CSCE 4813 – Computer Graphics
Programming Project 4 – Due Friday 03/13/2020

1. Problem Statement:

The goal of this programming project is to create an OpenGL program that uses Phong shading to display geometric models of objects that have been scanned with an RGBD camera. Color information (RGB) will be stored in one JPEG image and depth information (D) will be stored in a second JPEG image. We will be starting with the RGBD images of a penny below.

Task 1: Model Creation

We want to create a polygon model to represent the penny. The easiest way to do this is to model the top surface of the penny with a 2D surface D(x,y), where the D is the depth and (x,y) are the sample locations in the JPEG image.

Your first task is to write an OpenGL program that reads in a JPEG depth image and stores this information in a polygon mesh. In the case of the penny above, the D values range from [0..255] and the (x,y) coordinates go from [0..499, 0..499]. These values do not correspond to the real dimensions of a penny (the thickness of a penny is not half the diameter) so you will have to scale the D values into an appropriate range.

There are several ways to store the (x,y,z) locations of points in your polygon mesh. The most common choices are: (1) using three 2D arrays of float values x(u,v), y(u,v) and z(u,v), or (2) using one 2D array of 3D point values P(u,v) which store x,y,z values. You are welcome to choose whichever representation you are most comfortable with.

Task 2: Interactive Model Display

Once you have defined your surface, your next task is to extend your OpenGL program to display the model surface. To do this, you should add code in the display callback to loop over the polygons, and display them using GL_LINE_LOOP. The first time you run this, you will probably see a black/white grid of parallel lines. This is the top-view of the surface, so you will not see any depth information.

To view the surface from different angles, you need to extend your OpenGL program again to rotate it around three axes. To implement this, you need to add code to the display callback to call glRotatef(x_angle, 1,0,0), glRotatef(y_angle, 0,1,0) and glRotatef(z_angle,0,0,1). Then you need to add code to the keyboard callback to increase/decrease the global variables storing x_angle, y_angle, and z_angle when certain keys are pressed. (See object3.cpp for details).

If your penny surface looks too thick or too thin, you can adjust your scale factors until you get something that looks sensible.

Task 3: Saving Color Information

The color image of the penny provides us with the RGB values at each (u,v) location on the penny. Your third task is to extend your OpenGL program to read the color image, and store this information in three arrays R(u,v), G(u,v) B(u,v) or in the 3D point data structure.

If you print out these RGB values as you read them, you will see that pixels outside the penny will be almost pure white (255,255,255) and the pixels inside the penny will be a copper color (reddish brown). In order to display the penny as colored polygons, you need to extend your program to calculate and save the average RGB color of the penny pixels.

Task 4: Displaying the Penny with Stored RGB Values

Now that we know the original RGB color for every point on the penny surface, we can use this information to generate a more realistic rendering of the penny. Create a second display callback called “color_display” that displays polygons using GL_POLYGON instead of GL_LINE_LOOP. Then add code to specify the RGB color values of each polygon vertex using the glColor3f() function. Since this function expects colors in the [0..1] range, you will have to scale the original RGB values. As you interactively rotate your new penny image, it should look more realistic because depth information is also being displayed.

Task 5: Displaying the Penny using Phong Shading

Your next task is to create a third display callback method called "phong_display" that displays your penny surface using Phong shading. Before you can do this, you need to add code to your program to calculate and save the surface normal at each (u,v) location on the surface. This can be done right after you read and store the (x,y,z) surface information using the cross-product approach discussed in class.

To call the OpenGL functions for Phong shading, you should #include "shading.cpp" in your program. Then you need to modify init() to define your light positions and colors. To display the polygons using Phong shading, you need to specify material properties using the average RGB color of the penny before the polygon display loop. Finally, you need to remove calls to glColor3f() and add calls to glNormal3f() to specify the surface normal at each polygon point.

When you look at the resulting Phong shaded surface, it should look like it is all one material, but there should be color variations that let you see the shape of the penny surface. You may need to play with the Phong shading parameters Ka, Kd, Ks, Kp to get something that looks realistic. If you set your lights in the right locations, you may get a result that is close to the original RGB image of the penny.

Task 6: Complete the User Interface

Your final task is to extend the "keyboard" callback to allow users to switch between the three display callback functions you created. For example, you could use "1" for the wire frame display, "2" for the RGB surface display, and "3" for the Phong shaded display. This will let the user look at the penny from the same point of view with different display methods to compare results.

2. Design:

There are several design tasks you must complete for this project. The most important decision is to select data structures for the (x,y,z) coordinates, the corresponding surface normal values (Nx,Ny,Nz), and RGB the colors of every point.

There are also a number of scale factors and Phong shading parameters that you will need to select to get realistic looking images. You may want to do this by extending the program's user interface, or by a series of edit, compile, run experiments.

3. Implementation:

This semester we will be using C++ and OpenGL to implement all of our programming projects. If you are using a Mac with Xcode installed, then you can download the src.tar file and compile the sample graphics code using the enclosed Makefile. If you are using a PC, then your best option would be to download and install a Linux VM from the department's website. The instructions for doing this are

posted in README file the "Source Code" page of the class website. Once you have Linux and OpenGL installed, you can compile your graphics program using "g++ -Wall project2.cpp -o project2-lGL -lGLU -lglut".

Remember to use incremental development and 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 to check that it operates correctly for all of the requirements listed above. Also check for the error handling capabilities of the code. Try your program with several input values, and save screen shots of your output in jpeg images for inclusion 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. Does it work properly for all test cases? Are there any known problems? Save this report to be submitted electronically.

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.