Editors: Claudio Silva, csilva@cs.utah.edu
Joel E. Tohline, tohline@rouge.phys.lsu.edu

Robust Loop Detection for Interactively Placing Evenly Spaced Streamlines

By Zhanping Liu and Robert J. Moorhead II

The authors present a robust and fast loop-detection strategy to interactively place dense, uncluttered, evenly spaced streamlines, which help visualize complex flows.

Flow visualization is crucial to data analysis in many computational areas. The past decade has seen many flow visualization methods, but *streamlines*—instantaneous curves everywhere tangent to the flow—remain one of the fastest and most straightforward techniques. A streamline is generated through step-by-step advection/integration from a given point, the *seed*, usually in both negative and positive directions. One intrinsic problem with this geometry-based approach is that without an effective streamline placement scheme, an incomplete, nonuniform, or cluttered display of the flow often occurs. A global layout of evenly spaced streamlines, on the other hand, provides an aesthetic and intuitive pattern for facilitating cognitive flow reconstruction.

When creating evenly spaced streamlines, longer streamlines are better because shorter ones might cause distracting discontinuities. To generate an optimal placement, however, the advection of long streamlines must be balanced with the uniformity of the placement due to flow divergence and convergence. In addition, cavities might affect a placement when two streamlines, separated by an excessive distance, fail to accept another one in between to meet a specified density. Consequently, global placement of strictly evenly spaced streamlines might not occur. Furthermore, loops usually exist in a flow field and might clutter or even ruin a streamline layout.

Although loop detection is critical to streamline placement, few researchers have addressed this issue. Here, we describe an effective, robust, and fast loop-detection strategy for creating evenly spaced streamlines. At a negligible additional cost, it allows interactive streamline placement for complex flows on an ordinary PC.

Evenly Spaced Streamline Placement

We can roughly categorize existing algorithms for placing evenly spaced streamlines as *image-guided* and *sample-based*. Image-guided algorithms[1] adopt an iterative process to produce an optimal placement by creating, merging, repositioning, lengthening, or shortening streamlines by trial and error. They accept a trial operation only when the placement is refined—specifically, when the trial reduces the difference between the low-pass filtered version of the placement and a uniform grayscale image. Sample-based algorithms[2–5] employ *intersample* distance control to approximate *interline* distance control in streamline advection. They mainly differ from one another in the greedy seeding strategy that drives the placement process until a dynamic queue of candidate seeds is empty, and in the data structure they use to store streamline samples and perform intersample distance control.

Given a *threshold separating distance*, d_{sep}, which governs the density of an evenly spaced streamline placement, sample-based algorithms require that the interval between any two successive samples of a streamline be less than d_{sep} to make intersample distance control acceptable—any two streamlines are separated by a distance equal to or greater than d_{sep}. We can generate streamline samples using either a fixed step-size integrator [2–4] or a fourth-order Runge-Kutta integrator with adaptive step size and error control (RK4-ASSEC) coupled with cubic Hermite polynomial interpolation (CHPI).[5] RK4-ASSEC allows rapid but highly accurate streamline advection. CHPI is a fast, flexible curve-sampling scheme that produces evenly spaced samples along each streamline from *raw samples* directly generated through RK4-ASSEC. A brute-force but effective solution for evenly spaced streamline placement is to perform distance checking on the *n*ewest *s*ample of the *c*urrent *s*treamline (NSCS) against each *s*ample of the *o*ther *e*xisting *s*treamlines (SOES) to determine whether the current one is further advected or immediately terminated. In contrast with Delaunay triangulation,[4] a Cartesian grid with the cell size equal to d_{sep} is usually superimposed

on the flow field.[2,3,5] Each cell maintains, through sample registration, a list of pointers to the samples within the cell that have been accepted. A distance controller based on this data structure performs distance checking on the NSCS against only the SOESs within the NSCS's nine neighboring cells. If the distance between the NSCS, and each of the SOESs in those cells is equal to or greater than d_{sep}, the distance controller accepts the NSCS, and the streamline is further advected. Any sample refusal immediately terminates streamline advection, and the controller then registers the existing samples of the streamline.

Image-guided algorithms generate high-quality placements, but the huge computational cost restricts their practical applicability. Sample-based algorithms run faster than image-guided ones, though most of them[2–4] still can't interactively place high-density evenly spaced streamlines because they use a fixed step-size integrator. In addition, these algorithms tend to produce discontinuities and cavities in the placement, and either ignore streamline loops [2,3] or fail to provide a robust solution to this critical issue.[4]

A Novel Loop-Detection Strategy

Our fast, robust loop-detection strategy is an important component of our sample-based Advess (*adv*anced *e*venly *s*paced *s*treamline placement) algorithm.[5] By using RK4-ASSEC, CHPI, double-queue seeding, and adaptive distance control, Advess is an order-of-magnitude faster than Bruno Jobard and Wilfrid Lefer's algorithm,[2] with better placement quality, and is five times faster than Abdelkrim Mebarki and colleagues' algorithm,[4] with comparable placement quality but more robust loop detection.

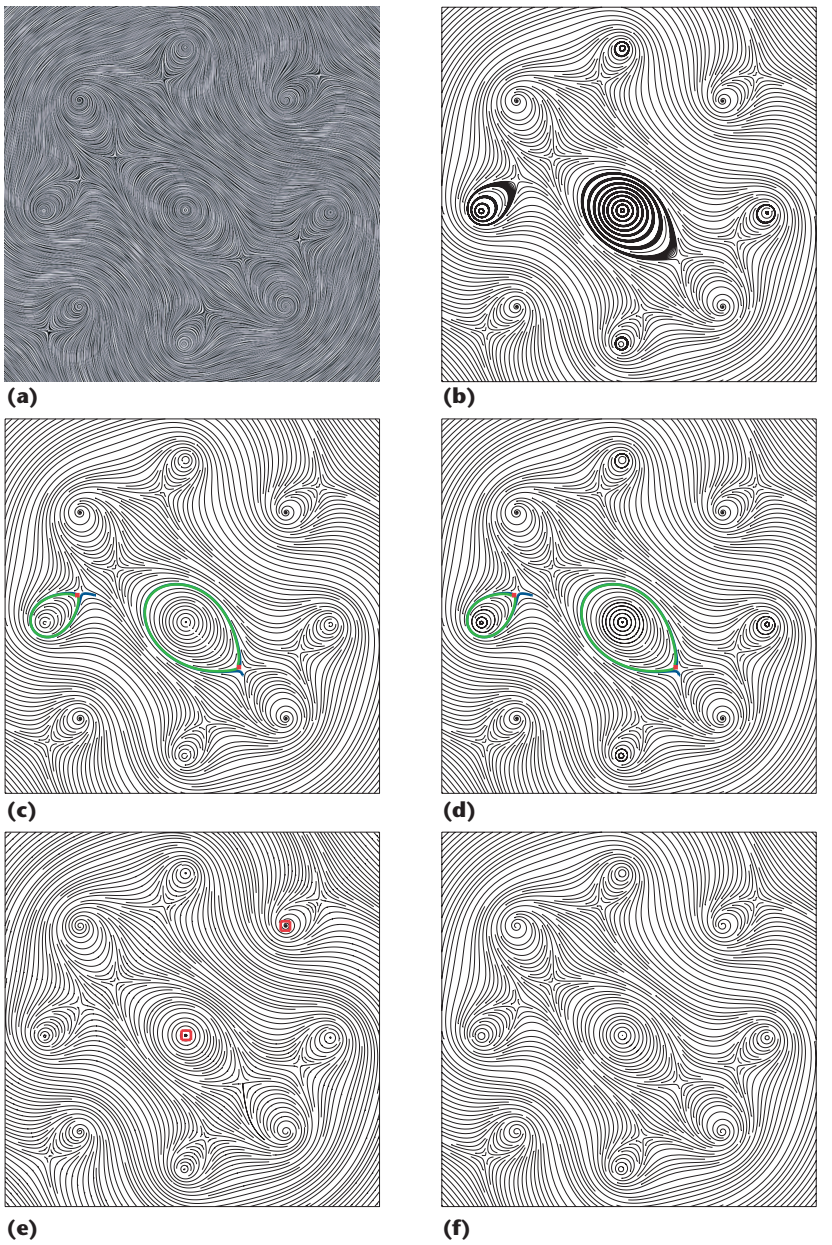


Figure 1. Some 1% density streamline placements of a flow field. (a) A line integral convolution (LIC)[8] image shows five centers and four tightly spiraling foci. (b) A placement generated without loop detection is heavily cluttered by closed and spiraling streamlines. (c) When we choose a small, seed-based curve length as the threshold beyond which to begin seed-based distance checking on the current sample of a streamline for loop detection, all closed streamlines are broken. The negatively advected part (in blue) and the positively advected part (in green) of each of two streamlines squeeze together to cause an ambiguity. (d) When we choose a large seed-based curve length threshold in a naïve loop detector as in (c), some closed streamlines are still broken, whereas some others are advected for excessive cycles. (e) The Mebarki loop-detection strategy isn't robust, and two streamline loops (red squares) leave artifacts. (f) Our novel loop-detection strategy is robust enough to avoid any ill-looping artifacts while retaining important flow features, such as closed streamlines and spiraling structures.

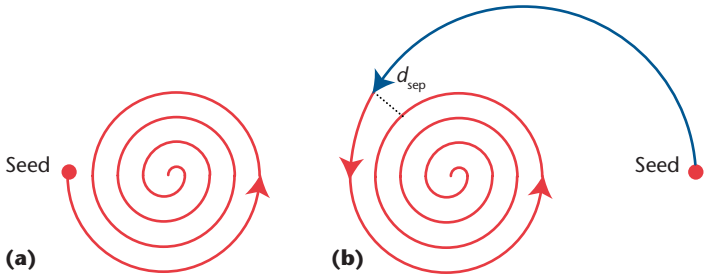


Figure 2. Two spiraling streamlines advected inward. (a) One streamline is ill-looping as a whole, whereas (b) the other is ill-looping only in part. The ill-looping part (shown in red) might (as in (a)) or might not (as in (b)) begin with the seed. A naïve loop-detection strategy based on seed-based distance checking might not detect the ill-looping part (as in (b)).

Loop Detector—A Sophisticated Intraline Distance Controller

In this article, we loosely define a loop as either a *closed* streamline or an open but tightly *spiraling* streamline. Figure 1 shows some 1 percent density placements of evenly spaced streamlines for a 400×400 flow field that demonstrate the critical role of loop detection in streamline placement. Sample-based algorithms are susceptible to loop cluttering or placement artifacts (Figures 1b, 1c, 1d, and 1e) due to locally oriented distance control. Some general algorithms are well suited to accurately detect closed streamlines,[6,7] although they're too computationally expensive to interactively place streamlines. Furthermore, a discrepancy exists between their global loop-detection schemes and intersample distance control mechanisms used for evenly spaced streamline placement; thus, it's difficult to adapt them for sample-based algorithms.

The distance controller described in the previous section fulfills only interline distance control. Samples of a streamline aren't registered in this controller until the streamline is terminated. This scheme is intended to prevent a sample from being checked against other existing samples of the same streamline, thus no loop detection occurs, and the resulting placement might be cluttered (as in Figure 1b). If sample registration isn't deferred such that the interline distance controller can check the intersample distance between intraline samples, streamline advection would be terminated from the very beginning because the interval between any two successive streamline samples is less than d_{sep}, as mandated for acceptable interline distance control. Mebarki and colleagues proposed the use of Delaunay triangulation[4] of streamline samples for both interline and intraline distance controls, the latter to detect loops. As discussed elsewhere,[5] this strategy isn't robust and could introduce artifacts (as in Figure 1e) or very noticeable cluttering.

To effectively and robustly detect loops, a sophisticated intraline distance controller, or *loop detector*, needs to be specifically designed for sample-based algorithms. The newest sample of a streamline being advected is first checked by the intraline distance controller and, unless rejected, is then checked by the interline distance controller. In this way, the two controllers work in combination to create evenly spaced streamlines without being cluttered by loops.

Toward a Universal Design

Without effective loop detection—that is, sophisticated intraline distance control—a closed streamline would incur excessive computation for many cycles of advection. Even worse, these cycles usually don't coincide precisely with one another when rendered to the screen. Consequently, a closed streamline might look thicker or bolder than nonlooping ones, introducing artifacts in the placement (Figures 1b and 1e). A spiraling streamline causes cluttering if it spirals tightly enough (relative to d_{sep}). From the viewpoint of creating evenly spaced streamlines, therefore, a streamline is *ill-looping* when the distance between any two successive cycles is less than d_{sep}. When a spiraling streamline spirals tightly enough, it is ill-looping either *as a whole*—that is, everywhere (Figure 2a)—or *only in part* (Figure 2b). Under this definition, a closed streamline is always ill-looping as a whole.

We can't use the distance between the current (newest) sample and the seed of a streamline for effective loop detection because a spiraling streamline might be ill-looping only in part, and the *ill-looping part* doesn't necessarily begin with the seed (as in Figure 2b). A loop-detection scheme based on seed-based distance checking might fail to detect an ill-looping spiraling streamline if the ill-looping part is on one (Figure 2b) or both sides of the seed (in blue and green in Figures 1c and 1d, respectively). To prevent streamline advection from being terminated from the very beginning while detecting loops, a naïve trial might choose a seed-based curve length threshold, beyond which the loop detector performs seed-based distance checking on the current sample. The problem is that a closed streamline tends to be terminated just prior to the first cycle's formation due to the seed-based distance being less than d_{sep} when the threshold is small relative to the circumference (Figure 1c). Given an unknown flow field in which a closed-loop region—that is, a center—could be considerably large, even a large threshold might not allow the

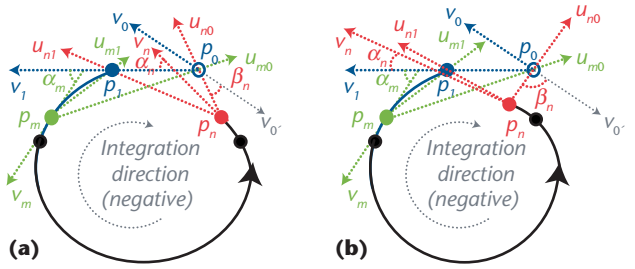


Figure 3. Our loop-detection strategy. Given the current sample p_0 and the previous sample p_1 of an outward-negatively advected streamline, the strategy distinguishes an opponent sample p_n ($|p_0p_n| < d_{sep}$ and $\alpha_n <= \alpha$; any such sample terminates a loop-detection process) from a proponent sample p_m ($|p_0p_m| < d_{sep}$ and $\alpha_m <= \alpha$; several samples of this kind might appear during a loop-detection process) by at least one negative (negated here) dot-product, regardless of p_n being (a) trigonometrically beyond or (b) between the segment p_0p_1. Samples along the black part of the streamline are not p_0's "α-neighbors" and hence are simply skipped without proponent–opponent disambiguation. We exaggerate segment p_0p_1 and α here for illustrative purposes.

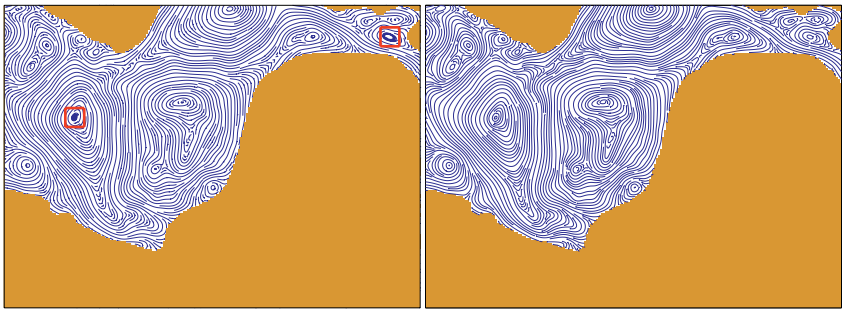
first cycle of a larger closed streamline to form, whereas smaller closed streamlines are advected many cycles to cause artifacts (Figure 1d). Closed streamlines are important flow features that a placement needs to retain by enabling the formation of the first cycle rather than excessive cycles or distracting broken loops. In fact, the distance between each existing sample and the current one should help focus loop detection on the ill-looping part of a streamline. We can't use the looping angle between the seed and the current sample as a threshold beyond which to begin simplistic seed-based distance checking. Instead, we must use the looping angle between each existing sample q and the current sample p_0—that is, the *ill-looping angle* of q (relative to p_0). To retain important flow features without loop cluttering, a universal loop-detection strategy needs to consider both the ill-looping angle and the intersample distance of each existing sample relative to the current sample.

A Robust Strategy

An ill-looping streamline might not clutter a placement if the largest ill-looping angle is limited to exactly one cycle for a closed streamline or near one cycle for a spiraling streamline. Allowing the ill-looping part to advect as long as possible within *the one-cycle limit* encourages a closed streamline to form the first cycle and a spiraling streamline to demonstrate the structure in detail without cluttering the placement.

Specifically, an ill-looping streamline is advected until the current sample p_0 makes both the ill-looping angle of an existing sample p_n equal to or greater than $2\pi - \alpha$ (α is a small threshold angle) and the distance between p_n and p_0 less than d_{sep}. A raw but effective loop-detection strategy is to check each existing sample against the current sample with respect to the distance and the ill-looping angle to find the first p_n-like sample, if any exist. In fact, each existing sample's ill-looping angle isn't obtained via segment-wise angle accumulation because trigonometric computation incurs a large cost; additionally, many accumulated angles would need to be updated nontrivially each time "the current sample" is updated. Instead, we measure it indirectly using a *dot-product* between two normalized vectors, which lets our loop detector add only a negligible additional cost to evenly spaced streamline placement. This is a critically important issue because most of the streamlines advected in an average flow field aren't ill-looping. However, given a threshold angle α, a sample p_m might exist such that the ill-looping angle is equal to or less than α, and the distance between p_m and p_0 is less than d_{sep}. Thus, the dot-product scheme causes an ambiguity between p_m and p_n in terms of the ill-looping angle. Our loop-detection strategy must distinguish between an *opponent sample* p_n (see Figure 3) and a *proponent sample* p_m when both are less than d_{sep} to the current sample p_0 and simultaneously the two associated flow vectors, v_n and v_m, are aligned, within a small threshold angle α, with the latest unidirectional line segment vector v_1 (that is, one of the bidirectional segment vectors that points in the positive flow direction). As p_0's "long-term α-neighbor," p_n approaches the one-cycle limit, whereas p_m, as p_0's "short-term α-neighbor," doesn't.

We use two vectors—(u_{n0}, u_{n1}) or (u_{m0}, u_{m1}), pointing toward the current sample p_0 and the previous sample p_1, respectively, from any opponent or proponent sample—to compute two dot-products with v_1. Therefore, we can distinguish p_n from p_m by at least one negative dot-product for positive advection. This holds for negative advection, too, if the two dot-products are negated. We might find several proponent samples through a loop-detection process, which runs for each newly generated sample. However, any opponent sample terminates such a process because the ill-looping part has been advected enough (ill-looping angle $>= 2\pi - \alpha$) to perform loop classification by testing whether the vector pointing from p_0 to p_n is aligned, within a small threshold angle β, with the integration vector (in the negative or positive flow direction) at p_0. Our test shows satisfactory results using α



(a) (b)

Figure 4. Two 1% density evenly spaced streamline placements. We compared (a) Mebarki and colleagues' algorithm and (b) our Advess algorithm in visualizing a 661 × 481 simulated ocean flow field. Our loop-detection strategy allows Advess to generate a layout without ill-looping artifacts, whereas the Mebarki approach fails to detect some spiraling loops, two of which (red squares) are very noticeable in the resulting placement.

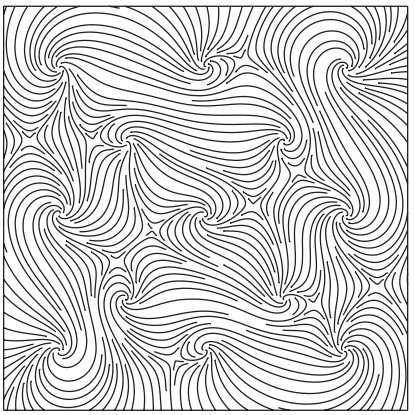


Figure 5. A 1% density placement of evenly spaced streamlines. This placement is the same using Advess both with and without our loop-detection strategy for a 400 × 400 flow field without centers or tightly spiraling foci.

$= 20^\circ$ and $\beta = 10^\circ$. This loop-classification scheme isn't as accurate as those used in general loop-detection algorithms,[6,7] although it makes our loop-detection strategy robust enough to avoid any cluttering. Figure 3 shows the two possible cases for outward negative advection.

A Fast Solution

We designed our loop-detection strategy to be fast; it checks the distance from each of the existing samples of a streamline to the current sample to ignore any samples greater than or equal to d_{sep} because they don't cause ill-looping artifacts. Only the remaining samples undergo proponent–opponent disambiguation to determine if an opponent sample exists to approach the one-cycle limit relative to the current sample. To reduce the amount of distance checking, our loop detector adopts an extended version of the data structure used in the interline distance controller. Thus, distance checking involves only the existing samples within the current sample's nine local cells. Furthermore, the signed RK4-ASSEC step index, a negative or positive integer determined by the advection direction, is attached to each raw sample as the stamp. Each of the evenly spaced samples along the streamline, which are generated through CHPI, is assigned a neighboring raw sample's stamp. If the stamp difference between an existing sample and the current sample is less than a given integer threshold δ, this existing sample is assumed to be a proponent sample and passes the stamp check. In this way, many proponent samples are simply skipped on a per-sample basis through fast stamp checking without undergoing distance checking against the current sample. In addition, each cell of the loop detector maintains the minimum (overwritten in negative advection) and maximum (overwritten in positive advection) stamps of the existing samples within it, and hence many samples are efficiently skipped on a per-cell basis by using the extreme stamp of the current direction for stamp checking. A conservative value for δ might be 4; a larger one could be used due to the smooth curves obtained by an RK4-ASSEC integrator.

The loop detector first checks the current sample of a streamline and, unless it finds an ill-looping cycle, sends the sample to the interline distance controller. Otherwise, the advection is terminated, and the ill-looping streamline is classified to either form a closed loop or simply refuse further spiraling. The loop detector is cleared upon the advection of each new streamline.

Results and Discussion

We've implemented our loop-detection strategy as part of our Advess algorithm using Visual C++ on a Windows XP/Dell notebook (1.70 GHz, 512 Mbytes RAM). The Mebarki loop-detection method based on Delaunay triangulation[4] is the only other scheme designed for and integrated into evenly spaced streamline placement. We compared their algorithm (see www-sop.inria.fr/geometrica/team/Abdelkrim.Mebarki) with Advess on the same platform. Figure 1a shows a line integral convolution (LIC) image of a 400 × 400 synthesized flow field with five centers and four tightly spiraling foci, which demonstrates how well the loop-detection strategies handle both closed and spiraling streamlines. Figure 1 also shows five 1 percent density comparative placements of evenly spaced streamlines generated for the same flow field. Figure 1b shows the place-

Table 1. Time consumption (in seconds) measured for evenly spaced streamline placements.

Flow fields used for creating 1% density evenly spaced streamline placements	400 × 400 flow with centers and foci	661 × 481 flow with centers and foci	400 × 400 flow without centers or foci
Mebarki and colleagues' streamline placement algorithm, with Delaunay triangulation-based loop detection	4.286	3.647	N/A
Advess with our loop-detection strategy	0.280	0.986	0.262
Advess without any loop detection	0.671	N/A	0.224

ment created using Advess without any loop detection. Figures 1c and 1d show the placements produced using Advess with a naïve ineffective loop-detection scheme, which works via seed-based distance checking beyond a small seed-based curve length threshold and a large one, respectively. Figures 1e and 1f show the placements generated using the Mebarki algorithm and Advess coupled with our loop-detection strategy, respectively. The Mebarki approach might introduce loop artifacts or noticeable cluttering into a placement, whereas ours is robust enough to detect all ill-looping streamlines, resulting in an image showing important flow structures in a detailed but clean manner. Figure 4 uses a 661 × 481 ocean flow field to compare these two strategies and adds strength to this conclusion.

Figure 5 shows a 1 percent density evenly spaced streamline placement generated using Advess for a 400 × 400 flow field without centers or tightly spiraling foci. This synthesized flow demonstrates our loop-detection strategy's negligible additional cost. Given a flow field with centers and tightly spiraling foci, such as that in Figure 1, our strategy's additional cost is outweighed by the large expense of conducting intersample distance checking on excessive samples of ill-looping streamlines without loop detection. Table 1 gives the time consumption, in seconds, of creating evenly spaced streamline placements for the aforementioned three flow fields. Source code for Mebarki and colleagues' algorithm is unavailable, so we can't give a very precise comparison between their approach and ours purely in terms of detection speed, although this speed difference contributes a lot to the large difference between Mebarki's streamline placement algorithm and ours in overall expense.

At a negligible additional cost, our loop-detection scheme is fast enough to support interactive and robust layout of high-density evenly spaced streamlines for complex real-world flows. We plan to extend this strategy for view-dependent placement of evenly spaced streamlines on curved surfaces. This work will address image-space-oriented streamline placement for visualizing large flows in perspective-view settings.

Acknowledgments

This work was supported by the US Department of Defense HPCVI program. The authors thank Abdelkrim Mebarki for the online executable demo and *CiSE*'s editorial staff for their valuable comments and suggestions.

References

1. G. Turk and D. Banks, "Image-Guided Streamline Placement," *Proc. ACM SIGGRAPH*, ACM Press, 1996, pp. 453–460.
2. B. Jobard and W. Lefer, "Creating Evenly Spaced Streamlines of Arbitrary Density," *Proc. 8th Eurographics Workshop on Visualization in Scientific Computing*, Eurographics Press, 1997, pp. 45–55.
3. V. Verma, D. Kao, and A. Pang, "A Flow-Guided Streamlines Seeding Strategy," *Proc. IEEE Visualization 00*, IEEE CS Press, 2000, pp. 163–170.
4. A. Mebarki, P. Alliez, and O. Devillers, "Farthest Point Seeding for Efficient Placement of Streamlines," *Proc. IEEE Visualization '05*, IEEE CS Press, 2005, pp. 479–486.
5. Z. Liu, R.J. Moorhead II, and J. Groner, "An Advanced Evenly Spaced Streamline Placement Algorithm," *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 5, 2006, pp. 965–972.
6. T. Wischgoll and G. Scheuermann, "Detection and Visualization of Closed Streamlines in Planar Flows," *IEEE Trans. Visualization and Computer Graphics*, vol. 7, no. 2, 2001, pp. 165–172.
7. H. Theisel et al., "Grid-Independent Detection of Closed Streamlines in 2D Vector Fields," *Proc. 9th Int'l Fall Workshop on Vision, Modeling, and Visualization* (VMV'04), ACM Press, 2004, pp. 421–428.
8. B. Cabral and L. Leedom, "Imaging Vector Fields Using Line Integral Convolution," *Proc. ACM SIGGRAPH 1993*, ACM Press, 1993, pp. 263–270.

Zhanping Liu is a research scientist with the Visualization, Analysis, and Imaging Lab in the High-Performance Computing Collaboratory at Mississippi State University. His research interests are in the areas of scientific visualization and computer graphics. Liu has a PhD in computer science from Peking University. Contact him at zhanping@hpc.msstate.edu.

Robert J. Moorhead II is the Billie J. Ball Professor of Electrical and Computer Engineering at Mississippi State University and the director of its Visualization, Analysis, and Imaging Lab. He's also the associate director for HPC for the MSU GeoResources Institute. His research interests are focused around visualization of enormous data sets, remote visualization, and the effectiveness of visualization techniques. Moorhead has a PhD in electrical and computer engineering from North Carolina State University. Contact him at rjm@hpc.msstate.edu.