CS180 Programming Assignment #4

Due: 11:59pm, December 8, Friday

In this homework, you are to implement a simple ray tracer. You will be given a scene description file with the following sections:

(1) Camera: the camera is always a perspective one with its “eye” (center of projection) located at the global origin. The camera axes line up with the global axes. The image plane is located at z=-1 (so the image is not inverted) and of a size 1x1 centered on the z-axis. The only parameter the camera has is its spatial resolution. Hence, a line like

```
camera 640
```

means the camera has a 640x640 image array. Note that to avoid distortion, the aspect ratio will always be one (or CCD width and height are the same and only one number is given in the camera line).

(2) Objects: we allow only two types of objects: spheres and planes. These primitives have the following common parameters: center location (x, y, z), dimension, intrinsic color (r, g, b, where 0<=r,g,b<=1), reflectivity (0<=R<=1), and transparency (0<=T<=1). An optional texture parameter can be specified as an image (in ppm format) to be pasted onto the surface. A plane has two additional parameters: normal (nx, ny, nz) and head-up (hx, hy, hz) directions.

For a sphere, dimension is a single number for its radius, and for a plane, dimension is a set of two numbers for the plane’s length and width in its own x and y directions, respectively. A plane’s x-axis is determined by the cross product of its head-up and normal directions, in that order. A plane’s y-axis is determined by the cross product of its normal and x directions, in that order. Furthermore, these parameters are used to determine the local coordinate system for the plane. As a plane is not used as the bounding surface for a solid, there is no front and back facing distinction here. Both faces of a plane are visible and can be “front facing.”

Note that if no texture parameter is specified, the surface color is specified in the “color” line. If a texture parameter is given, the texture color becomes the surface color. Texture mapping for a plane is done this way: We pick from the texture image the largest subarea that has the same aspect ratio as the plane and map that subarea onto the plane. Note that it may be necessary to rotate the texture image 90 degrees for mapping. For a sphere, we pick the largest rectangular area in the image with an aspect ratio of 2 (azimuth angle) to 1 (elevation angle). The long side will cover 0 to 360 degrees going around the sphere along the equator. The short side will cover -90 to +90 degrees from the South Pole to the North Pole. You can assume that the sphere has its axes aligned with the global axes.

So a sphere might be specified as:

```
sphere
       dimension      5
       center         10 10 -10
       color           1.0 0 0
       reflectivity   0.5
       texture         wood.ppm
```

and a plane might be specified as:

```
plane
       dimension      5 10
       center         10 -5 -10
       color           1.0 0 1.0
```

```
        normal        0 0 1
        headup        1  1 0
        reflectivity  0.7
```

(3) Lights: we allow only point light sources. Each light has the following parameters: location (x, y, z) and color (r, g, b, where 0<=r, g, b<=1). Or a light may look like this:

```
    light
        location       20 -5 -10
        color          1.0 1.0 1.0
```

You can expect only one camera section but multiple object and light sections, intermingled in no particular order. Regardless of the ordering of light and object sections, all lights shine on all objects (unless, of course, some intervening objects block the light).

Your ray-tracer is "basic" in the sense that only elementary ray-tracing functions need be implemented. In particular, (1) only one ray needs to be traversed per pixel (no anti-aliasing), (2) only one shadow ray is traced per light source (no extended light), (3) only one single reflective ray is traced per each ray-object intersection if reflectivity is not zero, (4) you can assume that T=0 always, so no refractive ray needs to be traced, and (5) our scene files are quite simple and contain very few spheres and planes, it is not necessary to build elaborate space partitioning data structures (e.g., Octree) for speeding up the intersection operations.

Programming Hint:

The skeleton of your program should look something like below.

```
for (each scan line) {
     for (each pixel in scan line) {
          compute ray direction from COP (eye) to pixel
          for (each object in scene) {
              if (intersection and closest so far) {
                  record object and intersection point
              }
              accumulate pixel colors (one level)
                      - shadow ray color
                      - reflected ray color (recursion)
                      - refracted ray color (recursion)
          }
      }
 }
```

You need to implement the basic ray-object intersection, depth-sorting (for determining ordering to the eye and to the light sources), and tree traversal (build top down and fill in colors bottom up). You can hardcode your recursion depth to 8 or experiment with the depth to determine a good tradeoff between image quality and runtime. When your ray reaches the background, you should fill in with some suitable background color (or ambient color, or a background texture pattern). Output your ray-traced pictures in any popular image format (e.g., jpeg or ppm) for viewing. Your program should be called raytracer and take two command-line argument: the name of the scene description file and the output image, or "raytracer scene_file_name output_image_file" to call your program. You can output the results in any image format that you feel comfortable in using. The recommendation is ppm format (anyway, you do need to be able to parse ppm format images as texture input) and you can use ImageMagick to convert ppm images to jpeg later.