Programming Assignment 1

Announcement: Session # 1
Submission Deadline: Session # 4

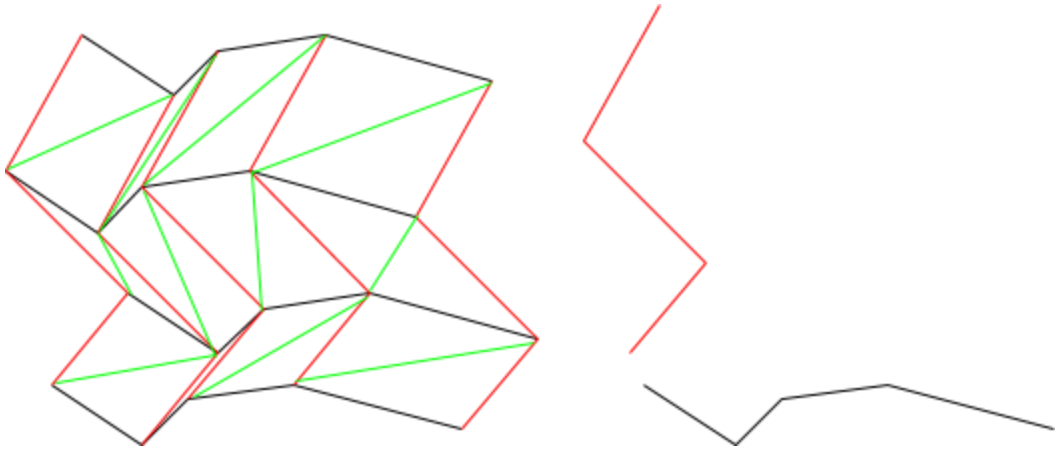
Description
In this OpenGL programming assignment you will learn how to program the building of a 3D mesh model using a simple method, sweep surfaces, how to display a mesh using points or lines or triangles, and also how to view it from different angles and distances.

A **mesh** is a set of polygons that share vertices and edges which describe the shape of a geometric object. In this assignment we will create triangle meshes to represent and display a translational sweep surface and a rotational sweep surface, which are described further below.

Translational Sweep Surface Definition:
Given a *profile* curve **P** and a *trajectory* curve **T** (both are 3D polylines in this assignment), a **translational sweep surface** is created by translating (sweeping) **P** along **T**. The following steps can be used to create a mesh representing a translational sweep surface.
Let $p_1, p_2, \ldots, p_n$ and $t_1, t_2, \ldots, t_m$ denote the sequence points in **P** and **T** respectively. Let $\mathbf{P^1}, \mathbf{P^2}, \ldots, \mathbf{P^m}$ denote successive translations of **P** obtained as follows:

1. $\mathbf{P^1} = \mathbf{P}$, $\mathbf{P^2} = \mathbf{P^1}$ translated by the vector (t_2-t_1), $\mathbf{P^3} = \mathbf{P^2}$ translated by the vector (t_3-t_2), and in general $\mathbf{P^{i+1}} = \mathbf{P^i}$ translated by the vector $(t_{i+1} - t_i)$.
2. Connect corresponding points between $\mathbf{P^1}$ and $\mathbf{P^2}$, $\mathbf{P^2}$ and $\mathbf{P^3}$ and so on to get a triangle mesh of the translational sweep surface.

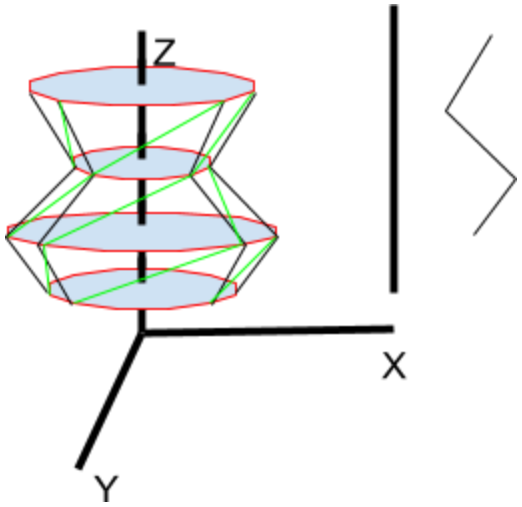
For an example, see illustration below, showing **P** (black), **T** (red) and green edges for the triangles.



Rotational Sweep Surface (also known as *Surface of Revolution*) Definition:

Given a planar *profile* curve **P** (for this assignment we will use a polyline in the XZ plane (y=0)), a rotational sweep surface is created by rotating every point in p_i (x_i, 0, z_i) about an axis in 3D (for this assignment we will use the Z axis) to get a circle $\mathbf{P}^i$ (actually a polyline approximation of a circle) in the $z = z_i$ plane. The mesh creation process is similar to the one above, i.e., connect corresponding points in adjacent $\mathbf{P}^i$ to get a triangle mesh

See illustration below (swept circles in red, connecting edges between adjacent swept circles in black).



Implementation Specifications

Develop an OpenGL application with the following functionalities and features:

- read the text file “input_a1.txt” with data for two polylines for translation sweep (profile curve data followed by trajectory curve data) and one polyline (profile curve) for rotational sweep as described below.
    - I (integer = 0 for translational sweep and ≠ 0 for rotational sweep
- If the first line is 0 then the subsequent lines will be input data for a translational sweep as follows:
    - N (integer number of points for profile curve)
    - p1x p1y p1z (3 reals denoting x y and z of first point of **P**)
    - p2x p2y p2z (3 reals denoting x y and z of second point of **P**)
    - …
    - pnx pny pnz (3 reals denoting x y and z of Nth point of **P**)
    - M (integer)
    - t1x t1y t1z (3 reals denoting x y and z of first point of **T**)
    - t2x t2y t2z (3 reals denoting x y and z of second point of **T**)
    - …
    - tnx tny tnz (3 reals denoting x y and z of Mth point of **T**)

- If the first line is ≠ 0 then the subsequent lines will be input data for a rotational sweep as follows:
  - S (integer number of spans)
  - N (integer number of points for profile curve)
  - p1x 0.0 p1z (3 reals denoting x y and z of first point of **P**)
  - p2x 0.0 p2z (3 reals denoting x y and z of second point of **P**)
  - …
  - pnx 0.0 pnz (3 reals denoting x y and z of Nth point of **P**)

- create a triangle mesh for the translational sweep surface defined by **P** and **T** or the rotational sweep define by **P**.
- create a GLFW window of size 800x800 with double buffering support.
- render the mesh on display.
- The application should use a perspective view to display the mesh and use the depth buffer for hidden surface removal.
- All vertices in the mesh must be colored. You can select whatever color you like provided that it somehow relates to the depth of that point. The simplest example would be to use the value of z to form a color (0,0,z) at each point.
- handle the following input:
  - the user can rotate the mesh using keyboard input i.e. left arrow → R_z, right arrow → R_{-z}, up arrow → R_x, down arrow → R_{-x}
  - the user can change the rendering mode i.e. points, lines, triangles based on keyboard input i.e. key 'P' for points, key 'W' for lines, key 'T' for triangles
  - the user can move the camera using the mouse i.e. moving forward/backward while left button is pressed → move into/out of the scene
- Window resize Handling:
  - The application should handle window resize events
- The application should use OpenGL 3.0 and onwards, and include brief comments explaining each step.

Submission (electronic submission through Moodle only)
Please create a zip file containing your C/C++ code, vertex shader, fragment shader, a readme text file (.txt).
In the readme file document the features and functionality of the application, and anything else you want the grader to know i.e. control keys, keyboard/mouse shortcuts, etc.

Additional Information
- You can use the skeleton code provided during the lab sessions to get started.
- The demo video shown in class can be found on YouTube:
  - https://youtu.be/5bpY58C5qrg
  - https://youtu.be/ccmit_YNJAs
- Sample input files and their output can be found here:
  - https://drive.google.com/drive/folders/0B4PJqE1tGP2cZHc1aDNlOVhoNFE?usp=sharing