

1 Goal

Implement ray-tracing. Your programs should be able to render triangles and spheres lit by a single light source (no attenuation) and viewed from any point and with any screen location. All the necessary data will be specified by the input file.

2 Input file format

The input file will specify:

1. The desired resolution of the output image
2. The coordinates of the viewpoint e (three floats)
3. The screen data: l (lower left corner), \vec{h} and \vec{v} (vectors running along the horizontal and vertical edges)
4. The light source (b , the coordinates and I , the intensity)
5. The ambient light intensity
6. List of primitives (spheres and triangles) to be drawn

The list of primitives will be preceded by an integer equal to their count. A description of a primitive will start with a letter, S for a sphere and T for a triangle, followed by end-of-line. For a sphere, we shall list the coordinates of the center and the radius (4 floats) in one line and the material description (8 floats): k_{dr} , k_{dg} , k_{db} , k_{ar} , k_{ag} , k_{ab} , k_s and n_{spec} (the specular coefficient), also in one line. These are all the coefficients of our local reflection model (see end of Section 4 or the ray tracing notes. For a triangle, we will list the coordinates of each of the three vertices (one per line). Then, the material properties in the same form as for spheres, will follow.

Thus, starting portion of the file will look like this:

```

m n
e_x e_y e_z
l_x l_y l_z
h_x h_y h_z
v_x v_y v_z
b_x b_y b_z I
I_a
N

```

Then, there will follow the description of the primitives. For a triangle, we will have:

T
 a_{1x} a_{1y} a_{1z}
 a_{2x} a_{2y} a_{2z}
 a_{3x} a_{3y} a_{3z}
 k_{dr} k_{dg} k_{db} k_{ar} k_{ag} k_{ab} k_s n_{spec}

And for a sphere:

S
 c_x c_y c_z r
 k_{dr} k_{dg} k_{db} k_{ar} k_{ag} k_{ab} k_s n_{spec}

3 Grading

100 points maximum. Things we will grade for:

1. Visibility (do correct object show up at the right pixels): 30
2. Shadows work: 30
3. Illumination OK: 30
4. Efficiency: 5
5. Clarity of the code: 5

4 Late policy

Late penalty: $3^{\#dayslate-1}$ points.

5 Test data

We will post some next week.

6 Output

You will have to produce an image in the form of an ascii ppm file. PPM files can be viewed under windows using e.g. the freeware program XnView. In other systems, viewing them is even easier. You can find information about the ppm file format at

<http://www.dcs.ed.ac.uk/home/mxr/gfx/2d/PPM.txt>. The resolution of the output image is specified by the input file (see 'input file format' section).

Start by allocating and filling $m \times n$ array with the RGB values (floats). Then, scale the values to 0...255, round them and use 255 as the maximum color-component value.

7 Turning in your code

Send a tar file including a makefile and your source code to `cs4451a@cc.gatech.edu`. It has to compile using plain 'make' from the command line. The executable's name has to be 'project1' and it should read the input file from the standard input and output the text ppm file to standard output. Under UNIX/LINUX/MacOSX, the command needed to run your program from terminal should be:

```
% project1 < input_file > output_image.ppm
```

8 Final things

Programs have to be written in C or C++.

This is an individual project. You are free to discuss high level implementation ideas but the code has to be your own.

We will try to use the new MAC lab (CCB 130) for testing your programs. The lab is not ready for use yet, but it should be very soon. This means that you can program anywhere, but before you turn your program in you should make sure that it compiles and runs on the MACs in the lab. To avoid problems, use standard C or C++.