CAP 4730 – Assignment 1
Due February 9, 2024

Start Early!

First-Hit Ray Tracer

In this homework you will write a simple ray tracer to render a scene composed of simple objects (e.g., multiple spheres and tetrahedra with different colors and sizes)[1].

You will start with a directional light and a parallel camera geometry for rendering purposes.

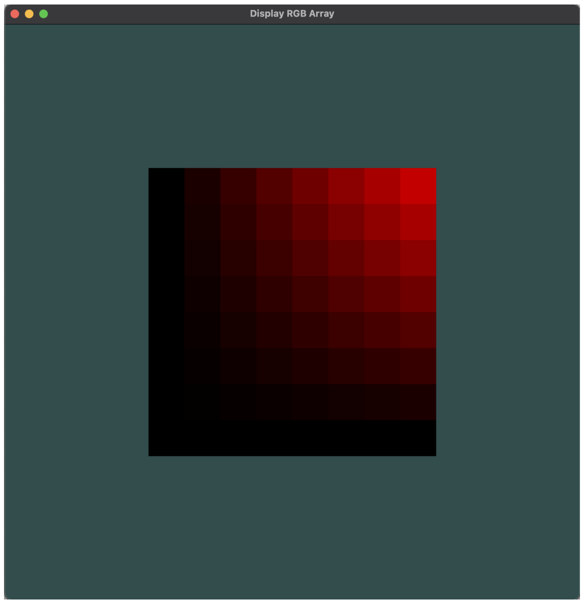
- Generate parallel rays and solve the ray-object intersection problems to determine the color of each pixel. Verify the correctness of your ray tracer by rendering the scene from multiple angles (e.g., change the `LookAt` vector). Document the results for this step and each of the following steps in your report. `20pts`
- Add ambient, diffuse and specular shading to your ray tracer. Pick the ambient, diffuse and specular constants and light intensities to reflect a realistic rendering of the scene. `15pts`
- Add a glazed surface to the scene. You may choose to set one of the objects in the scene or have a plane below these objects that appears as a glazed surface. `15pts`
- Create a small animation by moving the camera and some of the objects in the scene, and assembling the sequence of rendered images into a movie. You may use image libraries for exporting rendered images. `15pts`
- Once you have completed the above parts, give the user an option (e.g., by pressing a key) to switch to a perspective geometry. `15pts`
- Bonus: Implement an advance feature with creative ideas. `+10pts`
- Along with the source code and makefile (or a VS Code project file along with a README file for compiling instructions), submit a report (a PDF file) that describes and documents *each functionality* you implement. The report is graded on its comprehensiveness in documenting both the parts implemented and the parts that are partially (e.g., buggy) or not implemented. Here is a sample report from a previous year. `20pts`

This assignment can be done in groups of two students. You are welcome to discuss ideas/problems with other classmates but the source code and the report you submit MUST be your work. You need to clearly acknowledge sources (ideas, solutions, websites) that you use.

[1]For inspiration, have a look at this demo from a previous year (that included fancy advanced features)

Setting up OpenGL

The focus of this homework is on the topics we learned in class about ray tracing. You will only be implementing the ray tracer in C++ without any graphics libraries (do NOT look for OpenGL techniques to do ray tracing). However, only for the purpose of displaying the ray traced images you create we will use an OpenGL display program. For your convenience a simple OpenGL program is made available here. that displays the contents of an RGB array (e.g., output of your ray tracer). If you look at line no. 159 through 172 currently a gradient image of resolution 8×8 is being generated. The program shows the content of this image on a square area over a teal background



To be able to compile and run this program, you first need to set up your machine so you can access the OpenGL functionality its video card or GPU provides. Getting OpenGL going can be quite complicated (ask me why), but we will learn a bit about the system design a bit later. Follow the following steps (there are many helpful online resources for troubleshooting):

1. Ensure your machine supports OpenGL version 3.3 or above. Any hardware build in the past 5 years, with Windows/Linux/macOS should work. Older hardware may also work, but you might need tweaks. To find out what versions of OpenGL your system supports you can use the `GLView` (app) from here – a tool to test your system's capabilities.

2. Install `GLFW` (latest version). This provides a library for OpenGL development, but also functionality for creating a window, context and handling input events. Pre-compiled versions are available in common package managers (e.g., `brew install glfw`). However, if you are comfortable with using `CMake`, you may compile `GLFW` from source.

3. Install `GLEW`. This OpenGL Extension Wrangler library is necessary for setting OpenGL function pointers to the positions provided by the driver. Again pre-compiled versions are available in package managers (e.g., `brew install glew`), but you can also install from source.

4. Now you can compile: `g++ -lglfw -lglew simpleTexture.cpp` and run the program `./a.out`
   Note that on macOS you need to add a flag:
   `g++ -lglfw -lglew -framework OpenGL simpleTexture.cpp`

Submission Guidelines
Submit to E-learning site a **single file** as a .zip or a .tar.gz bundle that contains all the files to be submitted. Include the source codes for your programs in the submission bundle. Please include a ‘README’ file that clearly explains how to run and test the program. Also include a ‘Makefile’ (or a VS Code project file) that compiles and links the program from the source files.
Late submissions are penalized by 20% of the grade for each day (up to 3) past the due date.