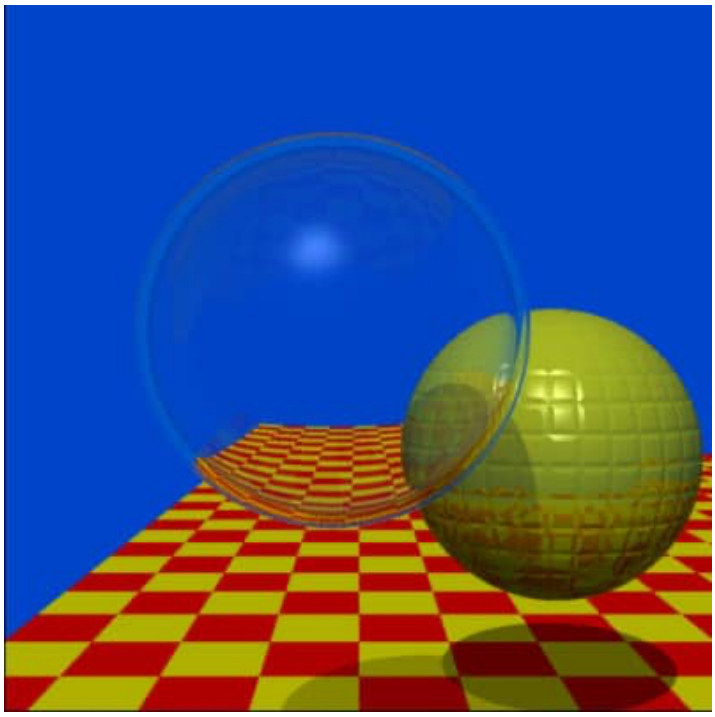
CSCE 4813 – Programming Project 6
Due Date – 05/02/2024 at 11:59pm

1. Problem Statement:

Ray tracing is capable of creating some amazingly realistic images of objects because it can simulate reflection, refraction, and many illumination effects. This realism comes at a cost, because in order to calculate the color of each pixel in the output image we have to cast one or more rays into the scene and calculate the intersection point with the object closest to the camera, and then recursively cast reflection and refraction rays to determine what is visible at each point.



In this project, you will be extending an existing ray tracing program to learn more about ray tracing internals as you add several new features. In particular, you are required to add **two** of the following four features to the ray_trace program in the CSCE 4813 src directory.

Feature 1 – Display moving spheres

The current ray_trace program displays a static image of a collection of N spheres with random colors in random locations. To make the program more interesting, you should add an "idle" or a "timer" callback to the OpenGL program that moves one sphere along a circular path around the other sphere like Turner Whitted's original ray tracing animation (see https://youtu.be/0KrCh5qD9Ho).

Feature 2 – Display different objects

There are not many realistic scenes that consist of only spheres, so it would be nice to extend the ray_trace program to display some different kinds of objects. If you look at the lecture notes and Wikipedia, you will find the necessary formulas to intersect rays with planes, cubes, cylinders, and cones. Pick **one** of these objects, and add the necessary functions/methods to intersect rays with this object. Then extend the ray_trace program to display N spheres and M new objects in random locations.

Feature 3 – Cast multiple reflection rays

The current ray_trace program only casts two rays per output pixel, one ray to determine which object is closest to the camera, and one ray from that intersection point towards the light source to see if the point is in shadow. To make more realistic images, we need to add reflection rays. To do this, you will have to calculate the ideal reflection direction, and cast a new ray in this direction and calculate the color of the object that is intersected (if any). Then you need to add Ks times the reflected color to the Phong shaded object color and store this in the output pixel. If you are feeling really ambitious, you could cast a second reflection ray and do two levels of reflection.

Feature 4 – Add multiple light sources

The current ray_trace program has only one light source. It would be nice to add multiple light sources to be able to create more realistic scenes. To implement this, you will need to add an array of light sources, and send "shadow rays" towards each of the light sources to calculate if the object intersection point is in shadow, and if not calculate the object color from that light source. You then need to add up the object colors from light sources and store this in the output pixel. You may find that the output image looks "over exposed". If so, you should reduce the brightness of your lights.

2. Design:

Your first design task is to do some research on the math required to implement the two features you have selected. The lecture notes and the Wikipedia page on ray tracing are good places to start. Once you have the math worked out, you should look over the code in ray_classes to see what classes/methods you can reuse, and what new classes/methods need to be added.

Your next design task is to decide what additional data structures will be needed to keep track of the moving spheres, new objects, and multiple light sources, and come up with algorithms to initialize and use these data structures. With OpenGL applications, you may need to make these global variables.

Finally, you need to come up with a plan for incrementally adding the features you have chosen. You may want to print or display intermediate information as you go. For example, you may want to work on a small portion of the output image, and print out the object ids for the spheres you intersect, and the ideal reflection direction before you actually trace the reflection arrays. When implementing multiple light sources, you could create an output image with the color from each light source separately before you try to combine them.

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. Once you have OpenGL installed, you can compile your graphics program using "g++ -Wall hw6.cpp -o hw5 -lGL -lGLU -lglut".

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 random number generator seeds 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 (e.g. too many lines, lines too short, etc.). 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. When you have completed the tasks above go to Blackboard to upload your documentation (a single docx or pdf file), and all of your C++ program files. Do NOT upload an executable version of your program.

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.