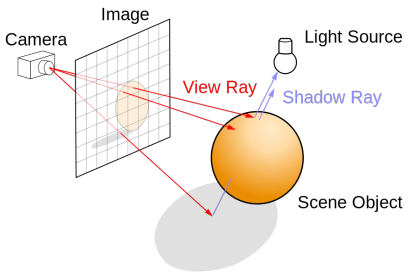
Project

Date of announcement: 30^{th} May 2018
Submission deadline: 18^{th} June 2018

Description

OpenGL's graphics pipeline is tailored for developing real-time interactive graphics applications hence computationally intensive calculations are often approximated or ignored altogether e.g. shadows, reflections, etc. One of the most popular techniques for simulating complex effects to produce realistic images is ray tracing. Ray tracing is an offline technique for rendering realistic images by tracing the path of light rays passing through each pixel and their interactions with the objects in the scene. In this homework you will be learning about ray-tracing by implementing a simple ray-tracer of our own.

You are required to implement a simple ray-tracer for rendering scenes containing multiple lights and simple objects.



Implementation Specifications - Grading Criteria

Develop an application in C/C++ with the following functionality and features:

- loads scene file (1)
- uniformly sends out rays from each pixel in the image plane (2)
- handles intersections (3)
- computes illumination (4)
- renders the scene (5)

1 Scene file - Description format

The scene file describes the contents of the scene. The scene can include the following objects: one camera, one plane (i.e. the "ground"), any number of lights, and any number of spheres. The first line in the file is a number indicating the total number of objects in the scene. Related information about each object is then specified from line 2 on-wards as follows:

1. for specifying the camera you need the following two lines:
   1. *camera*
   2. *pos: x y z //where x y z are floating point numbers specifying the location of the camera*
   3. *fov: theta //where theta is the field-of-view in degrees)*
   4. *f: focal_length //where d is the focal length of the camera*
   5. *a: aspect_ratio //where a is the aspect ratio of the camera*

2. for specifying the plane you need the following seven lines:
   1. *plane*
   2. *nor: nx ny nz //where (nx, ny, nz) is the normal*
   3. *pos: px py pz //where (px, py, pz) is the position of a point on the plane*
   4. *amb: ax ay az //where (ax, ay, az) is the ambient color of the plane*
   5. *dif: dx dy dz //where (dx, dy, dx) is the diffuse color of the plane*
   6. *spe: sx sy sz //where (sx, sy, sz) is the specular color of the plane*
   7. *shi: s //where s is the specular shininess factor*

3. for specifying a sphere you need the following seven lines:
   1. *sphere*
   2. *pos: px py pz //where (px, py, pz) is the position of the center of the sphere*
   3. *rad: r //where r is the radius of the sphere*
   4. *amb: ax ay az //where (ax, ay, az) is the ambient color of the sphere*
   5. *dif: dx dy dz //where (dx, dy, dx) is the diffuse color of the sphere*
   6. *spe: sx sy sz //where (sx, sy, sz) is the specular color of the sphere*
   7. *shi: s //where s is the specular shininess factor*

4. for specifying a light you need the following three lines:
   1. *light*
   2. *pos: px py pz //where (px, py, pz) is the position of the light*
   3. *col: cx cy cz //where (cx, cy, cz) is the color of the light*

Note: There is a space after the ":" and between every pair of values.

Sample file:

```
4
camera
pos: 0 0 0
fov: 60
f: 1000
a: 1.33
sphere
pos: 0 0 -50
rad: 10
amb: 0.5 0.2 0.7
dif: 0.2 0.4 0.2
spe: 0.1 0.7 0.2
shi: 0.1
sphere
pos: 0 30 -50
rad: 3
amb: 0.1 0.5 0.5
dif: 0.4 0.6 0.2
spe: 0.2 0.5 0.5
shi: 1
light
pos: 0 60 -50
col: 0.9 0.9 0.9
```

2 Raytrace

For each pixel p in the image form a ray starting from the center of projection (COP) of the camera with a direction passing through the pixel p. To do this you are required to compute the 3D point in world space corresponding to each pixel in the image space.

3 Intersections

The rays are send through each pixel into the scene. Compute the intersections (if any) with the objects in the scene. Since you have two types of geometric objects i.e. plane and sphere, you need to handle each case differently as discussed in the lecture. If there is an intersection between a ray and an object then a shadow ray should be send out to each light in the scene to determine whether the point of intersection is in shadow or not.

4 Illumination

If the point of intersection is in shadow [for a particular light] then its contribution from that light should be (0,0,0) i.e. black. If the point of intersection is not in shadow [for a particular light] then the contribution of that light should be computed using the Phong illumination model as discussed in the lectures given by:

$$pixel\_color = light\_color * (k_{dif} * (\vec{l}.\vec{n}_{inter}) + k_{spe} * (\vec{r}.\vec{v})^{\alpha}$$

In order to compute the *pixel_color* you will need to compute the normal at the point of intersection $\vec{n}_{inter}$, and the reflection vector $\vec{r}$ around that normal.

The final color of the pixel corresponding to the point of intersection is the sum of the contributions from all lights, plus the global ambient color. For example, if a point of intersection p is in shadow for all lights in the scene then the final color of the pixel corresponding to point p will be just the ambient color. If the value of the final color [per channel] is greater than 1.0, you should clamp it to 1.0.

Note: if $\vec{l}.\vec{n} < 0$, you should clamp $\vec{l}.\vec{n}$ to zero; similarly for $\vec{r}.\vec{v}$

5 Render

Compute the color at all pixels in the image and save out the image. You should use Cimg for saving out the image and displaying it on screen. For example,

```
#include "Cimg.h"
//Creates an image with three channels and sets it to black
cimg_library::CImg<float> image(width, height,1,3,0);
//Compute the color of each pixel and assign it
...
//Save out the image in BMP format. Pixel values must be in the range [0,255]
image.save("render.bmp");
//Display the rendered image on screen
cimg_library::CImgDisplay main_disp(image,"Render");
while (!main_disp.is_closed()) { main_disp.wait(); }
```

Sample files

You can download sample scene files and their expected results here.

Extra credit

1. Recursive reflection [20 pts]
2. Good anti-aliasing [10 pts]
3. Soft shadows [10 pts]

Submission (electronic submission through Moodle only)

Please create a zip file containing your C/C++ code, a readme text file (.txt). In the readme file document the features and functionality of the application, describe any extra credit work (if applicable), and anything else you want the grader to know.

Evaluation Procedure

You MUST demonstrate your solution program to the lab instructor during lab hours. You will be asked to download and run your submitted code, demonstrate its full functionality and answer questions about the programming aspects of your solution. Major marking is done on the spot during demonstration. Your code will be further checked for structure, non-plagiarism, etc. However, ONLY demonstrated submissions will receive marks. Other submissions will not be marked.