PH3170 (C++ Programming)
Project 1 : Ray tracing program

December 3, 2014

1 Introduction

This project is designed to make you develop a relatively large optical simulation program. These types of algorithms are very wide spread in physics, movies, games etc. This involves the simulation of a scene using optics principles to produce a realistic result. An example of the output of a simple program is shown in Figure 1. The types of algorithms used in here are just as applicable for simulation of a multiple element lens system as to Toy Story 3. The main difference between a *physics* and *artistic* simulation is that the physics cannot have any artistic license.

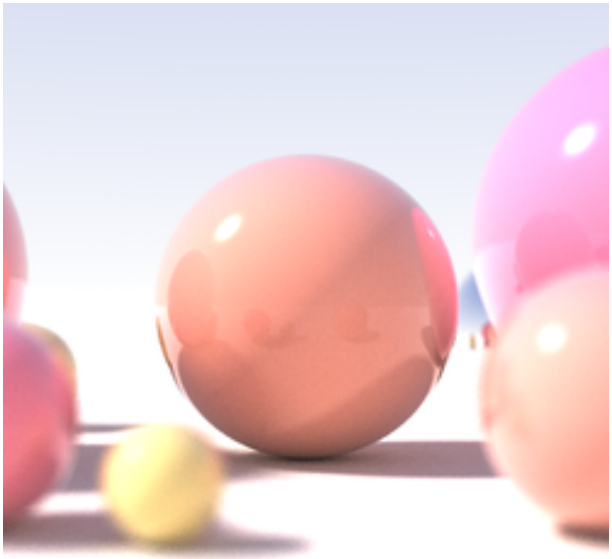


Figure 1: Example of a ray traced image

1.1 Basic optics

This project is the realistic simulation of the passage of light in a three dimensional geometry. The physics of the passage of light is well understood in terms of the laws of reflection and refraction. In one dimension the law of reflection is simply the incident angle θ_i is the same as the reflected angle θ_r

$$\theta_i = \theta_r \tag{1}$$

In three dimensions and with vector notation

$$\mathbf{v}_r = \mathbf{v}_i - 2(\mathbf{v}_i \cdot \mathbf{n}_s)\mathbf{n}_s \tag{2}$$

where $\mathbf{v}_r$ is the direction of the reflected ray, $\mathbf{v}_i$ is the direction of the incident ray and $\mathbf{n}_s$ is a surface normal of the reflecting surface. This equation is simple to understand in terms of momentum. The component of momentum normal to the surface is reflected, so changed by $2\mathbf{v}_i \cdot \mathbf{n}_s$.

The refraction of light rays is governed by Snell's equation

$$n_i \sin\theta_i = n_r \sin\theta_r, \tag{3}$$

where n_i and n_r is the refractive index in the incident and refracted medium respectively. Again this can be generalised into three dimensions and in vector notation, so

$$\mathbf{v}_r = \frac{n_1}{n_2}\mathbf{v}_i + \left(\frac{n_1}{n_2}\hat{\mathbf{v}}_i \cdot \hat{\mathbf{n}_\mathbf{s}} - \sqrt{1 - \frac{n_1}{n_2}\left(1 - (\hat{\mathbf{v}}_i \cdot \hat{\mathbf{n}}_s)^2\right)}\right)\hat{n} \tag{4}$$

where $\mathbf{v}_i$ is the vector representing the incident ray, $\hat{\mathbf{n}_s}$ is a unit surface normal vector and $\mathbf{v}_r$ is the refracted ray.

Finally we need to understand a little about illumination. Imagine the intensity of light reflected I from a surface with normal $\mathbf{n}_s$ at location $\mathbf{p}_r$ is given by

$$I = \hat{\mathbf{v}}_r \cdot \hat{\mathbf{v}}_{rs} \tag{5}$$

where $\mathbf{v_{rs}}$ is the direction between the refection point and the source, so $\mathbf{v}_{rs} = \mathbf{p}_s - \mathbf{p}_r$, where $\mathbf{p}_s$ is the location of the source. It is also possible to add a power to Equation 5 to give a stronger illumination to the primary reflection direction so something like

$$I = (\hat{\mathbf{v}}_r \cdot \hat{\mathbf{v}}_{rs})^\gamma \tag{6}$$

where the γ is a number greater than 1, so this number can be varied and the relative brightness of the specular reflection varies.

1.2 Basic three dimensional geometry

The project will require a good understanding of how to calculate intersections between rays (lines) and geometric objects, so planes, spheres etc. The vector equations for these shapes are briefly reviewed in this section. A vector equation of a line can be written as

$$\mathbf{v}_l = \lambda\hat{\mathbf{v}}_l + \mathbf{v}_{l,0} \tag{7}$$

where $\mathbf{v}_l$ is a point on the line, $\hat{\mathbf{v}}_\mathbf{l}$ is a unit vector in the lines direction and $\mathbf{v}_0$ is any point on the line. A plane is a little more complicated so points on a plane $\mathbf{v}_p$ satisfy the following equation

$$(\mathbf{v}_p - \mathbf{v}_{p,0}) \cdot \hat{\mathbf{v}}_p = 0 \tag{8}$$

where $\mathbf{v}_0$ is a point in the plane, $\hat{\mathbf{v}}_p$ is a normal to the plane. So the vector $\mathbf{v}_p - \mathbf{v}_0$ must lie in the plane and so all these vectors are perpendicular to the plane normal, so the dot product must be zero. A sphere is simply

$$|\mathbf{v}_s - \mathbf{v}_{s,0}| = R \tag{9}$$

where R is the radius of the sphere.

First we need to see if a ray actually does intersect an object. So taking the sphere as an example we can calculate the distance from a line and a point (in this case the centre of the sphere), if this distance is smaller than the radius the ray must intersect the sphere, so

$$\begin{aligned} (\mathbf{v}_l - \mathbf{v}_{s,0})^2 &= (\lambda\hat{\mathbf{v}}_l + \mathbf{v}_{l,0} - \mathbf{v}_{s,0})^2 & (10)\\ &= \lambda^2\hat{\mathbf{v}}_l^2 + 2\lambda(\mathbf{v}_{l,0} - \mathbf{v}_{s,0}) \cdot \hat{\mathbf{v}}_l + (\mathbf{v}_{l,0} - \mathbf{v}_{s,0})^2 & (11)\end{aligned}$$

Taking the derivative of the distance squared with respect to λ and also knowing that the square of unit vector is 1 gives

$$\frac{(\mathbf{v}_l - \mathbf{v}_{s,0})^2}{d\lambda} = 2\lambda + 2(\mathbf{v}_{l,0} - \mathbf{v}_{s,0}) \cdot \hat{\mathbf{v}}_l \tag{12}$$

So to find the minimum distance we have to determine the λ where the derivative is zero $\lambda_{\min}$ so

$$\begin{aligned} \frac{(\mathbf{v}_l - \mathbf{v}_{s,0})^2}{d\lambda} &= 2\lambda + 2(\mathbf{v}_{l,0} - \mathbf{v}_{s,0}) \cdot \hat{\mathbf{v}}_l = 0 & (13)\\ \lambda_{\min} &= (\mathbf{v}_{s,0} - \mathbf{v}_{l,0}) \cdot \hat{\mathbf{v}}_l & (14)\end{aligned}$$

So inserting this $\lambda_{\min}$ into the equation for a line we find

$$\mathbf{v}_{\min} = [(\mathbf{v}_{s,0} - \mathbf{v}_{l,0}) \cdot \hat{\mathbf{v}}_l]\,\hat{\mathbf{v}}_l + \mathbf{v}_{l,0} \tag{15}$$

If the length of the vector $\mathbf{v}_{\min} - \mathbf{v}_{s,0}$ is less than the radius of the sphere then the line must intersect, otherwise it does not.

To calculate the intersection point of a line with an sphere , the line equation must be inserted into the equation for the object. So for example for a sphere-line intersection we have $\mathbf{v}_l = \mathbf{v}_s$, so inserting Equation 7 into Equation 9 gives us

$$|\lambda\hat{\mathbf{v}}_l + \mathbf{v}_{l,0} - \mathbf{v}_{s,0}| = R \tag{16}$$

It is probably easier to deal with the square of this equation so

$$\begin{aligned} (\lambda\hat{\mathbf{v}}_l + \mathbf{v}_{l,0} - \mathbf{v}_{s,0})^2 &= R^2 & (17)\\ \lambda^2\mathbf{v}_{l,0} \cdot \mathbf{v}_{l,0} + 2\lambda\mathbf{v}_{l,0} \cdot (\mathbf{v}_{l,0} - \mathbf{v}_{s,0}) + (\mathbf{v}_{l,0} - \mathbf{v}_{s,0})^2 - R^2 &= 0 & (18)\\ a\lambda^2 + b\lambda + c &= 0 & (19)\end{aligned}$$

where

$$\begin{aligned} a &= \mathbf{v}_{l,0} \cdot \mathbf{v}_{l,0} & (20)\\ b &= 2\mathbf{v}_{l,0} \cdot (\mathbf{v}_{l,0} - \mathbf{v}_{s,0}) & (21)\\ c &= (\mathbf{v}_{l,0} - \mathbf{v}_{s,0})^2 - R^2. & (22)\end{aligned}$$

The problem reduces to the solution of a quadratic in λ which generally has 0, 1 or 2 real solutions, what do these solutions correspond to? Given a solution λ_1 then the point on the sphere can calculated, lets call it $\mathbf{p}_r$ for reflection. To calculate the reflection of a ray at a point we need the surface normal to a sphere, this is then simply

$$\hat{\mathbf{n}}_s = \frac{\mathbf{p}_r - \mathbf{v}_{s,0}}{|\mathbf{p}_r - \mathbf{v}_{s,0}|}. \tag{23}$$

2 Ray tracing

Physically an optical simulation would consist of randomly sending out rays from a light source and reflecting and refracting them until they arrive at an observer. This is known as *forward* ray tracing and is shown in Figure 2. This is computationally very inefficient as the vast majority of rays will not arrive at the observer at all.

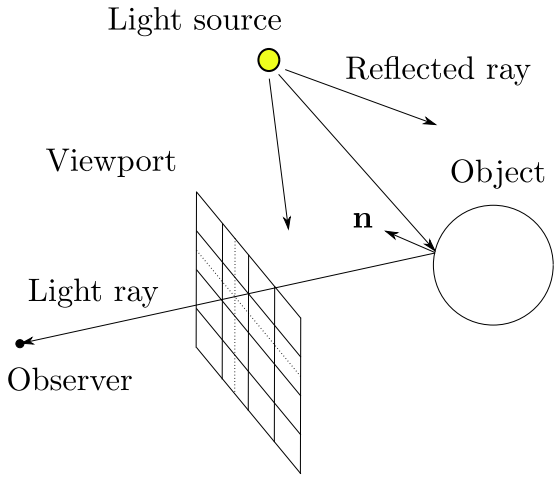


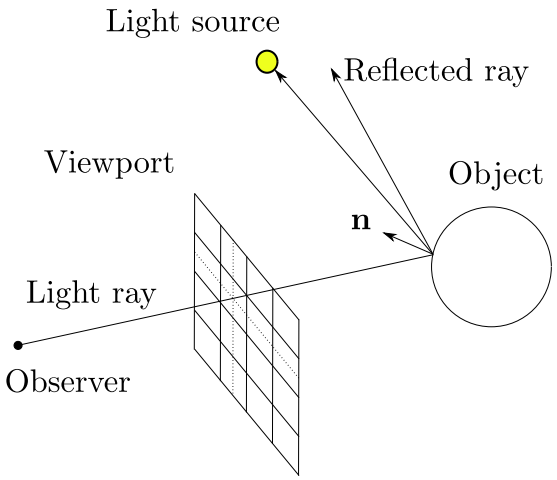
Figure 2: Diagram explaining the forward ray tracing algorithm.

Another method is called *reverse* ray tracing. The main idea of ray tracing is to send out rays from an observer and see where the rays intersect objects and then back to the source of illumination. Of course if objects are reflective then the reflected ray could hit another object. This process has to repeatably applied until we arrive at light source or a diffuse source of light. This process is shown in Figure 2.

We calculate a ray that passes from the observer through the view port into the world. This ray is the propagated back through the world until it intersects with an object. This means that we see if that ray intersects with any object in the world. Then if the object is reflective we create a new ray and repeat this process. If it is refractive we create a refracted ray and repeat the whole ray tracing process.

3 Structures and code flow

Here is a list of possible classes which could be implemented.



- 3D vector class
  - A light ray class
- 3D shape class
  - Sphere class
  - Cuboid class
  - Your own shape
- Viewport class (which transforms between screen coordinates and world coordinates)
- Scene class (possibly using an STL container) which holds the objects in the scene.
- Image input-output

The most important aspect of the code is the interaction between the

An outline of a ray casting algorithm might be the following

1. Define world (create objects, screen, observer, light sources).
2. Loop over all rays from observer to a screen pixel.
   (a) Create ray from observer through pixel in to the world.
   (b) Recursively see if the ray hits an object.
   (c) Reflect and refract the ray depending on the surface of the object.
   (d) Until hitting a diffuse object.
   (e) Calculate the diffuse reflection from the object. So compare the reflected ray with the direction of the light source.
   (f) Assign the pixel a value according to the ray casting.
3. Write image

4 Visualisation and plotting

There are many ways to view your output, here are a few methods which you might wish to choose

- Use and output filestream and plot your results in python,
- Use the easybmp library (http://easybmp.sourceforge.net/).
- Use the TGA file format (http://www.cplusplus.com/forum/general/6194/).

5 Ideas for extension

These are possible ideas for extension to the basic ideas presented. So to

- Add shadows
- Add diffraction,
- Add polarisation,
- Implement the Monte Carlo method from source to observer,
- Implement a gas or particles for diffuse scattering.

6 Assessment

The project will be assessed via a written 6 page report and the working code (to be uploaded to moodle). The report is to explain to a proficient programmer (i.e the course instructors) how your code works. It should highlight the code design and important classes and where applicable the important methods of classes which embody the algorithms. The report should also include a brief description of how to run the program and an example of the output. The code should be clearly commented, classes and methods should be carefully named so that the functionality is clear. The project should use as many of the programming concepts explained in the lectures as relevant. In most cases the project script with all the extensions cannot be completed in the time given. Good marks will be awarded for clear and usable code, which can be extended.