Project Design Proposal: Functional Ray Tracer

1. An overview of the purpose of the project

Motivation: Our team shares a deep interest in computer graphics, with ray tracing standing out as a fascinating and classic subject within this domain. A ray-tracer, renowned for its ability to create strikingly realistic images by mimicking how light behaves in virtual environments, has piqued our interest. Some team members boast prior experience and knowledge in computer graphics, which they can leverage to guide those less familiar with this field. While most ray-tracers are typically developed using languages like Python, our curiosity is drawn towards exploring the implementation of one using a functional programming language. Hence, we've set our sights on crafting a ray-tracer for our final project.

Purpose: In the realm of computer graphics, a ray-tracer is a significant tool that emulates light physics to craft images that resemble real-world scenes. Our project aims to delve into the fundamental concepts of ray tracing and execute them through a functional programming approach. The ultimate objective is to produce captivating visuals that replicate natural lighting effects and interactions within scenes. This exploration encompasses crucial aspects of ray tracing, including ray-object intersection, lighting models, reflections, and more.

Workflow: The project involves building a library consisting of interconnected modules, each serving as a component of our ray tracer. These modules encompass camera, color, light, output, parser, point-light, ray, scene, shape, vector, and subclasses for different shapes such as spheres. Once these modules are established, the parser can interpret incoming arguments from a JSON file. Subsequently, it will generate an RGB file as the final output, showcasing the synthesized visual representation derived from the ray tracing process.

2. A list of libraries you plan on using

a. Yojson
b. Domainslib (for parallelism, potentially)
c. Core_unix Command_unix

(Note: We decided not to use CamlImages to write output, as it supports only up to Ocaml 4. We will be using Ocaml5 and will write RGB files with our own modules, so that we can potentially incorporate Ocaml 5 parallelism later.)

3. **Commented module type declarations (`.mli` files) which will provide you with an initial specification to code to**
   - You can obviously change this later and don't need every single detail filled out
   - But, do include an initial pass at key types and functions needed and a brief comment if the meaning of a function is not clear.
    a. Please see the mli files in lib/
    b. Additionally, bin/main.ml parses command line flags and shows how the functions will be called from main. We have provided the ~doc parameter in main as documentation for each flag.
4. **Include a mock of a use of your application, along the lines of the Minesweeper example above but showing the complete protocol.**
    a. We think providing examples of the input file and the command line argument will be helpful as a mock use of our application.
    b. An input json file will specify all properties and entities of the 3D scene in which ray tracing will be performed.
        i. An example input json file is located at /example_input/input1.json
    c. The binary can be executed with the command
        i. $ ./_build/default/bin/main.exe --in example_input/input1.json --out output/out.ppm --height 500 --width 500 --rLimit 5 --cutOff 0.0001
    d. Ultimately, the ray-traced image of the scene will be written to the file specified by the --out flag.
5. **Also include a brief list of what order you will implement features.**
    a. Due to a relatively short time frame and the fairly complex structures of a ray tracer, we plan to implement features using a top-down approach, followed by incremental improvements.
        i. We will start by thinking how the ray_trace function should color and build the image representation functionally.
        ii. Next, we will implement the recursive get_color, which gets the color for a given pixel given a ray-shape intersection, which may involve use of first class modules for lights.
        iii. Then, we will implement the lighting calculations.

iv. Ray-shape intersection calculations.
v. Add more features as time permits, such as parallelism.

6. **If your project is an OCaml version of some other app in another language or a project you did in another course etc please cite this other project. In general any code that inspired your code needs to be cited in your submissions.**
    a. One of the team members had experience writing the functions for a C++ ray tracer skeleton code in the Computer Graphics course https://www.cs.jhu.edu/~misha/Fall23/Assignments/Assignment2/.
    However, the nature of our Ocaml ray tracer is rather different as we are designing our own modules from scratch and hopefully make it purely functional (except when writing output files).
    b. https://github.com/wiatrak2/raytracer inspired us to use json files as input for added readability and extensibility.
    c. https://github.com/CianLR/ocaml-raytracer is an Ocaml ray tracer using Ocaml classes.
    d. Both b. and c. had a fair amount of mutations and we'd like to make a more functional one. We also plan to use first-class modules for better abstraction.