**CSCE 4813 – Programming Project 6**
**Due Date – 05/01/2019 at 11:59pm**

## 1. Problem Statement:

As you know, ray tracing revolutionized computer graphics because it is able to create realistic looking images with reflection, refraction, and shadows all within one framework. The goal of this programming project is to create a program that uses ray tracing to create and display images of multiple spheres. To get you all started, you will be given implementations of the following six classes:

- ColorRGB – Stores RGB values between [0..255] and supports addition, subtraction and multiplication of color values.
- Point3D – Stores the (px,py,pz) coordinates of 3D points.
- Vector3D – Stores the (vx,vy,vz) components of 3D vectors. This class has methods to normalize vectors and calculate dot products.
- Ray3D – Defines a 3D ray (line equation) starting at point p, and going in direction d. This class has a method to get point samples along the ray.
- Sphere3D – Defines a 3D sphere centered at location (cx,cy,cz) with radius r. This class has a method to intersect a ray with a sphere.
- Phong – Performs Phong shading calculation for a given point in the scene given the camera position, light source information, object properties.

You will also be given a sample program "ray_trace.cpp" that calls methods from these six classes to perform ray tracing and display one large sphere on the screen. This program makes the following assumptions:

- The camera will be located at (0,0,-distance).
- There will be one light source with a fixed direction and color.
- The image projection plane will be the Z=0 plane centered on (0,0,0).
- The image will be a fixed size (XDIM,YDIM) defined at compile time.
- The scene will consist of spheres with positive z positions.
- The spheres will have different colors and material properties.

The goal of this programming project is to extend "ray_trace.cpp" to display multiple spheres with different colors and then improve the ray tracing technique to add more realism. To do this, you must complete the following tasks:

### Task 1: Extend the program to display multiple spheres.

Your first task is to create and initialize a data structure that contains N spheres with random positions, sizes, and colors. Once you have this data structure, you should modify the ray tracing loop to intersect each sphere with the starting ray to determine which sphere is closest to the image plane, and calculate the shade of that pixel using the Phong shading model that is provided.

**Task 2: Extend the program to include shadows.**
As you know, the Phong model uses the surface normal and the direction of the light source to calculate the shade of a pixel. To improve the realism of your ray tracing program, you should check to see if there are any spheres "in the way" of the light source at this point, and if so, you should "turn off" the diffuse and specular terms in the Phong model and only display the ambient component. This will make some of your spheres look like they are in the shadow of another sphere.

**Task 3: Extend the program to simulate reflection OR refraction (BONUS 10 points for undergraduates, required for graduates).**
In order to simulate reflection (mirrors), you need to calculate the ideal reflection direction using the formula from the Phong shading model. In order to simulate refraction (glass), you need to calculate and the ideal refraction direction using the index of refraction for the transparent material. Once you have this information, you can trace a new ray in this direction, to see what light is reflected/refracted back to the original sphere intersection point. It is beyond the scope of this project to consider the effects of multiple reflections/refractions, so you only have to consider the effect of one reflection/refraction when calculating the shade of the output pixel.

**2. Design:**

Your first design task is to select an appropriate data structure to store sphere information, and the loops necessary to randomly initialize the location, size and color of these spheres. Once you have this working, you can design the code for finding the closest sphere that intersects each ray.

Your second design task is to work out how to implement shadows. To do this, you must define a ray from the sphere intersection point towards the light source, and then intersect this ray with all of the other spheres to see if a sphere is blocking the light source at this location. If so, this point is in shadow, and only the ambient light term should be used to calculate the output color. You can accomplish this by setting the Kd and Ks values to zero before calling the Phong shader.

Your third design task (for optional task 3) is to work out how to find the reflection/refraction direction, trace this ray, and then combine this color information with the color you calculated using Phong shading. One thing to keep in mind is that the Ks term in the Phong model tells you how shiny an object is, so perhaps you can use this to tell you how much light is reflected to the camera.

**3. Implementation:**

This semester we will be using C++ and OpenGL to implement all of our programming projects. The instructions for downloading and installing a Linux VM and installing OpenGL are posted in README file the "Source Code" page of the class website.

You are encouraged to look at sample OpenGL programs to see how the "main" function and the "display" function are normally implemented. As always, you should break the code into appropriate functions, and then add code incrementally writing comments, adding code, compiling, debugging, a little bit at a time.

Remember to use good programming style when creating your program. Choose good names for variables and constants, use proper indenting for loops and conditionals, and include clear comments in your code. Also, be sure to save backup copies of your program somewhere safe. Otherwise, you may end up retyping your whole program if something goes wrong.

**4. Testing:**

Test your program with different user inputs until you get some images that look fun/interesting. Take a screen shot of these images to include in your project report. You may also want to show some bad/ugly images that illustrate what happens if there is a problem somewhere. You can discuss how you corrected these problems in your project report.

**5. Documentation:**

When you have completed your C++ program, write a short report using the project report template describing what the objectives were, what you did, and the status of the program. Be sure to include several output images. Finally, describe any known problems and/or your ideas on how to improve your program. Save this report to be submitted electronically via Blackboard.

**6. Project Submission:**

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic plagiarism analysis of all programs that are submitted. Please upload your project as follows:

1) If your code was developed on a Mac or Linux machine, please make sure you have the original #ifdef MAC lines at the top of your program so we can compile and run your code on either a Mac or Linux machine. If your code was developed on a PC, please edit your code so it can also be compiled on Linux. This can be easily done by adding something like this to your program.

```
#define LINIX
#ifdef LINUX
#include <GL/glut.h>
#else
#include <YOUR_OPENGL_INCLUDE.h>
#endif
```

When you want to compile on your PC, comment out the #define LINUX line. When you want to compile on turing, uncomment this line. The OpenGL libraries on turing are all installed in the normal way, so your code should compile. If you have X forwarding enabled, you can run your program on turing and display images on your PC.

2) Please copy your code over to turing, and create a small Makefile to compile your program. Something like this will do the trick (remember to use tab to indent the compile line).

```
# define libraries
LIB = -lGL -lGLU -lglut

# define g++ flags
CC = g++ -Wall -Wno-deprecated-declarations -O3

project: project.cpp
        $(CC) -o project project.cpp $(LIB)
```

3) When you get ready to upload into blackboard, please create a tar/zip file that contains your code, your makefile and report. This way we can download and untar/unzip your code, type "make" and then test it.

The dates on your electronic submission will be used to verify that you met the due date above. All late projects will receive reduced credit:

- 10% off if less than 1 day late,
- 20% off if less than 2 days late,
- 30% off if less than 3 days late,
- no credit if more than 3 days late.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so handing projects in on time is highly recommended.

**7. Academic Honesty Statement:**

Students are expected to submit their own work on all programming projects, unless group projects have been explicitly assigned. Students are NOT allowed to distribute code to each other, or copy code from another individual or website. Students ARE allowed to use any materials on the class website, or in the textbook, or ask the instructor and/or GTAs for assistance.

This course will be using highly effective program comparison software to calculate the similarity of all programs to each other, and to homework assignments from previous semesters. Please do not be tempted to plagiarize from another student.

Violations of the policies above will be reported to the Provost's office and may result in a ZERO on the programming project, an F in the class, or suspension from the university, depending on the severity of the violation and any history of prior violations.