **(c)** *Light direction* (**l**)

Figure 2: *Visualization of the (normalized) normal vector, view direction and light direction. The visualization uses the* `materialtest.obj` *object, and the vectors are visualized with their world space values. You can use the fact that red maps to the x coordinate, green to y and blue to z to verify that the displayed values make sense – e.g., do the normals point into the direction you expect at each point?*

Before you start implementing either shading model, you will need to do some prep work. First, decide which space you want to perform your shading computations in. The document will refer to this space as the *shading space*. The recommendation is to perform shading in world space, but you can pick a different space if you wish (make sure you've studied all tasks before deciding). All shading should be done per fragment, i.e., in CW 2 we will not perform any per-vertex shading.

Make sure you get the necessary data into the right place.

- Make sure you pass the normals through the vertex shader to the fragment shader. (You may continue to assume that the models are defined in world space, and hence omit the model-to-world transform.)
- Make sure that the fragment shader has access to the fragment's position in the shading space.
- The fragment shader will need to have access to the camera's position in the shading space. (Note: you can just extend the per-scene transform information to include the camera's position and make sure it is accessible in both the `SHADER_STAGE_VERTEX` and `SHADER_STAGE_FRAGMENT` shader stages.)
- The fragment shader will need to have information about the scene's lighting data (e.g., a per-scene ambient light value and information about one or more light sources. Light sources are defined by their position and color).
- The fragment shader will need to get the correct material information.

For now, place the light source at the coordinates $[0, 9.3, -3]$ in world space, and give a color of $[1.0, 1.0, 0.8]$. Screen shots in this document will use those settings. Information about the light must be passed to the relevant shaders using an uniform buffer/interface block (i.e., do not hard-code the light's properties in the shaders).

Decide on a reasonable setup with descriptor sets and descriptor bindings. Introduce additional uniform buffer objects as necessary. Recall the `std140` layout rules (see Lecture 15's slides if you need a quick refresher).

It is a good idea to verify that things are working as they should. Change your shader to visualize the per-fragment normals, view direction and light direction. Make sure the values behave as expected (e.g., should they change with the camera position or not?). You can compare your results to the screenshots in Figure 2.

In your submission's `README`, document the following:

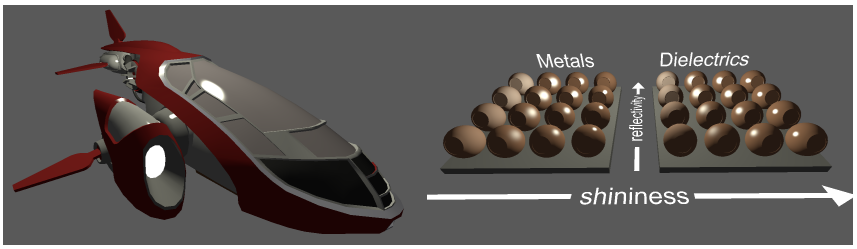
- Your choice of shading space
- Your choice of descriptor set layout and descriptor bindings. Motivate this choice *briefly*.

You may be asked to briefly show the visualizations of the above values (and/or other debug visualizations) in the live version of your program in the demo session.

2.2 Modified Blinn-Phong

6 marks

Now, it's time to implement the modified Blinn-Phong shading for reference. The modified version includes an extra normalization term that makes the model somewhat energy conserving when it comes to the specular highlights. This is a first step towards a more physically-inspired shading model.



(a) `NewShip.obj` **(b)** `materialtest.obj`

Figure 3: *Shading with the modified Blinn-Phong model. In the material test, there is no observable difference between the metals and dielectrics (non-metals). This is expected, as the Blinn-Phong model not distinguish between metals and non-metals. (Instead, one could try to tweak the material specular to emulate the look of metallic materials; however, the author of the models here has clearly not done so.)*

The modified Blinn-Phong model is defined as follows:

$$\begin{aligned}\mathbf{L}_o = \mathbf{c}_{\text{emit}} &\\ &+ \mathbf{c}_{\text{ambient}} \otimes \mathbf{c}_{\text{diff}} \\ &+ (\mathbf{n}\cdot\mathbf{l})_+ \frac{\mathbf{c}_{\text{diff}}}{\pi} \otimes \mathbf{c}_{\text{light}} \\ &+ \frac{\alpha_p + 2}{8} (\mathbf{n}\cdot\mathbf{h})_+^{\alpha_p} (\mathbf{n}\cdot\mathbf{l})_+ \mathbf{c}_{\text{spec}} \otimes \mathbf{c}_{\text{light}} \\ = \mathbf{c}_{\text{emit}} + \mathbf{c}_{\text{ambient}} \otimes \mathbf{c}_{\text{diff}} &+ \left(\frac{\mathbf{c}_{\text{diff}}}{\pi} + \frac{\alpha_p + 2}{8} (\mathbf{n}\cdot\mathbf{h})_+^{\alpha_p} \mathbf{c}_{\text{spec}} \right) (\mathbf{n}\cdot\mathbf{l})_+ \otimes \mathbf{c}_{\text{light}},\end{aligned} \tag{1}$$

where

- α_p Material shininess
- $\mathbf{c}_{\text{emit}}$ Material emissive color
- $\mathbf{c}_{\text{diff}}$ Material diffuse color
- $\mathbf{c}_{\text{spec}}$ Material specular color
- $\mathbf{c}_{\text{light}}$ Light color
- $\mathbf{c}_{\text{ambient}}$ Scene ambient light color
- $\mathbf{n}$ Surface normal (normalized)
- $\mathbf{l}$ Light direction (normalized), pointing *towards* the light
- $\mathbf{v}$ View direction (normalized), pointing *towards* the camera/viewer
- $\mathbf{h}$ Half vector (normalized), computed from the light and view directions

The $\otimes$ operator denotes a element-wise multiplication of two vectors/tuples, and $(\mathbf{a}\cdot\mathbf{b})_+$ denotes a "clamped" dot-product, $(\mathbf{a}\cdot\mathbf{b})_+ = \max(0, (\mathbf{a}\cdot\mathbf{b}))$.

The factor $(\alpha_p + 2)/8$ is the normalization factor mentioned earlier. Additionally, the specular term is multiplied with $(\mathbf{n}\cdot\mathbf{l})_+$. You can (optionally) read more about the derivation of this modified model in the course notes for *Crafting Physically Motivated Shading Models for Game Development* by Hoffman [2010], where the reasons for the additional terms are explained in detail. The version presented there includes an additional Fresnel-term that is omitted here. Computation of the normalization factor involves some simplification and assumptions, meaning that you can occasionally find different variants of it.

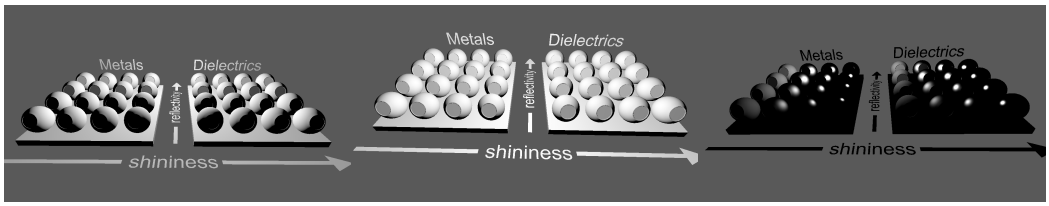
Implement the modified Blinn-Phong model (Equation (1)) and use it for shading. Compare your output to Figure 3. If you need to debug your shaders, see Figure 4 for visualization of some of the intermediate values (also check that the various values are behaving as they should!).

Important: In your submission, make sure to include a separate pair of shaders (vertex and fragment) that implement the Blinn-Phong shading. (If you use the CW 2 project template, use the `BlinnPhong.vert` and `BlinnPhong.frag` shaders).

2.3 Physically-inspired model

8 marks

You will now implement a slightly more complicated shading model, specifically a physically-inspired model that has a Lambertian diffuse component and a specular component based on a microfacet BRDF using the Blinn-Phong normal distribution function.



(a) $(\mathbf{n} \cdot \mathbf{l})_+$ **(b)** $(\mathbf{n} \cdot \mathbf{h})_+$ **(c)** *Blinn-Phong specular term*

Figure 4: *Visualization of the some of the components of the modified Blinn-Phong model.*

To introduce the model, we will start with the Rendering Equation, as shown in the lectures:

$$\mathbf{L}_o = \mathbf{L}_e + \int_\Omega f_r \, L_i \left(\omega\right) \, \left(\mathbf{n} \cdot \mathbf{l}\right) \, d\omega$$

We are dealing with discrete point lights, which are the only points in space that emit light, and ambient light. Consequently, we can sum over the light sources instead of integrating over a hemisphere (where the ambient light approximates all indirect illumination):

$$\mathbf{L}_o = \mathbf{L}_e + \mathbf{L}_{\text{ambient}} + \sum_{n=0}^{N-1} f_r \, \mathbf{c}_{\text{light},n} \, \left(\mathbf{n} \cdot \mathbf{l}_n\right)_+$$

For now, we'll focus on a single light source, meaning we can drop the sum (and the index n) all together:

$$\mathbf{L}_o = \mathbf{L}_e + \mathbf{L}_{\text{ambient}} + f_r \, \mathbf{c}_{\text{light}} \, \left(\mathbf{n} \cdot \mathbf{l}\right)_+ \tag{2}$$

The general form of the equation is already somewhat reminiscent of the modified Blinn-Phong model introduced earlier. In short, the BRDF (f_r) describes how much of the incoming light ($\mathbf{c}_{\text{light}}$) is reflected towards the camera/viewer.

For the BRDF, we will use a general microfacet model for isotropic materials [Burley, 2012]:

$$f_r \left(\mathbf{l}, \mathbf{v}\right) = \mathbf{L}_{\text{diffuse}} + \frac{D \left(\mathbf{n}, \mathbf{h}\right) \, \mathbf{F} \left(\mathbf{l}, \mathbf{h}\right) \, G \left(\mathbf{n}, \mathbf{l}, \mathbf{v}\right)}{4 \, \left(\mathbf{n} \cdot \mathbf{v}\right)_+ \, \left(\mathbf{n} \cdot \mathbf{l}\right)_+},$$

where $\mathbf{L}_{\text{diffuse}}$ is the diffuse contribution, and the specular contribution is constructed from the Fresnel term F, the normal distribution function D (Figure 5a) and the masking function G (Figure 5b).

Some care must be taken if one attempts to evaluate the above term directly. The denominator can become zero, which, in practice, produces a NaN value. This then cascades through the following computations. A simple workaround is to add a small ϵ to the denominator before the division. Alternatively, one can try to cancel out some of the terms in the division (e.g., compare to Equation (2)), and potentially avoid problems in the first place.

In theory, we need to differentiate between metals and dielectrics (non-metals), as these behave quite differently. Metals only reflect on the surface, meaning that they have a zero diffuse contribution. Additionally, the (specular) reflection from a metal is tinted. In contrast, dielectrics have both a diffuse and specular contribution, where only the diffuse one is colored. The specular reflection tends to have the same spectrum/color as the incoming light.

In practice, it is possible to merge the two cases into a single approximative method. The underlying idea revolves around the observation that the amount of specular base reflectivity. F_0, of dielectrics is *relatively* similar for most materials. The value 0.04 is frequently used as an approximation across the board. For dielectrics, the material's color then determines the diffuse color. For metals, the material's color is instead used to control specular base reflectivity:

$$\mathbf{F}_0 = \left(1 - M\right) \, \left[0.04, 0.04, 0.04\right] + M \, \mathbf{c}_{\text{mat}}$$

M describes the *metalness* of the material. In reality, a material is either a metal ($M = 1$) or a dielectric ($M = 0$). However, in computer graphics, we may encounter cases where this isn't true – for example, when the material properties of a metal and dielectric are interpolated between. The above formula deals with non-binary metalness.

(a) *NDF* **(b)** *Geometry term*

Figure 5: *Illustration of the D and G components of the microfacet model. The NDF, $D(\omega)$, describes the distribution of microfacets. It specifically tells us the density of facets that are oriented such that their normal points in the direction ω. The masking function $G(\omega_i, \omega_o)$ describes self-shadowing where microfacets block each other.*

The amount of specular reflection is given by the Fresnel term F, which we evaluate using the Schlick approximation:

$$\mathbf{F}(\mathbf{v}) = \mathbf{F}_0 + (1 - \mathbf{F}_0)\ (1 - \mathbf{h} \cdot \mathbf{v})^5 .$$

For the diffuse term we will just use a simple Lambertian. Only light that that wasn't reflected specularly will participate in the diffuse term (hence the $\mathbf{1} - \mathbf{F}$ term). Additionally, as previously mentioned, metals have a zero diffuse contribution. We model this with the $1 - M$ term:

$$\mathbf{L}_{\text{diffuse}} = \frac{\mathbf{c}_{\text{mat}}}{\pi} \otimes ([1, 1, 1] - \mathbf{F}(\mathbf{v}))\ (1 - M) .$$

More complex diffuse reflectance models exist. However, for now, the Lambertian is sufficient. Other models can be quite a bit more expensive and may only contribute with minor improvements [Karis, 2013].

For the normal distribution function D, we will use the Blinn-Phong distribution. As the name indicates, it is very similar to to the Blinn-Phong specular term from the previous exercise

$$D(\mathbf{h}) = \frac{\alpha_p + 2}{2\,\pi}\ (\mathbf{n} \cdot \mathbf{h})_+^{\alpha_p} .$$

The masking term from the Cook-Torrance model [Cook and Torrance, 1982] that you should use looks as follows:

$$G(\mathbf{l}, \mathbf{v}) = \min\left(1, \min\left(2\,\frac{(\mathbf{n} \cdot \mathbf{h})_+\ (\mathbf{n} \cdot \mathbf{v})_+}{\mathbf{v} \cdot \mathbf{h}},\ 2\,\frac{(\mathbf{n} \cdot \mathbf{h})_+\ (\mathbf{n} \cdot \mathbf{l})_+}{\mathbf{v} \cdot \mathbf{h}}\right)\right) .$$

For the ambient term, $\mathbf{L}_{\text{ambient}}$, you can assume a constant ambient illumination and just modulate it with the material's color:

$$\mathbf{L}_{\text{ambient}} = \mathbf{c}_{\text{ambient}} \otimes \mathbf{c}_{\text{mat}}$$

This model relies on a different set of material parameters:

- α_p Material shininess
- M Material metalness
- $\mathbf{c}_{\text{emit}}$ Material emissive color
- $\mathbf{c}_{\text{mat}}$ Material color

(Other quantities such as the various vectors and the light color are the same as in Section 2.2.)

Implement the shading model for one light source (Equation (2)). Use a separate pair of shaders (vertex and fragment – if you use the provided project template, use the included `PBR.vert` and `PBR.frag` shaders).

Compare your output to Figure 6. If you need to debug your shaders, see Figure 7 for visualization of some of the intermediate values (also check that the various values are behaving as they should!).

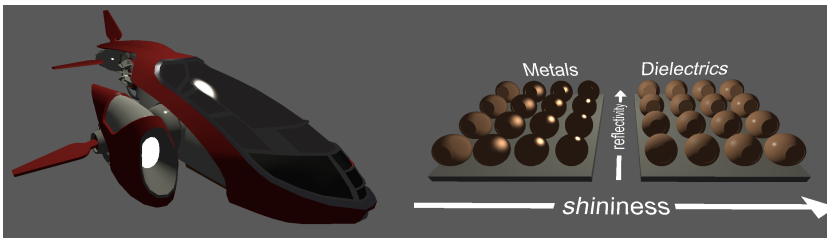
2.4 Animated light position

2 marks

Animate the light position. For example, make the light orbit around the Y axis above the model(s). Make sure that the animation is frame-rate independent (e.g., like the camera controls in CW 1).

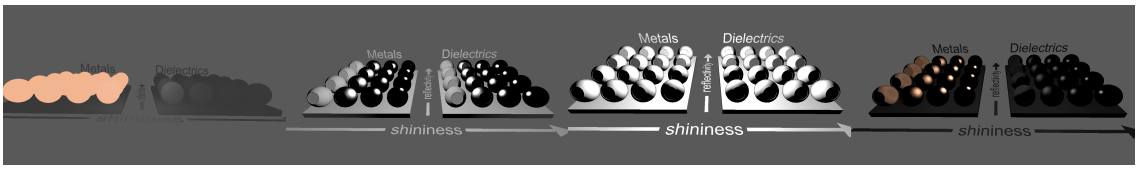
Let the user start/stop the animation by pressing spacebar.

In the submission's README, document what light path you choose.



(a) `NewShip.obj` **(b)** `materialtest.obj`

Figure 6: *Shading with the PBR model. In the material test, there's now a clear difference between the metals and the dielectrics. Metals benefit immensely from e.g. image-based lighting approaches, due to their lack of diffuse contribution. Note that we no longer have any variation in the "reflectivity" direction, as the corresponding parameter is not used by the PBR method.*



(a) $\mathbf{F}$ **(b)** D **(c)** G **(d)** *PBR specular term*

Figure 7: *Visualization of the some of the components of the PBR model used in CW 2.*

2.5 Multiple light sources

2 marks

Extend the application to support multiple (≥ 3) light sources.

Make sure that the light sources are placed in reasonable places (e.g., not inside the models and not on top of each other). Give the each light a separate color, to make sure that the different lights can be distinguished from each other.

Let the user select the number of active light sources via the number keys (use the number keys in the row at the top of the keyboard).

2.6 Custom PBR model

4 marks

Pick *one* of the following terms/factors in the PBR shading model to improve on:

- Diffuse term
- Normal distribution (D)
- Masking/geometry term (G)

Replace that term/factor with a different compatible term (e.g., replace Section 2.3's choice of D with a different normal distribution function). You must use an established model/function (i.e., you can't just come up with your own). This will likely require some research online.

In the submission's README, detail which term/factor you chose to replace, and describe the new choice (using words and equation, not code). Cite the source from where it stems. Describe how your chosen term/factor differs from the one's selected in Section 2.3.

3 Submission & Marking

Submission specifics are unchanged from CW 1, but are included here again to avoid any misunderstandings. If you studied the corresponding section in CW 1, 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 `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 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.*

References

BURLEY, B. 2012. Physically based shading at disney. URL https://media.disneyanimation.com/uploads/production/publication_asset/48/asset/s2012_pbs_disney_brdf_notes_v3.pdf.

COOK, R. and TORRANCE, K. 1982. A reflectance model for computer graphics. URL https://graphics.pixar.com/library/ReflectanceModel/paper.pdf.

HOFFMAN, N. 2010. Crafting physically motivated shading models for game development. URL https://renderwonk.com/publications/s2010-shading-course/hoffman/s2010_physically_based_shading_hoffman_b.pdf.

KARIS, B. 2013. Real shading in unreal engine 4. URL https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf.

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 2 (Student copy)

Date . . . . . .

Name . . . . . .

UoL Username . . . . . .

Student ID . . . . . .

Instructor name . . . . . .

Instructor signature:

Coursework 2 (Instructor copy)

Date . . . . . .

Name . . . . . .

UoL Username . . . . . .

Student ID . . . . . .

Instructor name . . . . . .

Instructor signature: