Ray Tracer II: Shading and Recursive Ray Tracing

Due date: 11:59pm May 7, 1999

NO LATE ASSIGNMENTS WILL BE COUNTED TOWARDS YOUR GRADE!

1 Overview

In this assignment you will add lights, material properties and recursive ray tracing to the ray caster you have written.

2 What your program should do

As before, your program should read a file specified on the command line and create an image of the scene described in the file.

In addition to the name of the file, on the command line you should be able to specify the size of the output image, the maximal depth of recursion, and the ray weight threshold.

3 File format description

In addition to the commands described in the previous assignment, you need to add several new commands, to specify lights and material properties.

Light sources. Light sources have no visible shape of their own. They are just points or areas which emit light. We will use only one type of light sources: point lights. The syntax for a light source is:

```
light_source { <X, Y, Z> color rgb<R, G, B> }
```

Where X, Y and Z are the coordinates of the location and R, G, B are the components of the color in the range 0 to 1. For example,

```
light_source { <3, 5, -6> color rgb<0.5, 0.5, 0.5> }
```

is a 50% white light at X=3, Y=5, Z=-6.

Pigment. The color or pattern of colors for an object is defined by a pigment statement. For example,

```
sphere {1
  pigment {color rgb<1.0, 0.0, 0.0> }
}
```

defines a red sphere. The color you define is the way you want it to look if fully illuminated. The parameter is called `pigment` because we are defining the basic color the object actually IS rather than how it LOOKS. We will implement only the simplest pigment type, which uses the `color` statement to specify the pigment. As in the case of light sources, the color statement has the form `color rgb<R, G, B>`. In addition, the color statement may have the form `color rgbf<R, G, B, F>`, where `F` stands for transparency (called "filter" in POV-ray). The default is 0, which means that the surface of the object is not transparent.

Finish. The finish properties of a surface can greatly affect its appearance. How does light reflect? What happens when light passes through? What kind of highlights are visible. To answer these questions you need a finish statement, which defines other parameters of the lighting model. An example of a finish statement is

```
sphere { 1
  pigment {color rgb<1.0,0.0,0.0> }
  finish {phong 0.3}
  }
```

In general, the finish statement contains a number of finish items, that control diffuse reflection, specular reflection and refraction (for solid objects).

Diffuse reflection items. The keyword `diffuse` is used in a finish statement to control how much light of the light sources is reflected via diffuse reflection. For example:

```
finish {diffuse 0.7}
```

means that 70% of the light is reflected, that is, the radiance of a ray coming from a light source is multiplied by 0.7 before being added to the reflected radiance. The default value is diffuse 0.6.

The light you see in dark shadowed areas comes from diffuse reflection off of other objects. This light cannot be directly modeled using ray tracing. However we can use a trick called "ambient lighting" to simulate the light inside a shadowed area. The ambient light term assigned to an object is added to the radiance of all outgoing rays; the kwyword introducing the item is `ambient`. The default value is very little ambient light (0.1). The value can range from 0.0 to 1.0.

Specular reflection. When light does not diffuse and it reflects at the same angle as it hits an object, it is called "ideal specular reflection". Such mirror-like reflection is controlled by the `reflection` keyword in a finish statement. This is the coefficient For example:

```
finish {reflection 1.0  ambient 0  diffuse 0}
```

This gives the object a mirrored finish. It will reflect all other elements in the scene. The value can range from 0.0 to 1.0. By default there is no reflection.

Phong highlights. The `phong` keyword controls the amount of Phong highlighting on the object. It causes bright shiny spots on the object that are the color of the light source being reflected – the specular term of the lighting equation.

The value of `phong` is typically from 0.0 to 1.0, where 1.0 causes complete saturation to the light source's color at the brightest area (center) of the highlight. The default phong 0.0 gives no highlight.

The size of the highlight spot is defined by the `phong_size` value, which is the Phong exponent, or shininess. The larger the `phong_size`, the tighter, or smaller, the highlight and the shinier the appearance. The smaller the `phong_size`, the looser, or larger, the highlight and the less glossy the appearance.

Typical values range from 1.0 (very dull) to 250 (highly polished) though any values may be used. Default phong_size is 40 (plastic) if phong_size is not specified. For example:

```
finish {phong 0.9  phong\_size 60}
```

If `phong` is not specified then `phong_size` has no effect.

Refraction. For closed transparent objects (spheres, cones, cylinders, boxes) one can define the index of refraction. The keyword used to define it is 'ior:

```
sphere{ 1
  pigment{ rgbf<1.0,1.0,1.0,0.5>
  finish {
         ior 1.5
       }
   }
```

The default ior value of 1.0 will give no refraction. The index of refraction for air is 1.0, water is 1.33, glass is 1.5, and diamond is 2.4. For the index of refraction to have any effect, the object should be transparent, that is, the color in the pigment statement should have a "filter" component different from 0.0.

The following table gives the correspondence between various POV finish keywords and coefficients used in the lecture notes. Suppose the pigment is defined as `color <R,G,B>`; we list the lighting model coefficients for the red radiance component; the green and blue components are computed in a similar manner, with R replaced with G and B.

K_{diff}	`diffuse*R`
K_{spec}	`phong*R`
k_{refl}, k_{trans}	`reflection`
t	"filter" component of color
p	`phong_size`
$K_{amb}L_{amb}$	`ambient`

Note that we use `reflection` for both k_{refl} and k_{trans}.

4 Ray tracing

Implement the basic ray tracing algorithm described in class; for each point, shoot rays to all light sources, the reflected ray (if `reflection` is not zero) and the refracted ray (if the "filter" component of the pigment is not zero). In addition, implement the coordinate-aligned bounding box acceleration, that is, for each finite object (any object excluding quadrics), compute a bounding box (it need not be the smallest one possible); before intersecting a ray with the object, intersect it with the bounding box. Make the code for ray-bounding box intersection as efficient as you can.

The recursive spawning of rays is terminated when the maximal depth specified on the command line is reached, or when the ray weight is below a threshold. The weight is computed as follows: for pixel rays (the rays starting at the camera) it is set to 1. Each reflected or refracted ray is assigned a weight equal to the product of its current weight with the coefficient with which its contribution enters the summation in the lighting equation.

5 What to turn in

Generate 512 by 512 images of several test scenes that will be posted.