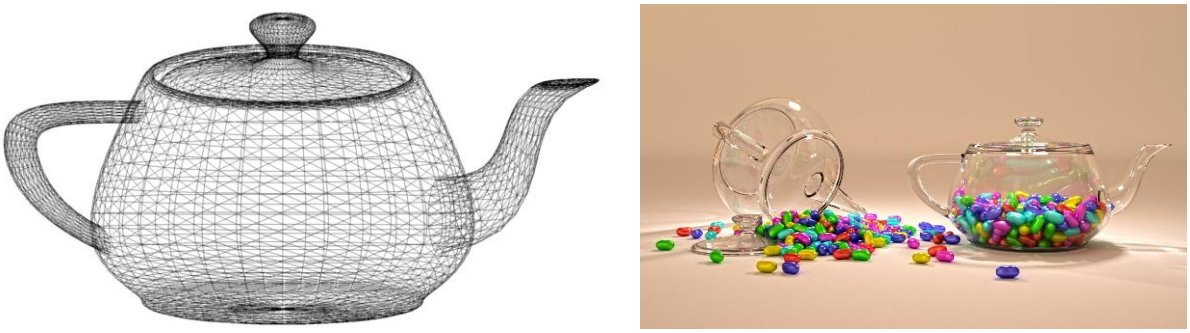
CM30075: Advanced Computer Graphics

Coursework Specification 2022-2023



(a) (b)

(a) Simple Wireframe Teapot, Ken Cameron
(b) Utah Tepot Rendering Competition Winner, Laura Marie Lediaev
Figure 1

Deadlines

All work must be uploaded to Moodle by the following deadlines.

Stage 1 Lab 1	Code, Image		
Stage 1 Lab 2	Code, Image		
Stage 1 Lab 3	Code, Images		
Stage 1 Lab 4	Code, Images	8pm Friday 11th November 2022	Week 6
Stage 2+3	Code, Report	8pm Friday 9th December 2022	Week 10
Stage 4	Image, attribution.txt [,Code]	8pm Friday 16th December 2022	Week 11

Introduction

This unit is assessed 100% by coursework. There is no written examination.

The coursework provides you with a framework written in C++. You will extend this framework to build a series of renderers of increasing complexity. We've split the coursework into four major pieces.

1. Four lab exercises. These are smaller mostly self-contained tasks with near weekly deadlines. Most students should be able to achieve full marks on these with support from the tutors. The assessment for these tasks will be by demonstrating output and code to a tutor during a lab session. There is no partial credit for incomplete tasks, but full marks will be given that meet the minimum requirements. The work must also be uploaded to moodle by the deadline to receive the marks. *(For those with a DAP that are unable to attend the labs, the work will be marked online.)*

2. Feature-rich Rendering. There are a series of tasks to complete. The complexity of the tasks in this stage varies considerably. Some tasks require a deeper understanding of the topic and may not be achievable by all students. For this stage partial credit will be applied in each task. This means that a full featured, good quality implementation is required to gain all marks. A minimal attempt will not gain full marks. This stage will be assessed by the lecturers using the implementation and a report on the work done.

3. Photon-mapping. This is an advanced technique that requires a deep understanding of the topic and is expected to be challenging for most students. For this stage partial credit will be applied in each task. This

means that a full featured, good quality implementation is required for full marks. A minimal attempt will gain minimal marks. This stage will be assessed by the lecturers using the implementation and a report on the work done.

4. A final rendered image. Marks are also available for a good quality image produced by the software you create. This will be assessed by the lecturers.

There are multiple deadlines to help ensure you make steady progress over the semester. You are free to begin tasks ahead of earlier deadlines. This document is the full coursework specification. Further implementation guidance can be found within the comments of the framework code.

Stage 1: Labs (30%)

30% of the overall marks are awarded for the four lab exercises that require you to build basic rendering programs. The lab tasks are designed to be independent for marking purposes but can be combined to produce more complex rendering.

Lab 1: Rasterising Lines (5%)

In this lab you will extend `main_lab12.cpp`. It provides a framework for a rasterising renderer. This includes a `FrameBuffer` class allows you to store your constructed image and write it out as an image file. Your task is to replace the line drawing function in linedrawer.cpp with an integer based one.

Lab 2: Reading Models (5%)

In this lab you will further extend `main_lab12.cpp`. Your task is to extend the `PolyMesh` class to read 3D object files. This will allow you to read the teapot model and using either the provided line drawing function or your own to draw a wireframe teapot.

Lab 3: Simple Raytracing (10%)

In this lab you will extend `main_lab34.cpp`. It provides a framework of a raytracer that can, as provided, render unshaded spheres and planes from a fixed viewpoint. There are two tasks in this lab:

a. If you examine the `SimpleCamera` class you will see that it generates a Ray for each pixel in a `FrameBuffer` and uses the `raytrace()` method within a Scene to trace this and apply the resulting colour and depth to the frame buffer. The camera is at a fixed location and looks along the z-axis. In this task you will create a new class called `FullCamera`, derived from `Camera`, that allows the location, look-at and up vectors of the camera to be specified as well as the field-of-view.

b. Extend the `PolyMesh` class to support ray-triangle intersection to allow the rendering of poly-meshes. You will need to extend the reader part of the class to compute both face and vertex normals. You will be allocated 5% for triangle intersection and normal calculation. You must demonstrate both vertex and face normals.

Use the `FullCamera` class to write out both a depth buffer image and a colour image to check that the code functions correctly. If you do not have the triangle intersection working, you can still get the structure 5% by using the supplied `Sphere` and `Plane` objects to demonstrate that functionality. Use the `FalseColour_Material` to assist with debug and demonstration.

Lab 4: Basic Lighting (10%)

In this lab you will further extend the program you developed in Lab 3 to perform local lighting. There are two tasks in this lab:

a. The `Material` class is used to assign material properties to an object. Your task is to provide a `Material` class called `Phong_Material` that provides a basic lighting model that supports local ambient, diffuse and specular lighting. The class requires two methods. The first is called once per intersection and the second is

called for each light. In this task, the first method should return the ambient contribution and the second the diffuse and specular value. A `Light` class is provided. This task is worth 5%.

b. Extend the `raytrace()` method of `Scene` to support shadows. The Scene class already contains a method to perform the core raytracing operation needed. This task is worth 5%.

Use the `FrameBuffer` class to write out the colour buffer image to check that the code functions correctly. If you successfully extended the `PolyMesh` class you will be able to use it to render the teapot as a lit object. If you did not, you can again use the provided `Sphere` and `Plane` objects.

Stage 2: Feature-Rich Rendering (30%)

After completing the labs you should begin work on this stage. You will extend the basic raytracer to add more advanced features. You will also provide a report that explains the theory behind your implementation. This is worth 30% of the overall unit mark.

This stage of the coursework requires you to have a working version of the lab code. After the deadline for the labs, a sample solution will be released. You are free to use this without penalty or you can continue with your own code. You may also mix and match the two.

Task 1: Reflection and Refraction (10%)

In this task you will extend the raytracer to implement reflection and refraction of rays to provide global illumination. You will gain more marks if you support the Fresnel term than if you don't. All of the functionality required for this should be placed in a `Global_Material` class that derives from `Material`. No not add the reflection/refraction directly to the `Scene` class.

Task2: Constructive Solid Geometry (15%)

In this task you will extend the raytracer to support two new kinds of object. You will add a `Quadratic` class derived from the `Object` class that implements a quadratic surface and a `CSG` class derived from the `Object` class that implements Constructive Solid Geometry. The `Quadratic` class will be worth 8% and the `CSG` class 7%.

Task3: Additional Feature (5%)

In this task you are free to choose one of the following features to add to your raytracer.

a. Depth of Field. (`Camera`)
b. Motion Blur. (`Camera, Scene, Object`)
c. Additional Object Types. (`Object`)
d. Texturing. (`Object, Material`)
e. Advanced Local Shader. (`Material`)
f. Ambient Occlusion. (`Material`)
g. Bounding Objects. (`Object`)
h. Another feature of similar complexity with the agreement of a lecturer. Ask by email.

When choosing your feature consider both the complexity to implement and the runtime. For example, techniques like depth of field and ambient occlusion that involve additional ray sampling require only a small amount of extra code, but significantly increase runtime for worthwhile image generation.

Stage 3: Photon Mapping (30%)

In this stage you will extend the raytracer to support photon mapping as introduced by Jensen[1]. This is an advanced technique and will require significant self-study to implement fully. It will have to extend some base classes you have not previously needed to change. It also increases runtime required significantly to produce useful images and this should be considered. The steps needed are:

1. Add a `Photon` class, extend `Environment`, `Light` and `Material` classes.
2. Add new `Material` classes to support photon generation.
3. Record photons and map to a suitable data structure such as KD-Tree for use in the second pass.
4. Support use of the photon-maps during the existing rendering phase via updated/added `Material` classes.

You also need to explain what you have done in the report.

Stage 4: Final (10%)

10% of the unit mark is allocated to the production of a final image. The image must be square, contain no border area and have a minimum resolution of 512x512 pixels. It should make use of the teapot dataset. The deadline for this image is a week later than the project code and report. You should use the code you submitted for Stage 2+3. You should not add new functionality but can fix minor bugs you discover during this period. Examples of minor bugs are using the wrong variable in an expression, forgetting to normalise a vector or needing to negate it. For further guidance, re-implementing or replacing a triangle intersection test with a new algorithm are not considered minor bug fixes. Neither is adding or finishing incomplete code. You are however free to adjust the scene and its settings. If the code or data used for the final image has been modified from that submitted in Stage 2+3, a copy of the modified version must be submitted as part of this Moodle Assignment.

Marking Guide

Submissions that extend the basic raytracer to support reflection and refraction can achieve a pass mark (40%-50%) with a good quality report and image. Submissions that further support CSG, add an extra feature and submit an interesting image with a good quality report can achieve a better mark (<=70). Submissions that additionally, successfully support diffuse photon mapping can reach a 1st class mark (70-80%) and to score higher will require transmitted caustics. Your report need only cover work done in Stages 2+3. You do not need to write about Stage 1.

Good quality code will:

a. Make use of sensible names for functions, classes, variables, etc.

b. Maintain the structure of the provided framework.

c. Make use of comments to assist the reader to follow the functionality.

d. Clearly indicate any code that is not your own and cite the source.

e. Function correctly.

A good quality report will provide:

a. A clear explanation of the objective of the code.

b. A brief, clear description of the theory/methods/algorithms employed.

c. An overview of the code developed, including any design decisions taken, problems encountered and how these were overcome. It will not include significant amounts of the code.

d. Appropriate diagrams/illustrations/renderings to assist the readers understanding.

e. Appropriate references to any literature that has been relied on.

Good quality images will:

a. Consist of a scene that is of sufficient complexity to demonstrate the key features of the raytracer.

b. Avoid artefacts that detract from the image such as excessive noise or low-resolution.

c. Show some level of artistic merit in the composition.

d. Make use of the teapot dataset.

Marks have not been explicitly split between the code and the report in the marking scheme. The lecturers will take a holistic view that generally gives them equal weight. However, while minor deficiencies in the report can be compensated for by a good implementation, and vice versa, you should not expect half marks regardless how good your code is if there is no or little report. This is also true should you only submit a report.

Submission

All work should be submitted to the appropriate assignment submission point in Moodle by the deadline given or by any extension agreed by your Director of Studies. **Unit Lecturers cannot grant extensions.** You must send any completed extension request form to your Director of Studies following the guidelines they have issued.

Stage 1

For each lab, you should submit your code and a sample image for each 5% task claimed. Images should be between 128x128 and 512x512 in size and in PNG format. Your code should be uploaded as a ZIP file. The uploaded images should NOT be included in the zip file.

Stage 2+3

On completion you are expected to submit the following:

1. The code you have developed.

2. A report of 4 to 6 pages that explains the theory behind the code you have created.

3. To demonstrate the functionality of your code, you may include an appendix of illustrative images in your report and reference them in the main text. This must not include any diagrams as those should be part of the main text. This will not form part of the report page count.

You must upload this to the Moodle Assignment. Your report must be in PDF format and uploaded as a stand-alone uncompressed file. You should upload your code and any data it requires as a single separate zip file. The uploaded report must NOT be included in the zip file.

Stage 4

You must upload your final example image in PNG format. If you have made any changes to the code/data, from that uploaded to the Stage 2 Assignment, you must upload a fresh zip file. As in previous years, the images will be used to create a class gallery. Please upload an attribution.txt file that contains the JSON formatted data indicated in Fig 2. This includes your name as you wish it to appear in any attribution and a title for the image. The title must be suitable for a general audience and not reference any other student. You may also indicate if you do not wish your image to appear in the gallery (opt-out.)

```
{ "name": "Your Name",
  "title: "Title",
  "include":"no"    ← optional opt-out
}
```

Fig 2: Attribution JSON

Plagiarism

The internet is full of raytracing code. While the license under which such code is shared may permit you to use it, this does not mean you can cut and paste that code into your work and try to claim it as your own, even if you changed the variable names or datatypes. The goal of this coursework is for you to demonstrate your understanding of the tasks set by implementing them yourself.

However, you may use those code examples as guidance for your own work. Where you do this, you should include a reference to the source in a comment in your code along with an indication of the extent to which you have relied on it. Cutting and pasting whole methods is not acceptable. Fragments or inspiration is. Where you have relied more heavily on such code you should include a citation in the report. This is treating code exactly as you would any other source you rely on in a piece of academic work.

Where the code is a supporting utility rather than the specific graphics task you are implementing, you are permitted to use the code without adaptation in this coursework. For example, libraries or classes that implement non-graphics specific data structures such as a KD-Tree may be used as is. Or libraries that support the reading or writing of additional file formats. When doing this, it is still necessary to indicate that you have done this. You should include a comment in the code and a citation in the report, where you will be expected to demonstrate that you understand what the code is doing and not treat it as a black box.

Refs

Henrik Wann Jensen: "Global Illumination using Photon Maps". In "Rendering Techniques '96". Eds. X. Pueyo and P. Schröder. Springer-Verlag, pp. 21-30, 1996

KMC 2022