Fluid Flow Visualization using Line Integral Convolution

By Kalin Johnson

For my Computer Graphics final project, I visualized a fluid flow using line integral convolution and other mathematical techniques. My original goal of the project was to display the fluid flow data coming from one of the senior research students in Dr. Stein's Lab. Due to some complications with the data, I was not able to complete this part of the project. Still, I was able to visualize a simpler velocity field using mathematical and graphics techniques. The first velocity field I pictured was a circular pattern that swirled around the center of the image. For this project, I worked off of a paper titled "Fast and Resolution Independent Line Integral Convolution" by Detlev Stalling and Hans-Christian Hege.

Now, I am going to go back to the process I went through with the original data and explain some of the problems I ran into. The data was in the form of a numerical vector field. For each location in the field, I had the x coordinate, y coordinate, x velocity, y velocity, pressure, and vorticity. The last two pieces of data were not used in my project, but they could supply fascinating additional insight if the project continued. I first tried to read this in data using fs, which is a Node package, but I quickly learned that CORS blocked me since I was running my code on the browser. Instead, I copied and pasted it into a separate Typescript file that is read at the beginning of the program. The velocity field was also created based on a mesh, so the points I supplied were not in neat squares. This led to difficulties matching up points at several places throughout my project and was the final reason I had to take a different direction with the project. The vector field I had was not at a high enough resolution that I could get away

with simply taking the closest point, and I did not have the time to build a method that took the weighted average of all the points around the desired location.

Using the velocity vector field, I used the Runge Kutta technique to calculate the streamline that went through each pixel in my final image. This was one of the parts of the project that I struggled with the most. The math was confusing both to understand and to implement. I used a variety of sources to try and accomplish this, such as Numerical Recipes and code supplied to me by Dr. Gossett. The first method I tried from the Numerical Recipes in C book did not have enough information outside of the code to understand what I needed to supply to the function. The second method, using Dr. Gossett's C++ code, worked well, and after some debugging on the side of the velocity field, it did the job I expected it to do. I also had to mess around with the tolerance that I inputted. The lower the tolerance, the more black dots are in the final image. The higher the tolerance, the fewer black dots are in the image, and the more of a clover effect occurs. At the end of this method, I had two arrays containing vector field locations of the x and y components of streamline points.

After I got the streamline points, I looped through all the points for the pixel and skipped anything with a value of -1. I then connected those valid dots with a straight line and sampled each pixel along that line. To do this, I started by defining two points and calculating the distance between the two points. The distance between the two points told me how many pixels I would sample. Next, I calculated the fractional space between each pixel in the x and y direction. These numbers are then rounded to get an array of x and y coordinates I could access from my array of white noise. Finally, I found the average of each streamline segment and the average of the entire streamline for each pixel. I tried to use the parametric representation of a

line to complete this step, but that caused me some problems, and this method made more sense to me. I had to mess around with the indices in this part of the project quite a bit. One of the lessons I learned during this project is that finals week is very busy, and I didn't have any time to work on this early in the week. I also learned that math is hard, especially when a computer is reading your math. In the end, I got a result using the techniques I set out to use in this project, even though the velocity vector field did not go as planned.

References:

Cabral, B., & Leedom, L. (Casey). (2023). Imaging vector fields using line integral convolution. *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 375–382. https://doi.org/10.1145/3596711.3596752

Stalling, D., & Hege, H.-C. (1995). Fast and resolution independent line integral convolution. *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '95*. https://doi.org/10.1145/218380.218448