CSC 433 / 533: Spring 2017
Assignment 4

Assigned: Thursday, April 6th
Due: 11:59pm Thursday, April 27th

For this assignment you extend a fully functional ray tracer program. Your program will be graded by running your modified ray tracer on the Mac workstations on the 9th floor in the Gould-Simpson building.

As before:

1. Provide a Makefile that builds the program. We will not grade programs that do not compile.
2. Your directory needs to contain all source code for the modified ray tracer and required scene files.
3. Report any errors to stderr and provide a usage message when the program is invoked with no scene file name.
4. Include a readme.txt file and include any special instructions or assumptions.
5. Comply with the University of Arizona Code of Academic Integrity. Review the course policies at https://www2.cs.arizona.edu/classes/cs433/spring17/policies.html

Tasks:

1. This assignment starts with Peter Shirley's raytracer and adds a procedural texturing capability. Shirley, one of the author's of the textbook we've been using, has created three minibooks for the Kindle. Each book costs $2.99. While they would likely help, I don't plan on you needing to buy these books.
    a. Ray Tracing in one weekend
    b. Ray Tracing: the Next Week
    c. Ray Tracing: The Rest of your Life
2. He provides source code, and that is what you will be starting with for this assignment.
3. **raytracer**
    a. In your makefile create an executable named ***raytracer***.
    b. Input to the ***raytracer***:
        1. The program takes one parameter which is a scene file name, for example
            raytracer oneSphere.txt > image.ppm
        2. The format for the scene file is described later in this document.
4. Compile Shirley's raytracer, produce a ppm file, and examine the resulting ppm file.
5. Modifications to the ray tracing code. Shirley in his second book discusses many additions to the raytracer, including creating a checkerboard texture, Perlin noise for texture, an image for texture, and bounding volumes. This assignment includes adding the checker board texture.
6. Modify the existing scene generation.
    a. Do not call the random_scene() function.

b. Create a 'hitable list' using the scene description in the scene file. Set the *world* to hold that hitable list.
c. Generate the scene to be ray traced by reading a scene file.
    i. Add checkerboard texture mapping. Peter's Shirley's second book on raytracing discusses this. Here is my summary.
        1. You will be adding a new class to the source code for texture mapping.
        2. You will be expanding the lambertian material class which will change how to create a solid color material property. Change from
           `new lambertian(vec3(0.5,0.5,0.5)`
           and replace the vec3(…) with
           `new constant_texture(vec3(…)`
           and will include the ability to use a checker board texture.

d. One step in texture mapping is to create a new class using the following header:

```
//
// new texture class to support solid colors and procedural textures
// new for assignment 4
//

#ifndef TEXTUREH
#define TEXTUREH

#include "vec3.h"

class texture {
public:
	virtual vec3 value(float u, float v, const vec3& p) const = 0;
};

class constant_texture : public texture {
public:
	constant_texture() { }
	constant_texture(vec3 c) : color(c) { };
	virtual vec3 value(float u, float v, const vec3& p) const {
		return color;
	}
	vec3 color;
};

class checker_texture : public texture {
public:
	checker_texture() { }
	checker_texture(texture *t0, texture *t1) : even(t0), odd(t1) {  }
	virtual vec3 value(float u, float v, const vec3& p) const {
		float sines = sin(10 * p.x())*sin(10 * p.y())*sin(10 * p.z());
		if (sines < 0)
			return odd->value(u, v, p);
		else
			return even->value(u, v, p);
	}
	// the checker odd/even pointers can be to a constant texture or to
some other procedural texture
```

```
        // this is in the spirit of shader networks introduced by Pat
Hanrahan back in the 1980s
        texture *odd;
        texture *even;
};

#endif
```

e. Include the texture header in the material header and in main.cc

f. You also need to replace the previous lambertian sub-class with this new one.

```
class lambertian : public material {
public:
        lambertian(texture *a) : albedo(a) {}
        virtual bool scatter(const ray& r_in, const hit_record& rec, vec3&
attenuation, ray& scattered) const {
                vec3 target = rec.p + rec.normal + random_in_unit_sphere();
                scattered = ray(rec.p, target - rec.p);
                attenuation = albedo->value(0, 0, rec.p);
                return true;
        }
        texture *albedo;
};
```

g. For example, to use the checker texture, you could change the base sphere in the random_scene function from

   list[0] = new sphere(vec3(0,-1000,0), 1000, new lambertian(vec3(0.5,0.5,0.f)

   to

   texture *checker = new checker_texture(new constant_texture(vec3(0.2,0.3,0.1), new constant_texture(vec3(0.9,0.9,0.9));

   list[0] = new sphere(vec3(0,-1000,0),1000, new lambertian(checker));

h. To use the constant texture for a solid color sphere change from

   new sphere(vec3(2.f, 0.5f, 2.f), 0.5f, new lambertian(vec3(0.9f, 0.9f, 0.9f))))

   to

   new sphere(vec3(2.f, 0.5f, 2.f), 0.5f, new lambertian(new constant_texture(vec3(0.9f, 0.9f, 0.9f)))

Hints:

1. This raytracer saves the image in PPM (Portable Pixmap) format. For MS Windows I've been using irfanview (http://www.irfanview.com/). Peter Shirley says he uses ToyViewer on the Mac.
2. There is a program, `preview`, available on the Macs in GS that you can use to display ppm files.
3. Once raytracer is compiled, you can run raytracer > image.ppm. Then, display the file image.ppm with a viewer to see the result.
4. To run the raytracer on MS Windows I needed a substitute for the function drand48(). I put this code at the top of camera.h to use the replacement function.

```
#ifdef _WIN32
double drand48(void);
#endif
```

   I have provided a simple replacement in the source file drand48.cc to use if you are using visual studio.
5. During testing you can substantially reduce run time by reducing image size, number of spheres, and number of samples per pixel.
6. I also found it useful to print out the line number once every 100 lines to provide some feedback during a run.
7. The first sphere created in the function random_scene is a very large sphere used for a base for other spheres. While not mandatory, it can be useful to have this in a scene.
8. Consider using an C++ STL vector to build list of hitable pointers while reading the scene file, and then copy that list to a hitable_list, and assign that hitable_list to the *world* variable. The function random_scene returns a hitable_list that was assigned to the world variable.

1. **Grading rubric – 100 points total**
    1. Read a scene file and create a ppm output file with:
        a. [10 points] A lambertian base sphere without checker board texture and one lambertian sphere
        b. [10 points] A lambertian base sphere without checker board texture and one metal sphere
        c. [10 points] A lambertian base sphere without checker board texture and one dielectric sphere
        d. [10 points] A lambertian base sphere with a checker board texture and one lambertian sphere
        e. [10 points] A lambertian base sphere with a checker board texture and one metal sphere
        f. [10 points] A lambertian base sphere with a checker board texture and one dielectric sphere
        g. [40 points] Read arbitrary scene files with a variety of spheres, and the spheres will have a variety of materials. No scene file will contain more than 250 spheres. Most scene files will have less than that.

Scene File Format

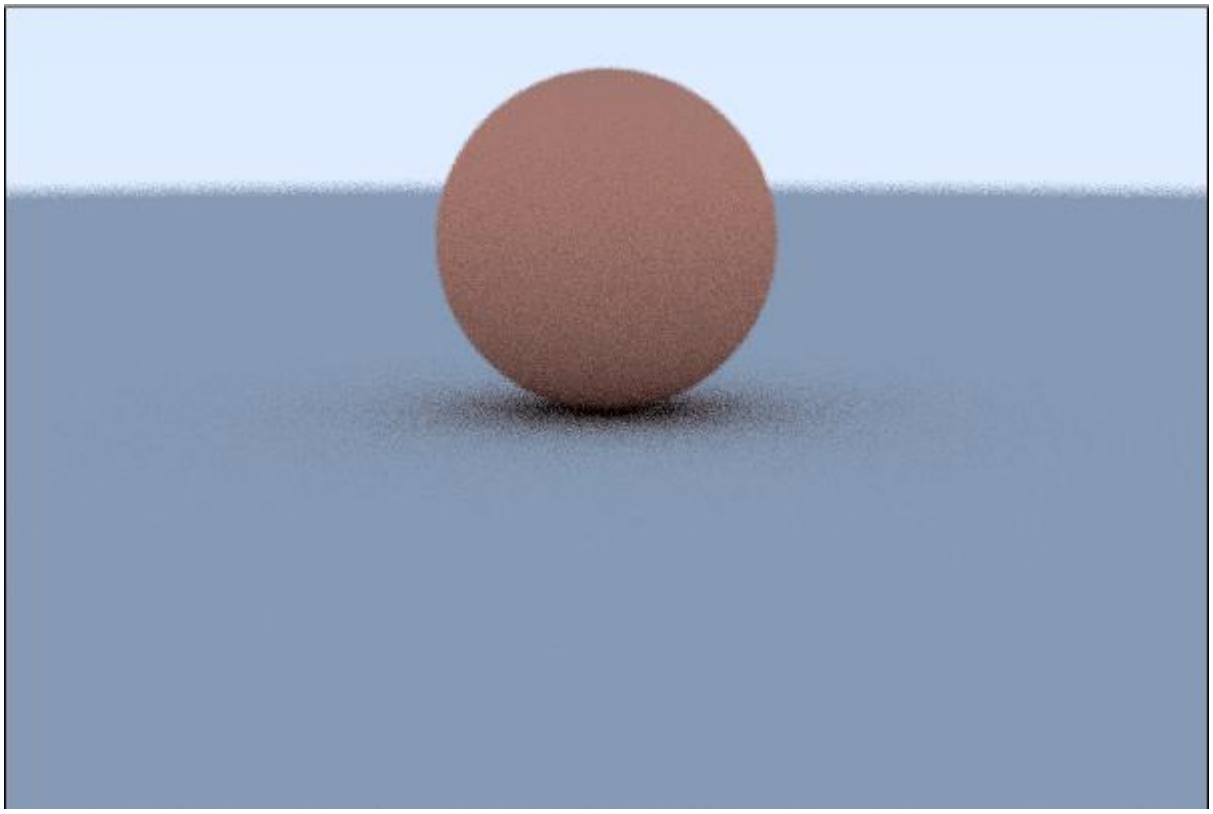
The format is keyword value pairs. Comments are included in this description following a //. However, a scene files will not have comments. You do not have to deal with comments when parsing the scene file. Square brackets indicate

```
nx <int>                    // screen resolution in x
ny <int>                    // screen resolution in y
ns <int>                    // number of samples (rays) per pixel
camera <float><float><float>  // camera position
center <float><float><float>  // camera is pointing to this location
viewup<float><float><float> // the up direction

// a sphere is described with a center, radius,
// and material
sphere center <float><float><float> radius <float>
        material [lambertian|metal|dielectric]
// for lambertian
        albedo  <float><float><float>
// -- or –
        checker_texture <float><float><float> <float><float><float>

// for metal
        albedo <float><float><float> fuzz <float>

// for dielectric ior (index of refraction)
        ior <float>
```

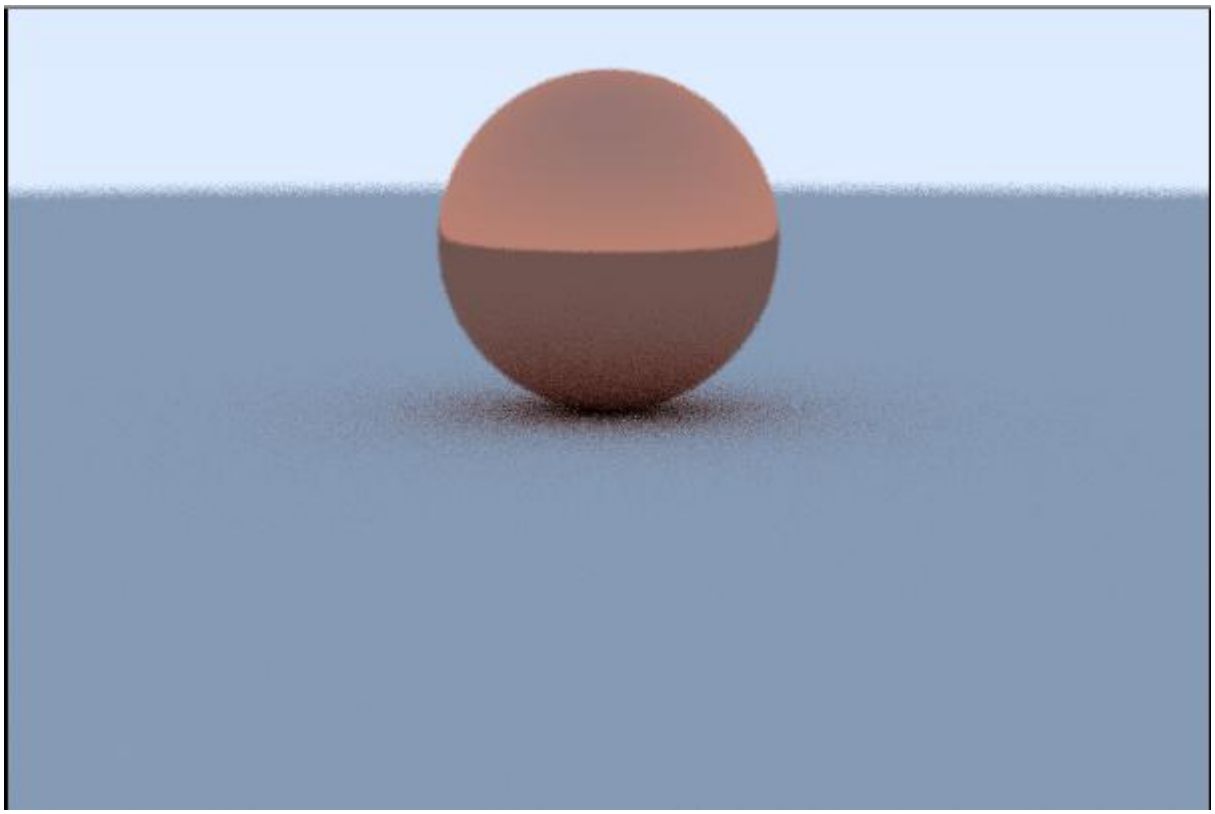
Notes:

1. albedo, formally in thermodynamics is a non-dimensional, unitless quantity the indicates how well a surface reflects solar energy. 0 indicates black or a perfect absorber and 1 indicates white or a perfect reflector. In astronomy it is a measure of the amount of light reflected from the surface of a celestial object, such as a planet, comet or asteroid. In this application we use a vec3 to attenuate the color (rgb) returned by a hitting an object. Look at the color function in the main.cc source file. So to create a red sphere, set the albedo to (1.f,0.f,0.f)
2. Fuzz a number from 0 to 1 to provide a “fuzziness” control
3. ior is the index of refraction. You can find indices of refraction with a Google search. In the sample program, 1.5 is used for a glass sphere.

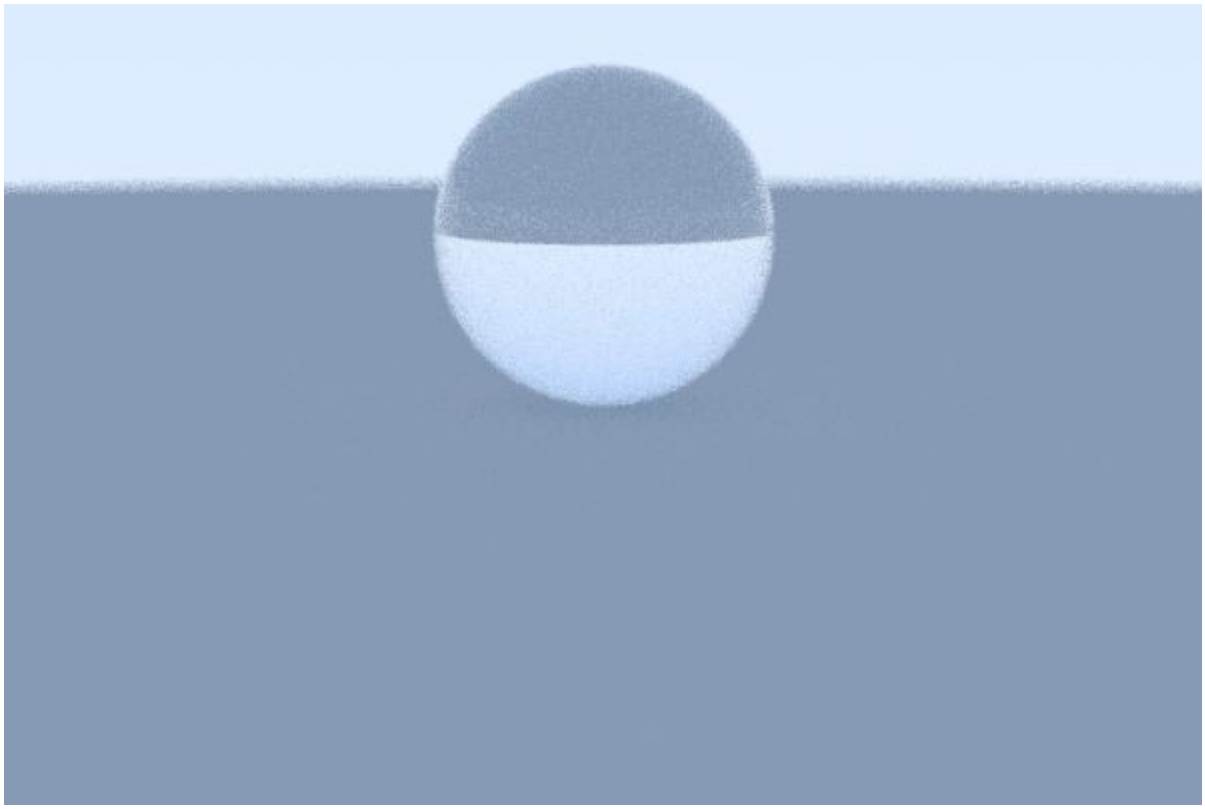
Sample output



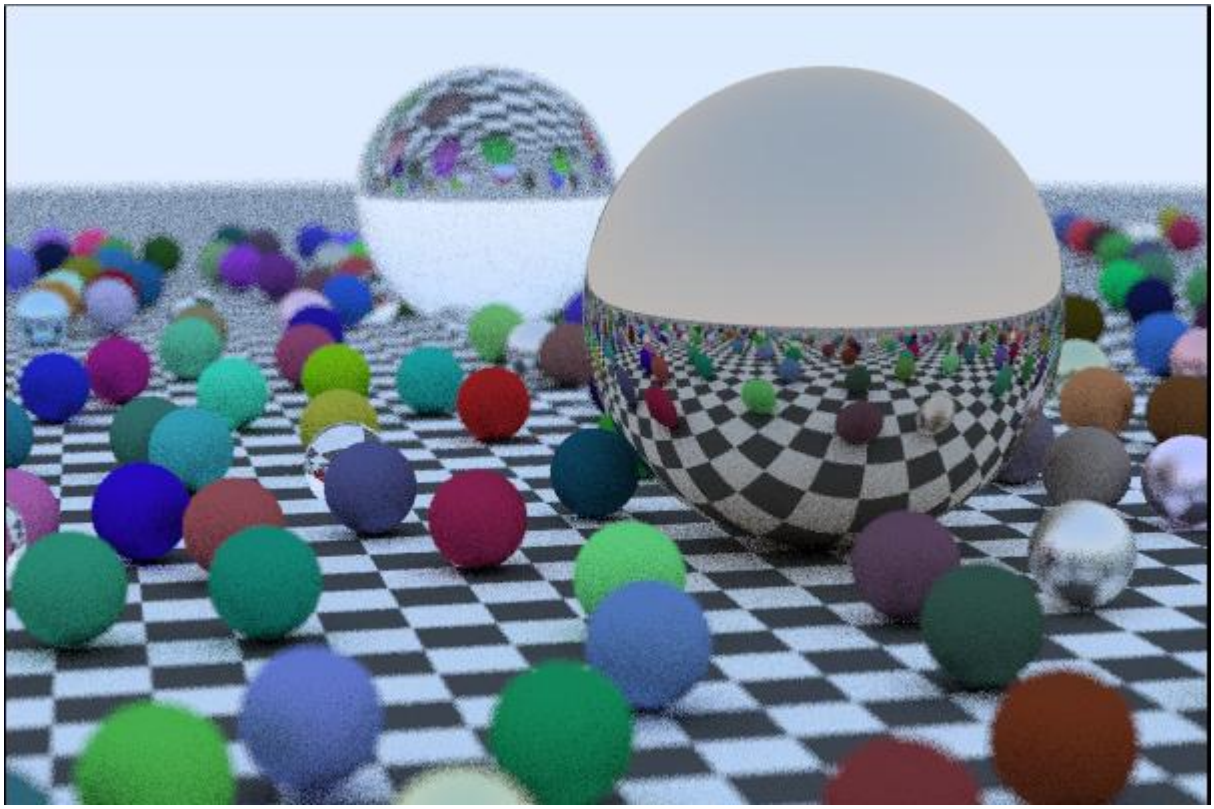
```
nx 600
ny 400
ns 15

camera 13 2 3
center 0 0 0
viewup 0 1 0

sphere center 0. -1000. 0. radius 1000
material lambertian albedo .5 .5 .5

sphere center 0  1  0 radius 1 material
lambertian albedo .7 .3 .2
```



```
nx 600
ny 400
ns 15

camera 13 2 3
center 0 0 0
viewup 0 1 0

sphere center 0. -1000. 0. radius 1000
material lambertian albedo .5 .5 .5

sphere center 0  1  0 radius 1 material
metal albedo .7 .3 .2 fuzz 0.
```



```
nx 600
ny 400
ns 15

camera 13 2 3
center 0 0 0
viewup 0 1 0

sphere center 0. -1000. 0. radius
1000 material lambertian albedo .5 .5
.5

sphere center 0  1  0 radius 1
material dielectric ior 1.5
```



```
nx 600
ny 400
ns 20

camera 13 2 3
center 0 0 0
viewup 0 1 0

sphere center 0. -1000. 0. radius 1000. material
lambertian checker_texture .1 .1 .1 .9 .9 .9

sphere center -10.2655 0.2 -10.9457 radius 0.2
material lambertian albedo 0.575633 0.630066
0.258336
sphere center -10.9356 0.2 -9.14393 radius 0.2
material lambertian albedo 0.45218 0.439454
0.0530051
sphere center -10.9437 0.2 -8.83851 radius 0.2
material dielectric ior 1.5
sphere center -10.5466 0.2 -7.72256 radius 0.2
material lambertian albedo 0.109579 0.162936
0.457401
sphere center -10.9516 0.2 -6.5648 radius 0.2
material metal albedo 0.828763 0.993489
0.989399fuzz 0.430978
sphere center -10.117 0.2 -5.8423 radius 0.2
material lambertian albedo 0.309237 0.363655
0.0693067
…
```

Submission Instructions

Submit your files from the host **lectura.cs.arizona.edu** using the command
turnin cs433s17-assg4 [files]

Assignment Advice

1. Start early.
2. Do your own work.
3. Check piazza regularly.