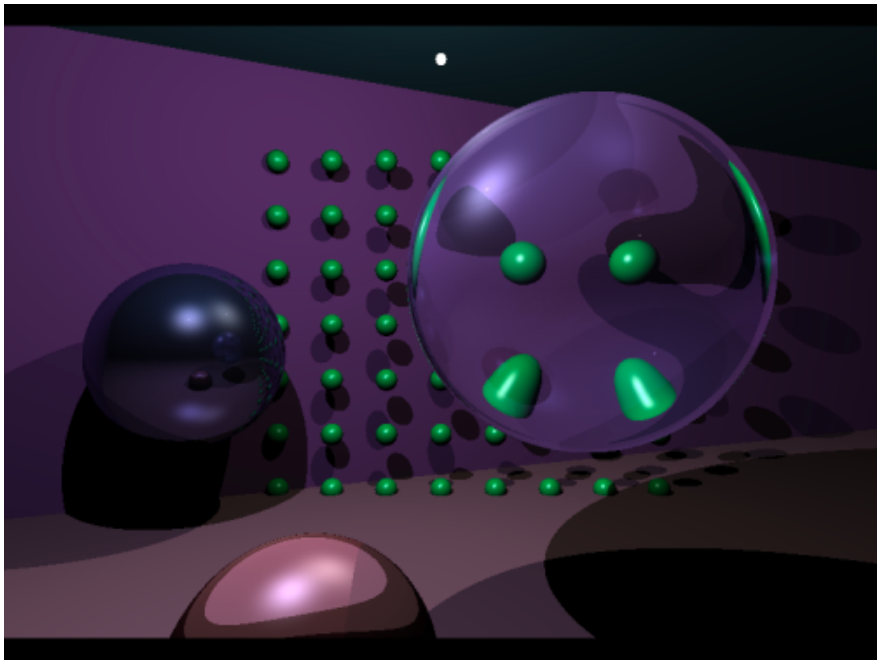
https://www.flipcode.com/archives/Raytracing_Topics_Techniques-Part_3_Refractions_and_Beers_Law.shtml



Assignment 02 Java Best Practices

Issued 20 October 2021
Due 21 November 2021
Assignment: **35 marks**
Type: Team Assignment (code and report)

OVERVIEW

There is a large amount of legacy code that has been written for older versions of the JDK. Legacy systems, implemented using legacy code will still run, but there are advantages to modernising these applications to take advantage of modern libraries and language features. These legacy systems are often monolithic (meaning they have little or no reusable code “snippets”) which can make these systems difficult to understand and modify.

As Java evolved, new language features have been added to enhance the language, some examples are

- JDK 1.5 Annotations, enhanced for loop, generics
- JDK 7 Type inference (declarations), try with resources, catch multiple exception types, Objects class
- JDK 8 Functional interfaces, lambda expressions, Optional class
- JDK 9 Modules, Optional class enhancements
- JDK 10 Local variable type inference
- JDK 11 Type inference for lambda parameters

CS3318

In this assignment you will work on a legacy codebase and identify where it can be modernised, develop a plan to modernise it and rewrite the code as a well-documented modern API.

Deliverables

An updated group repository on `csgate.ucc.ie` and a report submitted via Canvas.

Due Date

21 November 2021

THE MODERNISATION TASK

Introduction

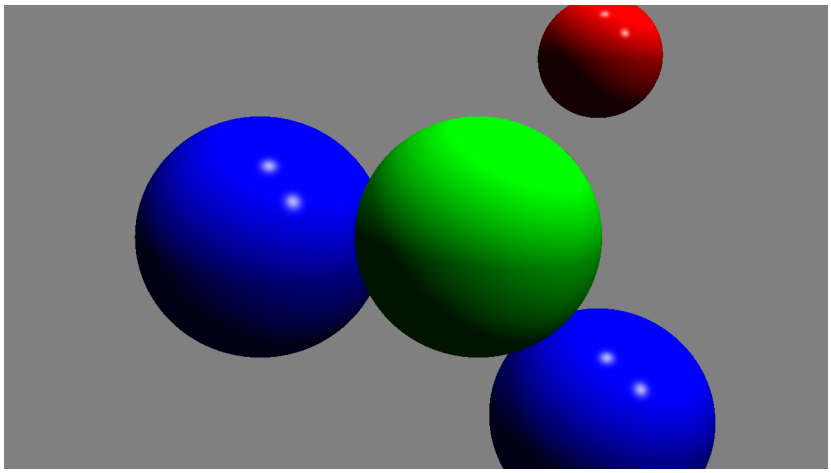
I have located some code from twelve years ago that implements a ray tracing application. The original code implemented a Java command-line tool that read a scene description file and applied a ray tracing algorithm to render the objects described in the scene. The code written by Idris Mokhtarzada is available on GitHub at https://github.com/idris/raytracer.

There has been no attempt to modernise the code – your team has a job on its hands.

As it is currently presented the application reads a scene description file and renders an image in bitmap (bmp) format based on its contents. An example scene description file is

```
0   0   0
0   0  -1
0   1   0
30

3
  0     0    0    0.2   0.2   0.2    1   0   0
 10   100   10    1.0   1.0   1.0    1   0   0
100   100 100    1.0   1.0   1.0    1   0   0

3
solid          1   0   0
solid          0   1   0
solid          0   0   1

2
0.4   0.6   0.0   1     0   0   0
0.4   0.6   0.7   500   0   0   0

4
0 1 sphere    3    3   -15     1
1 0 sphere    1    0   -15     2
2 1 sphere    5   -5   -25     3
2 1 sphere   -5    0   -30     4
```

The example scene description file above produces the following output.



There is scant documentation, but the implementation suggests that the scene description file reads the following elements in order

- View – the 3D position of the eye, centre of the scene, up direction and field of view
- Lights – the 3D position, the colour (RGB), three floating point values
- Pigments – type (solid, checker, gradient, texmap), three floating point values
- Surfaces – finish (ambient, diffuse, specular, shiny, mirror, transparency, index of refraction)
- Shapes – pigment number, finish number, type (cone, cylinder, disc, sphere), 3D position, size (shape dependent)

Your team will have to uncover the interpretation of the scene description that your implementation requires.

ASSIGNMENT REQUIREMENTS

The current implementation is an application that reads an external scene description file and produces a Windows bitmap format image of the rendered scene. In this assignment you team will

- Create a well-defined, documented Java API for ray tracing.
- Create an exemplar application that uses this API to render an image.
- Create a maven build file to create a Java archive (JAR) for the API.

The API presents Java functionality that a client developer uses to write code to programmatically create the elements required to construct a raytracing render job and save the results of the render job in an image format.

An example of using a programmatic API is (note this is not the API for this assignment)

```
objectList.addElement(new Sphere(spherePosition, size));
lightList.addElement(new Light(lightPosition, lightColour));

raytrace.assemble(objectList, lightList);
raytrace.render();
```

You should consult the existing readScene() method, to discover how the current solution constructs the render job from the input. In this assignment

- You do not have to implement support for all shapes. However, your solution must support a sphere and one other shape of your team's choosing.
- You need only support solid pigments.
- When the render job is run the result should be accessible as an image and a client developer should be able to specify the name and format of the output image.

When developing your solution your team must

- Assess the structure of the original application and identify any structural changes.
- Update where reasonable the solution to use modern Java features.
- Use Java coding conventions.
- Create a programmatic API for ray tracing, paying particular attention to the abstractions used.
- Document the API using Javadoc.

You are not required to (but may choose to)

- Re-implement the functionality of the methods.
- Provide a GUI to preview the rendered image.

The assignment requires teamwork and using a shared (for each group) repository. How well the git repository is managed is part of the assessment.

ASSIGNMENT RESOURCES

You have been divided into groups and details of membership are on Canvas. On `csgate.ucc.ie` there is a folder for each group located in `/users/shared/cs3318`. Each folder contains a shared git repository for that group. The groups are numbered from group01 to group27 (note the leading zero). Each member of the group should clone a copy of the shared repository for their group, so for example if I am in group 00, I would open IntelliJ and create a new project from version control using

```
username@csgate.ucc.ie:/users/shared/cs3318/2023/group00/raytracer.git
```

As the location of the git repository. Replace `username` with your lab username and you will be prompted for your lab password. Make sure that each commit to your repository has a single purpose and is documented using a commit comment.

To update the shared git repository on `csgate.ucc.ie` team members must push content from their local repository to the shared repository. You will need to manage how to organise updates to the shared repository to avoid and resolve conflicts.

You should also note that work you intend to do may already be done by a member of your team and if it has been pushed to the shared repository it can be pulled from there.

OUTCOMES

The assignment outcomes are

- Updated project on the group's repository, containing a well documented API for ray tracing.
- An example application that uses the API to render a demonstration scene and store it as a file.
- Well written commit messages.
- A report (maximum 1000 words) describing the issues your team identified with the original code and the choices your team made about the API.