CPSC 4050/6050 Fall 2021

Project 1

due date: Septeber 10 upload to handin

Summary

For this project you are to write C++ code for a triangle class. You will generate a collection of triangles and display them in OpenGL. You may use the starterkit in any way you wish. You may ignore the startkit entirely and write your own code from scratch. You may make minimal modifications to the starterkit so that it satisfies the description below. Or you may do something in between.

1 Description for All Students

You must create two files, Triangle.h and Triangle.C (or Triangle.cpp, depending on your personal preference for the extension of C++ files), holding a declaration of a triangle class (Triangle.h) and an implementation of a triangle class (Triangle.C or Triangle.cpp). The input to the triangle class is the set of three points that define the triangle. You must also have the triangle class compute the following quantities and make them available outside of the class as const references from class const method calls:

1. The three edge vectors
2. The unit normal to the triangle
3. The area of the triangle
4. The aspect ratio of the triangle

The three vertices must also be available as const references from class const method calls.

You must write function(s) that generate(s) multiple triangles and stores them in a data structure (your choice). The triangles must be generated according to the following rules:

1. The three vertices of the first triangle are positioned randomly in space, but each edge should be between 1 and 2 units in length.

2. The second triangle should use two of the vertices from the first triangle, and the last vertex is randomly chosen so that its edges are between 1 and 2 units in length. The angle between the normal of the first triangle and the normal of the second triangle should be less than or equal to 30 degrees.

3. The third triangle should use two of the vertices from the second triangle, and the last vertex is randomly chosen so that its edges are between 1 and 2 units in length. The angle between the normal of the second triangle and the normal of the third triangle should be less than or equal to 30 degrees.

4. The Nth triangle should use two of the vertices from the (N-1)th triangle, and the last vertex is randomly chosen so that its edges are between 1 and 2 units in length. The angle between the normal of the (N-1)th triangle and the normal of the Nth triangle should be less than or equal to 30 degrees.

Give each triangle of random RGB color. Display the triangles using the OpenGL glBegin() and glEnd() calls, and the GL_TRIANGLES option. Make sure you use glColor3f() or a similar call to display each triangle with its appropriate color.

The user of your executable should be able to navigate the camera around your triangles.

You must generate a minimum of 30 triangles.

Description for 6050 Students

As each triangle is added to the collection, there is a requirement:

> The angle between the normal of the (N-1)th triangle and the normal of the Nth triangle should be less than or equal to 30 degrees.

You must make the 30 degrees adjustable using a keyboard input (triggering a glut keyboard callback). There should be a running parameter for that angle. The 'a' key should reduce the value a small amount (but never go below 0 degrees), and the 'A' key should increase the value (but never go above 90 degrees). Each time the angle is changed, you must regenerate the collection of triangles and make the new collection available for OpenGL display.

Some general guideance

- There should be no "hardwired values", i.e. numbers that are not assigned to variables. For example, in the code

```
for(int i=0;i<5;i++)
{
   ...
}
```

the number 5 is a hardwired number. Dont do this. Instead, use a descriptive variable, i.e.

```
int count_of_items_in_list = 5;
for(int i=0;i<count_of_items_in_list;i++)
{
   ...
}
```

There is one exception to this: The 0 initialization of the loop variable i is very common and acceptible as long as it has the very simple interpretation of initializing a counter. If its meaning is deeper than that, then it should be defined in an appropriately named variable also.

- Remember that the code must be compilable and executable on a School of Computing Linux computer. It is not the grader's job to ferret out the obscure and complex set of steps that you have set up to compile and run. Use a makefile to handle the process, and make it trivially easy to understand. The Makefile in the starterkit satisfies those requirements.

- Part of your grade is based on the cleanliness and clarity of your code. Think of you code as a kind of essay, and this is an essay assignment. If you turn in an essay in the English department that reads like

  > Times bad. Good sometimes. i dont have pride in work so im jus fillin quickly not paying utention

  You will probably be graded accordingly. Same here. *Write code that is meant to be quickly understood by others.*

- The starterkit gives you a lot of functionality that you can exploit, but when you change something you will generally need to make sure that you understand what the thing is doing and what your changes do.

Upload to handin

Create a folder called `<username>`. Put all of the following files into that folder. There should be no subfolders.

Zip compress the folder into a single zip file, named `<username>.zip`. Upload this file to the handin system. The course webpage has more guideance and caveats if you need them.