Scientific Visualization
Project III

Milian Wolff

January 23, 2012

During the third project for the the scientific visualization class by Eugene Zhang we got an introduction to vector calculus and vector field design. We investigate the topology of vector fields and write a JavaView based application for LIC-based vector field design and visualization.

The source code of my exercise solutions can be found online under

`https://github.com/milianw/scivi`

Contents

1 Scalar Field Topology

In the following we analyze the topology of scalar vector fields defined on the unit sphere in 3D. We leverage the periodic boundary conditions by applying spherical coordinates with $r = 1$, which yields the following parametrization:

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\phi \sin\theta \\ \sin\phi \sin\theta \\ \cos\theta \end{pmatrix} \tag{1}$$

The value ranges are $0 \leq \theta \leq \pi$ and $0 \leq \phi \leq 2\pi$.

In the following we investigate scalar vector fields $f(\vec{r})$ and want to know whether they are Morse, i.e. smooth and have only isolated and non-degenerate critical points.

The critical points can be found by solving

$$\nabla f(\vec{r}) = 0, \tag{2}$$

with the Nabla-Operator ∇ in spherical coordinates and $r = 1$ being

$$\nabla = \frac{\partial}{\partial\theta}\vec{e}_\theta + \frac{1}{\sin\theta}\frac{\partial}{\partial\phi}\vec{e}_\phi. \tag{3}$$

Frankly I have no clue how to calculate the Hessian or the determinant of said in spherical coordinates. I'll thus limit myself to finding the critical points then.

1.1 Case a)

$$f(\vec{r}) = 1 \tag{4}$$
$$\nabla f(\vec{r}) = 0 \tag{5}$$

The vector field is not morse since every point is critical and thus non-isolated.

1.2 Case b)

$$f(\vec{r}) = x = \cos\phi \sin\theta \tag{6}$$
$$\nabla f(\vec{r}) = \cos\phi\cos\theta\vec{e}_\theta - \sin\phi\vec{e}_\phi \tag{7}$$

The above turns zero for $\sin\phi = 0$ and $\cos\theta = 0$, i.e. at $\phi = 0, 180$ and $\theta = \pm 90$. I.e. there are two isolated critical points at $[\pm 1, 0, 0]^T$.

1.3 Case c)

$$f(\vec{r}) = x^2 = \cos^2\phi \sin^2\theta \tag{8}$$
$$\nabla f(\vec{r}) = 2\cos^2\phi\cos\theta\sin\theta\vec{e}_\theta - 2\cos\phi\sin\phi\sin\theta\vec{e}_\phi \tag{9}$$

The above turns zero for $\cos\phi = 0$ and any value of θ. As such the critical points are not isolated and the function non-morse.

1.4 Case d)

$$f(\vec{r}) = x^2 - y^2 = \sin^2\theta(\cos^2\phi - \sin^2\phi) \tag{10}$$
$$\nabla f(\vec{r}) = 2\sin\theta\cos\theta(\cos^2\phi - \sin^2\phi)\vec{e}_\theta - 4\sin\theta\cos\phi\sin\phi\vec{e}_\phi \tag{11}$$

The above turns zero for $\sin\theta = 0$, i.e. at the north and southpole ($\theta = 0, \pi$). It is furthermore zero when $\cos\theta = 0$, i.e. on the equator, and additionally $\sin\phi = 0$ or $\cos\phi = 0$, i.e. at $x = \pm 1$ and $y = 0$ or $x = 0$ and $y = \pm 1$.

These singularities are isolated.

1.5 Case e)

$$f(\vec{r}) = \sum_{k=0}^{N} \left(\begin{array}{c} N \\ K \end{array} \right) x^k y^{N-k} \cos\left(\frac{N-k}{2}\pi\right) \tag{12}$$
$$= \sin^N\theta \sum_{k=0}^{N} \left(\begin{array}{c} N \\ K \end{array} \right) \cos^k\phi \sin^{N-k}\phi \cos\left(\frac{N-k}{2}\pi\right) \tag{13}$$
$$\nabla f(\vec{r}) = \sum_{k=0}^{N} \left(\begin{array}{c} N \\ K \end{array} \right) \ldots \tag{14}$$

For $N = 0$ we have case a), $N = 1$ yields case b). Anything larger boggles my mind. Lets skip this.

1.6 Case f)

$$f(\vec{r}) = x^4 + y^4 + z^4 = \sin^4\theta(\cos^4\phi + \sin^4\phi) + \cos^4\theta \tag{15}$$
$$\nabla f(\vec{r}) = 4(\sin^3\theta\cos\theta(\cos^4\phi + \sin^4\phi) - \cos^3\theta\sin\theta))\vec{e}_\theta \tag{16}$$
$$+4\sin^3\theta(\sin^3\phi\cos\phi - \cos^3\phi\sin\phi)\vec{e}_\phi \tag{17}$$

There are two critical points once again at $\sin\theta = 0$. For $\cos\theta = 0$, i.e. on the equator, are again four more singularities for $\cos\phi = 0$ and $\sin\phi = 0$, just like in case d).

2 Vector Field Design

The vector field design application is based on JavaView which already has built-in LIC visualization support. As such, this task was quite easy.

In my implementation, the user picks a type of design element (see below) he wants to add to the vector field. Then he can add such a feature to the vector field by clicking into the vector field domain. Existing features can be removed, altered and moved via drag'n'drop. The view gets updated automatically.

2.1 Design Elements

I have decided to implement the following design elements based on the equations given in the recitation:

1. constant, fig 1: $\vec{V}_i(\vec{r}) = \sigma\vec{e}_i$, with $\vec{e}_i$ being a normalized direction vector
2. generic, fig 2: $\vec{V}_i(\vec{r}) = \sigma\mathbf{A}(\vec{r} - \vec{r}_i)$, with $\mathbf{A}$ being an arbitrary 2×2 matrix
3. sink, fig. 3: generic with diagonal $\mathbf{A}$ and entries -1
4. source, fig 4: generic with diagonal $\mathbf{A}$ and entries 1
5. saddle, fig 5: generic with diagonal $\mathbf{A}$ and entries 1 and -1
6. focus, fig 6: generic with $\mathbf{A} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$
7. center, clockwise, fig 7: generic with $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$
8. center, counter-clockwise, fig 8: generic with $\mathbf{A} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$
9. converging, fig 9: $\sigma(\vec{e}_i + ((\vec{r} - \vec{r}_i) \cdot \vec{n}_i)\vec{n}_i$, with $\vec{e}_i$ being a normalized direction vector and $\vec{n}_i$ the normal to that
10. diverging, fig 10: like the converging element multiplied by -1

In the equations above, σ is the scaling factor

$$\sigma = s\exp(-d|\vec{r}_i - \vec{r}|^2), \tag{18}$$

with s denoting the strength and $\vec{r}_i$ being the base point of the design element, i.e. the point where the user clicked.

Note that user-placed vector field elements are shown as large dots in the LIC image. They are furthermore color coded such that centers, focuses and sources and diverging elements are green, while converging elements and sinks are red. Lastly saddles are colored blue while generic items are black.

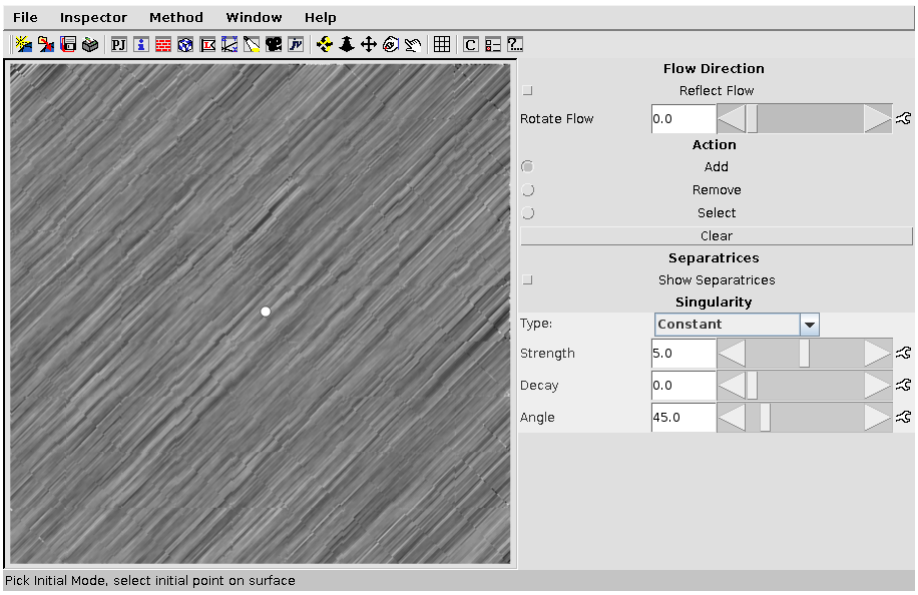


Figure 1: constant vector field design element, without decay

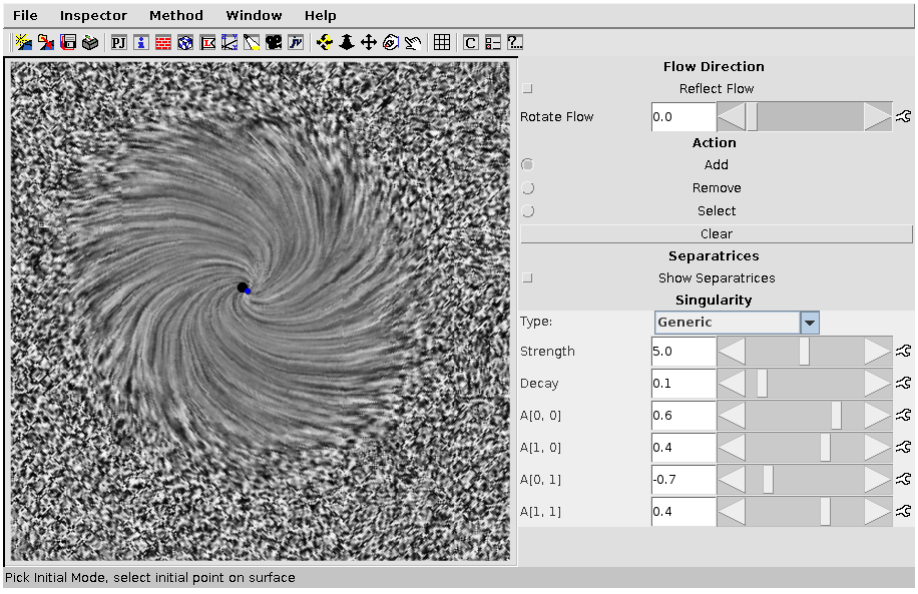


Figure 2: generic vector field design element

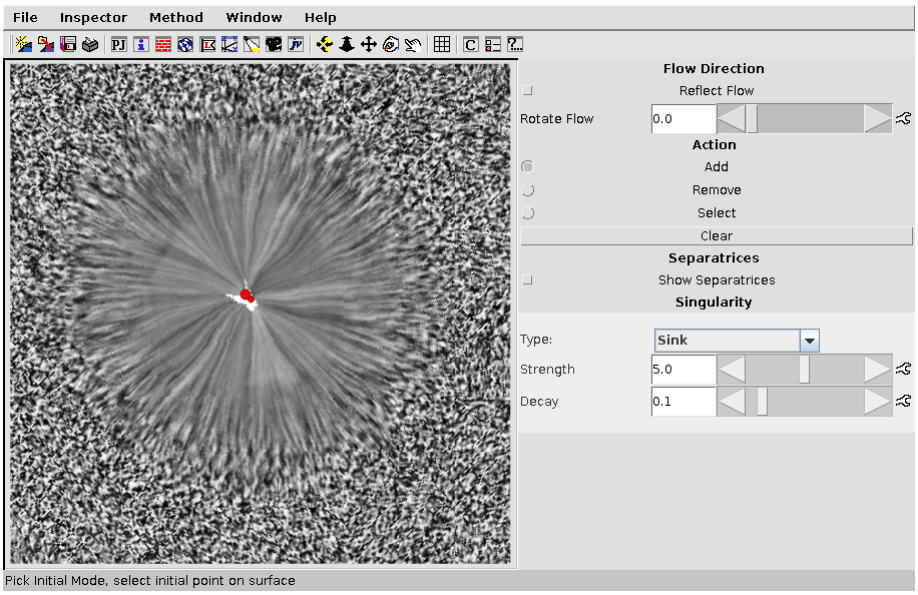


Figure 3: sink vector field design element

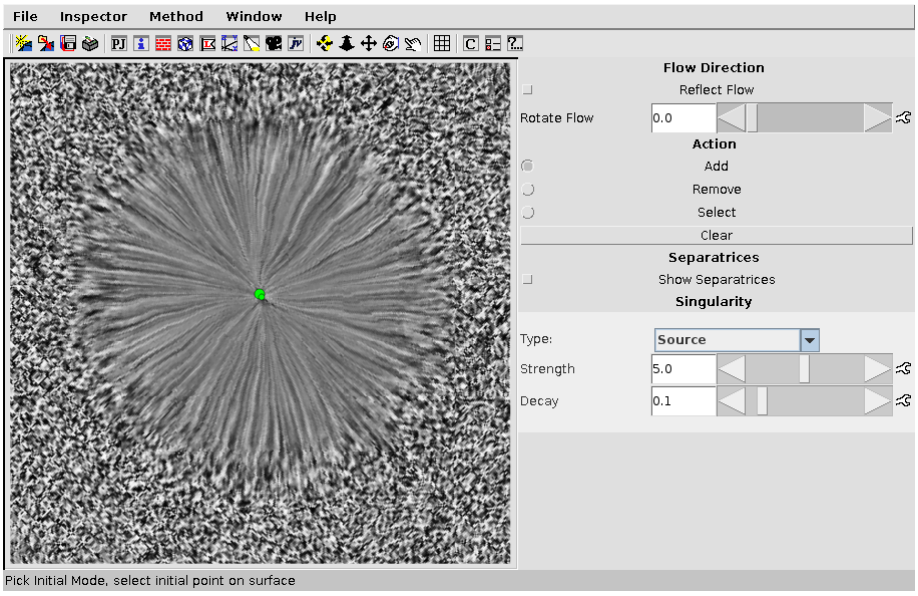


Figure 4: source vector field design element

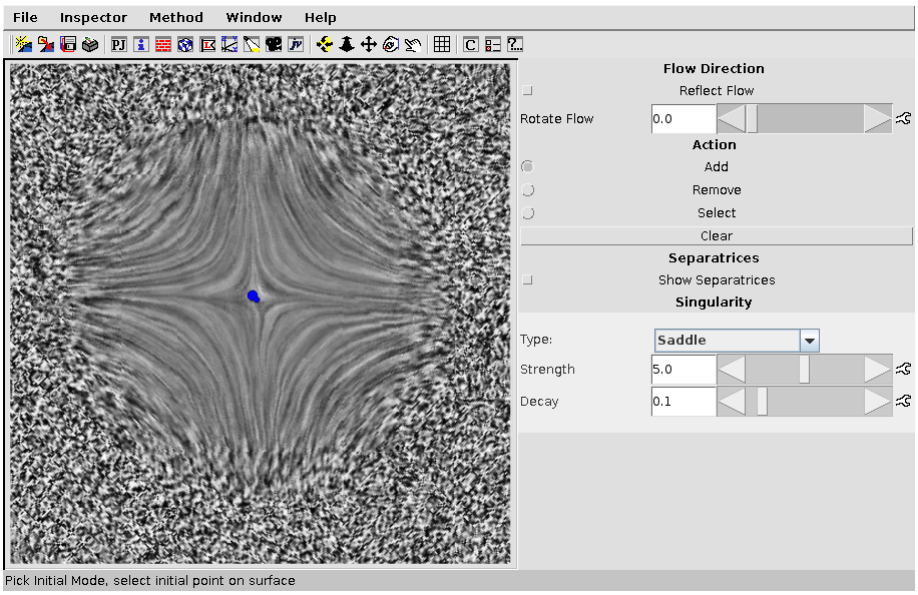


Figure 5: saddle vector field design element

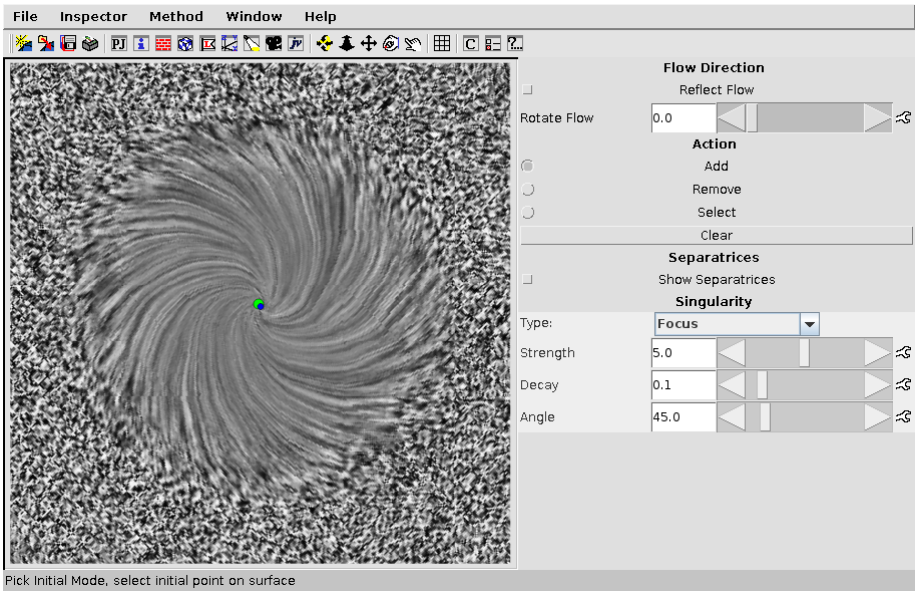


Figure 6: focus vector field design element

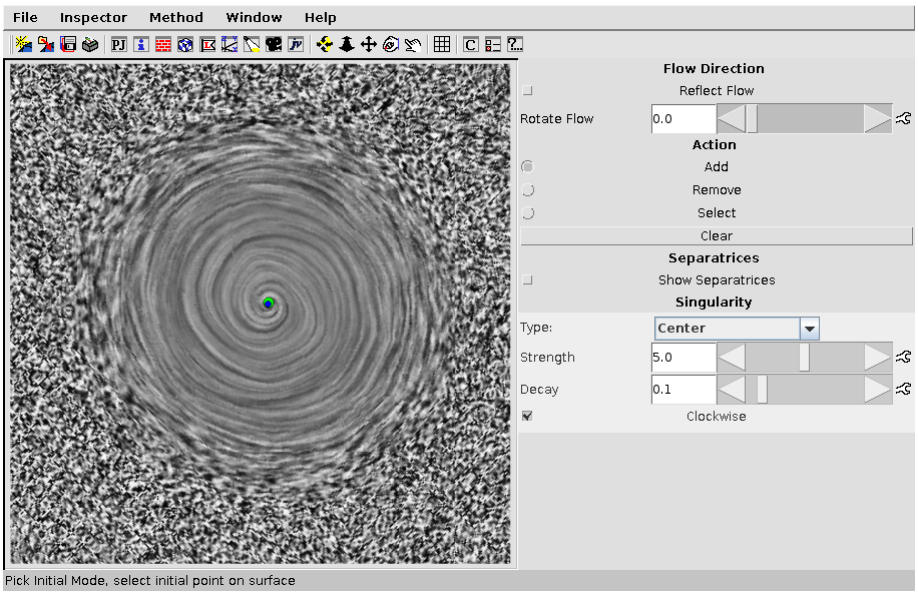


Figure 7: clockwise center vector field design element

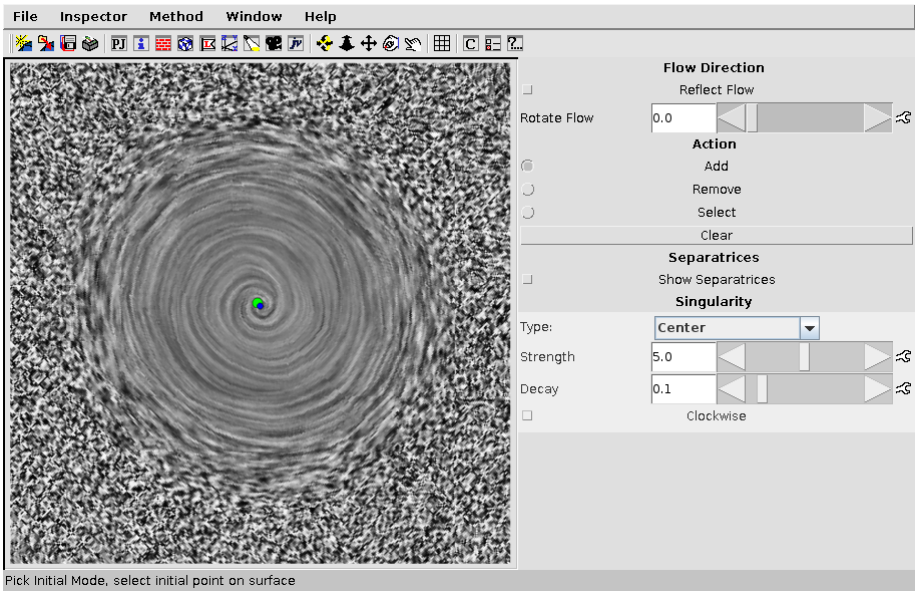


Figure 8: counter-clockwise center vector field design element

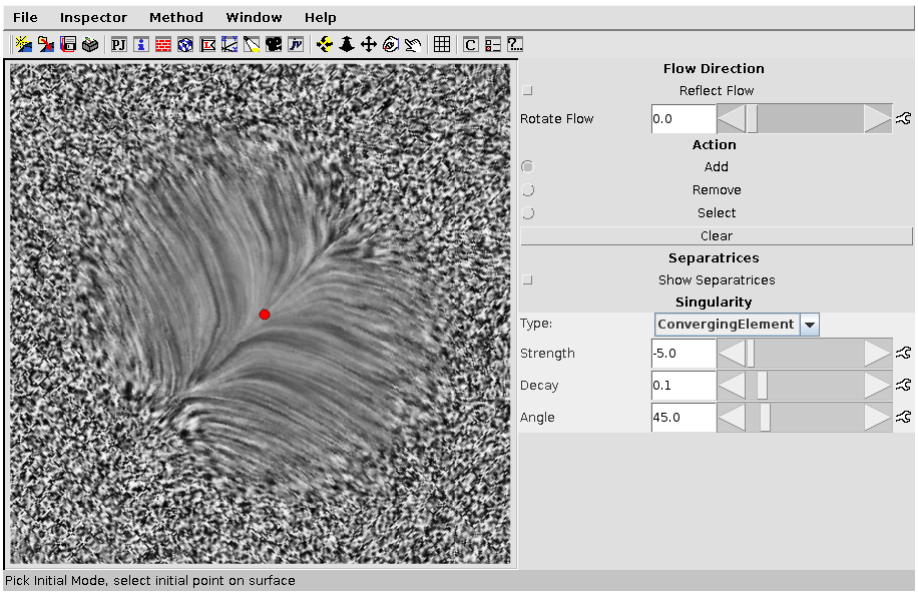


Figure 9: converging vector field design element

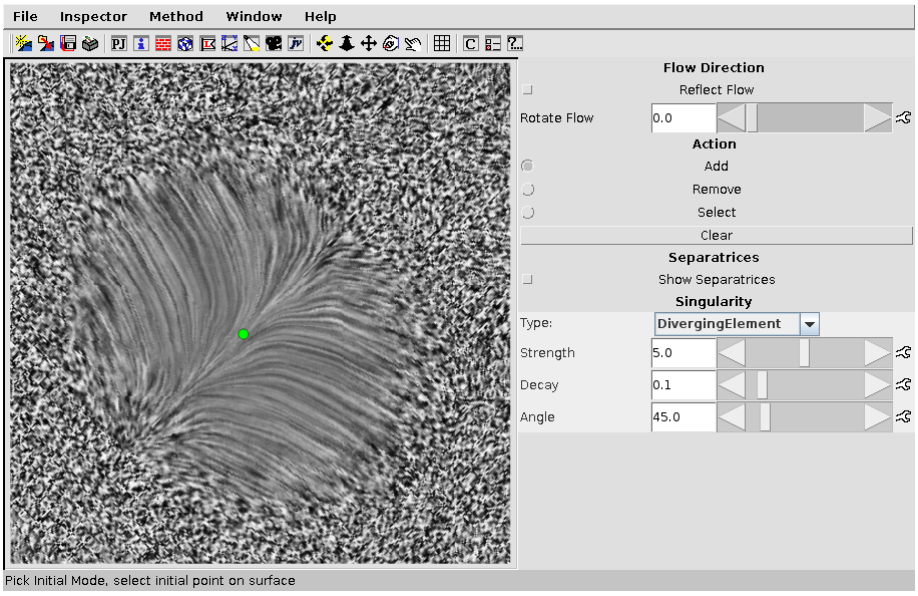


Figure 10: diverging vector field design element

2.2 Parametrization

I reuse the given parametrization of the plane in a triangulated quad. I have chosen a equidistant discretization of $10 \times 10 = 100$ vertices. The discrete vector field is then given by evaluating the sum of all design elements at each vertex point.

2.3 Flow Rotation

The flow can easily rotated by an arbitrary angle θ via matrix-multiplication of each vector in the discretized domain with a rotation matrix:

$$\vec{v}'(\vec{r}) = \mathbf{R}\vec{v}(\vec{r}) \tag{19}$$

The rotation matrix is given by

$$\mathbf{R} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{20}$$

An example can be seen in fig. 11.

2.4 Flow Reflection

Similar to above, flow reflection is achieved via matrix multiplication. In my implementation, reflection can be turned on and off and is supposed to be used in combination with flow rotation. When flow reflection is turned on, the sign of the second column in the rotation matrix changes sign:

$$\mathbf{R}' = \mathbf{R} \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{21}$$

An example for flow reflection is shown in fig. 12.

2.5 Singularities

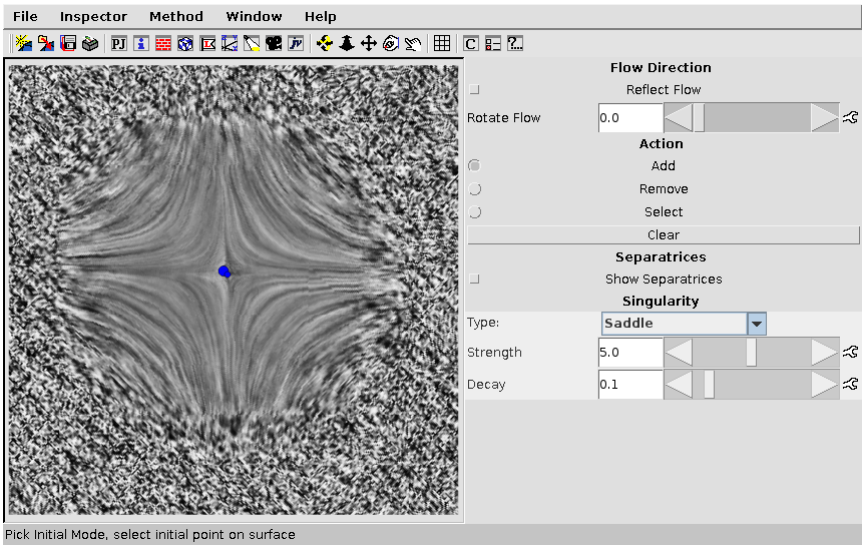
Whenever the vector field is updated, I extract the singularities while LIC is running in the background. To do so we first need to interpolate the discretized vector field, yielding an element-wise continous vector field.

2.5.1 Vector Field Interpolation

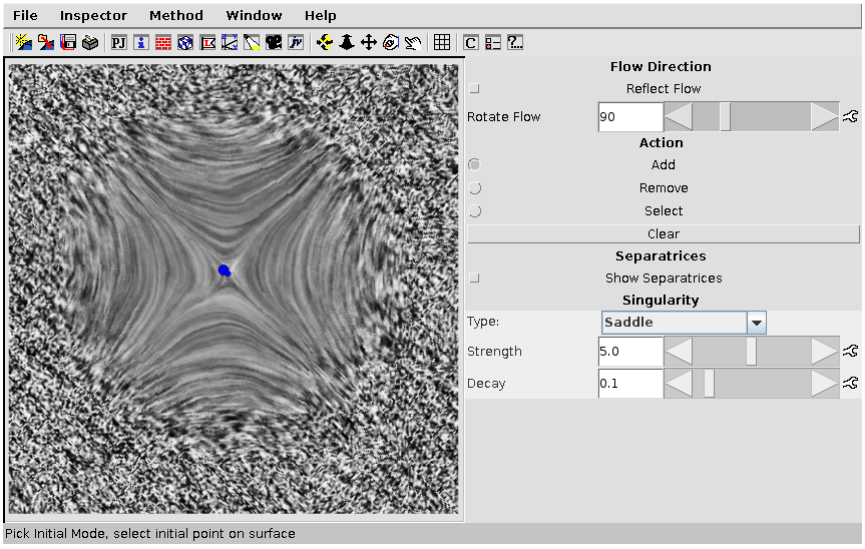
For each element, i.e. triangle we first combine the three vertex positions into a matrix

$$\mathbf{A} = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix}. \tag{22}$$

The actual values of the discrete vector field we store in two vectors $\vec{b}_x$ and $\vec{b}_y$ such that

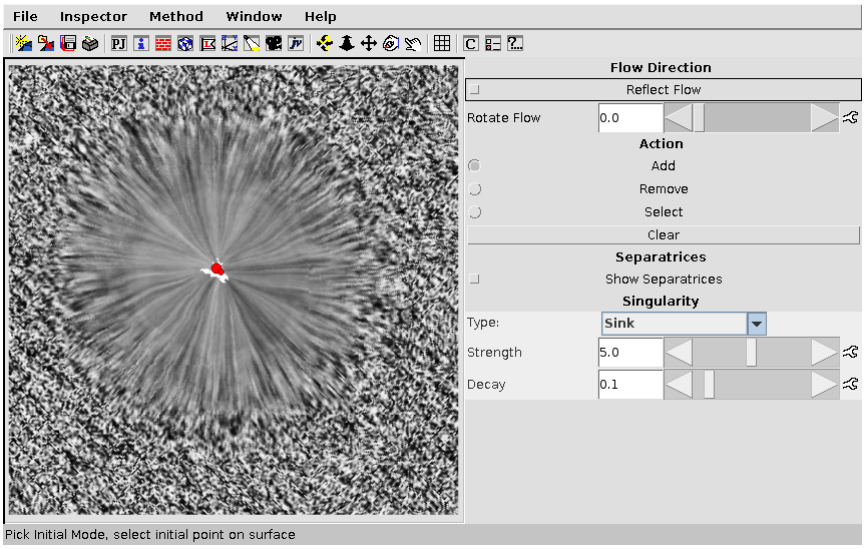


(a) basis

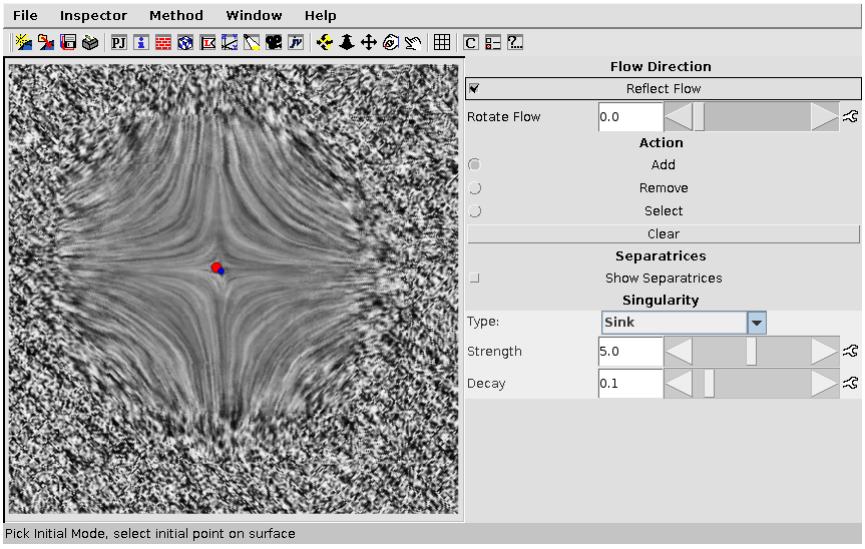


(b) rotated

Figure 11: flow rotation



(a) basis



(b) reflected

Figure 12: flow reflection

$$\vec{b}_k = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix}. \qquad k = x, y \tag{23}$$

Now we can solve, using the Cramer method, the following two equations:

$$\mathbf{A}\vec{c}_k = \vec{b}_k \qquad k = x, y \tag{24}$$

We obtain six coefficients which define our interpolated vector field:

$$\begin{aligned} \vec{V}_i'(\vec{r}) &= \begin{pmatrix} c_{x,1} & c_{x,2} \\ c_{y,1} & c_{y,2} \end{pmatrix} \vec{r} + \begin{pmatrix} c_{x,3} \\ c_{y,3} \end{pmatrix} && (25) \\ &= \mathbf{C}\vec{r} + \vec{c} && (26) \end{aligned}$$

The above is implemented in `InterpolatedField::interpolate()`.

2.5.2 Finding Singularities

Using eq. 26 we can easily find singularities by solving the equation

$$\mathbf{C}\vec{r}_s = -\vec{c}. \tag{27}$$

One must consider the boundary though, i.e. our interpolated vector field is only valid inside the triangle area. Hence a solution obtained by eq. 27 is only a "real" singularity if $\vec{r}_s$ is inside the triangle. The latter is easily checked by the usual vector multiplication approach as implemented in `::inTriangle()` and `::onSameSide()`. All the above is implemented in cmp. `::findSingularities()`.

2.5.3 Classification

To classify the singularities found by applying the above method, we look at the eigenvalues of the Jacobian, in our case $\mathbf{C}$. If the two eigen values have a different sign, i.e. $\sigma(\lambda_1) \neq \sigma(\lambda_2)$ then the singularity is a saddle. Otherwise the singularity is a sink if the eigenvalues are negative and a source otherwise, i.e. for positive eigenvalues.

2.5.4 Examples

Two examples of the above algorithm to find singularities can be seen in fig. 13. The singularities are again color coded similar to above and can be differentiated from the user-supplied design terms by their smaller size.

2.6 Separatrices

After having found the singularities in the vector field we can go one step further by also visualizing the separatrices, i.e. the two isolines originating at a saddle and going into the major and minor direction of the Jacobian $\mathbf{C}$.

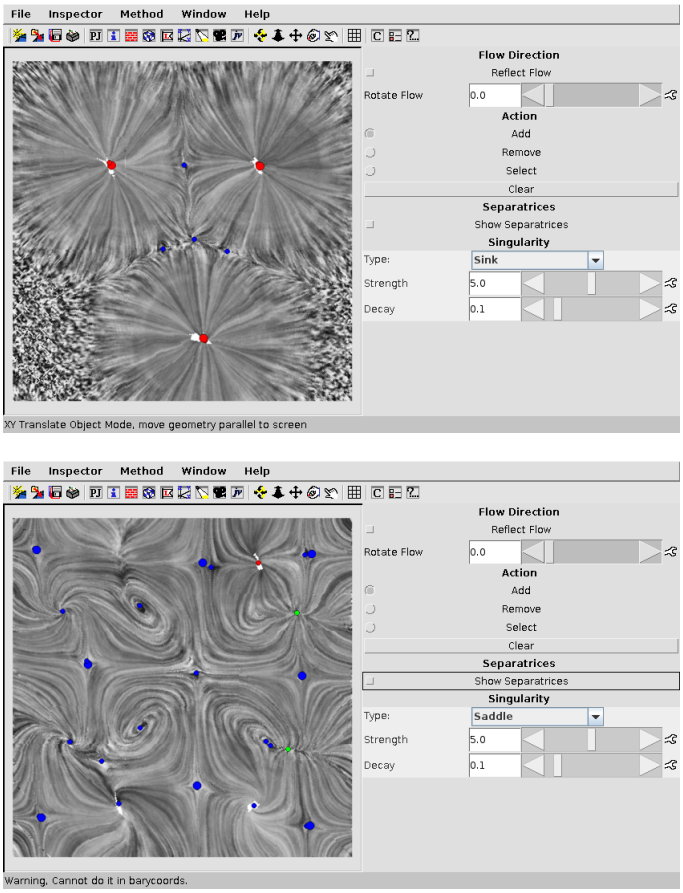


Figure 13: finding singularities

2.6.1 Line Tracing

To trace an separatrix we take a seed $\vec{y_0}$ that is slightly displaced from the singularity in the direction we want to go to. The next point on the separatrix is then found by numeric integration, e.g. via the explicit Euler method (cmp. `ExplicitEulerTracer`):

$$\vec{y}_{i+1} = \vec{y}_i + h\vec{V}(\vec{y}_i) \tag{28}$$

Here $\vec{V}$ denotes the interpolated vector field that is valid at the position $\vec{y}_i$, see `InterpolatedField::elementAt()` for more information.

A better approach is to apply the so called classical Runge-Kutta metod (cmp. `ClassicalRungeKuttaTracer`), which is more numerically stable. Here we obtain the next line node from the equation

$$\vec{y}_{i+1} = \vec{y}_i + 1/3(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4), \tag{29}$$

with

$$\vec{k}_1 = h\vec{V}(\vec{y}_i) \tag{30}$$
$$\vec{k}_2 = h\vec{V}(\vec{y}_i + 0.5\vec{k}_1) \tag{31}$$
$$\vec{k}_3 = h\vec{V}(\vec{y}_i + 0.5\vec{k}_2) \tag{32}$$
$$\vec{k}_4 = h\vec{V}(\vec{y}_i + \vec{k}_3). \tag{33}$$

2.6.2 Caveats

There are no built-in checks to prevent crossing separatrices, which may result in noisy images esp. when multiple saddles are in the vicinity of a vortex.

Furthermore the tracer does not stop when it gets close to a singularity, which sometimes might lead to a small overshoot, yet most of the time can be safely ignored.

Both of the above could be implemented but will impose a significant performance burden.

2.6.3 Examples

To see the above in action, look at fig. 14. Note that separatrices in major direction are colored cyan while those in the minor direction are orange. Furthermore each integration step is shown by a vertex node on the separatrix.

My implementation uses the classical Runge-Kutta integration scheme with a step-size of $h = 0.5$ and up to $N = 1000$ steps.

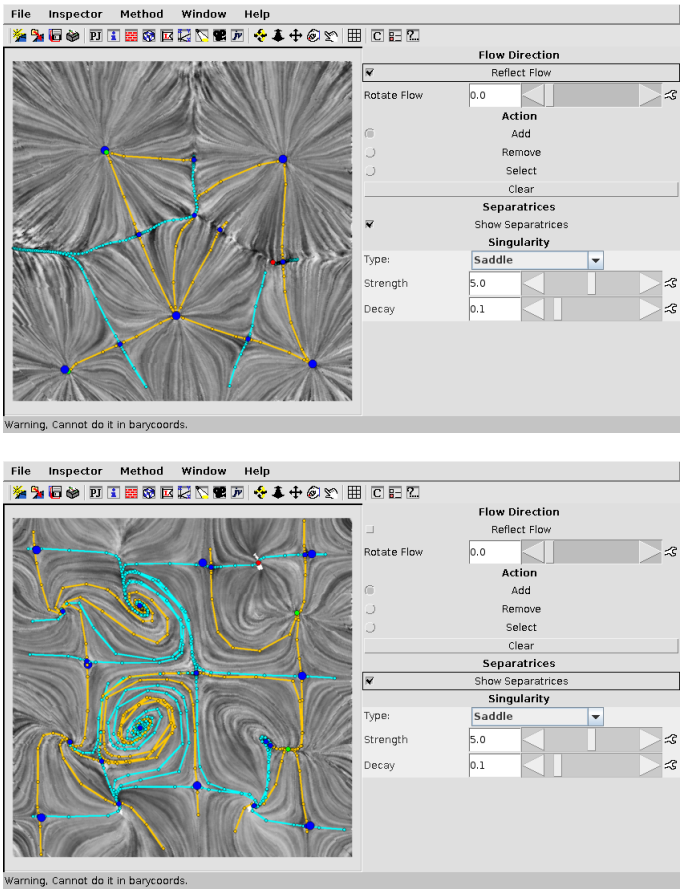


Figure 14: finding separatrices