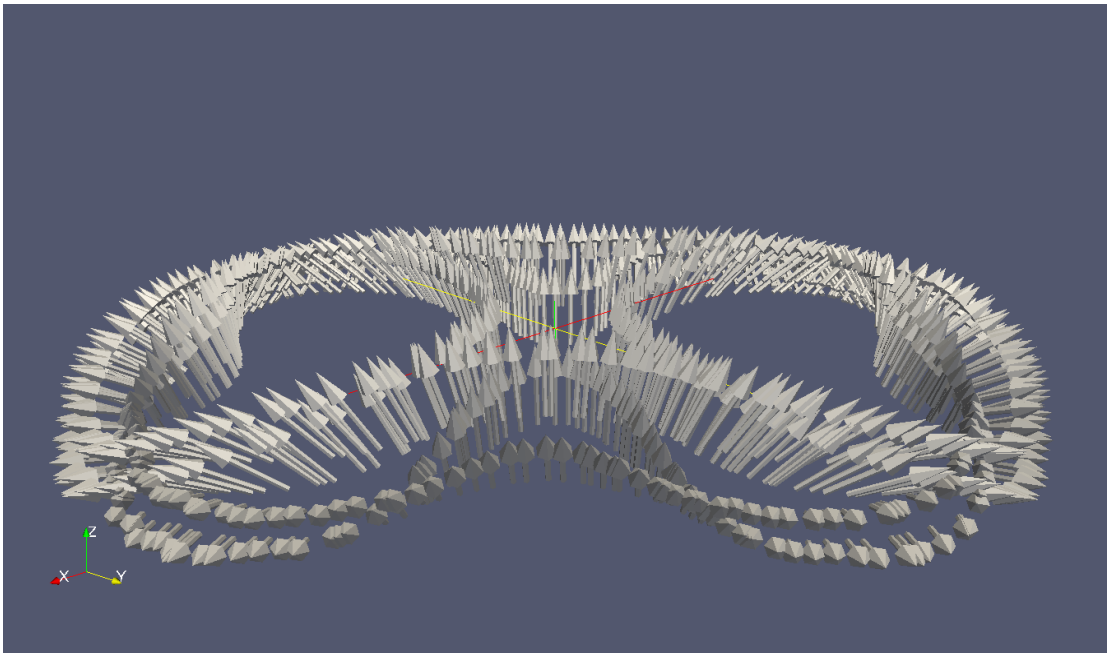
CMSC 634/435 Introduction to Computer Graphics

Project Assignment 1: Vector Field Rendering from Geometry Primitives



Goals of the Project:

Get hands-on experience with OpenGL; learn simple geometry primitives construction through **geometry tessellation** to construct **sweeping plane**.

Introduction:

Representing data graphically often is a great means for data scientists to gain insights into the data. In scientific visualization, three-dimensional data are depicted after feature extraction. In this project, we will draw one type of graphic features called contours. The four contours are derived from quantum physics simulations using data provided by physicists at NIST (National Institute of Standards and Technology.)

In this project, you will draw four contours and associated magnetic field using graphical primitives: cylinders and cones to represent magnetic field strength and orientation. The scene will look like the one above.

Input file:

An input file contains data points and is provided to you (./data/contours.svl). The format is rather simple. The first line of the file defines the total number of contours. Each contour is represented using a set of points. Each point contains six values: the position

(the first three values represent X, Y, and Z coordinates) and the orientation (normalized vector direction Vx, Vy, Vz).

To summarize the data,

1. Line 1: # of contours (here 4.)
2. Line 2: # of vectors on the first contour
3. Line 3 is followed by four sets of values. Each set consists of
    a. Count of number of lines and
    b. Data points with 6 values in each line
4. Each data point has 6 values, of which first three values represent a point followed by next three values representing orientation.
   E.g.: From the input file,

   ```
   -1.07316 -10.5732 3.5 -6.5807e-06 -5.90279e-06 2.49766e-06
   ```

   Here, position = (-1.07316, -10.5732, 3.5) and orientation = (-6.5807e-06, -5.90279e-06, 2.49766e-06)

Requirements:

Computer graphics often deals with 3D scenes. The concepts of Mesh, Rendering, Transformations, Viewport, Projections and camera angle (or view) are important to complete the project. Make sure to learn the concept behind how a 3D object is formed and depicted. The process of displaying the scene is important (also called graphics pipeline). In OpenGL, this is implemented by setting up geometry, camera, rendering window etc. Some of the functions are provided in this project. You only need to worry about geometry calculation.

We use the concept of sweep plane. It's an algorithm that is used to iteratively develop the surfaces. Consider as an example, creation of a cone. The curved surface of the cone can be computed by rotating a line around the central line of the cone. We record the points of these segments and link vertices on these segments to construct the surface.

Implementation

One of the most exciting aspects of this assignment is that the "building-block" nature of the assignments. In other words, you won't just hand in your code and expect never to use it again, but rather you can expect to see the fruits of your labor from this assignment even in the final one (or so we hope!). This of course means that careful planning and a good design are paramount.

You will use OpenGL to draw the vector field. Functions for setting up the view, clearing the color buffer, and camera initialization must be defined and are provided.

There are two methods to draw the arrows in the scene in OpenGL. Here are the pseudo code.

Algorithm 1:

- compute the transformation matrix for a vector given its position and the orientation.
- Use OpenGL matrix routine to push the matrix into the rendering and rendering a vector
- Draw using the OpenGL glutWireCone (or glutSolidCone) and glu cylinder.
- Repeat the first three steps to draw all vectors.

Algorithm 2:

- compute the points on the sweep plane by rotation
- connect these points to form a sweep plane
- Do the first two steps for each cone and cylinder given the start point and the orientation.

You can either use Algorithm 1 or Algorithm 2 (also see extra crdit.)

Some useful functions include:

glTranslatef(x, y, z) – to translate the current point to the point (x, y, z)

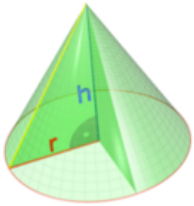
glRotatef(angle, x, y, z) – to rotate the drawing by the angle specified in parameters.

E.g.: glRotated(45, 1.0, 0.0, 0.0) – This function call changes rotates the object by 45 degrees with respect to x-axis.

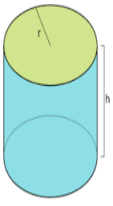
To improve efficiency, consider OpenGL display list.

Shape Specification:

Each data point in the file corresponds to one arrow in the scene. An arrow is the combination of cylinder with cone on top of it. Use your own judgment to size them to minimize occlusion. Here is a reference scale:



h = 0.35, r = 0.1



h = 0.65, r = 0.03

For 634 (graduate students):

Occlusion is an issue in computer graphics. One way to resolving occlusion is to morph the geometry. Here we will implement a single morphing mechanism, i.e., wrap the arrows pointing backwards to point forward.

What we do is to create a cutting plane in the middle of the geometry and rotate all arrows on the back to the front. We will then show side-by-side view of these two parts. An example picture using a human face is shown below.

Strategy:

Incremental development will probably result in the most efficient use of your time. For example, first try to program how to draw a line arrow in the center of the environment, and then implement transforms.

Supporting code:

Supporting code is provided to get you started. It can load the geometry and save to a list. It has also included camera set up and interaction technique using trackball. You only need to draw the geometry in this project.

What to turn in:

Source code should be sent only by email to Kevin Jones (jkevin1@umbc.edu).

- A README file, containing basic information about your design decisions and any known bugs or extra credit. Writing this information is crucial because your project will only be executed using the information contained in this file.
- In the README me file, you should include how to compile and run your code as if you are telling a colleague that is to continue the development.
- Your code must run and be tested on gl.umbc.edu.
- Submit the code as a tar ball (tar.gz) following the naming convension specified in the syllabus. The command to create tar.gz is
  > tar cvzf [yourLastNameFirstName_project1.tar.gz [yourLastNameFirstName-project1 directory]

Project 1 Rubric (the orange part is for Algorithm 1 and purple shading for Algorithm 2 implementation mentioned above.)

Points Possible	Description
25 (435)/10 (634)	Code compiles
5	Code is well documented and contains README file
45	Translation, rotations, and scaling matrix for a vector
25	Depict cylinder and cone
10	Vector translation
30	Rotations and mesh for cone
30	Rotations and mesh for cylinder
15 (634 only)	For creating two halves of the scene
10 (Extra credit)	Color the arrows based on their orientation and explain why the color mapping works or does not work.
10 (Extra credit)	Color the arrows based on their magnitude and explain why it works or does not work. (you can add an arrow to toggle between the orientation and magnitude coloring if both are implemented.

Total: 100. If code does not compile on gl, Score will be 0.