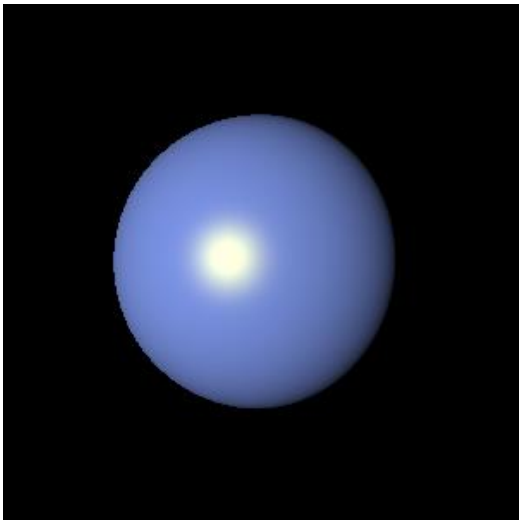
CMSC 435 / 634 Introduction to Computer Graphics

Project Assignment 5: Raytracing, Due May 9



Goals of this project:
Understand how images get rendered on your screen using computer graphics techniques. Once this project is completed, you will understand how rays and object(s) intersect. Since you will be working on your final project this week, this assignment excludes the more challenging (and perhaps more interesting) reflection, refraction, and texture that can be generated using ray tracing in graphics. You can take the challenge to implement them as extra credits. Some of the images I rendered is also included.

The Assignment
Ray tracing is a simple and powerful algorithm for rendering images. Within the accuracy of the scene and shading models and with enough computing time, the images produced by a ray tracer can be physically accurate and can appear indistinguishable from real images. The ray tracer we do in this class will not be powerful enough to produce physically accurate ones and if you are really interested, go on to take CMSC 635, where you will learn more on how to build modern physically accurate images.

In this assignment, your ray tracer will have support for:

- Spheres
- Specular highlight, diffuse shading, and ambient shading
- Arbitrary orthogonal cameras

Your output should be:

- An image in PPM format (the PPM format is an ASCI format and is human readable.) https://en.wikipedia.org/wiki/Netpbm_format has the format

description. You can also try to save the example image files yourself to check out the results.

Input File Format

The input file for your ray tracer is in plain text including the following definitions:

One is a *scene* element that contains two others, called camera and surface. The *surface* element has an attribute named *type* that has the value *Sphere*. It also contains a center element that contains the text "0, 1, 1", which in this context would be interpreted as the 3D point (0, 1, 1).

Note that you can use the same input file as your view assignment and reuse your loadFile routine.

An input file for the ray tracer always contains one *scene* element, which is allowed to contains tags of the following types:

- *Surface*: This element describes a geometry object. It must have an attribute *type* with value *Sphere* (we won't use any other geometries). For Sphere, *center* containing a 3D point, and *radius* containing a real number.
- *Camera*: This element describes the camera. It is described by the following elements:
    - *viewPoint*, a 3D point that specifies the center of projection.
    - *viewDir*, a 3D vector that specifies the direction toward which the camera is looking. Its magnitude is not used.
    - *viewUp*, a 3D vector that is used to determine the orientation of the image.
    - *projNormal*, a 3D vector that specifies the normal to the projection plane. Its magnitude is not used, and negating its direction has no effect. By default it is equal to the view direction.
    - *projDistance*, a real number *d* giving the distance from the center of the image rectangle to the center of the projection.
    - *viewWidth and viewHeight,* two real numbers that give the dimensions of viewing window on the image plane.

  The camera's view is determined by the center of projection (the viewpoint) and a view window of size *viewWidth* and *viewHeight*. The window's center is positioned along the view direction at a distance d from the viewpoint. It is oriented in space so that it is perpendicular to the image plane normal and its top and bottom edges are perpendicular to the up vector.
- *Image:* This element is just a pair of integers that specify the size of the output image in pixels.
- *Light:* This element describes a light. It contains the 3D point position and the RGB color *color*.
- *Shader:* This element describes how a surface should be shaded. It must have an attribute type with value Lambertian. The Lambertian shader uses the

RGB color *diffuseColor*. A shader can appear inside a surface element, in which case it applies to that surface.

Your code only needs to handle the following scene which contains a single sphere.

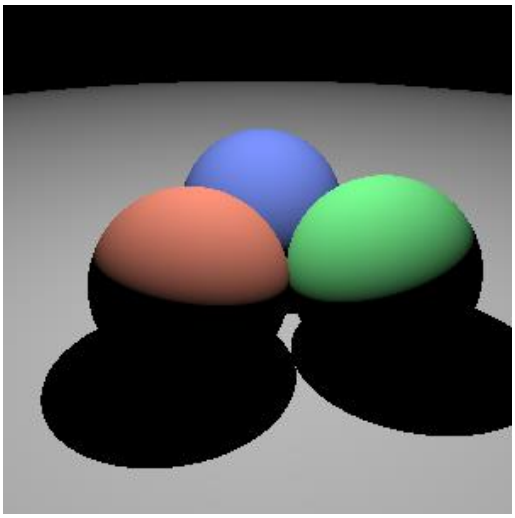
```
viewPoint: 5 4 3
viewDir: -5 -4 -3
projNormal: 5 4 3
viewUp: 0 1 0
projDistance: 5
viewWidth: 2.5
viewHeight: 2.5
image size: 300 300
shader: Phong
shader-diffuseColor:.2 .3 .8
shader-specularColor: 1 1 0
specularExponent: 50
sphereCenter: 0 0 0
sphereRadius: 1
lightPosition: 3 4 5
lightIintensity: 1 1 1
Ka = Kd = ks = 0.5;
```

Your ray tracer framework:
Your ray tracer may contain a parser. You can assume that the parameters are space delimitated. You can use the vector calculator lib provided in class. And finally it is okay to hard-code your ray tracer given the time constraints.

Start **now**! Or you will probably not finish. Really, I promise you will not be able to do it in the last two days.

Extra credits

- For 20 points, render your scene with three spheres and shadow similar to the scene as the following (you will need to define the plane and two more spheres.) **This part is mandatory for graduate student to get full credit. This scene includes a plane and three spheres. The shadow is hard shadow.**



- For 20 points, implement two of the visual effects among soft shadow, transparency, reflection, refraction, mirror, and texture.
- For 10 points, implement something you find interesting from those I rendered.

The maximum points are 150 for 435 students or 130 for 634 students.

What to turn in

Source code only by email to Kevin. Please do not include any .o files. Please include:

- A README with your handin containing basic information about your design decisions and any known bugs or extra credit;
- How to compile and run your code as if you are telling a colleague that is to continue the development.

Note: Please comment on your code. The better Kevin understands your code, the higher your grade is likely to be.