CS598APE	Due: **Feb 17, 2025**

Mini-Paper 1: Raytracer

Name: Don't forget to add your name *Collaborators: List all sources used*

The first mini-paper this semester is to analyze and improve the performance of a raytracer program. This project provides you the opportunity to explore optimizations for profiling performance, reducing work, and introducing parallelism. For this project, you will work in teams of two (section 1), and the grading will be based on your report (section 5.1) and your artifact evaluation (section 5.2). You will also peer review the reports and artifacts of others.

1 Working in Teams

In this project you will be working in teams of two. Both members of a team will get the same grade.

For this project you *must* use pair programming and both of you should take turns being the driver (who types) and the observer (who watches, advises, and plans ahead). You should switch roles at a relatively fine grain, e.g. switch between writing the optimization and the test for it. Make sure that you are both familiar with the algorithmic concepts and infrastructure, enough that you could do the entire project on your own.

Your report should include a paragraph which describes how you worked together on this project, including the design, implementation, and testing of your optimizations.

You are welcome to use any source of inspiration or tool to create your optimizations, subject to the course policy on the website. These may include new code/changes to code and the use of different tools to generate or change code. You are welcome to also try more advanced techniques that do not fall into this category (e.g. specific hardware). For precise details, check section 5.2.

2 Accessing Course Infrastructure

You are welcome to use any hardware of your choosing to test and perform optimizations, assuming that the optimizations you write can be replicated by others. This does not mean that others need to replicate your exact performance results (e.g. exactly 12% boost by doing X), but that the optimization you write generally results in an improvement and does not rely on non-standard hardware features. If you want to use your own machine for setup and optimization, you are responsible for ensuring the requisite requirements are available.

We have given students access to UIUC teaching VMs. You may use them for completing the homeworks and project.

3 Raytracer

We are optimizing a raytracer. Raytracing is a technique for generating realistic-looking images of a virtual scene. The scene is defined by a number of objects and images are created by simulating light being emitted from a camera and bouncing off of objects until it hits light sources. See Figure 1 to get a feel for how raytracing works.

3.1 Where Is the Code?

The application code for this project is available on GitHub. Fork this code, but please keep your code and optimizations in private (and not public) repositories.

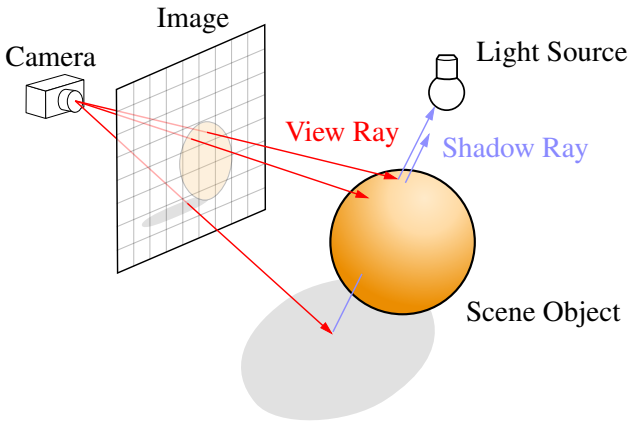


Figure 1: High level view of a ray tracer in action.

The README.md contains more information about the layout of the Raytracer code and how to execute it. The Raytracing application can emit images, or a sequence of images which can be used to create a video.

3.2 What are your Tasks?

Your task in this project is to ensure that the four sample inputs to the program run as fast (and as high resolution) as possible. They are as follows:

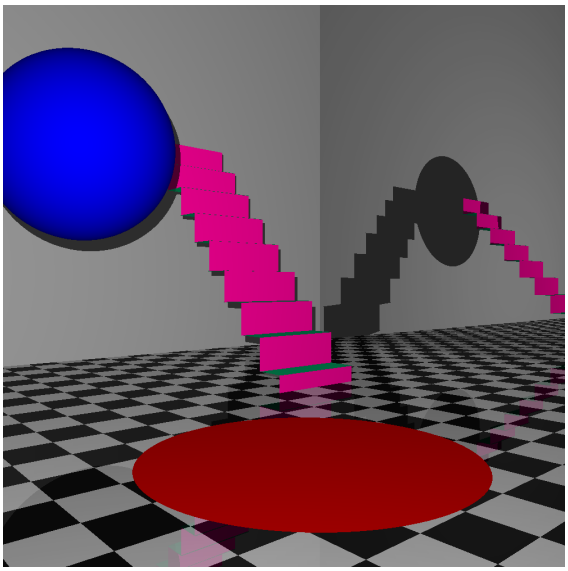
- A *piano room* with a checkered floor and a pink staircase
- A *rotating globe* on the sea
- A mesh which has the following 2 variants:
  - A *sphere* (a simpler one)
  - An *elephant* (a slightly more complex one)

See Figure 2 to see how all the 4 testcases look like. Instructions to run the testcases are specified in the README.md

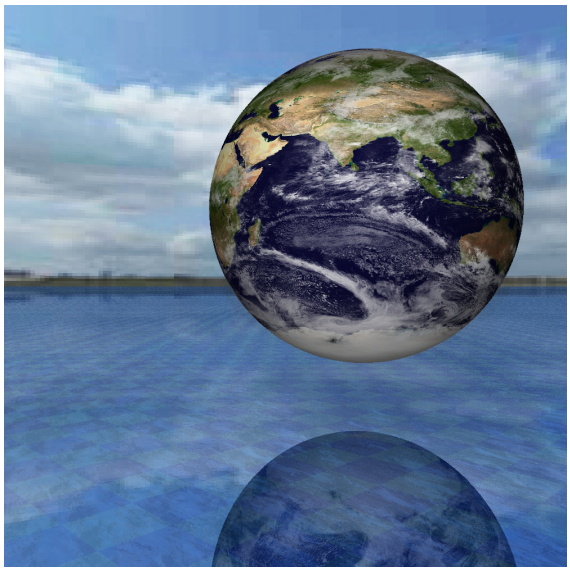
Intro to docker and packaging

4 How to Tackle this Homework

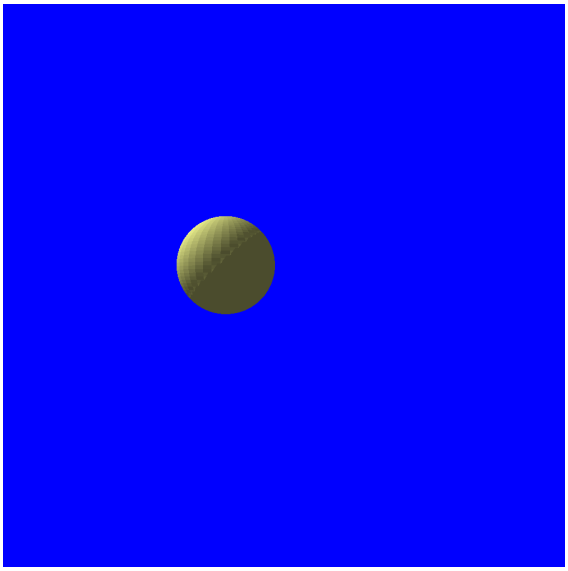
1. Create a private fork of the homework repository on GitHub. Instructions on how to do that are in this GitHub gist.
2. Look at the Raytracer code, see where you can make some obvious improvements. Profile, identify bottlenecks, and optimize them. Keep doing this again and again. Don't prematurely optimize.
3. Use `perf` to identify performance hotspots in your code. One super easy way to get started is to use `perf` to produce a flamegraph, and see the busiest functions in your code.
4. Use debuggers like `gdb` or `rr` to assist you with debugging. Ensure that your programs are safe from memory leaks using tools like `valgrind`.
5. Implement one optimization at a time. This helps in reasoning about your optimizations, and reproducing the effect of a specific optimization if needed.



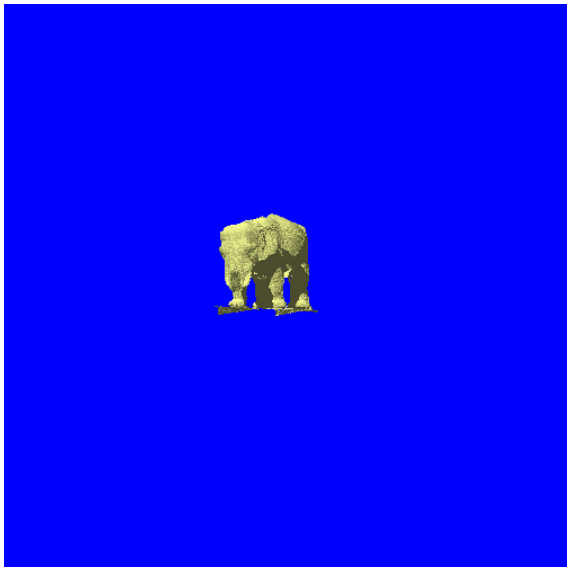
(a) Piano Room



(b) Globe on the sea



(c) Sphere on a meshgrid



(d) Elephant on a meshgrid

Figure 2: The 4 test cases that you will optimize.

6. Document your approach in the report. Make sure your code is reproducible (the whole point of `Docker` is to avoid the *oh but it runs on my computer...*)

7. Review & run the code your peers write!

5 What to Hand In

5.1 Report

Your report should be about (and no more than) 3 pages long(excluding references). Use the ACM Sigplan Template (follow the formatting in `sample-sigplan.tex`). It should include:

1. A **brief description** of your optimizations. This can include optimizations which helped performance, optimizations which did not help performance, or potential optimizations you did not implement and why (for example as profiling indicated it not to be helpful). Describe what kind of optimization you tried (e.g. algorithmic, data-structure, parallelism, caching, work-reduction, etc).

2. **Specify the assumptions** which an optimization relies on, and the scope in which your optimization applies. For example, one could hardcode the application to directly output the desired image (perhaps conditional on the exact input specification) – but this would only provide a speedup for that scene at that resolution.

3. **Include graphs** to indicate performance improvements/slowdowns (if any). Remember, the goal here is to not just get blind speedups, but also attempt to investigate why we are getting those speedups (or perhaps why we are not).

4. A brief **introspection** section for your optimizations: Performance engineering involves coming up with ideas that can improve performance, and profiling/deciding which ones will be impactful in practice. Sometimes you can do something but speedup up a function $100\times$ that represents $< 1\%$ of runtime won't change much. Similarly, sometimes you need to do prerequisite optimizations before the fruit of a bigger optimization can apply (e.g. rewrite a datastructure to be a linear scan is required before caching can be improved).

5.2 Artifact

You will be expected to submit an artifact, which will be independently evaluated by random team. To ensure the ease of reproducibility, the artifact should be a well documented git repo which should have at-least the following things in it's README.md

1. **All the tools needed to construct a binary from your source code**. We recommend modifying `docker/Dockerfile` and specifying the packages needed to install/build your tools.

2. **Instructions on how to build and run** your code. This includes how to run and benchmark your baseline code, and verify that your optimizations hold water. Remember, part of your grade depends on whether your peers are able to replicate the results stated in your paper.

3. **List your optimizations, and what commands, flags, or earlier git commits** are needed to evaluate their effectiveness.

Custom Hardware You are welcome to also try more advanced optimizations that require specific hardware to run – for example GPU tensor cores, RTX, DLSS, Intel AVX-512 or Arm NEON intrinsics(for the M1 Mac folks...).

However, **in this case you must submit two artifacts**: one artifact which can be run by anyone with a standard Unix machine (like the provided course VMs or Docker files), and a second artifact detailing optimizations based on non-replicable setups. We recommend using separate git branches or commit hashes to easily distinguish both artifacts.

6 How to Submit?

You will be using HotCRP to submit and peer-review your reports. We will open up the submission link as soon as the deadline nears. As for the code, it is sufficient to provide a link to the github repo you will be using for writing the code.

7 Some Tutorials

Please refer to this gist on a tutorial to help you out with the following aspects of the homework:

- Forking a private copy of a public repo.
- How to use `perf`

The README.md on the project page has enough information on how to build and run your own `Dockerfile` if you need more pre-existing software.