University of Calgary, CPSC453
Assignment 4
Raytracer

Release Date: Monday, November 16, 2015
Due Date: Monday, December 7, 2015 at 10:59 am
Weight of this assignment: 10%
Total Marks: 100 Marks (+ 30 Bonus)

Overall Description

For this assignment, you are to write a recursive ray tracer. For ray tracing, you need to create an image of a scene including several objects and models. You may specify the scene in the code and output the resulting image to a file or to the screen. The type of image file you use is your choice, but the user must be able to specify the desired width and height of the resulting image through the command line or an input file.

You are are allowed to use the C++ standard template library, `algebra.h`, `algebra.cpp` files from Assignment 2 and the provided files. No other libraries files are to be used.

The Interface

No graphical user interface is required for this assignment. The user must be able to specify the image resolution (width and height), for example through command line arguments. All other interactions are designed at your discretion.

Donated Code

You have been provided with the following files:

- `main.cpp` - The main point of entry into the application, including an example of how to populate and save a 2D image.
- `polyroots.cpp, polyroots.h` - Routines for solving quadratic functions. Maybe useful for intersection tests.

To compile and run the provided program, execute:

```
qmake -project QT+=widgets
qmake
make
./a4
```

The provided code illustrates an example of populating and saving a 2D image to a file, `output.png`. As a test, it loops through the pixels and sets the RGB tuple for that pixel. You need to modify this code to colour the pixels based on the algorithm for a recursive ray tracer. A suggested to-do list, in an order that will help you is as follows:

- Generate scene specifications for the camera and image plane. Compute the rays for each pixel.
- Implement sphere intersection.
- Generate scene specifications for a virtual sphere object that should be visible by the camera.
- Colour the scene using a binary colour scheme (eg. red versus blue) where an intersection with the sphere generates one colour, and no intersection generates another.
- Generate scene specifications for a light source. Implement the Phong illumination model.
- Generate shadow rays and shadows.
- Add support for triangles and quads.
- Support reflection via recursive secondary rays.

You may wish to create helper methods, helper classes or use data structures such as queues or stacks, depending on your design. C++ standard template library offers stacks, queues and other collection mechanisms which may be used. Describe your design decisions in your `README`.

Bonus (30 Marks)

This assignment is a decent amount of work. However, if you have time, and the creative inclination, there are a lot of ways this can be made much more interesting. You are encouraged to experiment with the code and implement these sorts of changes, as long as you have already met the assignment's basic objects. The maximum additional marks from bonuses is 30. It is at the discretion of the TA to determine coolness factors in awarding bonus points.

- Read scene and light specifications from an input file. (up to 5 marks)

- Dynamic eye position and viewing direction specification. (up to 5 marks)
- Refraction (recursive). (up to 10 marks)
- Simple anti-aliasing as discussed in class. (up to 5 marks)
- Use a triangle mesh in OBJ format in the scene. (up to 5 marks)
- Animated MD2 (a sequence of images generated by ray-tracing). (up to 5 marks)
- Add other primitives such as cones, cylinders or other implicit surfaces. (5 marks each, up to 15 marks)
- Generate an OpenGL user interface to create and preview the scene for ray tracing. This interface should include controls to position, orient and scale objects within the scene. (10 marks)
- Support texture mapping and/or bump mapping (5 marks each)
- Support CSG operations (intersect, union, diff, etc). (10 marks)
- Efficiency Improvements:
  - Intensity Threshold: If the accumulated mirror reflectance is less than epsilon, cease recursive on the branch. (up to 5 marks)
  - Bounding Boxes: (Extends mesh bonus) Compute a bounding box to check for intersections before performing whole OBJ intersections. (up to 5 marks)
  - Spatial Partitioning: Modify your intersection routine to use a spatial partitioning scheme (eg. BSP trees, uniform spatial subdivision or octrees). (up to 10 marks)
- Provide an alternative rendering style (eg. toon shading). (up to 5 marks, based on coolness factor)

If you make an amazing modification (an actual feature, not an unintended bug...), document it in your `README` and it will be considered and graded at the discretion of the TA for a maximum of 10 marks.

If you make extensive changes, additionally offer a "compatibility mode" by default. You should support at least the user interface required by the assignment. You can activate your extensions either with a special command line argument or a menu item. Document this in your `README` file.

Non-functional Requirements (20 Marks)

Documentation

1. You must provide a **README** file. A sample one has already been provided.
2. Your README file should contain:
   (a) Your name and UCID.
   (b) Short description of algorithms you implemented to complete the program. This includes how you have designed your ray tracer.
   (c) A brief description of the data-structures used to implement the assignment. This includes what classes you chose to use and their purpose.
   (d) A brief description of each of image and what portion of the assignment it illustrates.
3. Due to speed issues, you must provide at least three screenshots of your assignment demonstrating its capabilities. Your submitted images **must** show each of the features you have implemented. You will not receive marks for features that are not demonstrated in the images you submit. Additional screenshots are necessary to demonstrate any implemented bonuses. Screenshots should be named 'firstname.lastname_a3_#.png' where # is $1 - n$ for n screenshot images.

Source Code

1. All your source code must be written in **C/C++** and properly commented. **OpenGL** and **Qt** are not to be used for this assignment unless it is for a bonus. Your source code must compile on the lab machines in MS 239 without any special modifications. Your source code must be clear and well commented.
2. You will lose marks for inefficient code and a slow demo sample.
3. You may reuse source code:
   (a) which has been provided by the instructor for use in the course,
   (b) which has been written by you which implements basic data structures, such as linked lists or arrays,
   (c) which you have received permission from the instructor or one of the TAs of CPSC 453 prior to handing-in your assignment,
4. Any instances of code reuse by you for this assignment must be explicitly mentioned within the README file. Failure to do so will result in a zero in the assignment. Please read the University of Calgary regulations regarding plagiarism `http://www.ucalgary.ca/honesty/plagiarism`.

Functional Requirements (80 Marks Total)

1. Objects in the scene: spheres (15 marks), triangles and quads (15 marks)
2. Phong illumination model, and point light sources (20 marks)
3. Shadows (15 marks)
4. Reflection (recursive) (15 marks)

Lost Marks

Possible areas for deductions:

- Unable to enter image resolution (width, height).
- Demo scene takes too long.
- You need to give us comparison images for things like antialiasing.
- If your eye point is too far, or your fov is too wide, the objects near the edges will appear distorted (stretched).
- For scaling to work properly, you need to normalize the ray every time you transform it.
- The images shouldn't be upside down.
- The background shouldn't obscure the view especially if using a unique background.
- If images look washed out and shadow areas are gray, it is probably because you are adding the ambient light to your colour instead of ambient*material.ka.
- Missing screenshots, README.

Demo

You are required to give an approximately 5 minute live demo to your TA. For this demo, you'll need to construct a simple example which will compile and raytrace your scene within 30 seconds. Failure to show up at the presentation will result in a zero in the assignment. You will need to schedule your demo with your TA - they will have details about how your tutorial section demos will run.