Computing Science CMPT 464/764 **Spring 2019**
Instructor: Richard (Hao) Zhang **Simon Fraser University**

Assignment #1 (Total mark: 30 + 6 bonus marks)
Due via electronic submission by 23:45 on February 8

On the cover page of your homework, write and sign the following statement: "I have read and understood the policy concerning collaboration on homework and lab assignments". Without such a signed statement, your work will not be marked. A sample cover page can be found on the course webpage.

Problem 1 (18 marks + 6 bonus marks) Mesh data structures, GUI, and subdivision

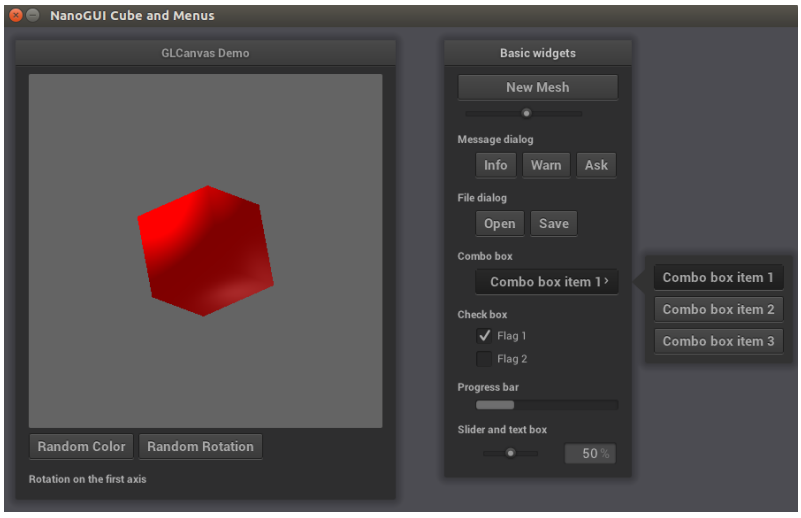
You will design and program a **graphical user interface (GUI)** for visualizing triangle meshes. The GUI may be enhanced in later assignments and/or the course project.

1. *[4 marks] Learn NanoGUI and design your GUI*

   The interface you program should ideally use NanoGUI and I expect you to learn it mostly on your own, with some initial guidance by the TA. NanoGUI is a minimalistic cross-platform widget library for modern OpenGL (3.x or higher). It supports automatic layout generation, stateful C++11 lambda callbacks, and a variety of useful widget types. The following screenshot shows the screen generated by the demo code:

   https://coursys.sfu.ca/2019sp-cmpt-764-g1/pages/demo_code/download

   It demonstrates how to generate a custom GUI and also how to render and interact with a simple cube:



You do not need to download and set up a NanoGUI environment, it is provided to you in the demo code. However, if you want to download it, you can find the latest build at the following link:

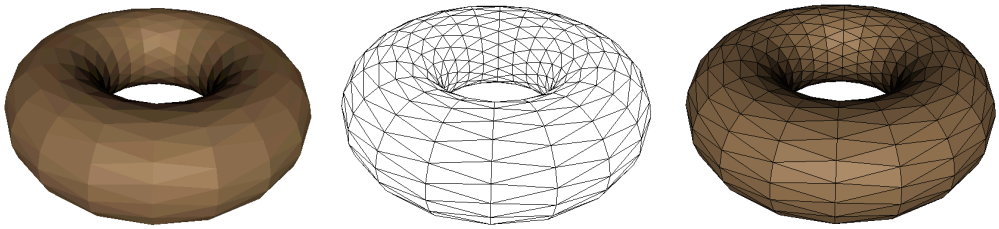
https://github.com/wjakob/nanogui

CSIL and Big Data lab machines have all packages necessary to compile your application on their Ubuntu environments. This library is compatible with Windows and OSX environments, however you need to **make sure that your code runs in the CSIL/Big Data Lab Ubuntu environments.** To compile the sample code, simply download it and run these commands using the terminal:

cd <path-to-folder>
cmake <path-to-folder>
make
./mytest

You are required to design a GUI that responds to user inputs and displays a mesh model in different ways. At least the following components should be provided:

- A combo box selection to modify how to present the mesh as shown below. These options are: *flat shaded, smooth shaded,* in *wireframe,* or *shaded with mesh edges displayed*.



- Allow rotation, zooming, and translation. Use the proper controls or events from NanoGUI. There are two viable options: use mouse event listeners to interact with the mesh directly or use sliders to interact indirectly.
- Implement the logic to input from and output to a mesh file. The widgets for getting a file paths for both these operations were given in your demo code.
- A "Quit" button to exit from the GUI.

You will receive full marks for this part as long as your program can generate such an **interface;** there will be no need to bind the controls with the intended actions.

2. *[10 marks] OBJ parser and winged-edge mesh data structure*

Write code to input a triangle mesh given in **OBJ** format and store the connectivity and geometric information in the **winged-edge data structure.** There is a lot of information about this well-known data structure. One good description is here:

http://tinyurl.com/m6d7x2

You can also learn face-based mesh data structure and the half winged-edge data structure there. You may assume that the input mesh is closed (i.e., no boundary) and there are no non-manifold vertices or edges in the mesh. Both input from and output to **OBJ** files should be implemented in this part.

Note that in the **OBJ** format, a mesh is given by a *vertex list* followed by a *face list*. Each line in the vertex list starts with the character 'v', followed by the x, y, and z vertex coordinates. Each line in the face list starts with the character 'f', and followed by three integers indexing into the vertex list. The vertex indexes start with 1 and are given in *counterclockwise order*, viewed from the tip of the triangle's outward pointing normal. Any comments start with the character '#'. The very first line of the **OBJ** file is of the form "*# n m*", where *n* is the number of vertices and *m* the number of faces in the mesh. Some sample useful **OBJ** files can be found in the assignment directory:

http://www.cs.sfu.ca/~haoz/teaching/cmpt464/assign/a1/

3. *[4 marks] Mesh display*

   Write code to display the mesh (in various modes, as specified in part 1) in the display box of your GUI. Note that you want to ensure proper width and stable visibility of the mesh edges. You may choose lighting and material parameters on your own. As far as efficiency is concerned, your program should be able to load and display meshes with up to 50,000 faces reasonably interactively. Obviously, you cannot rely on a quadratic-time OBJ parser. Try your program on larger models, e.g., the horse model with about 20,000 vertices.

4. Bonus problem [6 extra marks]: Butterfly and Loop subdivision

   You are challenged to implement the Butterfly and Loop (the original, not Warren's, as seen in Slide 16 of Lecture slides on Subdivision Zoo) subdivision schemes. If you do so, you should update your GUI to provide additional controls (e.g., another combo box to choose between Butterfly and Loop, a slider to choose number of subdivision levels, and a button to start the subdivision processing) in the GUI you implemented so that the results from your subdivision program can be properly generated and displayed.

   A simplifying assumption you can make is that the input mesh is a closed manifold mesh. Some small test meshes and results from subdivisions can be found in

   http://www.cs.sfu.ca/~haoz/teaching/cmpt464/assign/a1/subdivision

What to submit electronically:

All the source **cpp** code plus the **CMakeLists.txt**, in a directory called **A1_obj_view**. You should submit a **single ZIP file** (**a1q1.zip**), which when opened, will contain this directory with the appropriate files in it. Please, do not send the rest of the environment/libraries. The executable program produced by the make command should be called **obj_view**. Running **./obj_view** will start the GUI so your implementation can be tested. Also submit a **README** file that contains at least the following information:

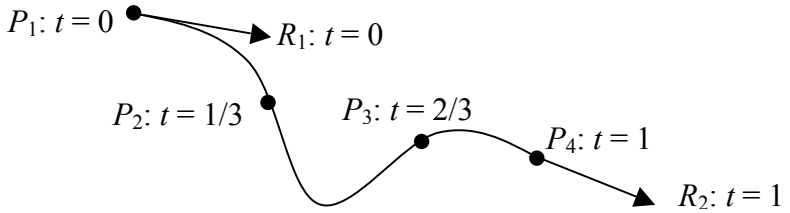
- Which feature(s) listed above were *not* implemented.
- Any special instructions or information you wish to provide for the grader.

If you are submitting code for the bonus question, similarly, please submit all the source code plus a **CMakeLists.txt**, in a directory called subd. Make the zip file **a1bonus.zip**.

The executable program will be called subd and running ./subd will start the GUI from which your programs can be tested. Also submit a README as above.

Problem 2 (3 marks): Parametric curve design

Determine the change of basis matrix for a quintic (that is, degree-5) parametric curve which is defined by four points and two tangent vectors, as shown below. Use the transpose of $[P_1\ R_1\ P_2\ P_3\ P_4\ R_2]$ as your vector of control points or observable quantities. Expressing your solution as the *inverse of a computed matrix* is sufficient.



Problem 3 (3 marks): Continuity of cubic B-splines

Think about how the change-of-basis matrix for cubic B-splines can be derived. Note that you *need not* submit a solution for this. You need to complete the following however. Recall that given the vector of four control points $P = [P_0\ P_1\ P_2\ P_3]^T$ and the monomial basis $T = [1\ t\ t^2\ t^3]$, the cubic B-spline curve piece defined by P is given by

$$p(t) = TM_{\text{B-spline}}P, \text{ where } M_{\text{B-spline}}$$

$$M_{B-spline} = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

is the cubic B-spline change-of-basis matrix,

Prove that piecewise cubic B-spline curves are $\mathbf{C}^2$.

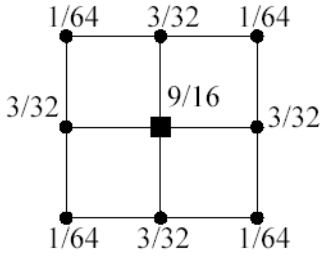
Problem 4 (3 marks): Approximating a circular arc using a Bezier curve

Very efficient algorithms exist to draw cubic Bezier curves. In fact, these algorithms are so efficient that other types of curves are often converted to Bezier curves for display purposes. You are to approximate a 90° circular arc (it is part of the unit circle located in the first quadrant) with a Bezier curve. Derive the *coordinates* of the four control points of a Bezier curve that approximates the arc. This approximation should touch and be tangent to the arc at both of its endpoints as well as at its *midpoint*.

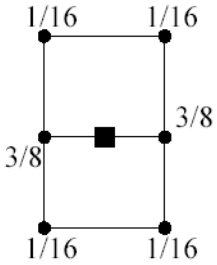
Problem 5 (3 marks): Midpoint subdivision

Refine an arbitrary closed manifold triangle mesh, whose connectivity is characterized by the graph G, by subdividing each triangle uniformly into four sub-triangles. Call the resulting graph G_1. Then compute the centroids of all the sub-triangles. Connect these centroids in the fashion of a dual graph G'_1 of G_1. This results in a polygon mesh composed mostly of hexagons. Finally, compute the centroids of all these polygons and connect them in the fashion of a dual graph G''_1 of G'_1. These constitute one step of a *midpoint subdivision scheme* for arbitrary triangle meshes.

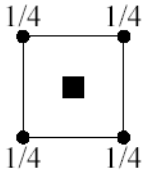
Compute the *masks* (showing topological and geometric rules of subdivision, as given in the figure below for Catmull-Clark subdivision over regular regions) for both regular and (general) extraordinary cases of this scheme.



Vertex rule



Edge rule



Face rule