Program 1: ray tracer

out: Thursday March 28, 2024
due: Thursday April 11, 2024 (24pm)

1. Overview

Ray tracing is a simple and powerful algorithm for rendering images. With careful attention to scene details and shading models, coupled with sufficient computing time, the images produced by a ray tracer can be physically accurate and can appear indistinguishable from real images. Your ray tracer will not be able to produce physically accurate images though.
In this assignment, your ray tracer will have support for:
• Spheres (50%)
• Lambertian (25%) and Phong shading (25%)

Optionally, implementing the following functionalities would be interesting. However, please note that no additional points will be awarded for these features.
• Axis-aligned boxes
• Point lights with shadows
• Arbitrary perspective cameras
Some framework code (rayTracer.py - less than 60 lines) is provided to save you the time to implement I/0, an XML parser, and vector operations.
However, the framework does not contain any code that does any actual ray tracing—you will develop that code yourself.
When you write the ray tracing code, you have the freedom to determine the design you believe is best. You may want to add some classes to the program, and there are many choices about where to put the various parts of the computation. Any solution that correctly meets the minimal requirements below and is clearly written will receive full credit. For example, spheres without glossy shading at all will still receive 75% of the full credit to encourage submission. The textbook, the lectures, and the course staff (including the professor) are all sources of information about good approaches to coding up a ray tracer.
This is not to say that you need to write a lot of new code. For reference, the framework contains about 60 lines of code, and our complete solution supporting all the optional features contains three small additional classes and about 200 additional lines of code.

2. Requirements

1. Use a ray tracing algorithm.
2. Support spheres.
3. Support the Lambertian and Blinn-Phong shading models, as defined in Shirley 9.1–9.2 and in the lecture notes.
4. Support point lights that provide illumination that does not fall off with distance.

You do not need to worry about malformed input, either syntactically or semantically. For instance, you will not be given a sphere with a negative radius or a scene without a camera.

3. File format

The input file for your ray tracer is in XML. An XML file contains sequences of nested elements that are delimited by HTML-like angle-bracket tags. For instance, the XML code:

```
<scene>
    <camera>
    </camera>
    <surface type=Sphere>
        <center>1.0 2.0 3.0</center>
    </surface>
</scene>
```

contains four elements. 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 "1.0 2.0 3.0", which in this context would be interpreted as the 3D point (1, 2, 3). 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 geometric object. It must have an attribute type with value Sphere or Box. It can contain a shader element to set the shader, and also geometric parameters depending on its type:
  – for sphere: center, containing a 3D point, and radius, containing a real number.
  – for box: minPt and maxPt, each containing a 3D point. If the two points are (xmin , ymin , zmin)
  and (xmax , ymax , zmax) then the box is [xmin , xmax] × [ymin , ymax] × [zmin , zmax].

• 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 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 by 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 or Phong. The Lambertian shader uses the RGB color diffuseColor, and the Phong shader additionally uses the RGB color specularColor and the real number exponent. A shader can appear inside a surface element, in which case it applies to that surface. It can also appear directly in the scene, which is useful if you want to give it a name and refer to it later from inside a

surface (see below). If the same object needs to be referenced in several places, for instance when you want to use one shader for many surfaces, you can use the attribute name to give it a name, then later include a reference to it by using the attribute ref. For instance:

```
<shader type="Lambertian" name="gray">
        <diffuseColor>0.5 0.5 0.5</diffuseColor>
</shader>
<surface type="Sphere">
        <center>0 0 0</center>
        <shader ref="gray"/>
</surface>
<surface type="Sphere">
        <center>5 0 0</center>
        <shader ref="gray"/>
</surface>
```

applies the same shader to two spheres. Really, the file format is very simple and from the examples we provide you should have no trouble constructing any scene you want.

4. Framework

The framework for this assignment includes a simple main program, some examples for basic image processing and parsing for the input file format.

4.1 Parser

The framework utilizes Python's built-in XML parsing. The parser simply reads a XML document and instantiates an object for each XML entity, adding it to its containing element.
For instance, the input

```
<scene>
    <surface type="Sphere">
        <shader type="Lambertian">
            <diffuseColor>0 0 1</diffuseColor>
        </shader>
        <center>1 2 3</center>
        <radius>4</radius>
    </surface>
</scene>
```

results in a parse tree which are automatically generated from the parser. There is more detail for the curious in the rayTracer.py.
The practical result of all this is that your ray tracer is handed a scene that contains objects that are in one-to-one correspondence with the elements in the input file. You only need to use the information that is already there.

4.2. rayTracer.py

The main method holds the entry point for the program. The main method is provided, so that your program will have an command-line interface compatible with ours. It treats each command line argument as the name of an input file, which it parses, renders an image, and writes the image to a PNG file. You need to write some additional code to do the actual rendering.

4.3 Image

The framework creates an array of floats and provide the requisite code to set pixels and to output the image to a PNG file.

4.4 The Color class

The framework contains a class to represent RGB colors.
It supports gamma correction.

5. Submission and FAQ

5.1. How to run the skeleton code

After installing the necessary packages (python3, pil and numpy), unpack the zip file.

Then run:

```
python3 rayTracer.py scenes/one-sphere.xml
```

This will render to a PNG file of the same name, but with a ".png" extension.
The supposed output files are already in the folder, and the above command will overwrite the scenes/one-sphere.xml.png file with a mostly empty image.

5.2. Notice

Submit a single zip file – [studentID]-PA1.zip
All the necessary python files should be in the zip, and the main file should be rayTracer.py