UNIK4660 - Assignment 2

Wilhelm Karlsen and Magnus Elden

April 29, 2016

In this assignment we are first to visualize two sets of vector fields by their field lines. Secondly, to create a Line Integral Convolution(LIC) implementation and show its functionality on these fields. The vector field are given in the form of two .hdf5 files and are two different hurricane simulations, Isabel and Metsim.
We are also to implement and compare different integration methods, specifically Forward Euler and Fourth Order Runge-Kutta, and use differently sized convolution kernels. The results are shown and discussed at the end of this report.

1 Field Line Visualization

Visualization of field lines are intended to represent the vector field as numerous lines conforming to its orientation. This method is very dependent on the points, called seed points, from which the streamlines are calculated. Passing over the vector field and using everything for streamline would simply give a chaotic and unusable image. Seed point strategies are therefore employed to find the best start point for each streamline.
We have chosen to test evenly spaced seed points, as well as randomly generated points with no restrictions.

2 Line Integral Convolution

LIC is a visualization technique used to show the motion and flow in fluids. Using a simple gray-scale noise image it generates a texture by taking every pixel in the image and running it through a convolution kernel. The intended effect is lines that follow the vector field, and gives good, intuitive understanding of the flow direction. The lines are kept distinct by the varying gray scales, though they can also be given different colors to help

show additional information, such as the vector magnitude.
Given a field line σ, the operation can be described by

$$I(x_0) = \int_{S_0+L}^{S_0-L} k(s - s_0)T(\sigma(s))ds$$

Where $I(x_0)$ is the intensity of a pixel at $x_0 = \sigma(s_0)$. k is the convolution kernel with length $2L$, and s is the arc-length.

2.1 Fast LIC

Our LIC implementation uses the the "Fast LIC" algorithm presented in the lecture. With a constant box function as the filter kernel, the integration of the streamline origin is:

$$I(x_0) = k \int_{i=-n}^{n} T(x_i)$$

This allows the convolution of the streamline in both directions as we pass over the input image by the simple function below:

$$I(x_{m+1}) = I(x_m) + k[T(x_{m+1+n}) - T(x_{m-n})], m = 0, 1, ..., M$$
$$I(x_{m-1}) = I(x_m) + k[T(x_{m-1-n}) - T(x_{m+n})], m = 0, 1, ..., M$$

This should help speed up the execution. However, as we did not implement normal LIC, we do not have anything to compare it to.

3 Implementation

For our implementation, both geometric field lines and LIC, we have chosen to use Simple DirectMedia Layer(SDL) library for the graphical representation. The library have allowed us to quickly see the effect of any changes we have done in the code.
During the integration we handle critical points by simply ignoring the point if it is used as the origin. If the field line integration reaches one, it stops there. Interpolation uses it as any other vector in the field.
Should the integration try to pass beyond the boundaries of the vector field, we stop the integration in that direction. In the LIC implementation this has the effect of cutting short the convolution in both directions since we are using the Fast LIC algorithm.

3.1 Forward Euler integration

The forward Euler integration scheme is the simplest of the two schemes.

$$\psi_{n+1} = \psi_n + hf(x_n, y_n)$$

With ψ_n as the current coordinates for the output image, h as the step size and f as the vector field function.

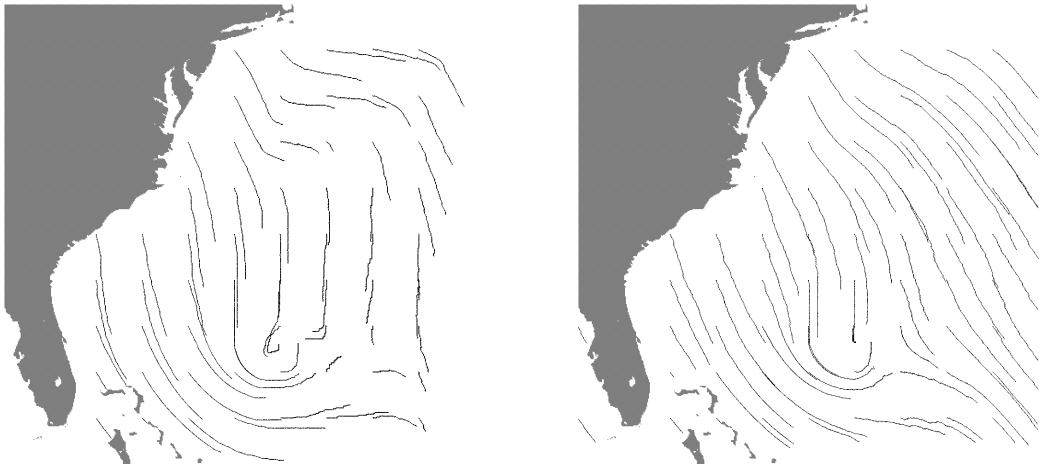
3.2 Fourth Order Runge-Kutta integration

Runge-Kutta is a bit more complicated, and where Euler takes the step in one go, RK divides it into four parts.

$$\begin{aligned}
k_1 &= f(x_n, y_n)h \\
k_2 &= f(x_n + \frac{k_1}{2}, y_n + \frac{k_1}{2}) \\
k_3 &= f(x_n + \frac{k_2}{2}, y_n + \frac{k_2}{2}) \\
k_4 &= f(x_n + k_3, y_n + k_3 \\
\psi_{n+1} &= \psi_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}
\end{aligned}$$

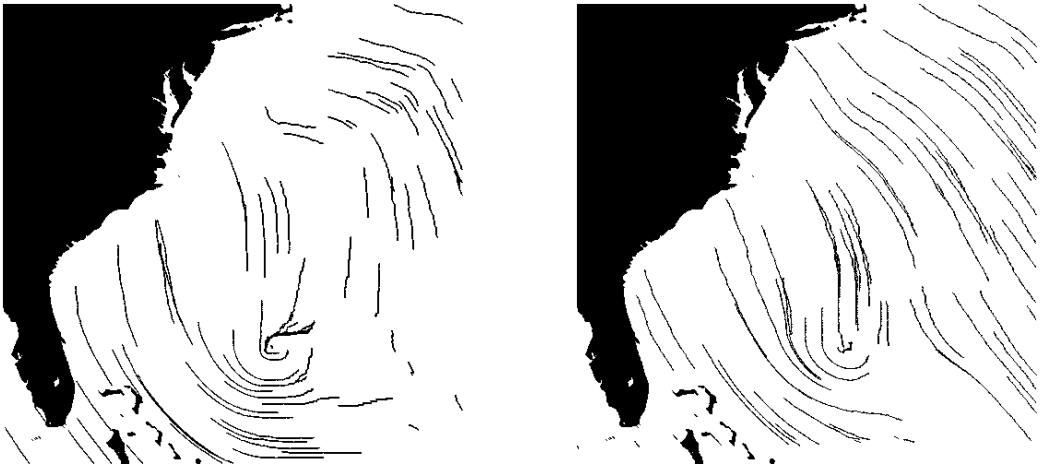
4 Results

4.1 Streamlines



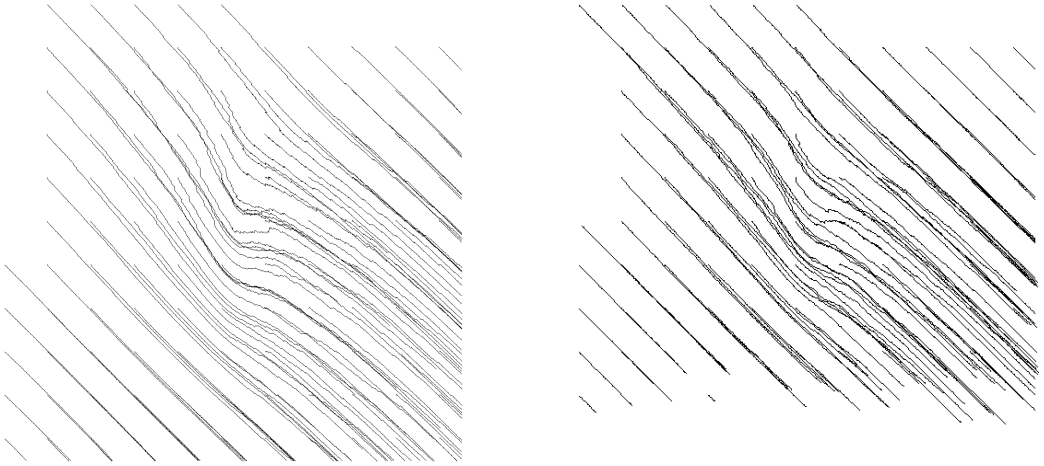
(a) Euler (b) Runge-Kutta

Figure 1: Isabel: Seedpoints evenly dispersed as a 10x10 grid.



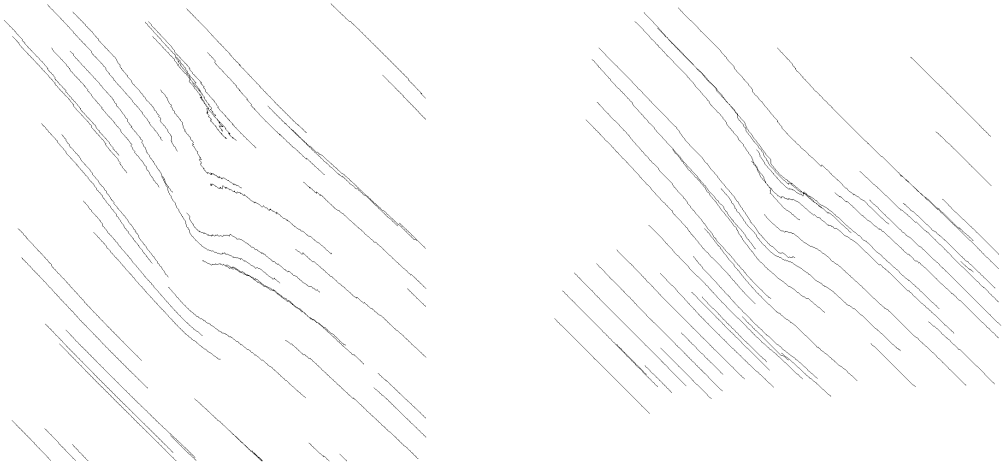
(a) Euler (b) Runge-Kutta

Figure 2: Isabel: 100 randomly generated seedpoints



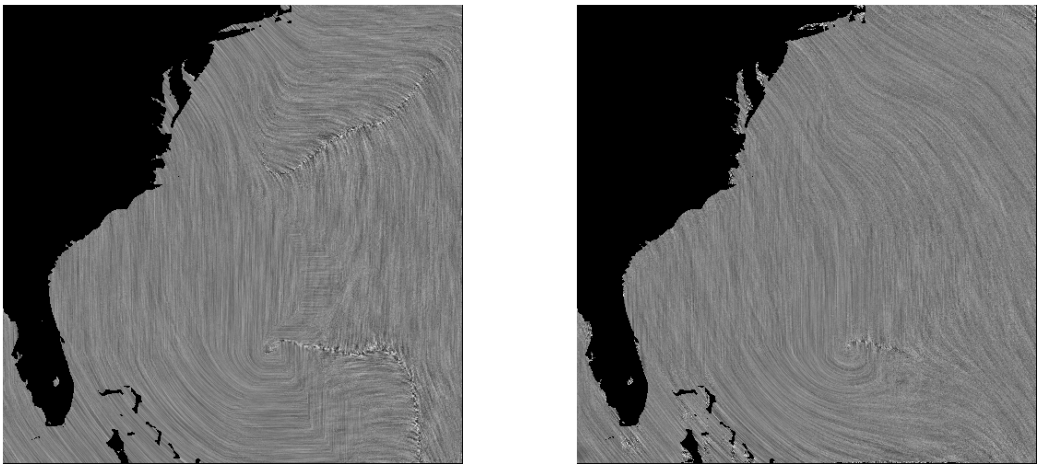
(a) Euler (b) Runge-Kutta

Figure 3: Metsim: Seedpoints evenly dispersed as a 10x10 grid.



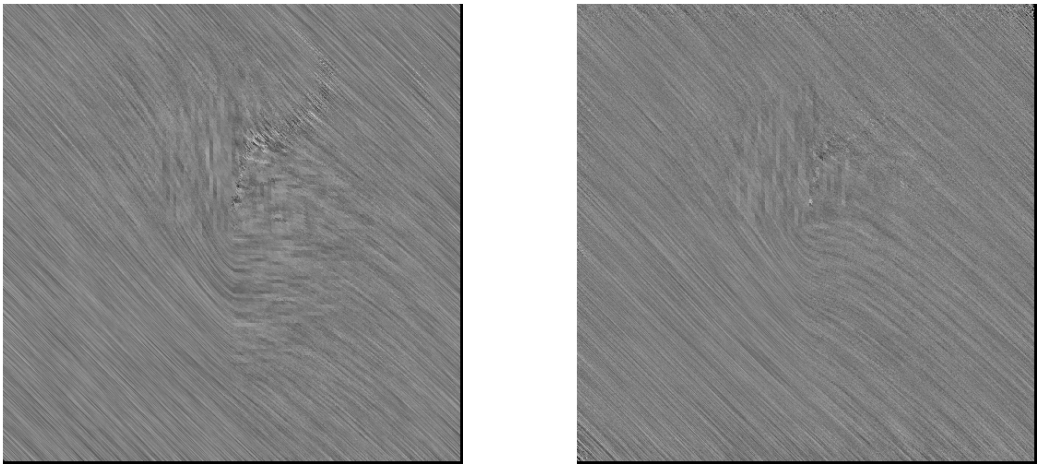
(a) Euler (b) Runge-Kutta

Figure 4: Metsim: 100 randomly generated seedpoints



(a) Euler (b) Runge-Kutta

Figure 5: Isabel



(a) Euler (b) Runge-Kutta

Figure 6: Metsim

5 Comparisons and Conclusion

Unfortunatly, we have a flaw in our algorithm which flipps sections of the vector sets causing our results to differ greatly from the examples provided. This is especially noticable in the visualization of the Metsim dataset, where the vectors are almost all diagonal in contrast with the oval example. In Isabel some of the vectors seem mirrored, but the vortex seems to be in the correct position.

5.1 Streamlines

In general the Runge-Kutta method proved itself to create the smoothest streamlines. Forward Euler had a tendency to rotate its direction more rapidly resulting in sharper turns and more ragged looking lines. The randomly selected seed points usually resulted in worse results with more cluttering when used with the Isabel dataset, while due to the specific nature of our Metsim dataset it generally performed better. Since evenly spaced seed points in a field moving along the diagonal caused most points to follow the same path causing major cluttering and drastically lowered the informative value of the visualization. We used a max streamline length of 500 for the Isabel set, while 200 seemed to be appropriate for Metsim. Choosing a smaller or higher length ended up with partial flow loss or increased cluttering, respectively.

5.2 Line Integral Convolution

Just like in the streamlines Runge-Kutta creates much smoother curves, but it also creates less distinction between them, slightly lowering the ease of interpreting the results.
In the Isabel set, Euler gives some strange detail in the upper and lower right side of the image. This is not shown when using Runke-Kutta, but we suspect this is an artefact from the vector data read errors.
In the Metsim set, the disturbance in the middle is very noticeable when using Euler, compared to the smoother Runge-Kutta.
We did experiment with varying the length of the convolution. However, it gave almost no variation in execution time, barely a second longer for each additional hundred points. This is likely because we implemented the fast LIC algorithm, which places a specific limit to how many times a single image pixel can be affected by the convolution. So the extra time spent on there was regained when skipping image pixels in the main loop.

5.3 Geometric field lines vs LIC

The geometric field line visualization gave a good overview, though much detail was lost compared to the LIC. As we did not use any complicated seeding strategies it was almost instantaneous. We still saw the begininng of the spiral in Isabel, but only a slight curve in Metsim. It is likely that with another seeding strategy, we would have gotten better details.
In contrast, the LIC gave a highly detailed result which were easy to interpret. Though there is a use for both, the quality and ease of interpretation of the LIC makes it superior to the geometric method, when calculation time is not an issue.