COMP 3490 Assignment 2
Fall 2024

Due date and submission instructions. This assignment has a written component as well as a coding component.

- **Processing code**. Submit **one zip file** containing **all** of the pde files required to run your submission. Name your zip file in the style of `VaughanJenniferA2.zip`, but with your name and not my name. All submissions should be uploaded to the Assignment 2 folder on UM Learn.

- **Written work**. Upload scans or photos of your answers to the written component of Question 2 on Crowdmark. A Crowdmark invitation will be sent to your U of M email address.

All submissions must be uploaded to their respective locations by 11:59pm on **Friday October 18**.

In this assignment, you are going to build your own ray tracer. There is no template code supplied. You are free to organize your code however you wish. The course content that you need can be found in `Week4-Sep25.pdf`.

Suggestions for geometry. It is simplest to work in a coordinate system in which the eye is at the origin. Set up your raster in 3D space by choosing its z-coordinate N, and setting its left, right, top and bottom boundaries. Choose dimensions for your canvas in pixels, and choose a location for the light source.

i dont get how the LRBT numbers are in terms of x,y units. But then coordinates/pixels come into play? How is N=16 same size as 800w 800h. And how is 800w 800h related to BT and LR

My demo was built using $N = 16$, $L = -10$, $R = 10$, $B = -10$, $T = 10$. You can use whatever values you like, but you will probably get the best results if N is around the same size as the width and height of the raster. I used an 800×800 pixel canvas, but you can make yours smaller if you encounter significant lag.

Make a plan on paper before you start writing code! Sketch your 3D space, and decide where you want to put your objects relative to the raster. For best results, the raster should be between the eye and the objects being drawn.

Suggestions for code. You will be drawing the canvas pixel by pixel. Recall that you can access the array of pixels in Processing as follows.

```
loadPixels();

// commands that set individual pixels
// by storing colors to pixels[i] for 0 <= i < width*height

updatePixels();
```

The above code can be put into the `setup()` function. My demo implemented all of the required features in this assignment without putting any code into `draw()`. I only redrew my image when something changed. In your solution, `draw()` can be empty.

Use lots of classes! Each type of shape in my demo has its own class, all of which inherit from an abstract base class. I also had classes for the rays, the ray cast results, and the various intersection calculations. Split your code up across multiple tabs so that you don't have to scroll through hundreds of lines to find something.

There are relatively few key concepts that have to be implemented, but they interact in complex ways. If you make an effort to keep your code organized and readable from the start, that will probably save you some trouble later.

1. **Create the rays (6 marks)**. This question and question 2 aren't really independent – you will probably be working on both of them simultaneously.

   Drawing is something that is done to pixels, not to objects. For each pixel on your canvas, create a corresponding ray. To draw that pixel, determine whether its ray intersects any of the shapes in your world. If one or more intersections occur, choose the closest one to the eye.

   To get the basic ray casting structure working, you can start by giving each object a flat color. Once you are successfully intersecting rays with objects, apply the Phong lighting model at the intersection point to determine the color. This will involve choosing a location for your light source; I recommend putting it above and behind the eye. You can reuse the Phong lighting code that you wrote for Assignment 1.

   (2 marks for constructing rays from the eye to each pixel, 2 marks for finding the closest intersection, 2 marks for Phong lighting at intersection point)

2. **Draw some surfaces (20 marks)**. To draw a surface using a ray tracer, you need to be able to answer two questions: (1) where does an arbitrary ray intersect this surface, and (2) what is a normal vector to this surface at an arbitrary point?

   (a) Implement spheres using the intersection and normal vector formulas that can be found on the class slides. Draw a few spheres of different sizes and colors.

   (b) Now implement planes. A plane is described by a position in 3D space, and a normal vector. The normal vector is the same at every point on the plane; it remains for you to determine the intersection point between a plane and an arbitrary ray. Add at least one plane to your image.

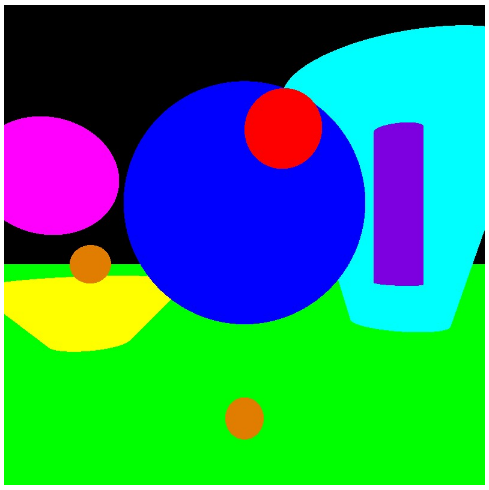
   Written component. Write up and submit your derivation of the intersection point between an arbitrary ray and a plane.

   (c) Implement **two** additional shapes of your choice, and add at least one of each shape to your image. Some options, each of which counts as a different shape:

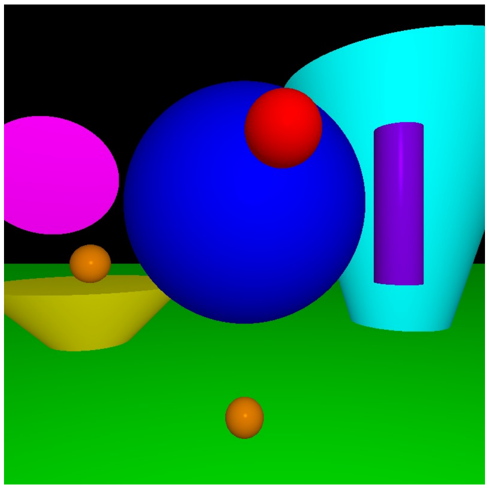
   - An infinite cylinder (i.e., a cylinder that consists only of the curved portion and has no endcaps)
   - An infinite cone
   - An infinite hyperboloid – see Assignment 1 (if you want to control the rate at which the curved portion expands, you can add another parameter to the equation: $x^2 + z^2 = ay^2 + r^2$, where $a > 0$ is a constant)
   - A compact planar figure like a circle or a square
   - A finite cone, cylinder or hyperboloid, which is made up of a curved portion together with one or more planar circles (takes a bit more work)
   - A cube (takes a bit more work)

   Written component. For each shape that you choose, write up and submit your derivation of the intersection point between an arbitrary ray and the shape. Also provide an expression for a normal vector to your shape at an arbitrary point, and briefly explain how you found that expression.

   (Code: 3 marks for spheres, 3 marks for planes, 3 marks for each of the two additional shapes. Written: 2 marks for planes, 3 marks for each of the two additional shapes.)



A collection of shapes, each of which is filled with a flat color.



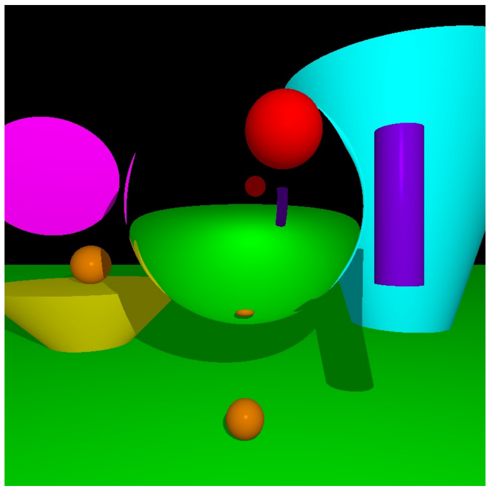
The same shapes with Phong lighting implemented.

3. **Visual effects (8 marks)**. Implement **two** effects that involve casting rays from a location other than the eye. Some examples:

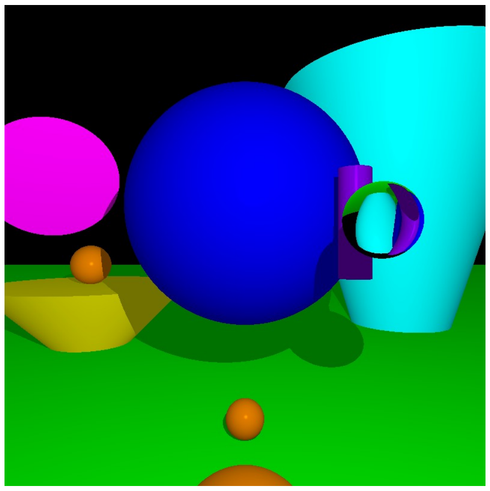
   - Make your shapes cast shadows.
   - Make one of your shapes into a mirror.
   - Make one of your shapes transparent and cause it to refract incoming light, like the surface of water. (I recommend trying this with a plane. For a 3D object like a sphere, the light has to bend twice – once on entry and once on exit.)

   If you have built your rays with sufficient generality, the ray casts themselves should take very little new code. The problem is how to integrate the results into your drawing procedure.

   (4 marks per effect)



Reflections and shadows.



Challenge: a glass sphere that refracts light. If you haven't taken physics in a while, you can look up refraction on Wikipedia or ask me about it in office hours.