Department of Computer Science, University of Otago **Paper 342 - Graphics 2011**

Ray Tracing

Assessable assignment 2 **Weighting: 20%**

DUE DATE: Friday, 13th May 2011 at 5pm
NO LATE ASSIGNMENTS ACCEPTED

For this assignment you are asked to write a basic ray tracer that can draw 'spheres' and 'lenses'.
A sphere will be specified by the location of its centre, and its radius (these are all in 'world space').
A lens is the intersection of two spheres in world space.
The scenes you will raytrace will be made up of these objects. It is possible to be 'inside' an object.

The camera is at a fixed position on the z-axis looking through the origin. The camera can be rotated about the z-axis.

1. Starting with the skeleton program that we provide, write and test routines that ray trace and shade the objects in the scene. The skeleton program defines the required user interface, reads scene information from a file (format described overleaf), and sets up many useful data structures.
2. Test your program with our sample file and with scenes of your own design.
3. Write a *short* report, describing how your program works and how you tested it. Describe any known flaws in your program.

The ray tracer skeleton is in the directory /coursework/342/pickup/ass2/, along with a sample test file. Submit your work by using the **submit342** script. Make sure your submission is in a directory with your name on it, and that it contains your source code, makefile, report and at least one of your own test scenes which we will use in an art display for the course. Also make sure that your name is included in the report and in the introductory comments in your program. We will e-mail you to confirm that we have received your assignment within a week of the due date. Keep a copy of your work at least until the assessment has been returned to you.

The report must be submitted as a plain text file (**not** OpenOffice or MSword) and *not more than two pages long.* We do not want any assignments handed in on paper!

The skeleton code and test file will be available 5th April.

The marking scheme for this assignment will be added later, but will include:

- Basic ray intersection showing colours and geometry of objects with ambient light.
- Diffuse shading.
- Phong shading.
- Shadows.
- Mirror reflections.
- Viewing from inside objects.
- Rotating the camera.
- Report and design of scene file.

Scene description file format:

Important details not included in the scene file:

We are looking from the point (0, 0, -2) along the z-axis to the origin.

Our default view plane is a square that extends from (top-left) (–1.0, –1.0, 0) to (bottom-right) (1.0, 1.0, 0).

-- the viewing plane can be rotated around the z-axis (if specified in scene file).

We have a right handed coordinate system.

Use the object's diffuse coefficients when calculating ambient lighting effects.

Note: The viewing point could be inside any of the objects. We will test this.

First lines:

Any of the following settings may be specified.

If a setting is not included in the scene description file, then the associated default value(s) will be used.

```
imagesize      n
```

The text "imagesize" followed by an integer, n, which means the final picture is n by n pixels.

You should fire exactly one ray through the centre of each pixel.

The default image size is 320

```
background     br  bg  bb
```

The text "background" followed by a set of parameters.

br, bg, bb gives the red, green and blue components of the background colour.

These components will have real values in the range 0..1 .

The default background colour is 0.0 0.0 0.0

```
ambient        ar  ag  ab
```

The text "ambient" followed by a set of parameters.

ar, ag, ab gives the red, green and blue components of the ambient light.

These components will have real values in the range 0..1 .

The default ambient light is 0.0 0.0 0.0

```
angle     a
```

The text "angle" followed by a float, a, which means that the viewing plane has been rotated about the z-axis (X->Y rotation) by angle a (in degrees).

The default angle is 0.0

Successive lines:

```
light  x  y  z  r  g  b
```

The text “light” followed by real numbers:

x, y, z	gives the location of a point light source.
r, g, b	gives the red, green, and blue intensity of the light in the range 0..1.

The intensity of the light source does not diminish with distance.

We will test your program with **eight** or fewer point light sources.

.... OR ....

```
sphere cx cy cz  r
```

The text “sphere” followed by real numbers:

cx cy cz	the x, y, z coordinates for the centre of the sphere
r	the radius of the sphere.

and immediately following will be the line:

```
material dr dg db mr mg mb pr pg pb p
```

The text “material” followed by real number:

dr dg db	diffuse reflection coefficients for red, green, and blue.
mr mg mb	mirror reflection coefficients for red, green, and blue.
pr pg pb	Phong lighting coefficients for red, green, and blue.

(The diffuse, mirror, and Phong coefficients will have values in the range 0..1).

p	Phong shading exponent.

.... OR ....

```
lens  cx1 cy1 cz1  r1  cx2 cy2 cz2  r2
```

The text “lens” followed by real numbers:

cx1 cy1 cz1	the x, y, z coordinates for the centre of sphere1
r1	the radius of sphere1
cx2 cy2 cz2	the x, y, z coordinates for the centre of sphere2
r2	the radius of sphere2

and immediately following will be the line:

```
material dr dg db mr mg mb pr pg pb p
```

Same description as the sphere material

.... OR ....

```
lensmodel  cx cy cz  nx ny nz   r  h1  h2
```

The text “lensmodel” followed by real numbers:

cx cy cz	the x, y, z coordinates for the 'centre' of the lens
nx ny nz	the x, y, z coordinates for the normal of the 'circle plane'
r	the radius of circle
h1	the 'height'/thickness of one side of the lens
h2	the 'height'/thickness of the other side of the lens

and immediately following will be the line:

```
material dr dg db mr mg mb pr pg pb p
```

Same description as the sphere material

NOTE: your program *does not require anything extra* to get a 'lensmodel' working, it is provided as an alternative way to define a lens (as the intersection of two spheres), without having to explicitly specify the two spheres (see next page).

We will test your program on scenes with **fifty** or fewer objects.

Last line:

```
endview
```

The text “endview”.

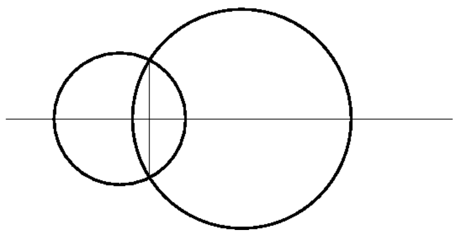
Remember this is an assignment in graphics not in production programming. You will not get any extra credit for extravagant solutions. High marks will be given for a well explained solution that is easy to follow. Programs must be commented of course, but no more than necessary for us to be able to read them.

You may discuss conceptual issues relating to this assignment with others, but all the work that you hand in must be your own, except for the parts of the given skeleton program.

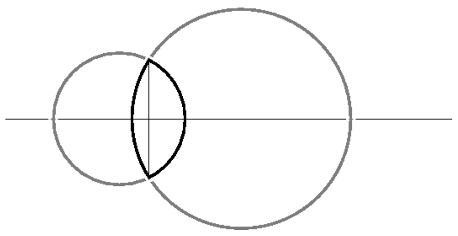
Creating scenes:

Creating lenses where we want them to be, and with the orientation we want, can be very difficult when we use the basic definition (as the intersection of two spheres). (If I want to draw the lens on the right – what spheres should I use?)

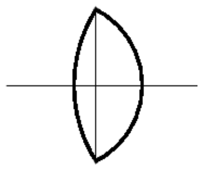
Two intersecting circles



'the lens'



the 'visible lens'

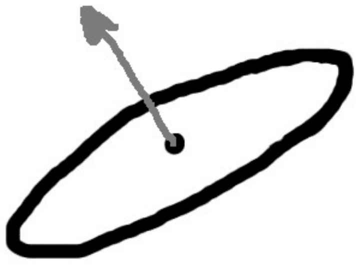


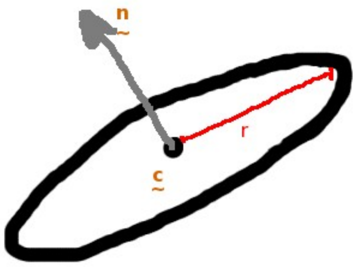
-- a more useful definition is to imagine a 2D circle in space. We can define that by

>>> **c**, the position of the centre of the circle

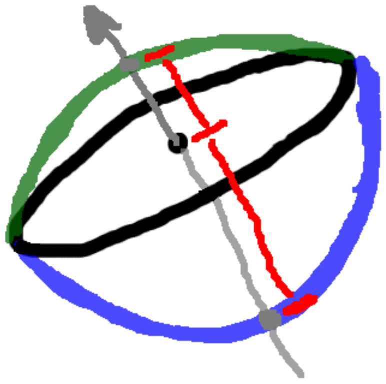
>>> **n**, the surface normal for the plane that the circle lies in

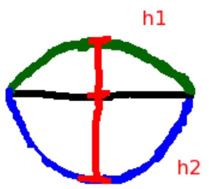
>>> r, the radius of the circle





We can 'inflate' the 2D circle into 3D by giving the distance the surface moves from the centre point along the normal in either direction ('height 1', and 'height 2').





Code has been added to the file reading, which will read a special “lensmodel”, and use this information (**c**, **n**, r, h1, h2) to construct the two spheres (**c1**, r1, **c2**, r2) which generate that lens.

So once read from file, the 'lensmodel' object is just an ordinary 'lens'.