CS3388 - Assignment 2, 2016

Posted: 11th October 2016
Due: 28th October 2016, 11:55 PM
(late submission till 31st October 2016, 11:55 PM)

Description

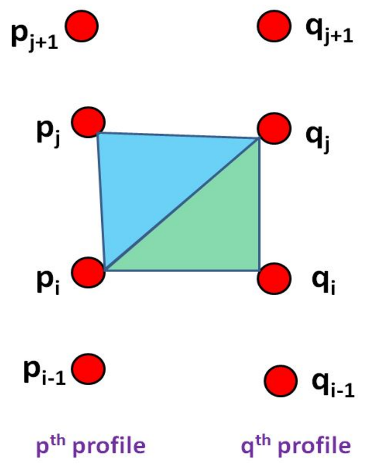
This assignment is about implementation of a wiremesh renderer. You have to create a 3D object which is defined around an axis of symmetry. You should write the code in C/C++ and OpenGL libraries. The assignment is worth **10%** of the final mark.

You are supplied a text file *vase.txt* which consists of a profile (of discrete points) defined in one of the planes of the 3D coordinate system. You have to read the points, rotate this profile around a specified axis by a small angle increment, collect these newly formed points and create a polygonal mesh made up of triangles, and render it. The steps are as follows:

- Read the profile points from *vase.txt*
- Consider z-axis as the axis of rotation and 2° angular increments of the given profile to form the discrete object boundary. That means, there will be 180 profiles around the axis of symmetry ($360/2 = 180$). However, you can use more profiles if you want, to make it look better in terms of smoothness, but 2° increment works pretty well. Use the matrix $R_z(\theta)$ for rotation, as in the course notes.
- Once you generate the discrete profiles, triangulate the data to form a polygonal mesh of the object. You can use any triangulation method (say Delaunay triangulation), but to keep things simple, you can perform a naive triangulation as follows.

  Consider two consecutive profiles (having same number of discrete points in each), which you have generated by applying discrete rotation to a single profile. Let's consider neighbouring points (p_i, p_j) on p$^{\text{th}}$ profile, and points (q_i, q_j) on q$^{\text{th}}$ profile. For the triangulation, we simply consider points (p_i, q_i, q_j) forming the first triangle, and points (p_i, p_j, q_j) forming the second triangle. In the figure below, these two triangles are shown as green and sky colours respectively.

  Use the same technique to triangulate all the points in all the profiles.



- Use appropriate data structure to store mesh attributes (i.e. vertices, faces of the triangles, etc.).
- Next is the rendering step. You will write code of wiremesh renderer that can be used to visualize wiremesh objects. Define and open a window, create a synthetic camera, define the light source, material properties (choose any colour as you wish), and display the object (that you have stored as polygon mesh) in the window. You can use this tutorial to get an idea of how to use camera and specify material properties in OpenGL (this is just a suggestion, feel free to use any resource, there are plenty of tutorials and examples available in the web). You can get help on wiremesh (and also general mesh data structure) from this tutorial.

Bonus Mark: There is bonus mark (5% of the assignment) if you can show the coordinate axes and animate the rendered vase by moving the light source around the vase.

What to submit?

Submit the program file(s) you have implemented. Put all the file(s) into a zip and submit via OWL (no files will be accepted by email). Please don't submit any unnecessary files (such as the whole project).

Late penalty

The late policy is a penalty of 5% per day up to 3 days of lateness. Saturday and Sunday count as one penalty day.

Plagiarism

Copying the code is a serious academic offence, which will be treated as per university norms. Remind that changing variable names and white spaces do not make your code unique, it's very easy to detect these cases using softwares.

General marking scheme

The marks will be distributed as follows:

- Working program: 70%
  - (will be divided into different parts of the assignment)
- Documentation: 10%
  - Main comment block identifying the student (name, student number, email address): 4%
  - Defining input and output parameters for a function: 3%
  - Purpose of functions/blocks of code: 3%
- Program style: 10%
  - Meaningful variable names: 3%
  - Constants instead "magic numbers": 2%
  - Readability (complete sentences, indentation, white spaces, etc): 2%
  - Code flows "nicely": 3%
- Program structure: 10%
  - Modular code: 4%
  - Uses appropriate data structure: 3%
  - Loops when needed/no loops when not needed: 3%