

Ray Tracer II: Shading and Recursive Ray Tracing

Due date: 12:00pm December 17, 1999

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 statements described in the previous assignment, we add several new statements, to specify lights and material properties.

Light sources. Light sources have no visible shape of their own. 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 { <0,0,0>, 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 “filter”, and determines how transparent the object is; the actual transparency for each color component is determined by the product of the filter value and the color component value. E.g. $\langle 1,0,0,0 \rangle$ means an object that allows only the red light through. The default filter value is 0, which means that the surface of the object is not transparent.

Finish. The finish statement, which defines other parameters of the lighting model. In general, the finish statement contains a number of items, that control ambient, diffuse and specular reflection. A complete finish statement looks like this:

```
finish {  
    ambient 0.2  
    diffuse 0.7  
    phong 0.5  
    phong_size 25  
    metallic 1  
    reflection 0.8  
}
```

Any item can be omitted. When it is the case, it is assigned a default value. The default values are listed below:

```
finish {  
    ambient 0.1  
    diffuse 0.6  
    phong 0.0  
    phong_size 40  
    metallic 0  
    reflection 0.0  
}
```

Thus, `finish {}` is exactly equivalent to the expanded statement above. The `ambient` component has effect only if there is a non-zero ambient light, which is specified using a special statement like this:

```
global_settings {ambient_light COLOR}
```

where `COLOR` is of the form `rgb <r,g,b>` as for a light source. The default is no ambient light.

Refraction. For closed transparent objects (spheres, cones, cylinders, boxes) one can define the index of refraction. The keyword used to define it is `ior`, which as to be inside a special object modifier `interior`.

```
sphere{ 1  
    pigment{ rgbf<1.0,1.0,1.0,0.5> }  
    interior {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.

Complete lighting formula. We state the complete formula for one component (red); the formulas for the other two components are obtained by replacing red with green or blue. Assume that for a given point there are M visible lights, with colors r_i, g_i, b_i , $i = 1 \dots M$. Assume that the pigment for the object was given by `pigment { color rgbf < $m_r, m_g, m_b, m_f$ >}`, and the finish is

```
finish {
    ambient  $k_{amb}$ 
    diffuse  $k_{diff}$ 
    phong  $k_{spec}$ 
    phong_size  $p$ 
    reflection  $k_{refl}$ 
}
```

Further, assume that the direction to the i -th light source is L_i , the direction to the eye (or, in the case of a recursive ray, the incoming direction) is V , the normal is N , and the reflected direction is R .

The total red intensity at the point is given by

$$(1-m_f)k_{amb}m_r r_{amb} + \sum_{i=1}^M (1-m_f)r_i \left(k_{diff}m_r(N, L_i) + k_{spec}\{m_r \text{ or } r_i\}(R, L_i)^p \right) + k_{refl}r_{refl} + m_fm_r r_{trans}$$

In the formula above r_{refl} and r_{trans} are the red components of the recursively traced reflected and refracted rays. Note that for a transparent object ($m_f \neq 0$) the ambient component and the diffuse component are attenuated and the specular and reflected components are not. This is a peculiar POV-Ray feature and this may be not true in other ray tracers. In addition for the specular component is scaled by either m_r (the material red) if `metallic` keyword is present in the finish, or by r_i (the light red) if it is not.

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 (or, if your ray tracer is too slow, 256 by 256) of the test scenes posted on the class page.