UNIK4660 - Assignment 2

Wilhelm Karlsen and Magnus Eden

April 25, 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 very dependent on the points, called seed points, from which the streamlines are calculated. Passing over the vector field and using everything for streamline will 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 the "equal distance" seed point strategy.

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 be given different colors to help show additional system 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 for the Field Line Integration 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, handling of critical points is done by simply ignoring the point if it is used as the origin. 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.

3.1 Forward Euler

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

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

Step size streamline/filter-core length Euler vs RungeKutta

4.1 Forward Euler: FilterVariations

4.2 4th order Runge-Kutta: FilterVariations

5 Conclusion and Comparison