1 Goal

Implement ray-tracing. Your programs should be able to render triangles and spheres lit by a single white 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 (two integers, m and n if the output is to be $m \times n$)

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 exponent), also in one line. The first three are the diffuse reflection coefficients for red green and blue components, then come the ambient reflection coefficients and the specular coefficient and exponent. Note that we won't have separate RGB components for the specular coefficient: we'll just use the same value for all of them (i.e. set $k_{sr} = k_{sg} = k_{sb} = k_s$).

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:

$m\ n$
$e_x\ e_y\ e_z$
$l_x\ l_y\ l_z$
$\vec{h}_x\ \vec{h}_y\ \vec{h}_z$
$\vec{v}_x\ \vec{v}_y\ \vec{v}_z$
$b_x\ b_y\ b_z\ I$
I_a
N

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

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

And for a sphere:

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

3 Grading

100 points maximum. Things we will grade for:

1. Visibility (do correct object show up at the right pixels): 40

2. Shadows work: 20

3. Illumination OK: 20

4. Behavior according to the rules laid out here (correct interpretation of the input file, reading from standard input/writing to standard output and such): 10

5. Efficiency: 10

4 Late policy

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

5 Sample input files

A few are posted, more will appear next week.

6 Output

You will have to produce an image in the form of a binary ppm file. PPM files can be viewed under linux using GIMP (this is what we'll be using for grading). Of course, there are other possibilities, e.g. XnView, also available for windows. You can find information about the ppm file format in the man pages (`man ppm`). The resolution of the output image is specified by the input file (see 'input file format' section).

The input files will be designed so that the intensity of a pixel should never exceed 1 (except possibly for a tiny numerical error). Please use 255 for the maximum color component value of the PPM file. For each pixel, run ray tracing to get RGB intensities. Then, multiply them by 255 and round to the nearest integer to get the intensity to be written to the output (the provided code is already doing this).

7 Turning in your code

Send a tar file (possibly compressed with either bzip2 or gzip) including a makefile and your source code to the class email that will be posted in a day or two. Make sure that your tarfile behaves like the skeleton code I posted, in particular:

- your tarfile extracts to a subdirectory called 'proj1'
- it compiles by typing 'make' in that subdirectory
- the resulting executable's name is 'proj1'
- your program reads input from standard input and writes output to the standard output

To test your code, we'll be running it like this from command line:
`% proj1 < input_file > output_image.ppm`
Then, we'll examine the output under gimp.

8 Final things

Programs have to be written in C or C++. They *have* to compile and run on the rh9 linux machines in the intel lab in order to be graded. Please test them on one of the machines in the lab before turning them in, especially if you choose to develop them using a different platform.

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