COSC363 Computer Graphics
Lab06: Vectors and Matrices (GLM)

Aim:

This lab gives an introduction to the GLM library and familiarizes you with vector and matrix operations commonly used in graphics applications.

I. Coil.cpp:

The program displays only a floor plane created using a grid of lines. The arrow keys rotate the camera's position around the origin. Please go through the following functions defined in the program in the order given below.

(a) **generateCurve**(): This function generates a set of 36 vertices on an elliptical curve, and translates the curve to (xshift, yshift) as shown in Fig. 1(a). The points are added to the array "vertices" and the centre point (xshift, yshift) is added to the array "centres". We will use this curve as the base curve for generating the model of a coil (a.k.a helix). The extruded shape is generated by revolving the curve about the y-axis, and simultaneously translating it along the y-axis. We generate a sequence of curves, where each curve is obtained by transforming the last curve in the sequence, by a matrix T. The points on the curves are then connected using quad strips to form the surface of the coil model. The curves are referred to as "slices" of the coil.

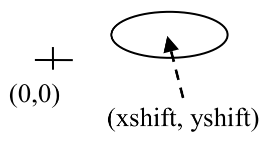

Fig 1(a)

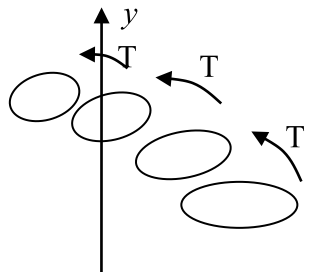

Fig 1(b)

(b) **generateModel**(): This function populates the array "vertices" with the coordinates of points on the slices (36 vertices per slice), and the array "centres" with the centre points of each generated slice. The transformation matrix T is the product of a rotation matrix and a translation matrix. The parameters of the transformations are already defined in the function. This transformation is applied to the *last* slice (i.e., the last set of 36 points in the array "vertices") to generate a new slice. The centre point of the last slice is also transformed using the same matrix to get the centre of the new slice. The code between the two comment lines "//-----" when completed, creates *one* transformed slice. Repeat the process to generate 150 slices. The program should generate the following output (Fig. 2)

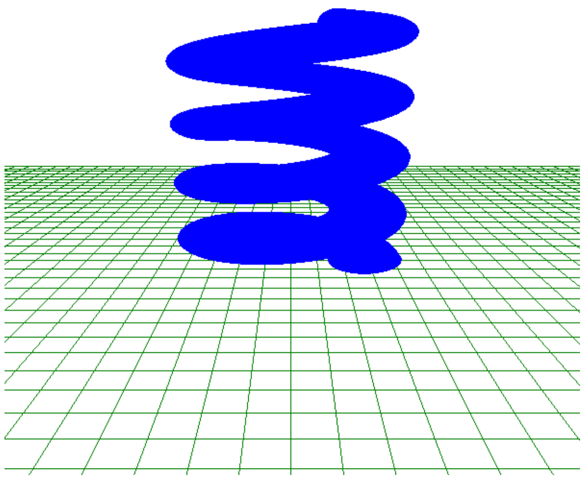
Fig. 2.

(c) **drawCoil**(): This function generates quadstrips between consecutive pairs of slices. vertices[i] on slice with index 'c' corresponds to vertices[i+36] on the next slice with index 'c+1' (Fig. 3). All vertices are assigned a constant colour (0, 0, 1). The program does not use any light sources, and lighting is not enabled.

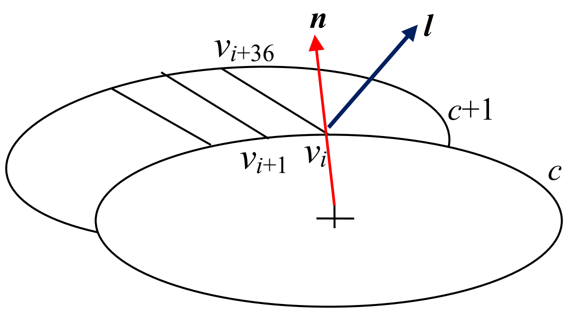

Fig. 3.

(d) **color**(): We will create our own illumination model and use it to compute the colour values at each vertex. The function color() receives a vertex on a slice and the centre of that slice as input parameters. The light's position and material colour are defined inside the function. Define the light vector ***l*** as the vector from the point to the light source (the blue arrow in Fig. 3). Note that light's position is specified using 3 coordinates (vec3) while the input point has four coordinates. You can cast point to an object of type vec3 using "glm::vec3(point)". Normalize the light vector using the function provided by GLM. The normal vector ***n*** at the given point can be specified by the vector from the centre of the slice to that point (the red arrow in Fig. 3). This is only an approximation, not the true normal vector. Normalize this vector also, and compute the dot product ***l.n***. If the value is negative, set it to 0. Compute the final colour as (ambient*material) + (***l.n***) material. We will now use the outputs of this function to assign colour values of each vertex specified in drawCoil().

In the drawCoil() function, remove the first line which assigns the blue colour to all vertices. Call the color() function for each vertex of the quadstrip. The function call for the first vertex "vertices[i]" is given as an example. Note that vertices[i+36] belongs to slice with index c+1. The final output of the program with the above simple illumination model is shown in Fig. 4.

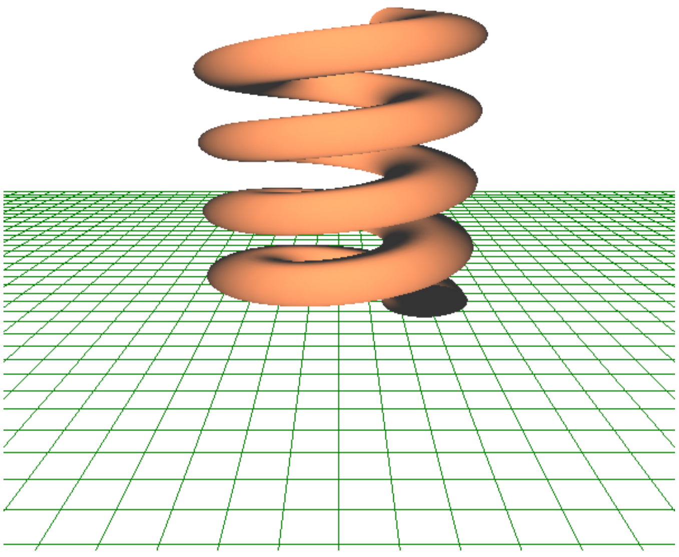
Fig. 4.

II. Shell.cpp

Create another copy of your program and modify the base curve parameters in generateCurve() function and the transformations in the generateModel() function to generate different shapes. For example, when a base curve with parameters a=10, b=10, xshift=10, yshift=10 is transformed using a combination of a rotation matrix and a ***scale*** matrix, we get the model of a shell shown in Fig. 5.

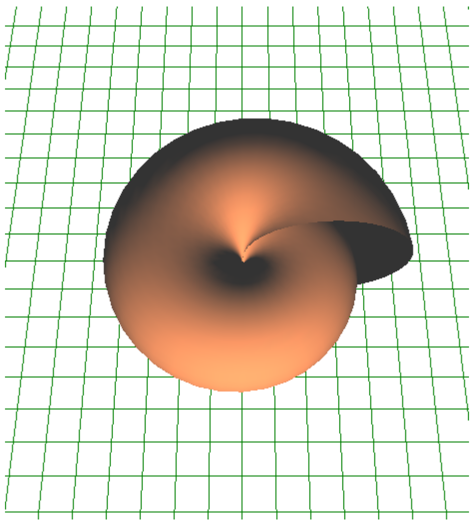
Fig. 5

III. Quiz-06

The quiz will remain open until **5pm, 4-May-2018.**