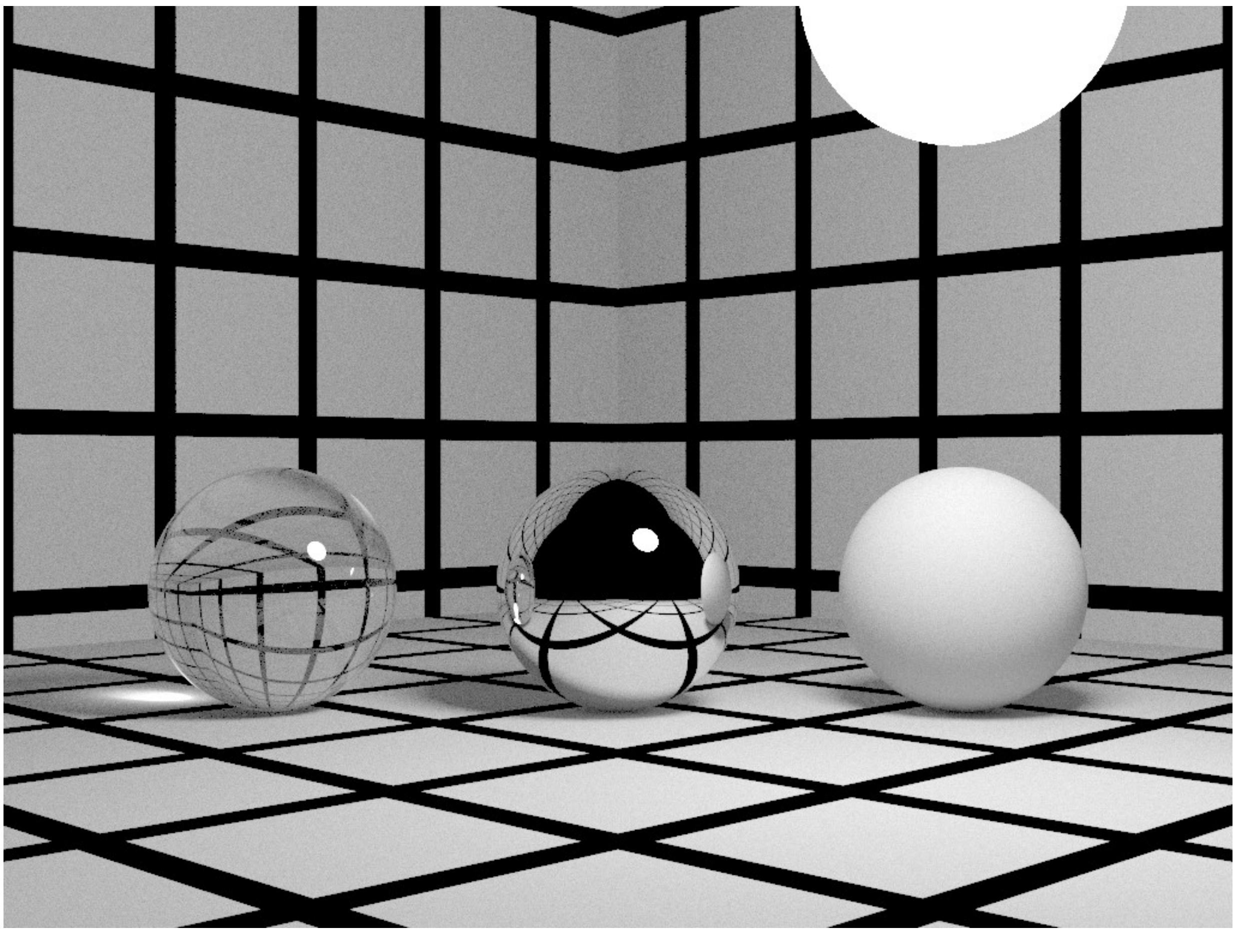
Introduction to Computer Graphics

Assessed Coursework 2

Description

The aim of this coursework is to create a simple ray tracer that is capable of rendering simple mathematical objects. A simple camera model and lighting model will be adopted.
The learning objectives of this small project are:

- Practical application of the ray tracing algorithm
- Better understanding of a simple lighting model
- Enforcing the concepts of ray intersection and normals



Requirements

The following must be implemented. Write a small ray tracer with the following attributes:

- The scene is bounded by 3 black-and-white cross-striped planes as depicted above.
- The scene contains 3 spheres: 1) A semi-transparent sphere; 2) A 100% reflective sphere; and 3) An opaque sphere.
- Smooth Gouraud shading
- Two-point light sources should be supported, placed in any position.
- A full Phong illumination model is to be implemented (ambient, diffuse, and specular).
- Recursive refractions: Implement the refraction component in the material shade method. After computing the reflection component, check whether the refraction color is zero. If it is not, create a *refraction ray*. This ray should start at the intersection point like the reflection ray, but its direction should be computed by *Snell's law* of refraction, using the refractive index "*ior*" of **pure flint glass**. Trace the refraction ray.
- Reflected rays to a maximum depth of 8 reflections should be implemented.
- The output should be to both a window (at rendering time) and an image file (BMP, TIF, GIF, or any format).

Hints:

- Snell's Law for Refraction:
  https://en.wikipedia.org/wiki/Snell's_law

- To compute the refraction ray, you need to compute the ratio of two indices of refraction: The refractive index of air (which is 1), and the refractive index of the material (*ior*). The numerator and denominator of the ratio depend on whether the ray hit the surface from the inside or the outside. If the ray hit the surface from the outside, you should use *r = 1/ior;* otherwise, you should use *r = ior/1*.

- You can check the sign of the dot product of the ray direction and the surface normal to determine whether the ray intersected the surface from the inside or the outside

- Don't forget to normalize your vectors before using them in the refraction formula

Submission

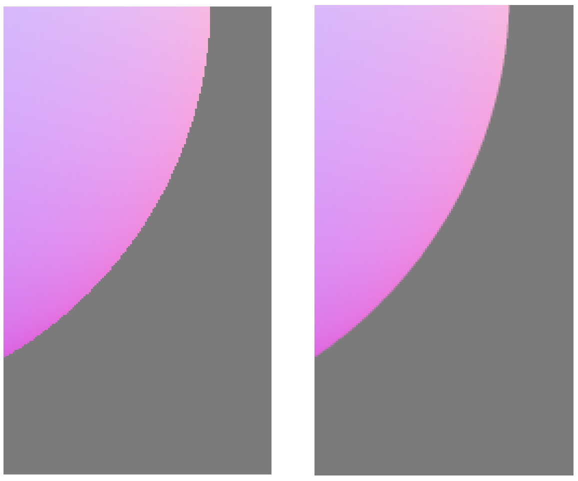
You will need to electronically submit a couple of renderings showing the output of your ray tracer rendering, the source code, executable version, and one-page stating team members and describing program features. Deadline for submission is ***Sunday 11 December 2016***. After submission, each team will have to demonstrate the program and explain implementation details.

Bonus Extra Features

To gain bonus marks, you may add one or more of the following features:

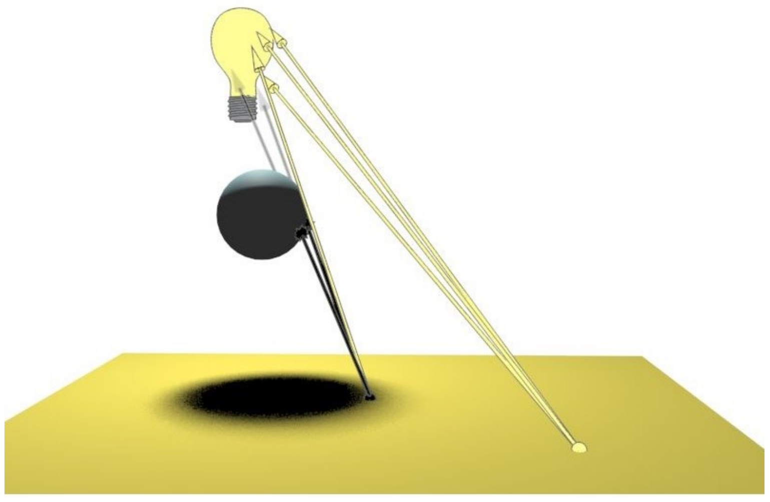
- **Anti-Aliasing using Oversampling**

Add some simple anti-aliasing to your ray tracer. You will use supersampling and filtering to alleviate jaggies and Moire patterns using the following steps



1. Jittered Sampling.
   For each pixel, instead of getting a colour with one ray, we can sample multiple rays perturbed randomly. You are required to subdivide a pixel into 3 × 3 sub-grids. This is equivalent to rendering an image with 3x resolution. Then, for each pixel at (i, j) in the high resolution image, instead of generating a ray just using (i, j), generate two random numbers ri, rj in range $[-0.5, 0.5]$ to get $(i + ri, j + rj)$.
2. Gaussian blur
   http://en.wikipedia.org/wiki/Gaussian_blur
   Use the kernel $K = (0.1201, 0.2339, 0.2931, 0.2339, 0.1201)$.
   First blur the image horizontally and then blur the blurred image vertically. To blur an image horizontally Each pixel colour I' (i, j) in the new image is computed as a weighed sum of pixels in the original image.
   $I'(i, j) = I(i, j - 2)K(0) + I(i, j - 1)K(1) + I(i, j)K(2) + I(i, j + 1)K(3) + I'(i, j + 2)K(4).$
3. Down-sampling.
   To get from the high-resolution image back to the original specified resolution, average each 3×3 neighbourhood of pixels to get back 1 pixel.

- **Shadow Casting**



Implement cast shadows by sending rays toward each light source to test whether the line segment joining the intersection point and the light source intersects an object. If there is an intersection, then discard the contribution of that light source. Recall that you must displace the ray origin slightly away from the surface.

Note that in this naive version, semi-transparent objects still cast opaque shadows. Implement something better for extra credit.