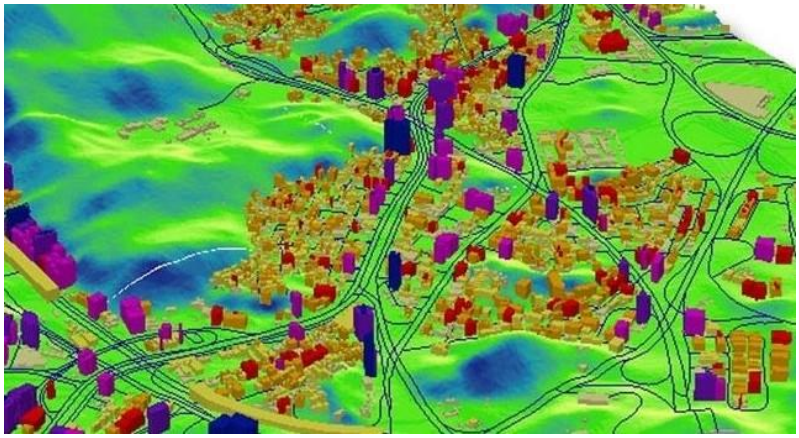
CSCE 4813 – Programming Project 3
Due Date – 03/13/2019 at 11:59pm
EXTENSION

1. Problem Statement:

The background of a graphics image is often overlooked because it is in the background and not as interesting as the main characters in the foreground. But without a realistic looking background, the scene just looks empty.



The goal of this project is to create and display a background image of a terrain model using Gouraud shading and a distance weighted diffuse reflection model. To do this, your program must do the following:

Model creation: The simplest way to represent a terrain surface is with a 2D grid of rectangular polygons. We can store the (x,y,z) coordinates of vertices of these polygons in three 2D arrays. To create this terrain surface, you can start with a low order polynomial model, and add random noise to each coordinate location. If the resulting surface looks too rough, you can loop over the 2D arrays and perform 3x3 or 5x5 averaging of coordinates to smooth it out.

Normal calculation: Before we can calculate the color of each vertex in the terrain, we need to know the surface normal direction for each vertex point. To do this, you need to loop over the coordinate arrays, calculate two tangent directions T1 and T2, and then calculate their cross product to get the surface normal N at that point. Remember to normalize the surface normals so they have unit length.

Specify lighting: For this project we will be using two point light sources L1 and L2 at locations (x1,y1,z1) and (x2,y2,z2) with colors given by (r1,g1,b1) and (r2,g2,b2) respectively. It is often difficult to guess in advance what light locations and colors will produce a nice looking image. To make this process interactive, you should add keyboard callbacks so when the user types '1' or '2' they select that light, and when

they type ‘x’, ‘X’, ‘y’, ‘Y’, ‘z’, or ‘Z’ they change the location of the selected light, and when they type ‘r’, ‘R’, ‘g’, ‘G’, ‘b’, or ‘B’ they change the color of the selected light.

Calculate vertex colors: For this project we will be using a distance weighted diffuse reflection model. To start, we will assume that the polygons on the surface all have the same (R,G,B) color. To determine how much light reaches the surface from each light source, you must find the vector V from the vertex point to the light source, and then calculate the dot product of V with the surface normal N. To account for the distance from the light, you must then multiply by $1/(a+bD+cD^2)$ where D is the Euclidean distance from the vertex point to the light source and (a,b,c) are predefined constants. Using this information, you can calculate and store the visible (r,g,b) colors for all vertices in three 2D arrays.

Model display: Now that we know the (x,y,z) locations and (r,g,b) colors for all polygons in our terrain model, we can use OpenGL’s color interpolation feature to display the Gouraud shaded rendering of our terrain. To do this, you simply need to loop over all of the polygons, and specify the color of each point using glColor, and the coordinates of each point using glVertex. In order to view this model from different angles, you should add keyboard callbacks so when the user types ‘R’ the program goes into ‘rotate mode’, and when they type ‘x’, ‘X’, ‘y’, ‘Y’, ‘z’, or ‘Z’ they decrease/increase the rotation angles in the X,Y,Z directions respectively.

2. Design:

Your first design task is to select appropriate data structures to store the vertex information for the terrain. You need to store the coordinates, the surface normal and color values for each vertex. You also need to store the light positions and colors, and rotation angles as global variables.

Your second design task is to work out the exact formulas needed to calculate distance weighted diffuse reflection. To start, you may want to implement basic diffuse reflection for one light source, and then add the second light source and distance weighting afterwards.

Finally, you need to decompose the tasks above into a number of functions that can be called from within the display and keyboard callbacks to create images of your terrain surface.

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 gouraud.cpp -o gouraud -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 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 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.