**Programming Project Report**

Rhett Brandon

010722727

**Problem Statement:**

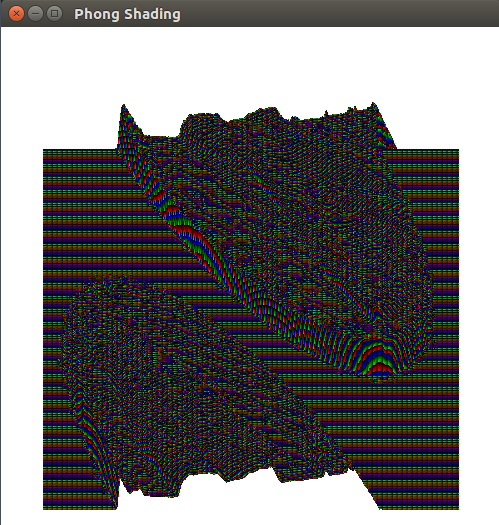
The goal of this project was to create a program that would take in a RGBD reading and produce a 3-D image of the object. The object should be able to be displayed in regular color mode, and in the Phong Display model. The 3-D object should allow users to rotate it using an interface of some sort, as well as having functionality for users to switch between Phong shading and color mode. The program accepts normal inputs from two files, one file containing RGB colors of each pixel where each value is between 0 and 255. The second file contains the depth map of the object, where 0 represents no depth and 255 represents maximum depth. The result should be an interactive representation of the 2-D image transposed into a 3-D space. No error handling is required.

**Design:**

For the design of this program I had to discover the best way to store all of the data, I decided to store this in a polygon mesh using the suggested Px(u,v), Py(u,v), and Pz(u,v) arrays. These afore mentioned arrays store the (x,y,z) values for the given 500x500 pixel image. The RGB values are stored in a similar fashion, where each value R(u,v), G(u,v), and B(u,v) are stored for their respective 500x500 pixel position. The advantage to using this method is that it made keeping track of everything easy to manage, the downside is that I had to initialize 6 different arrays that each store up to 250,000 values, this takes up a significantly more amount of memory for the program.

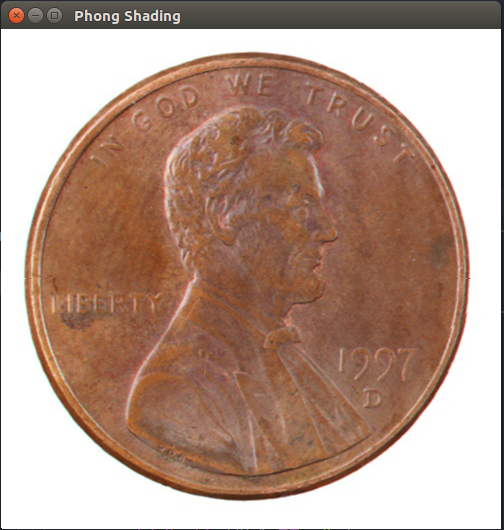
**Implementation:**

The actual implementation for each ask was more challenging than expected. The first task I completed was to create the surface for the 3-D model. The most challenging part of the first task was attempting to validate. As seen in the picture below, the object appears all white, making it difficult to see if it resembles a penny or not. The second task asked us to simply add rotation to the program, this helped the with the validation of the first task but it was still difficult to tell that the object was a penny.

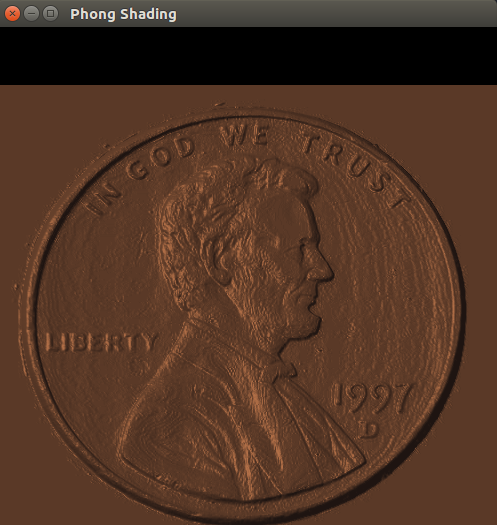


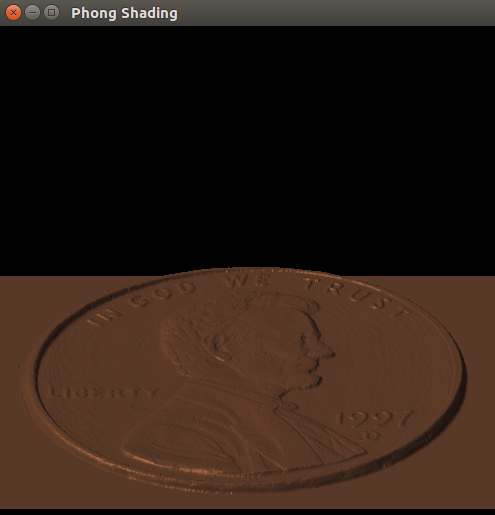
The third ask asked us to add color to the image. The main issue I had with this task was that my penny appeared with a bluish tint. Upon discussing this with some classmates we discovered that the values in the given penny-image.txt file appear in the order blue, green, red, and I was reading the values assumed as red, green, blue. After switching the color arrays around, the penny showed up as expected. See the pictures below for the color image of the blue penny, and the penny as it should appear.

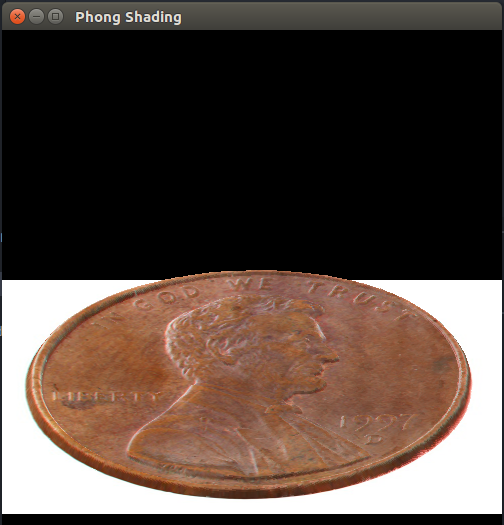




The next and final task asked us to implement the penny using a Phong shading algorithm. In order to implement this, I used the shading4.cpp source code to model after. Upon implementing this, the light was coming from the top right of the screen, and casting a bluish tone onto the penny. To make the penny look more like the image, I change the tint of the light to a more copper tone. I also moved the light to the top left corner of the screen, this gave it a similar look to that of the penny-image.jpg file. The development timeline took about 8 or 9 hours to complete.







**Testing:**

In order to test this program, I would implement a task, and then test the validity before moving on. As described earlier, first I tried to test my surface was correct, however with an all white object this was difficult to see. The next task was to add color, this was pretty easy to see, from a top down view it looks exactly like the penny-image.txt. The Phong shading was tested in a similar way.

**Conclusions:**

Overall the program works as described in the task description given in class. Values are read from the penny-image.txt and penny-depth.txt files and converted to a 3-D object. The object is interactable using the ‘x’, ’X’, ’y’, ’Y’, ’z’, ’Z’, ‘1’, ‘2’, and ‘3’ keys. The next time I implemented this project, I might try to implement this using other objects, not just pennies. I would be curious to see if this would work similarly if other objects are scanned and inputted into this program. The timeline on this project only took about 8 to 9 hours to finish.