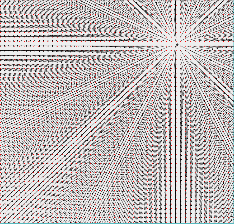
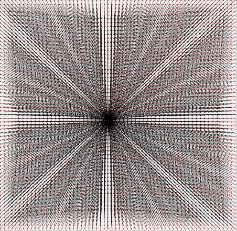
**Abstract:**

The objective for this project was to apply the information presented by creating a prototype of the original project but re-implemented on a different language and framework that replicated the visual animation. By creating a prototype and re-exploring the fundamentals of previously established research, models are drawn from the stage of development. This degree of analysis grants the opportunity to study and draw new ideas on top of the previous research to push and branch out the groundwork further. A comparison can be performed on vector field visualization to other visualization techniques in terms of implementation and effectiveness. With a broader understanding of vector fields, different levels of understanding can be worked to use these visualization techniques and models to its full effectiveness.

**Introduction**

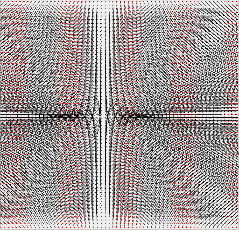
With the implementation of a vector field, additional visual features can be incorporated such a integrating streamlines and creating textures over the field that simulate flow and the interactions between the elements. Despite vector field’s inability to account for increasing complexity of data, the simplicity of vector visualization allows an opportunity to continuously develop and analyze the data through an interpreted design.

**Tool Exploration**

Working with and understanding the original design allowed me to map out all concepts of what the research wanted to represent. The expansive functionality dealt with multiple fields of visual designs. The code also allowed me to model my own code. The data structuring and program design allowed me to grasp how my model would function.

**Vector Field Generation**

Generating the vector field started with an understanding of mesh generation. The program needed to have flexibility with how the user wants to represent their visualization and so the mesh was created starting with user-defined input of sample points that determine the graph columns and rows. Through each sample point, the vertices of clockwise and counter-clockwise triangles could be generated and stored. This would be the starting data structure and what we would base our vector field representation on. *The clockwise triangle is generated by vertices {i, i+1, i+(Nx+1)} for any i: 0 ≤ i ≥ Ny-1. The clockwise triangle is generated by vertices {i, i+(NX+1), i+(Nx)}. After triangulation we can start storing all the triangles in a list which gives each triangle an index to retrieve from.* We would then give each triangle and id and store it in a list for retrieval later. Each triangle and its vertices will be used to store important information such as vector values.

To recreate vector behavior with singular elements, Basis Field Summation is applied which deals with defined jacobian matrix that determines the type for each singularity. The pattern types we can produce are source, sink, saddle, clockwise center, and counter clock-wise center. Each singularity will store a jacobian matrix that can be manipulated through user interface to change its orientation and scale.

**Figure 1**: Different design elements showing source (left), sink (middle), saddle (right).

Regular elements that converge and diverge also follow their own formula. They follow the same scheme and structure of singular elements but instead of stored matrices, regular elements change their interaction depending on the value of their determined constant value.

|  |  |
| --- | --- |
| Degree1Degree2 | Original Formula with assumed angle 0 degrees.  Updated formula with user input theta. |

**Figure 2**: Mathematical formula for regular elements.

Storage of all elements will be in each of their respective list so rendering all the elements would simply require iterating those lists and plotting the elements during each display. The interaction between the each of the elements are determined through weight values that are each calculated differently for their respective element. The base weight equation is *weight=C\*e^(-k\*distance\*distance)* where C is a constant determining element span and K is a constant determining element sharpness. Vector values are the summed values of all the elements and so the finalized value will visually present the merging of multiple elements.

**Rotation and Scaling of elements**

Singular elements had the functionality of rotation and scaling by adding transformation matrices into their calculations which creates a new Jacobian matrix with manipulated values. These transformation include the rotation matrix and the scale matrix. An updated formula was used for regular elements for rotation which incorporated a user designed angle value.

User interface was also implemented to rotation to user specification. This was done by either having rotatable boxes around elements. Additionally, users could also select elements with a selection mode to modify weight values and their scaling.

**Choosing our Platform**

Java is a common platform for running on many hardware types. Java applications also have the ability to run on systems given that they have a JVM. The original flow based program which was reference to begin this project was written in C++. However, the application was unable to work on a different operating system than the initial OS that it was developed in until some tinkering was done. The MFC libraries within the C++ programs faced inconsistences when developing on a different type of supported OS. Changes in the Windows header of standard system include files were done to run the application and would need to continuous be done when switching operating systems.

We chose Java OpenGL for our visual platform since it was easier to reference and work off the existing use of the OpenGL’s libraries.

A given the task for this project was translating an existing C program that sampled the flow visualization texturing using OpenGL. However, due to my inexperience and the differences between JOGL and OpenGL, I could not replicate the same visualization behavior in my prototype using JOGL. A static texture image would be generated and stored onto the graphics card using OpenGL’s displaylists and then displayed over. A simpler texturing design was turned to which was Line Integral Convolution (LIC). LIC is CPU based rather than working on top of the graphics card. It dealt more with manipulating images which was simpler to implement.

JOGL drawing capabilities are handled by embedding the drawing canvas onto a JPanel. The drawing domain of JOGL’s canvas is different from how swing graphics is since the swing graphics draw on a pixel-based graph and its domain is dependent on the panel’s size. JOGL represents their drawing domain in a floating point scale which uses world coordinates rather than window (pixel) coordinates. This difference changes the needed sizes of the weight values of the elements and the interaction that was observed between the elements. However, since Java’s mouselistener receives windows coordinates, those given window coordinates had to be converted to world coordinates.

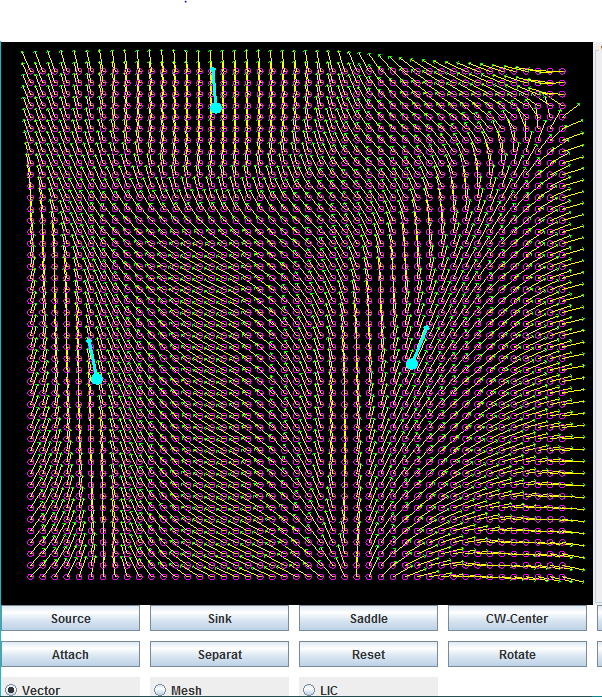
**float** **x**= 2.0f\* mouse.getX( / canvas.width -1.0f;

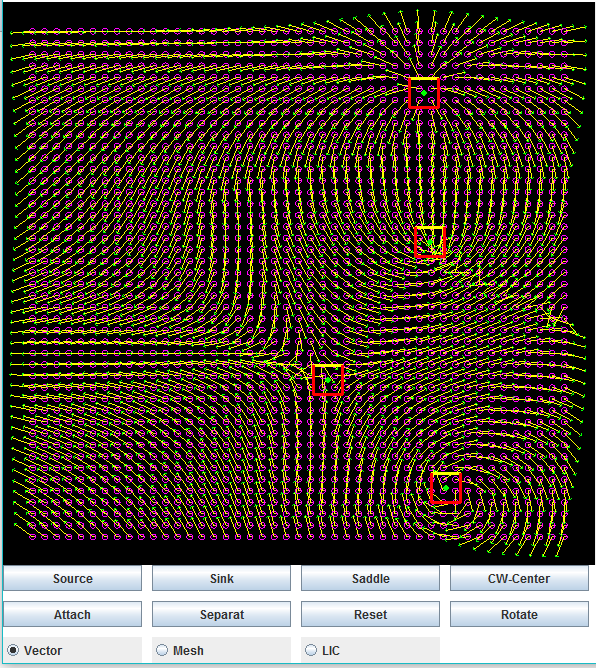
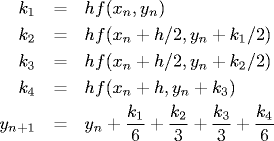
**float** **y**=-2.0f\*mouse.getY(/canvas.height + 1.0f;

**Figure 3**: Pseudocode for converting window (pixel) coordinates to world coordinates in OpenGL.

**Visualization**

Representation of the flow of the field is delivered through vector arrows, streamlines, and line integral convolution textures. When drawing the arrows, OpenGL can render the arrows using starting and ending position. The program iterates through all sample points and uses the location of those sample points to plot the vector arrows. However, to reduce visual clutter all vectors are normalized by a specific length value to evenly create a visual field representation.

**Figure 4**: Singular element interaction (left) with rotation boxes and Regular element interaction (right).

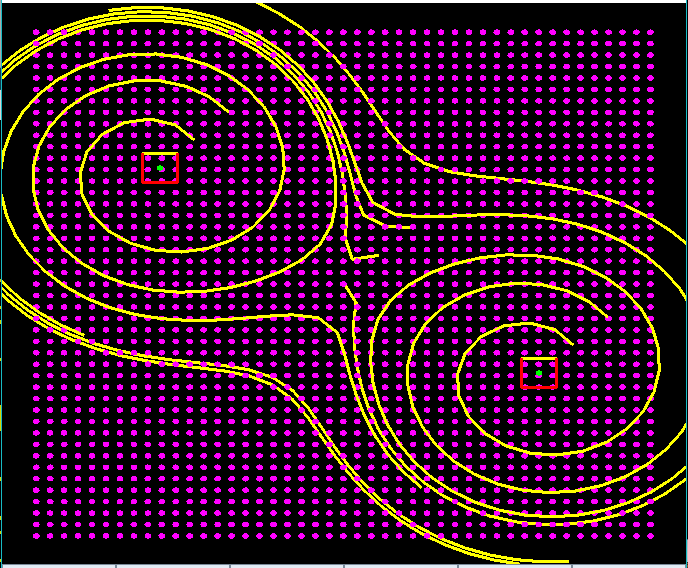
Streamlines are the lines tangent to the vectors at all points. To start calculating streamlines, there needs to be a way to integrate the differential function of the field. Vector values will need to be calculated at any given point instead of just at the values stored at the sample points. Since the vector values are stored only at the vertices of the triangle mesh, barycentric interpolation will be used which uses the relativeness of the coordinates to each vertices to calculate the vector value. We start with Euler integration for an example representation and then apply Runge-Kutta integration for precise streamline values.

|  |
| --- |
| v+u+w=1  vx = u\*vx1+v\*vx2+w\*vx3  vy = u\*vy1+v\*vy2+w\*vy3 |

1. (b)

**Figure 5**: (a) RK4 Integration used to draw streamlines and (b) Barycentric coordinates to find relative vector values.

Streamlines are also drawn onto the field using the vector values of each point. At a determined point around the center of an element we perform RK4 integration to create an accurate estimate of a slope for the field and then take a step and repeat till our streamline is finished drawing. LIC allows to better visually represent the flow of the field and it is down by creating a white noise and convoluting it with a streamline. This convolution is done by recording the pixel values across the streamline, summing up each of their color values and dividing the total by the number of pixels. The final color should be gray and show a contrast between the pixels located around the streamline. The final LIC image should be stored in a 2D array so that it can be used by OpenGL to paste the image on the field.



**Figure 6**: Representing streamline using Euler (left) and streamlines convoluted with white noise to show LIC (right).

**Conclusion**

Incorporating and applying pre-established visualization techniques presented an extraordinary learning in the field of computer graphics. The prototype developed was made to implement an existing design for vector fields and examine the concepts and fundamentals of flow visualization. Although many of these visualization topics have been explored, this project served to reiterate and identify areas of the development which can be optimized. Getting started with this research project required thorough reading of research papers which allowed an opportunity to be familiarized with the mathematical analysis of vector fields and the way they modeled different effects.