# LIC, UNIK4660 Obligatory Assignment

Joseph Knutson

April 2, 2018

# Introduction

The aim of this assignment is to present a geometric and texture based visualization method for visualizing vector field data. The geometric method is simply the visualization of streamlines. The streamlines are produced from numerical integration of the velocity field. The texture based method used is called LIC, proposed by Brian Cabral and Leith Leedom[1]. Finally, we will compare the resulting imagery.

# Data

To start off with, the 2 vector field data files we're looking at has different, but quadratic resolutions. The set called "metsim1" is 127 by 127, while the one called "isabel" is 500 by 500. In order to increase the resolution, I've implemented a simple interpolator which increases the resolution by any order of 2.

The datasets are in a vector form, stored in the HDF5 format. In order to extract the vector-field data, we need to use the HDF5 library, written in, but perhaps not exclusively for, PYTHON and C/C++. For this assignment I'll be using C++ for the integration and texture visualization and PYTHON(2.7) for the geometric visualization.

# Geometric visualization

The first half of the assignment is to extract the data from the HDF5 format, implement the integrators and choosing how to seed the streamline's initial position.

## HDF5

Below is a function that extracts the vector field data to the 1D vectors "rdata" and "rdata2":

```
1  void
2  init(hid_t file, hid_t dset,hid_t dset2, herr_t status,herr_t
       status2,double* rdata,double* rdata2){
3
4        file = H5Fopen (FILE, H5F_ACC_RDONLY, H5P_DEFAULT);
5        dset = H5Dopen (file, DATASET, H5P_DEFAULT);
6        dset2 = H5Dopen (file, DATASET2, H5P_DEFAULT);
7
8        status = H5Dread (dset, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL,
       H5P_DEFAULT,rdata);
9        status2 = H5Dread (dset2, H5T_NATIVE_DOUBLE, H5S_ALL, H5S_ALL
       , H5P_DEFAULT,rdata2);
10       }
```

[1]Cabral, Brian; Leedom, Leith Casey (August 2–6, 1993). "Imaging Vector Fields Using Line Integral Convolution"

This function takes two arrays, rdata and rdata2, and fills the arrays with the vector-field data. First the function opens the file whose name is stored as a string in the "FILE" variable. It then splits the dataset into its x and y sub-datasets seperately. These datasets are then "read", or rather inserted, into the rdata arrays thanks to the H5Dread method from the HDF5 library.

## Integration and Seeding

I've implemented both the forward Euler and 4th order Runge Kutta method. The timestep seems to be optimal around the value 1, causing the streamlines to jump to its neighbouring pixel. This might be due to the velocity normalization. The Euler and RK4 method seem to not give different results visualization-wise. I believe this is due to the extreme discreteness of our data/grid. A 3 times speed up is therefore achieved when keeping to the Euler method.

I've only tried 2 types of seeding, random and uniform. With random, I mean that the initial position of the streamlines have been chosen randomly throughout the grid. The uniform technique consisted of homogenously planting the lines' initial positions across the grid. In figure 1 and 2 you can observe both seeding strategies in effect.

The homogenously distributed lines are evenly dispersed, while the random seeding has chunks of lines overlapping. My favorite is the uniform/homogenous one, but it too has problems detailing the center and corners of the field.

Seeding isn't the only thing we can tweak, we can also experiment with line length. Figures 3, 4 and 5 show how the images can vary when the line length goes from 1/10th of the image's length to 4/10. To keep things fresh, these are from the Isabel dataset.

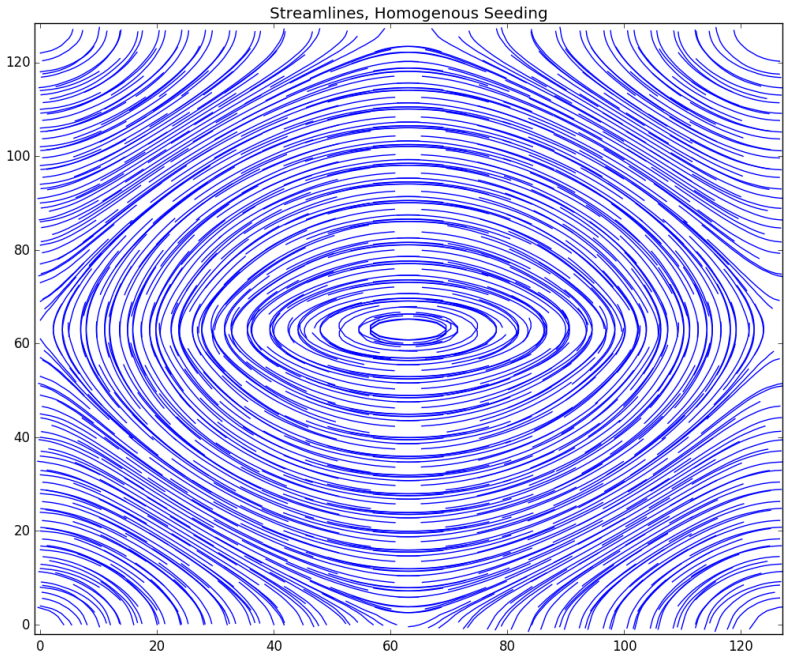


Figure 1: 1000 streamlines from the Metsim set. The seeding is homogenously distributed through the vectorfield.

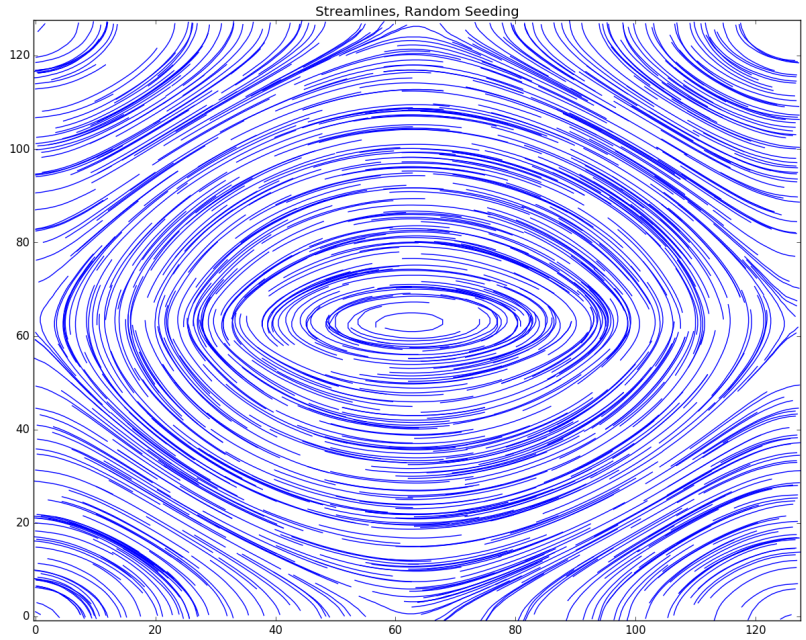


Figure 2: This time the seeding is randomly distributed through the vectorfield.

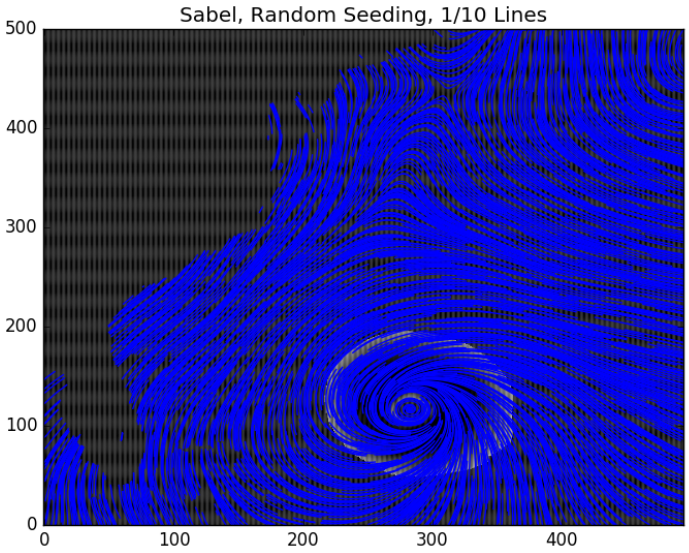


Figure 3: The short lines' cutoffs are visible and not esthetically pleasing.

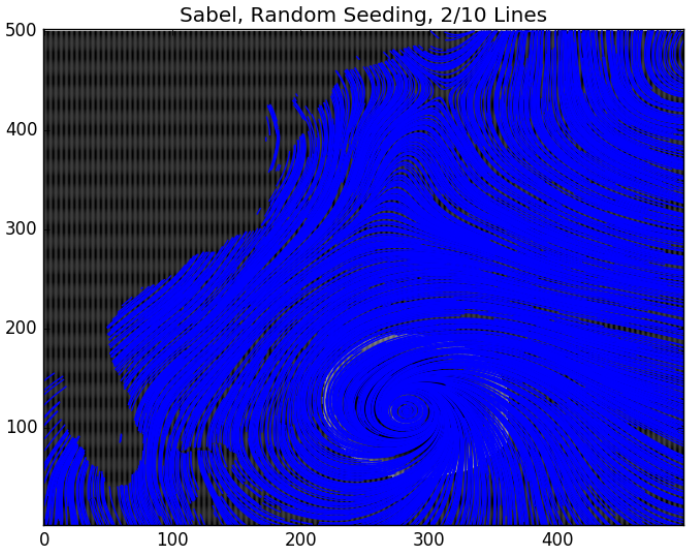


Figure 4: The medium length lines give a smoother feel, but the lines lie rather dense together.

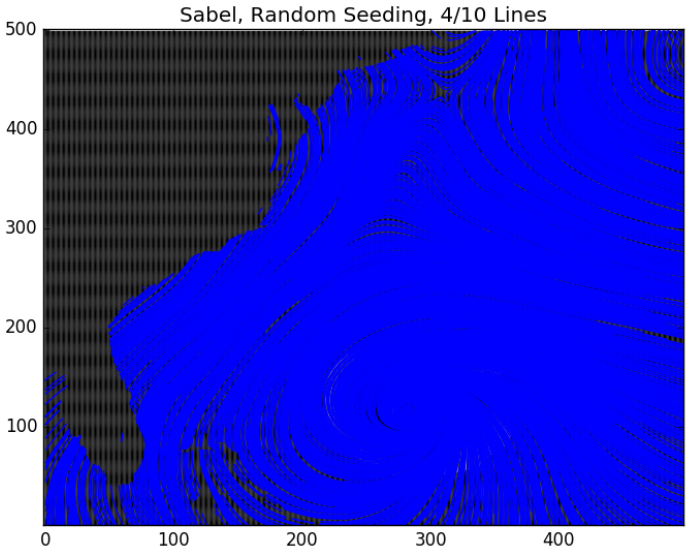


Figure 5: Blue, blue everywhere.

# LIC

The implementation of the LIC algorithm went something like this:

- Superimpose your vector field over a noise filter
- Create streamlines, one for each pixel, more if your vectorfield grid is denser than your filter.
- Convolve the streamlines. Do so along the noisefilter so that the weighted average intensity is produced. This is now that pixel's grey-image-value. Do this multiple times for pixels that initiate more than one streamline and take the average.

My implementation was quite simple. I created a grey image noise filter with values between 0 and 255. Its dimensions are always the same as the vectorfield and the vectorfield is interpolated to any degree I wish. Changing the dimensions of the filter is unecessary as it has no effect on the operations.

Below are LIC images of both datasets with increasing line lengths. You can see smoothening of the image the longer the lines, but as the image gets smoother, it also gets harder to distinguish field lines and extract information. In contrast, shorter lines will sometimes be too short to visualize a continuing line, ruining the pattern of the field.

The increase in line length seems to cause a linear increase in processing time. Setting the line length equal to 10% of the image width seems to make the most prominent pattern.

# Conclusion

When done right, LIC can beautifully render 2D veolocity fields in a much more fashionable way than streamlines alone. However, if you choose to visualize with streamlines alone, pure random seeding will cause ugly areas of overlapping lines (or voids). Uniform seeding might remove the problem with overlapping lines, but due to topological complications, certain areas will become underrepresented.

Euler and RK show no noticeable differences in the images, due to our low image resolution. The timestep's optimal value being 1 is likely due to the vector field normalization in the integration scheme.

When visualizing geometrically, too short lines will give an ugly and tacky image, while a textured image would simply be chaotically patternless. Long lines and random seeding seem to cause streamlines to overlap to the extent that entire parts of the picture are single colored chunks. While in LIC's case, the dynamics and directional sense of the image is lost due to long lines causing a bland, contrastless smoothness in the field. This is perhaps only visible in figure 11.

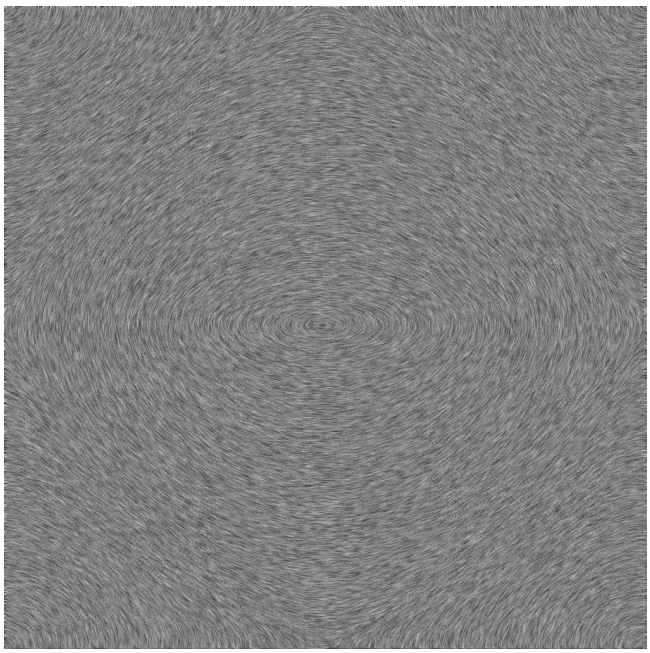

Figure 6: Textured LIC image from streamlines whose length is 5% of the image width or height. Quite tacky

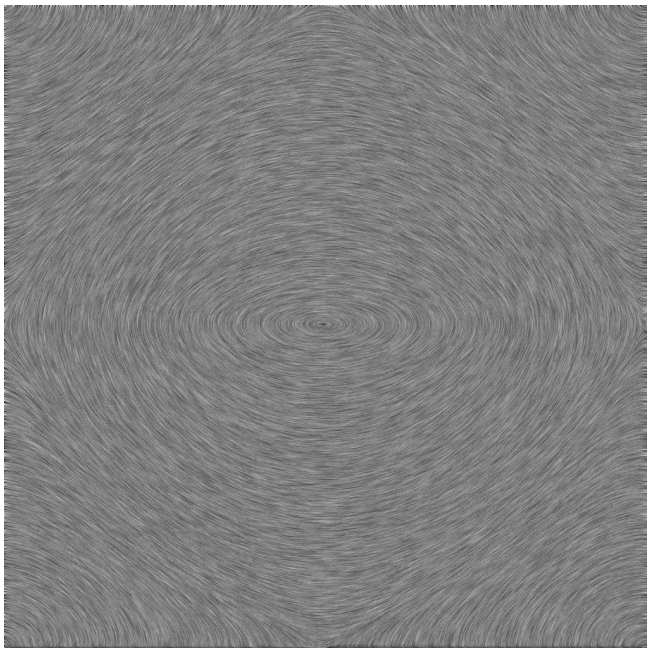

Figure 7: Medium Lines (10% of the image width)

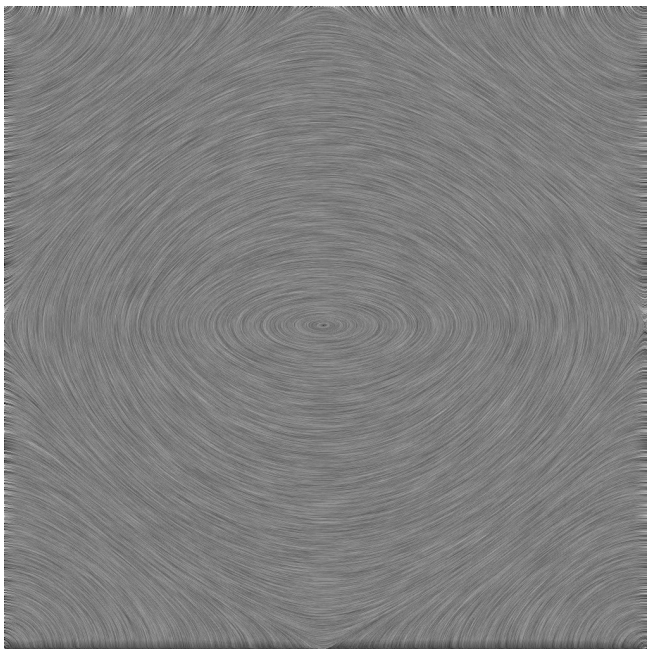

Figure 8: Long lines, with the length of 20% the image width seems an apt length for this resolution.

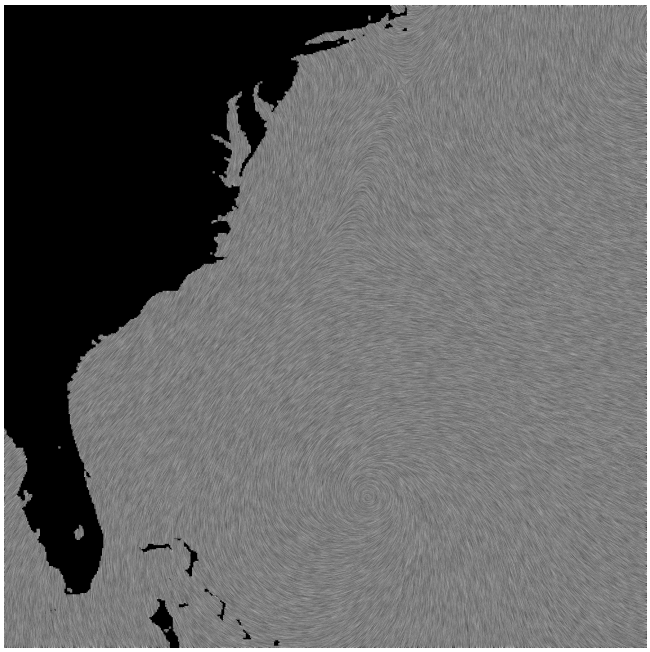

Figure 9: Isabel set, same gimmick, 5%

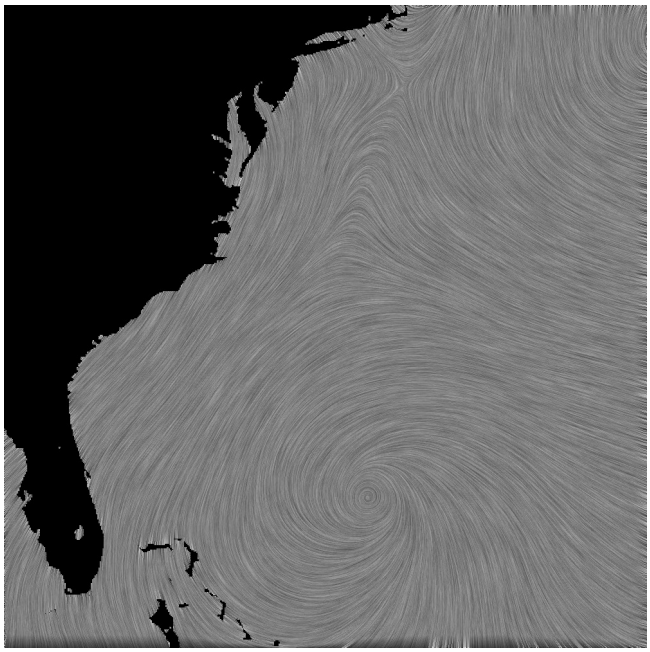

Figure 10: Medium Lines (10% of the image width)

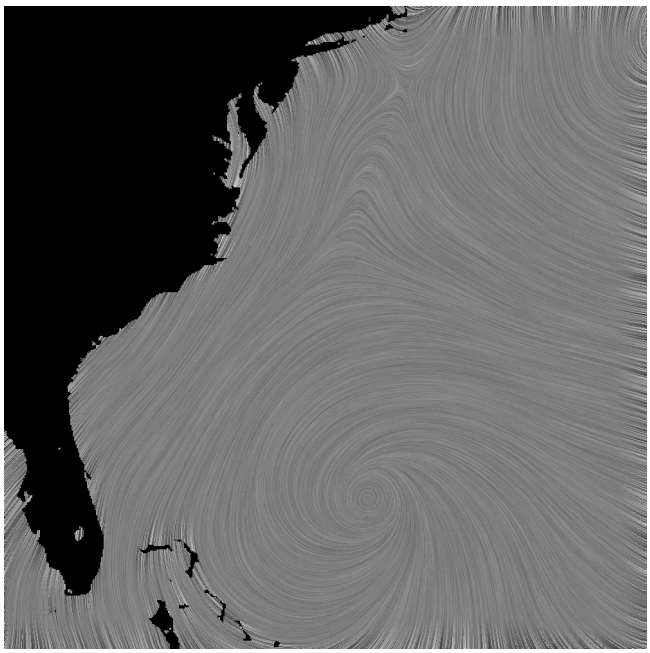

Figure 11: Long lines (20%)