Evenly Spaced Streamline Placement

(December 2019)

Parth Sanghani, Hitisha Damani

Abstract— A paper by Bruno Jobard and Wilfrid Lefer presents a new evenly-spaced streamlines placement algorithm to visualize 2D steady flows. The main technical contribution of this work is to propose a single method to compute a wide variety of flow field images, ranging from texture-like to hand-drawing styles. In this method, the control of the density of the field is very easy since the user only needs to set the separating distance between adjacent streamlines, which is related to the overall density of the image. They show that their method produces images of a quality at least as good as other methods and is computationally less expensive and offers a better control on the rendering process. In this project, we are implementing the proposed algorithm, to generate evenly spaced streamlines in C/C++ using OpenGL libraries like GLUI and GLUT.

1 Introduction

The problem of visualizing vector fields has been widely

A solution was offered by Dr. Wilfrid Lefer and Prof. Bruno Jobard of University of Pau and Pays de l'Adour, France, which provided a new algorithm to visualize 2D steady vector fields. This method can compute variety of flow fields from texture like styles to hand-writing like styles. The proposed method is up to the mark in terms of quality and is computationally less expensive than other methods.

This report aims to explain our project, in which, we have implemented the said algorithm in C/C++. We have developed a program which generates evenly spaced streamlines for any given steady vector field.

2 Related Work

To properly render fine grain details of a vector field, high spatial resolution techniques are required which makes field presentation dense. But there are several situations where sparse field presentation is also required. Methods that visualize a flow field typically fall into two categories: Dense field representations and hand-drawing style.

First method, spot noise texture synthesis, a dense field representation, has been proposed by Prof. Jarke J. van Wijk of Eindhoven University of Technology. This method creates a directional texture by superimposing many flow-oriented ellipses. Each ellipse is generated by projecting a spherical spot onto a surface and by adverting the spot with the direction and magnitude of the vector field at the projection point. This amounts to the flow field-controlled generation of a band-limited noise. Initially straight, the spots are now bent along short streamlines to follow the curvature of the vector field. An important feature of this method is the local control on the generated image. more spots provide more accuracy. Generation time is dependent on total number of spots used while generating the texture. Therefore, we can get a tradeoff between Image Quality and Rendering Time.

Another interesting method is the Line Integral Convolution or LIC which was proposed by Dr. Leith Leedom and Dr. Brian Cabral. A LIC texture is generated by convoluting an input texture with a streamline-oriented one-dimensional filter kernel. The images obtained with this technique are very effective, showing more details than the previous one. But it also requires a lot of computational power.

Prof. Greg Turk from University of Norht Carolina at Chapel Hill and Prof David Banks from Mississippi state University proposed a image guided streamline placement method. It uses a stochastic mechanism to refine the placement of streamlines iteratively.

First, some streamlines are randomly generated. Then at every step of refinement process, a small change is randomly performed. the change can be any combination of three valid operations: (1) changing the position and/or length of a streamline, (2) joining streamlines that are very close to each other, and (3) creating a new streamline to fill a gap. AN energy function is used to measure variation of energy in current and updated images. Modification is only accepted if energy variation is negative.

This method creates very high-quality flow fields, but its convergence is very slow as it sometimes takes several minutes to compute field for a single image. This method will also not work with dense field images because number of possible modifications increases exponentially.

3 Methodology

In order to implement evenly spaced streamlines, we have selected 4 random points on the field from which we generated initial streamlines. Then we are finding two seed-

points at distance d_sep for every point in the streamline. Then we are generating streamlines from the seed-points. If at any point, a streamline comes closer than distance d_test, then the streamline is terminated immediately. This procedure is repeated for every streamline, which finally gives us a field in which all streamlines are evenly spaced from each other.

Algorithm which we are using is as follows:

Compute an initial streamline and put it into the queue
Let this initial streamline be the current streamline
Finished = False
Repeat
  Repeat
    Select a candidate seed-point at d = d_sep apart from the current streamline
  Until *the candidate is valid or there is no more available candidate*
  If *a valid candidate has been selected,* ***Then***
    Compute a new streamline and put it into the queue
  Else
    If *there is no more available streamline in the queue,* ***Then***
      Finished = True
    Else
      Let the next streamline in the queue be the current streamline
    EndIf
  EndIf
Until *Finished=True*

4 Implementation

We have Implemented this project in C/C++. We have used .ply files to load the vector field data. For the GUI and Visualization, we are using OpenGL libraries like GLUI and GLUT.

In order to store seed-points, we are using a struct which contains several float variables like x, y, vx, and vy. We are storing a single streamline in a vector of seed-points. And those streamlines are again stored in a vector. In essence, we are using 2D or nested vectors to store the streamlines and seed-points data.

We are changing the y co-ordinate of current streamline by d_sep distance in order to get new seed-points.

To get the streamlines, we are using Euler's integration method. i.e. going forward one unit at a time in the direction of vector of current pixel. We are retrieving the vector value of a certain pixel, then converting the vector to a unit vector by finding its magnitude and dividing the vector by its magnitude. We then move forward to next pixel by adding the vector value to the current co- ordinates (x = x+ vx and y = y + vy). For every point of the streamline traversed, it is checked against the termination condition in order to decide if the point should be displayed or not.

For our termination condition, we have created a grid with size of resolution of the image (i.e. 512). For every point of a streamline displayed, we are setting a flag in the grid indicating that pixel is used in a streamline. When checking for new point, we traverse though every point which is in d_test (d_test = d_sep/2) radius of the current point to check if it is used in any other streamlines or not. If yes then we terminate the current streamline. Otherwise, we continue to create a streamline unitil (1) its magnitude is lower than a certain threshold (sink point) or (2) streamline crosses the image boundaries.

In the project, Stream-line tracing algorithm was implemented by Parth Sanghani and Termination condition as well as UI was handled by Hitisha Damani

5 Results and Discussions

Streamlines displayed are evenly spaced. The algorithm works with different types of flow fields.

We can change value of d_sep which controls the distance between each streamline.

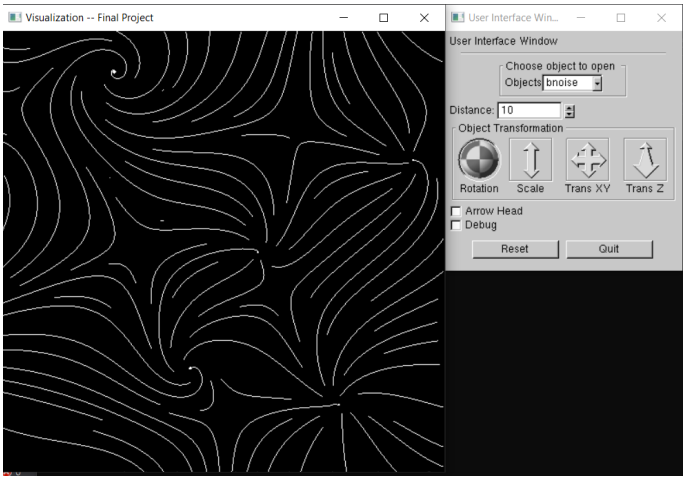

Figure 1: BNoise - d_sep = 10

Figure 2: Bruno3 - d_sep = 10

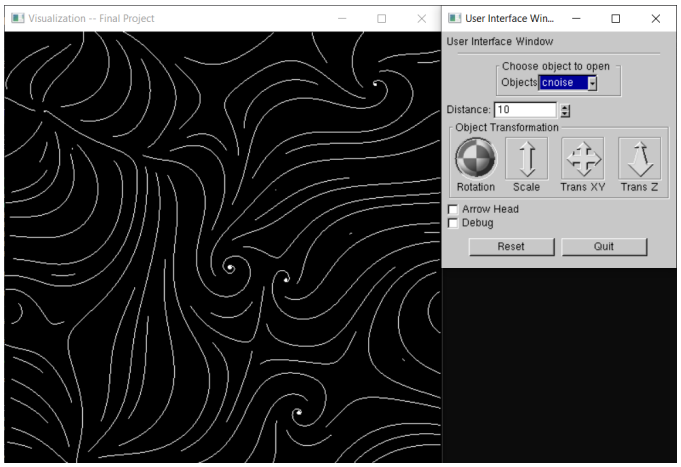

Figure 3: CNoise - d_sep = 10

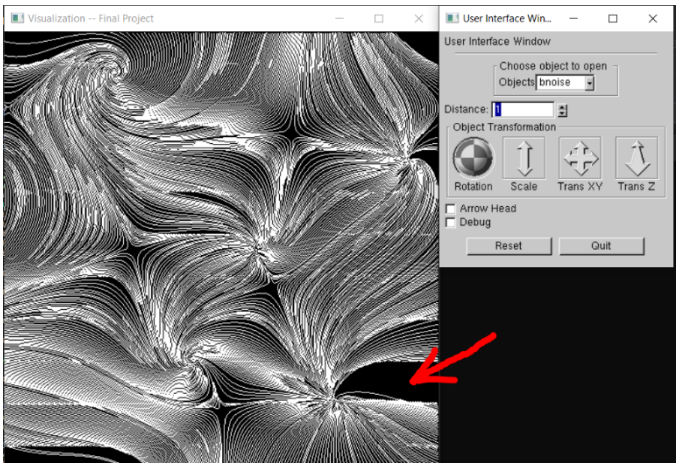

Figure 5: BNoise - d_sep = 1

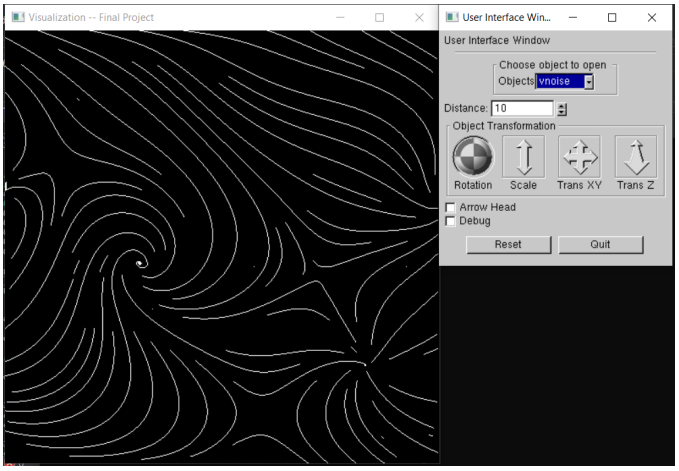

Figure 4: VNoise - d_Sep = 10

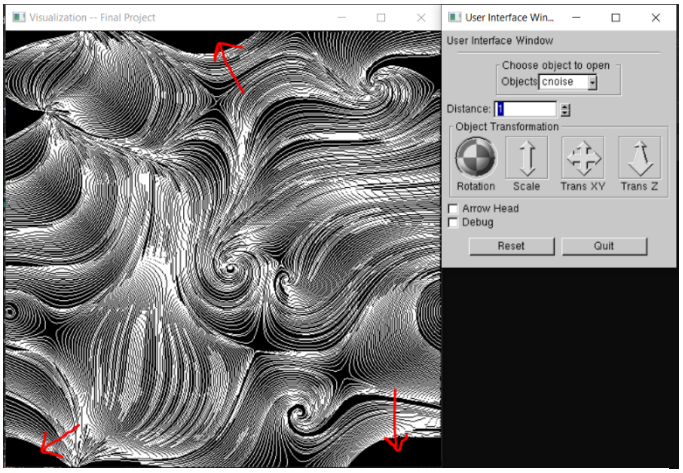

Figure 6: CNoise - d_sep = 1

In the images, we can see different type of vector fields, visuzalized using the said algorithm. The d_sep distance is set to 10. 10 is number of pixels on a 512x512 grid. Therefore, Stream-line seedpoints will be generated at every 10 pixels.

Termination condition uses distance d_test which is 0.5*d_sep (i.e. half of d_sep). If a streamline is within d_test distance of another streamline, it is terminated there.

As we can see, there is a slight problem in the implementetaion. Not every stream-lines are bring displayed. There are certain "Blind-Spots" where seedpoints are not being traced. We can see it more easily, if we decrease the distance from to 10 to 1.

From the images, we can say that there are some spots on which, streamlines are not being traced. This problem will be solved in the future iterations of this project.

Other than that, we have used euler integration method, which sometimes give jagged streamlines. Dipole

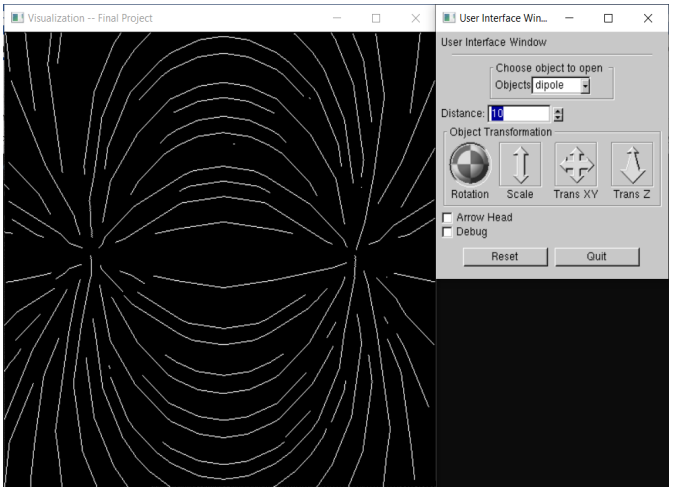

Figure 4: Dipole - d_sep = 10

Flow

The effect is more prominent if we decrease the d_sep from 10 to 5.

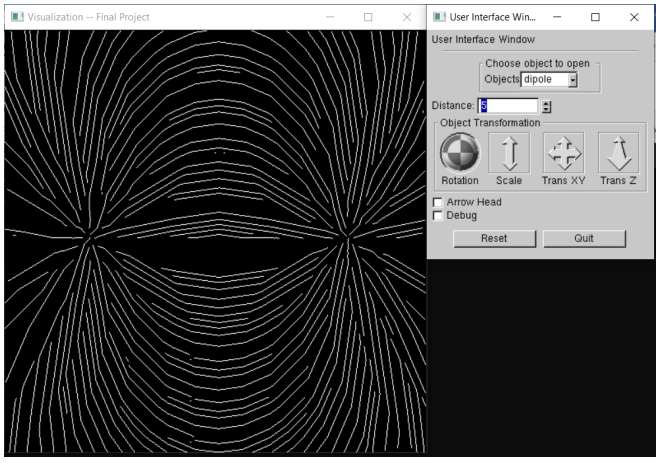

Figure 5: Dipole - d_sep = 5

We can reduce the jaggedness by using other methods instead of Euler's integration like n-degree RK Integration.

6 Conclusion and Future Work

6.1 Conclusion

After implementing the algorithm for evenly spaced streamlines, we can conclude that, it does not require as much computing power as some other methods, albeit this method has its own drawbacks, but if we need to visualize flow field faster and with relatively higher quality, this method is to be used.

6.2 Future Work

There are several 'bugs' in this implementation which we can solve in future, like we can implement RK Integration instead of Euler's method and change the code in order to reduce the blind spots I nthe vector field.

The paper from which we implemented the algorithm for evenly spaced streamlines, also outlines some other methods and other visualization techniques which can be used for vector fields as well like tapering effect and glyph mapping.

We can also extend this implementation to use generate textures using LIC or Enhanced LIC.

References

[1] Documentation of OpenGL , GLUI and GLUT at https://www.khronos.org/opengl/wiki/Primitive

[2] Jobard, Bruno & Lefer, Wilfrid. (1997). Creating Evenly-Spaced Streamlines of Arbitrary Density. Proceedings of the eight Eurographics Workshop on visualization in scientific computing. 7. 10.1007/978-3-7091-6876-9_5.