Projects for
Chapter 6: Vector Visualization

1 PROJECT 1

Implement *directional color coding* for a 2D vector field (stored, for simplicity, on a uniform quad grid), as described in Sec. 6.3, but using the different colormaps described for scalar visualization in Sec. 5.2, Fig. 5.2. Try also the zebra colormap described in Fig. 5.3. In all cases, use the vector orientation (as an angle between 0 and 360 degrees) as being the scalar quantity to map to colors. Take care about the fact that the colormap should have matching colors at its endpoints (which correspond to the orientations of 0 and 360 degrees, respectively). How would you modify the colormaps shown in Fig. 5.2 to achieve this effect? Compare the vector-field visualizations generated for different colormaps, and comment on which one you find the most effective in conveying the vector fieldÕs direction.

2 PROJECT 2

The standard *displacement plots* discussed in Section 6.4 for visualizing 3D vector fields take an input shape (typically a 3D surface) and warp it by linearly displacing each input-shape vertex for a (small) distance along the vector field at that location. Implement a generalization of this technique, where we displace each input-shape vertex along a 3D streamline computed from that vertex downstream in the vector field. The amount of displacement (expressed as integration time, using either the given vector field or a unit-length normalized version thereof) should be the same for all input-shape vertices. For this, proceed as follows:

- First, compute all streamlines generated by considering all all input-shape vertices as seed points, and save these streamlines as ordered point-sets.

- Next, construct the warped shape by selecting the points in the above point-sets that have the same integration time, and connect them by reusing the topology (cells) of the input shape.

Produce several visualizations for a given vector field and different integration time values. Which are the perceived advantages (and disadvantages) of this technique as opposed to displacement plots?

3 PROJECT 3

Implement a *particle-based animation* technique for visualizing 2D vector fields. For this, proceed as follows: Given a 2D vector field, represented *e.g.* for simplicity on an uniform 2D grid with quad cells,

1. Compute a dense set S of randomly-placed seed points $\mathbf{s}_i$ over the domain of the vector field. For this, study the 'dense streamlines' sample code provided in Chapter 6.

2. Define a particle set $P = \{\mathbf{p}_i \in \mathbb{R}^2\}$. Each point $\mathbf{p}_i$ represents a massless particle being advected in the 2D vector field.

3. Particles are created at the locations of the seed points $\mathbf{s}_i$, and are subsequently advected in the vector field, in an endless loop.

4. Besides position, each particle has an *age*, representing the integration time. Also, each particle knows the seed point $\mathbf{s}_i$ it was started from.

5. Keep the size $|P|$ of the particle set P fixed, by removing the oldest particles and/or relaunching particles from the corresponding seed points. The particle-set size $|P| \geq |S|$ should be an user-supplied parameter.

6. Draw particles using *e.g.* arrow, line, or sphere glyphs. Additionally, you can color the particles using random grayscale values.

Experiment with various values for the size of the seed-point set S and particle-set P. Experiment with animating the particles in both the normalized and un-normalized vector field, and comment on the differences. Compare your results with those obtained by the image-based flow visualization (IBFV) algorithm provided in the samples for Chapter 6.

4 PROJECT 4

Implement the *farthest point streamline seeding* method of Mebarki *et al.* for the visualization of 2D vector fields (see Chapter 6, page 210) using 2D distance transforms, as follows. Given a set S of traced streamlines, the distance transform DT_S of the set S gives us, for each point in the 2D plane, the minimal distance to *any* streamline in S. As such, we can find the farthest point $\mathbf{x}$ from any existing streamline in S by finding the maximum of DT_S over the vector field's domain. This point $\mathbf{x}$ is the location where we want to seed a new streamline.

In detail, your implementation should proceed as follows:

1. Initialize DT_S to infinity.
2. Initialize the first streamline seed-point $\mathbf{s}$ to the center of the computational domain.
3. Trace a streamline upstream and downstream from $\mathbf{s}$ until it exits the domain or enters a region where DT_S is lower than a user-given threshold ϵ which gives the maximal local streamline density.
4. Compute the distance transform DT_S of all already-drawn streamlines. For this, you can use the GPU-based sample code provided in Chapter 9.
5. Find the maximum $\max(DT_S)$ of the field DT_S, and set $\mathbf{s}$ to this location.
6. Repeat the process from step 3 until $\max(DT_S)$ becomes lower than the threshold ϵ.

Compare your results, for various vector fields and values of the streamline density ϵ, with the results delivered by the algorithm of Mebarki *et al.*

End of Projects for
Chapter 6: Vector Visualization
