Ray Tracing Spheres

Computer Graphics: Project 3A

1 - Objective

The goal of this project is to write a ray tracing renderer. Your program should be able to read scene data from a text file according to a defined scene description language. From this, your program will then render an image of the scene and write out the image to a file. This project is the first of a pair of projects which will accomplish this objective. For this first part you will cast eye rays into the scene for each pixel, test these rays for intersection with sphere objects, and then use the diffuse shading equation to find the color for each pixel. In the next project you will expand your Ray Tracer to detect intersections between rays and cylinders. You will also expand your shading function to include ambient and specular color as well as cast shadows and reflection. Keep this in mind when deciding implementation details for part A of this project.

2 - Deadline

This project should be submitted on T-Square by 11:55PM on Friday, March 9, 2018.

3 - Process

3.1 Download the base source

Download and unzip the folder with the base code for this project.

3.2 Project description

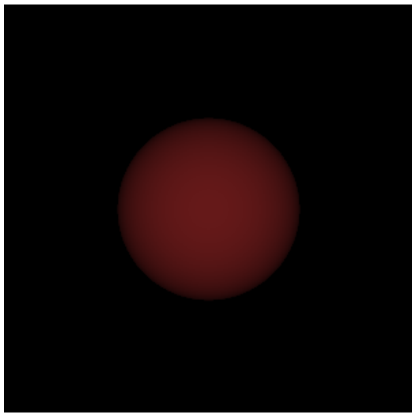
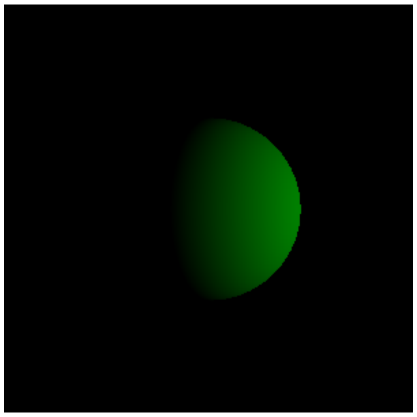
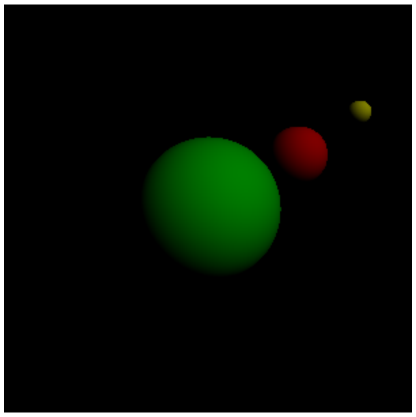
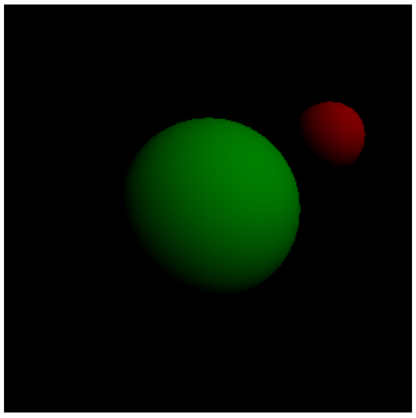
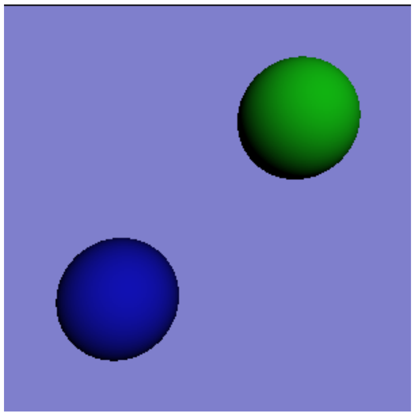
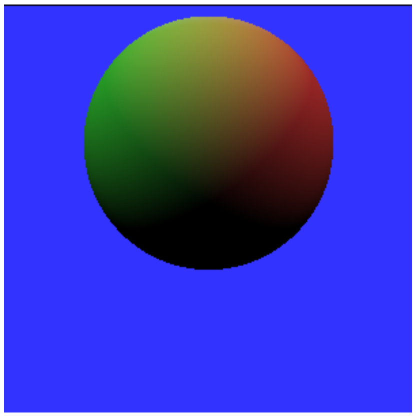
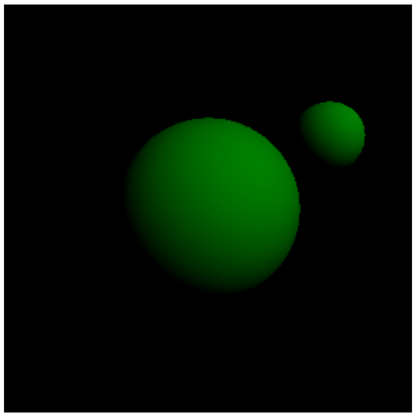
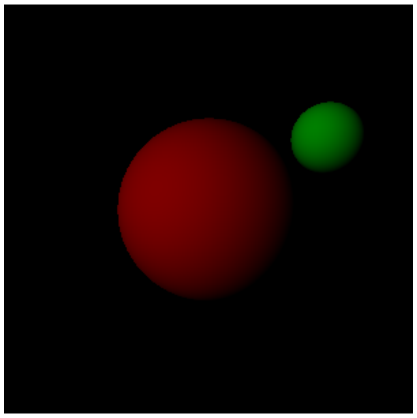
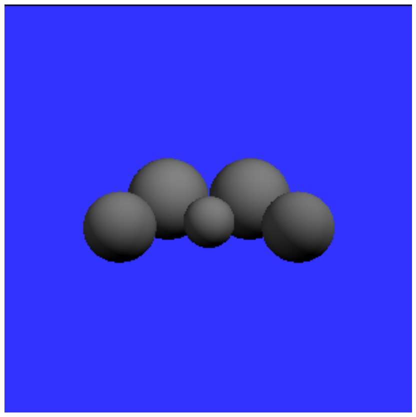
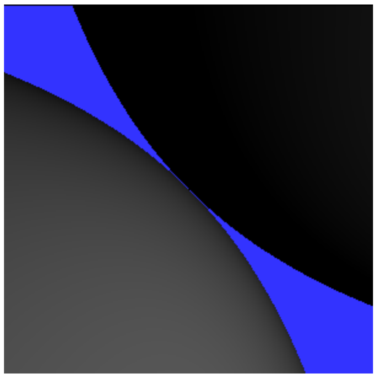
You have four primary goals for the first part of this project:

1. Initialize the scene based on commands from a .cli file
2. Cast eye rays for each pixel
3. Implement detection of ray intersection with spheres
4. Implement the diffuse shading equation

You can accomplish these goals however you see fit. A good approach would be to use object oriented programming practices and create objects for each of the major scene components (scene, light, surface material, ray, intersection point (hit), sphere, and later triangle). Global lists of scene objects could be stored to allow for easy access. You can then create a Ray object for each pixel and write a method which will take a Ray as input and test it against the scene objects for intersections, returning the closest Hit. This Hit object, containing all necessary information could be passed to a shading function which would implement the shading equation (for multiple light sources) and return the pixel color. Such an approach would be easy to test and to extend in the next stage of the project.

However you implement the ray tracer, your results should appear exactly like the examples on the following page.

Below are the correct images for the nine given scene files (i1.cli through i10.cli).

3.3 Source code

The source code provided parses a .cli file describing the properties of the scene. Each number key is assigned to a single example file and pressing it should reset the current scene and load the new one. These .cli files are simply text files. You can edit them using almost any text editor, either to look at them or to change them around to help with debugging.

You should modify the provided python source code in any way you see fit. Please include your name in the header as a comment. The source code is written in Processing/Python. Visit “Processing.org/reference/” for more information on built in functions and structure. You are NOT allowed to use most of the built in Processing/OpenGL graphics functions in this project (e.g. matrix stack, sphere and polygon drawing). The exception to this is that you may use the PVectors and the standard math functions. When in doubt about this, ask. You must set the color of each pixel manually. The easiest way to set a pixel’s color is to using the set() function, as is shown in the example code.

3.4 Scene Description Language

Each scene is described in a .cli file using the grammar described below. These files are contained in the /data folder. The suffix “cli” stands for “command language interpreter”.

fov **angle**

Specifies the field of view (in degrees) for a perspective projection. The viewer's eye position is assumed to be at the origin and to be looking down the negative z-axis (giving us a right-handed coordinate system). The y-axis points up.

background **r g b**

Background color. If a ray misses all the objects in the scene, the pixel should be given this color.

light **r g b x y z**

Point light source at position (x, y, z) and its color (r, g, b). Your code should allow up to 10 light sources. For the second part of this assignment, you will cause these lights to cast shadows.

surface **Car Cag Cab Cdr Cdg Cdb Csr Csg Csb P Krefl**

This command describes the reflectance properties of a surface, and this reflectance should be given to the objects that follow the command in the scene description, such as spheres and triangles. For this first part of the project, you only need to use the *second* group of three coefficients for diffuse color: Cdr Cdg Cdb. The first three values are the ambient coefficients (red, green, blue). The last group of three values specular coefficients. Next comes the specular power P (the Phong exponent), which says how shiny the highlight of the surface should be. The final value is the reflection coefficient (0 = no reflection, 1 = perfect mirror). You do not need to implement ambient and specular shading and reflections until the second part of this assignment (P3B).

Usually, 0 <= Cd,Ca,Cs,Krefl <= 1.

sphere **x y z radius**

A sphere with its center at (x, y, z) and the given radius.

cylinder **radius x z ymin ymax**

A vertical cylinder of a given radius. The axis of the cylinder is parallel to the y-axis, so all of your cylinders will be oriented up-and-down. The position of the cylinder’s axis is given by the (x,z) values. The lower and upper caps of the cylinders are at ymin and ymax. You do not need to implement cylinders until the second part of this assignment (P3B).

write [filename].png

Ray-traces the scene and saves the image to a PNG image file.

Note on color specification: Each of the red, green, and blue components range from 0.0 to 1.0.

3.5 Authorship Rules

The code that you turn in entirely your own. You are allowed to talk to other members of the class and to the Professor and the TA's about general implementation of the assignment. It is also fine to seek the help of others for general Processing/Python programming questions. You may not, however, use code that anyone other than yourself has written. The only exception to this is that you should use the provided source code that is particular to this project. Code that is explicitly not allowed includes code taken from the Web, from books, from previous assignments, from Github or from any source other than yourself. You should not show your code to other students. Feel free to seek the help of the Professor and the TA's for suggestions about debugging your code.

3.6 Submission

In order to run the source code, it must be in a folder named after the main file. Please keep the data directory and its .cli files in the directory that you turn in. When submitting any assignment, leave it in this folder, compress it into a zip file (not tar or rar) and submit via T-square.