KTH Stockholm
CSC :: CST
Visualization, Autumn 2017, DD2257
Tino Weinkauf

Homework assignment No. 05

Due October 5, 2016

Download the zip-File with the code and place the contents in the same folder as the previous modules. Follow the same process to activate the `IVW_MODULE_DD2257LAB3` module in CMake and build Inviwo.

You find the workspaces for this assignment under *File → Example Workspaces → DD2257Lab3* in the Inviwo application or in the `dd2257lab3/data/workspaces` folder. The workspace for the first task is `euler_rk4_comparison.inv`. The second and third task use the setup in `streamline_integrator.inv`. Note that for the interviews you might want to save some copies of that workspace with different property settings. The initial output for both workspaces displays the same: A single point that can be moved through changing the corresponding property value or by mouse interaction (hold the left mouse button pressed to move the point). See the files `eulerrk4comparison.h/cpp`, `integrator.h/cpp` and `streamlineintegrator.h/cpp` for additional comments and instructions.

Task 5.1: Euler vs. Runge Kutta — 2+3 P

Integrate a tangent curve in forward direction starting from a given seed point, e.g. $\mathbf{x} = (1, 0)^T$, in the vector field

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} -y \\ \frac{x}{2} \end{pmatrix}.$$

The comparison processor receives this vector field as a 16x16 grid where x and y have been sampled with ranges between -8 and 8. $\mathbf{x}$'s position in grid coordinates is therefore $\mathbf{x}_{\text{Grid}} = (9, 8)^T$. Implement the integration using:

(a) the Euler method,

(b) the Runge-Kutta method of 4th order.

Allow the user to specify the *number of integration steps* and the *step size* for each integration method independently. Draw the resulting tangent curves with different colors.

Hint: Our goal is to compare the two integration schemes. See Figure 1 for how it can look like.

You will need to add new properties to the `Euler RK4 Comparison` *processor. A new property has to be treated in at least three places: It needs to be defined in the header file and initialized as well as registered in the implementation.*

Task 5.2: Full-blown Stream Line Integrator — 10 P

Implement a versatile set of functions for integrating stream lines in 2D vector fields. You will need these functions in later assignments.

To test these functions, add corresponding properties to the `Streamline Integrator` processor and integrate a stream line from a given seed point. Display the seed and the stream line. Include the following functionality:

(a) Allow integration in forward and backward direction.

(b) Allow different step sizes.

(c) Allow integration in the direction field (normalized vector field).

(d) Stop the integration after a certain number of steps.

(e) Stop the integration after a certain arc length of the stream line.

(f) Stop the integration at the boundary of the domain.

(g) Stop the integration at zeros of the vector field.

(h) Stop the integration when the velocity becomes too slow.

The assignment comes with several data sets for testing. Make sure to test the last point (slow velocity) with *Sink.dat*.

Hint: Extend your integration functions for a single step from the previous task with additional settings.

Task 5.3: Seeding of Stream Lines **2+3 P**

Display the stream lines for a given 2D vector field.

(a) Seed n stream lines randomly in the domain of the vector field. The user can set n.

(b) Seed the stream lines on an uniform grid. The number of grid points in each direction can be defined by the user. The bounding box of the grid corresponds to the bounding box of the vector field.

(c) (Bonus Points: +5) Seed n stream lines such that the distribution of the seeds corresponds to the distribution of the magnitude of the vector field, i.e., there are more seeds in areas with high magnitude and fewer seeds in areas with low magnitude.

Good data sets for testing are *ANoise2CT4.dat*, *Cylinderclose2CT10.dat*, and *boussinesq2CT522.dat*. An example result is shown in Figure 2.

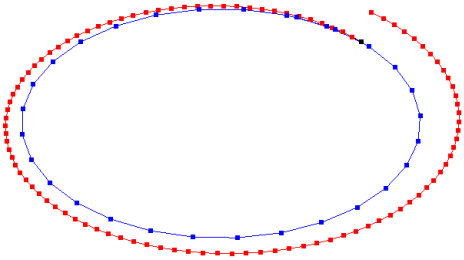

Figure 1: The Runge-Kutta method (blue) approximates the ellipse well with just 30 samples, whereas the Euler method (red) cannot approximate it well with 100 samples.

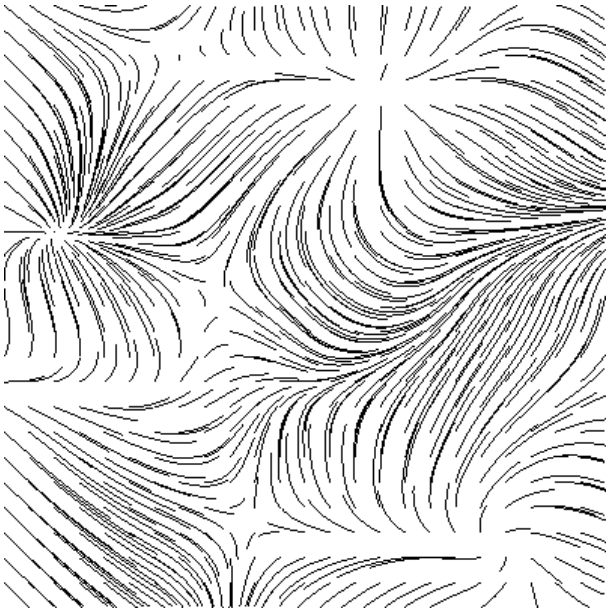

Figure 2: Streamlines on a uniform grid for the *ANoise2CT4.dat* data set.