Raytracer: CGR 2025 Coursework Specification

1. Overview, objectives and guidelines

For this coursework, you are required to develop a modular raytracing renderer from scratch in C++ and to implement a series of core rendering features, which are broken down into three mandatory modules and a set of final features. Alongside your implemented code, you are expected to produce a report detailing your methodology, implementation steps, and evaluation of your renderer. Your coursework aims to check the following objectives:

1. *Develop a Modular Raytracing Renderer:* Code a functional raytracing renderer from scratch, implemented in a suitable language (such as C++), ensuring a modular design for easy testing and flexibility.
2. *Implement Core Geometric and Camera Features:* implement a virtual pin-hole camera model (including coordinate transformation) and robust ray-primitive intersection tests.
3. *Integrate Advanced Visual Techniques:* Implement and demonstrate Whitted-style recursive ray tracing with advanced features like antialiasing and texture mapping.
4. *Optimise for Performance:* Develop and evaluate a spatial acceleration structure to speed up rendering times for scenes with thousands of primitives.
5. *Achieve System Integration:* Ensure all implemented modules are cleanly integrated into a raytracer with command-line options for enabling or setting parameters for different features. e.g. to set output resolution, run it with or without the acceleration, etc.
6. *Document and Evaluate Work:* Produce a comprehensive report that explains the step-by-step implementation methodology, includes visual examples (rendered images) of each feature, and provides a clear evaluation of the work.

Please adhere to the following policies while solving this coursework.

1. You may use GenAI coding assistants to build elements of this coursework. Although you may use any model, it is recommended that you use ELM.
2. This coursework is to be solved *individually*. You may discuss algorithms, maths, and methodologies but do not share code or prompts used to generate code.
3. It is strongly recommended that you use g++ from the GNU compiler collection. If you have trouble obtaining it for your platform of choice (say Windows), you may use an alternative C++ compiler. Please ensure that it is available freely (so that the marker can obtain it) and provide explicit directions on how your project should be built. Please avoid build packages. They will be overkill for this coursework, which should simply contain a set of your own headers and cpp files with no use of external libraries.
4. Your main submission should not include any Operating System specific calls (e.g. for GUI). You may use these for marks in the exceptionalism category, but if you do, this should be separated from your main submission (which will be marked separately).
5. You are not allowed to use any external libraries. This includes for maths objects, reading or writing images, scene files, geometry files, etc. Your final submission should only use your own implementation or those found in the C++ standard library.
6. You are not allowed to directly import code from online tutorials, most notably the Raytracing in One Weekend series. Your code will be checked for similarities with other submissions (this year and previous years) and publicly available codebases such as

RTiOW. A high degree of similarity will lead to further investigations. You may test the similarity (against RTioW) yourself by using JPlag. Simply follow the instructions provided on the github page! If you are concerned that the reported similarity is high, please check with the instructor immediately

7. Your implementation should be modular, using classes to encapsulate concepts appropriately.
8. For marking your submission, the marker should be able to:
    a. build your code into a binary executable by running the Makefile
    b. test your raytracer using a .blend file. For this they should be able to first use your Export.py script to output your scene to ASCII, and then run your raytracer on this exported file to obtain an output image (as a .ppm). Some such .blend files will be shared after the checkpoint deadline for Module 1.

2. Marking scheme

You are required to implement the raytracer in four distinct development stages---three intermediate modules and the final system integration. The table below provides the marks allocation to different topics, each of which will be explained. The suggested working hours are provided to help you plan your time; they are estimates to achieve about 50 out of the 80 marks for this CW, assuming capability in relevant topics from lectures, reading and tutorials.

	marks	**time (h)**
Module 1	**6**	
Blender exporter	1	0.5
Camera Space	4	4
Image R/W	1	0.5
Module 2	**8**	
Ray intersection	4	3
Acceleration	4	4
Module 3	**12**	
Whitted-style	8	4
Antialiasing	2	4
Textures	2	2

	marks	**time (h)**
Final raytracer	**16**	
Sys. Integration	4	2
Distributed RT	8	5
Lens effect	4	2
Timeliness Bonus	**20**	
Module 1	4	
Module 2	6	
Module 3	10	
Report	**8**	6
Exceptionalism	**10**	

To earn full marks for a feature, it needs to be:

1. Designed and implemented as per spec. e.g. use class hierarchies, comments, etc.
2. Listed as implemented and working in a text file named FeatureList.txt
3. Demonstrated in your report, with an example.

3. Deadlines and Submission

The final submission for all code, rendered test images, and the accompanying report is **5th December, 12:00 GMT.** To encourage incremental development and modular planning, three optional intermediate checkpoint deadlines are provided. Submitting functional, working code for a module by its checkpoint deadline will earn bonus marks (see Sec. X).

Module Checkpoint	Bonus Marks Available	Optional Deadline
Module 1 Checkpoint	4 marks	October 17th, 12:00 GMT
Module 2 Checkpoint	6 marks	October 31st, 12:00 GMT
Module 3 Checkpoint	10 marks	November 21st, 12:00 GMT
TOTAL BONUS AVAILABLE	20 marks	

For each submission, all material must be contained in a single compressed ZIP file named with your student ID (e.g. s123456.zip) and submitted via Learn. The folder structure for the final submission must be organised as shown below. Deviation from the required format will incur penalties. Maintain the same structure for all modules, building incrementally. That is, for the submission at the end of Module 1, FeatureList.txt will contain two entries: First that the camera class is implemented and second that the PPM image reader and writer have been implemented. For Module 1, the Code folder might only contain camera.h and image.h and the Output and Report folders may be empty (but should still exist).

The required folder structure is:
```
s123456/
├── System.txt
├── FeatureList.txt
├── Code/
│   ├── Makefile
│   └── (All source and header files. raytracer.cpp, camera.h,…)
├── Blend/
│   └── *.blend
│   └── Export.py (exports scene.blend to scene.txt)
├── Textures/
│   └── (*.ppm files used for texture mapping)
├── ASCII/
│   └── (all exported ASCII files of blender scenes. E.g. scene.txt)
├── Output/
│   └── (Generated output images in PPM format showcasing features)
└── Report/
    └── s123456.pdf
```

System.txt should contain two lines. On line 1, the name and version of the operating system your code was developed on (e.g. 'Ubuntu_xxx', 'Macos X.X', Windows XX). On line 2, the name of the C++ compiler you used (e.g. g++-X.XX, clang-X.XX). FeatureList.txt should have a list of features from this specification that your raytracer has demonstrably implemented. You can have a '#Partial' section for features that you have partially implemented.

Please ensure that your code can be built by the marker! If you have implemented some features that cause the code to not compile or link, please do not include them in the submission. If you have used a compiler other than g++, or a different type of Makefile than a script that can be executed at command line, please include 'build' instructions.

4. Detailed specification

4.1 Module 1

Module 1 consists of three tasks.

1. *Blender exporter*: Write a python script within blender to export objects setup in a Blender scene to a file in ASCII format (txt, json, etc). You may choose your own format provided it contains the following information
    a. For cameras: location, direction of the gaze vector, focal length, sensor width and height and its film resolution.
    b. For point lights: location and radiant intensity.
    c. For spheres: location and radius (scale in Blender).
    d. For cubes: translation, rotation and 1D scale.
    e. For planes: 3D coordinates of its four corners.
2. *Camera space transformations*: Write a C++ camera class with methods to
    a. read the file written by the exporter in 1 and stores camera information.
    b. convert pixel coordinates from (float px, float py) a ray (origin and direction) in world coordinates.
3. *Image read and write*: Write a C++ image class that can read and write image files in .ppm format. The class should have
    a. a constructor that takes a string as input (the filename)
    b. members to read & modify the pixel values and
    c. a method that writes the image to a specified file.

4.2 Module 2

Module 2 consists of 2 tasks.

1. Ray intersection: This will require the following steps.
    a. Develop 'Intersect' routines for three shapes---cubes, planes and spheres.
    b. Structure them within a class hierarchy of Shapes which exposes an Intersect method that returns a Boolean hit/miss.
    c. Pass a 'hit' tructure to the intersect routine that stores information pertaining to the intersection. E.g. the intersection point, distance along the ray, etc. The information in this structure will be modified if there is an intersection.
    d. For objects that are have been transformed, remember to transform the ray appropriately before testing for intersection.
2. Acceleration hierarchy: This step involves building a hierarchical data-structure that will make intersection tests efficient when there are many shapes in the scene. In your report, show evidence of speedup for at least one scene. Compare runtimes for raytracing a scene with and without this feature.

4.3 Module 3

1. *Whitted-style raytracing*: Assemble features in Modules 1 and 2 to achieve the following.
    a. For each pixel in the cameras image plane, trace a ray into the scene.
    b. Test intersections with all shapes in the scene.
    c. Shade the closest intersections according to the Blinn-Phong model.

    d. In addition to the shading for that intersection, trace a reflected and/or refracted ray depending on the material.
    e. Shade intersections recursively for reflected and refracted rays and add their contributions to the image weighted by material properties.
    f. Compare your image to the same scene rendered in Blender. It is expected that the colours will not match entirely (because of the different shading model used in Blender), but the perspective and projection of shapes should match.
2. *Antialiasing*: Instead of tracing one ray through each pixel, add average contributions of samples located at slightly different positions on the camera's image plane.
3. *Textures:* Modify the shape intersection routines so that the hit structure returned also contains surface texture coordinates (typically u,v). Then modify the scene description file output by blender to also include a filename (.ppm) where the texture is located. Finally, read the image and use the (u,v) coordinates to select the color used for Blinn-Phong shading.

4.4 Final raytracer

1. *System integration*: Combine and streamline all the above modules. Ensure that your code is modular and extensible. Implement command line arguments to turn various features on or off. Modify your export script in Blender so that all features may be integrated.
2. *Distributed raytracing*: Implement soft shadows and glossy reflection as distributed raytracing effects.
3. *Lens effects:* Implement motion blur and blur due to defocus (depth of field).

4.5 Timeliness bonus

To obtain the timeliness bonus you should have

1. Submitted a .zip file by the prescribed deadline and included the relevant source and/or output files within the folder structure.
2. Also try to include a short description of how you tested/debugged the module. You can put it in the report.pdf for the module submissions. Later, for the final submission these will be overwritten by the final report.
3. Justified, within the report, any deviations from the checkpointed implementation of code found in the final version. This should be under a section titled 'Timeliness Bonus' with subsections for each module. E.g. compare before/after output comparisons.

4.6 Report

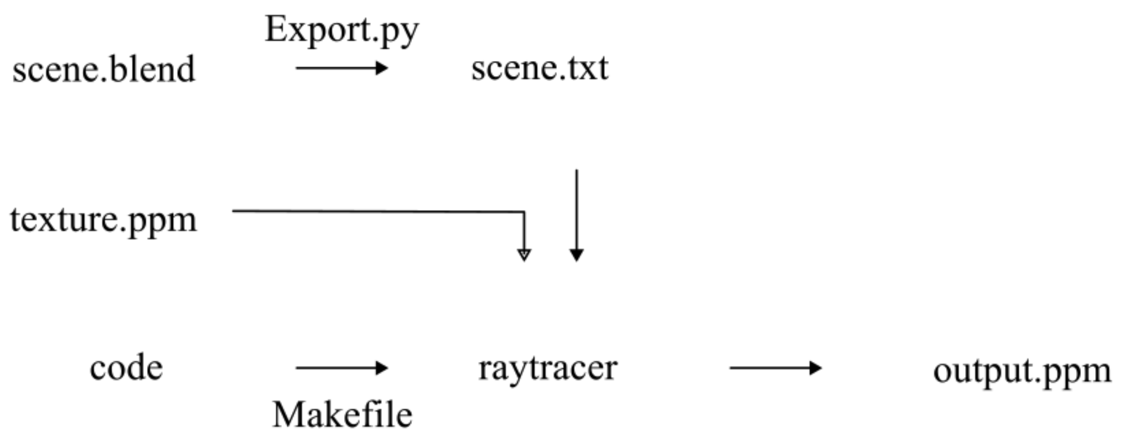
You may choose your own structure for your report. Some tips to avoid losing marks…

1. Keep your report concise.
2. List all the features that you have implemented. For each feature, provide examples of prompts and explain any modifications that you performed to the code supplied by your coding assistant.
3. Show example outputs with references to the filenames of the outputs in your submission and the corresponding blend files.
4. Compare your output with Blender's renderings of the corresponding scenes.
5. Include a table corresponding to the one in Section 2, showing what percentage of each of those topics you completed.

6. Include a section on Timeliness Bonus justifying your deviations (if any) of your final code from checkpointed versions in the modules.
7. Write a short summary (about 250 words) on what you think the strengths and weaknesses were of using coding assistants for this assignment specifically.

5. Illustrated workflow

The following figure illustrates the process described previously in the specification.



Although it is possible that you might support additional features and therefore require further inputs to the raytracer, this diagram represents a necessary configuration. In case you add additional inputs or features, ensure that you have documented this in your report and perhaps output relevant help messages from the raytracer. e.g. ‘File X not found’, ‘Expecting an input on the number of samples’, ‘Expecting material constant’, etc.

--------- All the best and have fun! ---------