Farthest Point Seeding for Efficient Placement of Streamlines

Abdelkrim Mebarki* INRIA Sophia-Antipolis
Pierre Alliez† INRIA Sophia-Antipolis
Olivier Devillers‡ INRIA Sophia-Antipolis

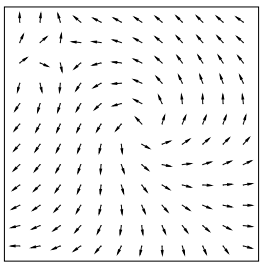
ABSTRACT

We propose a novel algorithm for placement of streamlines from two-dimensional steady vector or direction fields. Our method consists of placing one streamline at a time by numerical integration starting at the furthest away from all previously placed streamlines. Such a farthest point seeding strategy leads to high quality placements by favoring long streamlines, while retaining uniformity with the increasing density. Our greedy approach generates placements of comparable quality with respect to the optimization approach from Turk and Banks, while being 200 times faster. Simplicity, robustness as well as efficiency is achieved through the use of a Delaunay triangulation to model the streamlines, address proximity queries and determine the biggest voids by exploiting the empty circle property. Our method handles variable density and extends to multiresolution.

Keywords: Streamline placement, farthest point seeding, Delaunay triangulation, variable density, multiresolution.

1 INTRODUCTION

Vector and direction fields are commonly used for modeling physical phenomena, where a direction and magnitude, or a vector is assigned to each point inside a domain. A typical example of a vector field is given by the direction, orientation and velocity of a steady wind. Examples of direction fields include the principal curvature directions of a smooth surface. In this paper a *flow field* refers to either a direction or to a vector field.



Visual depiction of flow fields is motivated by the analysis and exploration of results in scientific computing. One popular method consists of choosing a set of samples throughout the field, and depicting associated *arrow icons* to present a view of the direction, orientation and magnitude in a single picture (see inset). The most delicate task of this technique lies into the choice of sample positions so as to best balance between sparse sampling for clear depiction at the risk of missing fine details, versus fine sampling at the risk of a cluttered visualization. For high range vector fields where the magnitude cannot be used to scale the arrow icons, the iconic representation is frequently composed of unit vectors instead, combined with color coding to depict magnitude. Other techniques have been developed to obtain a denser depiction by *imaging flow fields*, where the images are obtained by advection of a random noise image [2, 3, 4]. Conversely, some techniques propose to extract only salient features of the flow and depict them as geometric icons to facilitate comprehensive visualization [5].

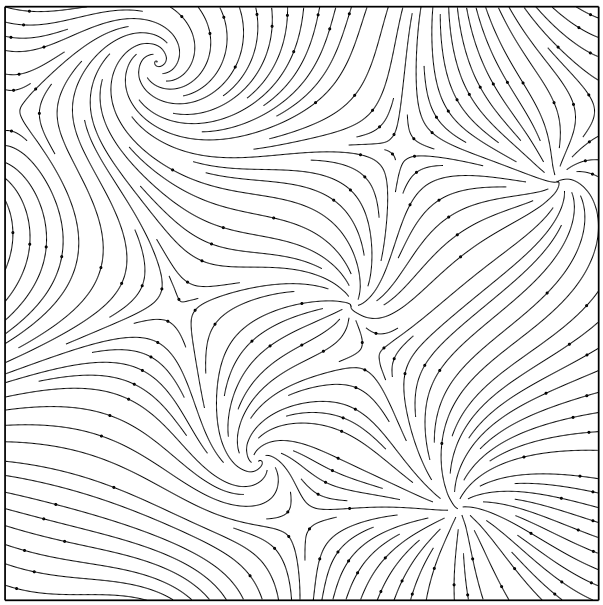


Figure 1: Placement of streamlines with a uniform density (with a tapering effect [1]). The black dots depict the seed points used for numerical integration. Processing time: 160 ms on a 2GHz Pentium IV.

One of the most popular methods in flow visualization consists of placing a set of *streamlines* which are always tangential to the flow in order to emphasize the global field coherency (see Fig. 1). Beside, high quality placement of streamlines has recently proven relevant for other applications such as non-photorealistic rendering [6, 7] or curve-based surface remeshing [8, 9]. We next give a few definitions and an overview of existing techniques for high quality placement of streamlines.

Definitions A *streamline* is a curve everywhere tangent to the field. A streamline can be considered as the path traced by an imaginary massless particle dropped into a steady fluid flow described by the field. In practice, a streamline is often represented as a polyline iteratively elongated by bidirectional numerical integration started from a *seed point*, until it comes close to another streamline, hits the domain boundary, reaches a critical point or generates a closed path. A *valid* placement of streamlines is obtained by saturating the domain with a set of tangential streamlines in accordance with a specified density. A *high quality* streamline placement for visualization has no formal definition, although it is admittedly related to the uniformity of streamlines as well as to the spacing with respect to the desired density. Moreover, long streamlines are preferred to short ones, the goal being to better emphasize the global field temporal coherency. At the intuitive level, and as already pointed in [10], most streamline terminations not coincident with the flow field singularities are perceived as artificial singularities and thus are potential distracters for an observer.

*Abdelkrim.Mebarki@sophia.inria.fr
†Pierre.Alliez@sophia.inria.fr
‡Olivier.Devillers@sophia.inria.fr

2 Related work

When constraining the streamlines to be tangential to the flow, the critical stage of any streamline placement algorithm reduces to the choice of seed points used to start the numerical integration. One trivial solution consists of choosing the seed points on a regular grid, but the resulting streamlines are not evenly spaced and some undesirable patterns frequently appear in the final placement. Another solution is to randomly generate the seed points with a uniform law, but has not shown to improve the placement quality nor to guaranty the domain saturation.

In a pioneering work, Turk and Banks [1] proposed an energy-minimizing approach to generate high quality placements. Their algorithm initially creates a set of so-called *streamlets* (*i.e.*, very short streamlines), then applies a series of energy-decreasing elementary operations to combine, delete, create, lengthen, or shorten the streamlets. To obtain a uniform density of streamlines, the energy to be minimized is related to the difference between a low-pass filtered version of the current placement and a uniform grey image. This approach generates high quality placements, with significant computation times due to the pliant aspect of the algorithm (a method is *pliant* if it incorporates both insertion and deletion, as well as local optimization, see [11]).

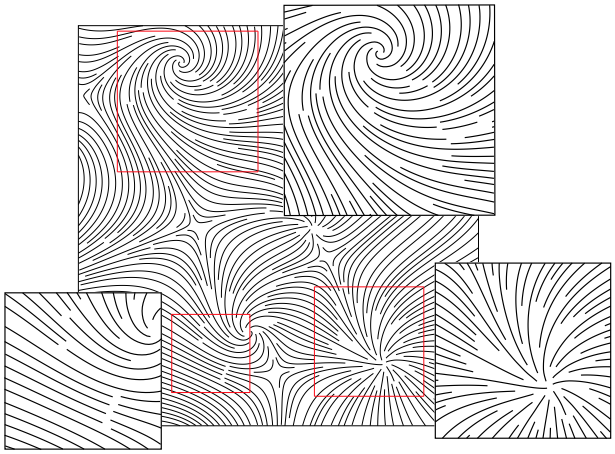


Figure 2: Local seeding strategy produces empty spaces due to the consecutive stopping of a series of streamlines (bottom right closeup). Some discontinuities also appear near singularities (top right closeup) and in laminar areas (bottom left closeup). Figure reproduced from [12].

Jobard and Lefer proposed an algorithm to create evenly spaced streamline placements [12]. Their solution is a greedy placement seeded *in the neighborhood* of previously placed streamlines. During the numerical integration of a streamline, a set of seed points are spread on both sides, and stored into a container of candidate seed points for future placements. The next streamline is placed from a valid seed point randomly chosen and popped out of the container. The algorithm terminates when no more valid seeds remain, *i.e.* when the container is empty. Although providing a good balance between efficiency and quality, some empty spaces may remain in the placement, and the length of streamlines is not fully satisfactory. A careful examination of a series of placements produced by this technique led us to characterize the undesirable effects produced by such a local seeding strategy. The first effect is a large empty space due to the consecutive stopping of a series of streamlines placed sequentially (see Fig. 2). These effects are more frequent when the flow is locally turbulent. Several discontinuities also appear near critical points as well as in homogeneous areas. Another undesirable effect is the concentration of a residual space incident to the last streamline of a sequence when there is not enough room to insert another streamline between the last one in the sequence and a previously placed streamline. One desirable effect would be to evenly distribute this residual on a larger area, even at the price of a more global deviation from the local ideal spacing.

To better depict the topology of the flow, Verma *et al.* [13] proposed a streamline placement technique where the streamline integration is seeded nearby critical points. In a final step the remaining space is filled by resorting to a random seeding strategy. Although this method provides placements that enhance the flow features and topology, it does not provide precise control over the density. Moreover, seeding streamlines nearby a singularity where the flow is often highly turbulent does not intrinsically favor long streamlines.

One important component for all greedy streamline placement algorithms is the data structure. It is commonly used to model the streamlines and the empty spaces, to accelerate the point location as well as the streamline-to-streamline distance queries. Some specific data structures have also been developed in order to cope with large data sets [14]. A streamline is frequently approximated by a polyline, each integration step adding a new point at one of its extremities. A common acceleration step consists of approximating the distance queries by simpler point-to-point queries, and the number of associated computations is often reduced by using regular boolean grids combined with region growing procedures [12, 13]. Unfortunately this choice is not suited to high density, and even less suited to variable density.

A visual qualitative study of existing streamline placement algorithms led us to distinguish all greedy methods from the pioneering optimization technique introduced by Turk and Banks [1]. The latter has not been really improved in terms of quality since 1996, while the greedy methods trade efficiency for a lower quality (they are typically two orders of magnitude faster).

3 Contributions

The main contribution of our work is a new *greedy* algorithm which improves over previous work by the efficiency of the placement without compromising the result quality, algorithmic simplicity, robustness and scalability. Our greedy method generates placements of comparable quality with respect to the optimization approach introduced by Turk and Banks [1], while being conceptually simpler and 200 times faster. To obtain a high quality placement with long and evenly spaced streamlines, our method departs from the usual greedy seeding strategies [12, 13] by choosing the seed points at the furthest away from *all* existing streamlines, namely at the center of the biggest voids within the domain. This more global criterion, already used successfully for point sampling and meshing [15, 16] drives the placement procedure by covering the biggest voids in priority. At the intuitive level, seeding a streamline at the farthest point favors *long* streamlines. Our experiments show that the same idea equally distributes the spaces between streamlines over the domain to obtain an evenly spaced placement, and is amenable to multiresolution placement by retaining the increased density at each newly inserted streamline.

Another important contribution of our work is the data structure. We use a Delaunay triangulation combined with a robust arithmetic from the Computational Geometry Algorithms Library [17]. It has proven both robust and efficient for point location and proximity queries, even for extreme ranges of density specified as input in our experiments. Notice that the Delaunay triangulation not only solves for proximity queries in our algorithm, but it also provides us with maximal empty circles, which are candidates for the biggest cavities. Finally, we drastically improve the efficiency of our algorithm by selecting only a small subset of all Delaunay circumcircle centers as candidate seeds for streamline integration. This optimization is shown to be not detrimental to the placement quality.

4 PLACEMENT OF STREAMLINES

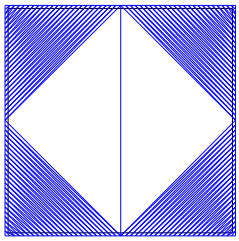
The *input* of our algorithm is given by (i) a flow *field*, (ii) a *density* specified either globally by the inverse of the ideal spacing distance, or locally by a density field, and (iii) a *saturation* ratio over the desired spacing required to trigger the seeding of a new streamline. The input flow field is given by a discrete set of vectors or directions sampled within a domain, associated with an interpolation scheme (*e.g.* bilinear interpolation over a regular grid, or natural neighbor interpolation over an irregular point set [18]) to allow for an evaluation at each point coordinate within the domain. The *output* is a streamline placement, represented as a list of streamlines.

The core idea of our algorithm consists of placing one streamline at a time by numerical integration seeded at the farthest point from all previously placed streamlines. The streamlines are approximated by polylines, whose points are inserted in a 2D Delaunay triangulation. The empty circumscribed circles of the Delaunay triangles provide us with a good approximation of the cavities in the domain. After each streamline integration, all incident triangles which circumcircle diameter is larger (within the saturation ratio) than the desired spacing distance are pushed to a priority queue sorted by the triangle circumcircle diameter. To start each new streamline integration, the triangle with largest circumcircle diameter (and hence the biggest cavity) is popped out of the queue. We first test if it is still a valid triangle of the triangulation, since it could have been destroyed by a streamline previously added to the triangulation. If not, we pop another triangle out of the queue. If yes, we use the center of its circumcircle as seed point to integrate a new streamline (see Fig. 4). Our algorithm terminates when the priority queue is empty. The size of the biggest cavity being monotonically decreasing, our algorithm guarantees the domain saturation. We summarize our algorithm in the following pseudo-code:

```
PLACEMENT(field,density,saturation)
Parameters: field (vector or direction field)
density (inverse of spacing between streamlines)
saturation (ratio over the spacing, > 1)
Variables:
Visualization_domain domain
Delaunay_triangulation triangulation
list<streamline> placement
priority_queue<triangle> queue

Algorithm:
begin
    initialization()
    streamline s=PLACE_STREAMLINE(domain.center())
    ADD_STREAMLINE(placement,s)
    while (!queue.empty())
        triangle t=queue.pop()
        if (t still in triangulation)
            s=PLACE_STREAMLINE(t.circumcenter())
            ADD_STREAMLINE(placement,s)
        endif
    endwhile
    return placement
end
```

initialization() This function inserts in the triangulation a set of points uniformly sampled on a rectangle surrounding the domain boundary(see inset) . The rectangle is constructed by enlarging the domain boundary with a distance equivalent to the desired separating distance between streamlines. This step is required to ensure a proper domain saturation.



place_streamline() The streamlines are integrated both forward and backward from a seed point. The integration, performed using first order Euler or second order Runge-Kutta [19], is stopped when the newly integrated point p is located outside the domain, or when the distance between p and the current placement (including part of the current streamline) is smaller than the separating distance. In the next section we explain how we *approximate* this distance instead for the sake of efficiency.

```
PLACE_STREAMLINE(seed)

begin
    point p=seed
    streamline s
    do % forward integration %
        s.insert(p)
        triangulation.insert(p)
        p=integrate_forward_from(p)
    while(domain.inside(p) and
            distance(p,placement) < 1/density)
    do % backward integration %
    ...
    return s
end
```

4.1 Adding a Streamline

The function ADD_STREAMLINE($placement$, $streamline$) inserts the newly integrated streamline $streamline$ to the placement, and subsequently pushes the set of triangles incident to $streamline$ in the priority queue. To reduce the number of triangles pushed in the queue, all triangles which circumcircle diameter is smaller than $saturation\ ratio/density$ are not inserted to the queue. Since the saturation ratio is greater than 1 (we used 1.6 in all our experiments), the latter procedure allows us to avoid the generation of short streamlines when the domain is close to be saturated.

```
ADD_STREAMLINE(placement,streamline)

begin
    placement.insert(streamline)
    for_each(t incident to streamline)
        if (t.circumradius is_local_maximum and
              t.circumradius > saturation × 1/density)
            queue.insert(t)
        endif
    endwhile
end
```

4.2 Approximating the Distances

When choosing the integration step significantly smaller than the desired separating distance, a good approximation of the distance between streamlines is given by the minimal distance between the newly integrated point and all other points of the previously integrated streamlines. Notice that when the current streamline comes back to itself or spirals (see Fig.3, right), things become more complicated since the distance computation should also include some – but not all – points of the current streamline (adding all points would always return the integration step as distance). To alleviate this problem and reduce the number of distance computations as well, we insert all integrated points in a Delaunay triangulation, and approximate the distance by the *minimal circumcircle diameter* of the triangles incident to the newly integrated point. The skinny triangles due to the ratio between the integration step and the desired separating distance provide us with a sufficiently accurate approximation of the distance with both regular and spiraling streamlines (see Fig.3).

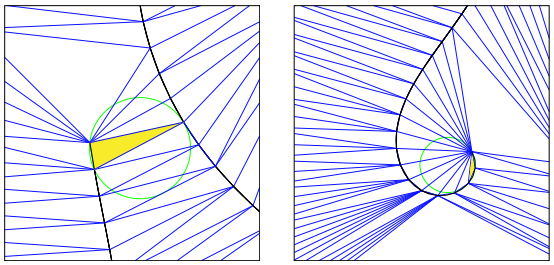


Figure 3: The separating distance between streamlines is approximated using the minimal circumdiameter of the triangles incident to the newly integrated point.

4.3 Optimizations

Although the previous section completes the description of our basic algorithm, there is room for improving efficiency. As mentioned in Section 4.1, adding a streamline to the placement leads to the insertion of all its incident triangles to the priority queue. The priority queue is therefore quickly populated with a large number of triangles. Moreover, a lot of triangles in the queue are in fact short-lived in the triangulation, since placing a new streamline breaks all triangles on its way in the triangulation (see Fig.5).

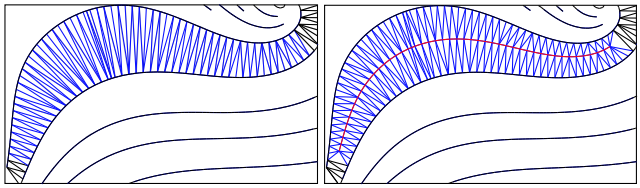


Figure 5: Each newly integrated streamline breaks all triangles on its way.

The vast majority of triangles pushed in the queue will therefore be invalidated when popped out of the queue because they do not belong to the triangulation anymore. To reduce the number of triangles inserted to the priority queue, we walk along the right side of the newly placed streamline, examine the associated sequence of circumcircle radii of all incident triangles, and insert to the queue the triangles which correspond to local maxima. We apply the same procedure for the left side, and finally insert all triangles incident to the streamline extremities to ensure a proper domain saturation. With almost no impact on the placement quality (see Fig.6), such an optimization reduces on average by 30 the number of triangles pushed in the queue, and therefore produces a significant speed-up of our algorithm. Notice that this step does not disrupt the saturation guarantee, since each integrated streamline creates a new set of triangles and hence new local maxima as candidate seed points.

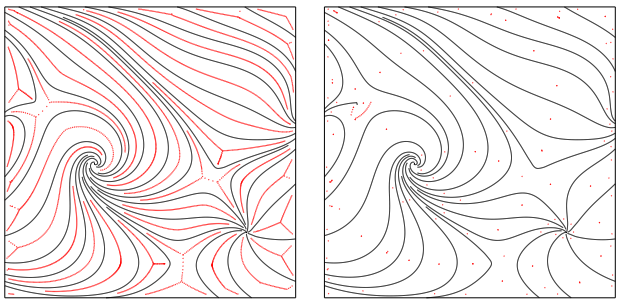


Figure 6: Seed points pushed in the priority queue at an intermediate step of the algorithm, when considering all triangle circumcenters (left), and when choosing only the ones corresponding to local maxima of circumcircle radii (right).

4.4 Variable density

Our algorithm can take as input a non uniform density field, or simply evaluate any function of the flow such as velocity or vorticity. The desired separating distance simply becomes a function of any point coordinates within the domain (see Fig.7). The triangulation data structure has proven particularly efficient and robust at handling even extreme ranges of density (up to 20K streamlines composed of 8M points in our experiments).

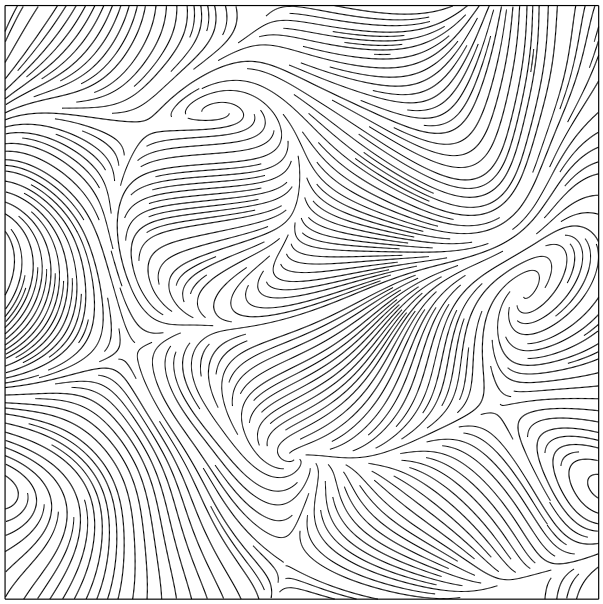


Figure 7: Placement of streamlines with a variable density, in this example related to the flow velocity.

4.5 Multiresolution

Visual exploration of complex flow fields may require multiresolution streamline placement. A typical exploration scenario is a rough depiction of the field on the whole domain, followed by a focused depiction on the regions of interest. One method proposed by Jobard and Lefer [20] performs a sequence of nested streamline placements with increasing density. The streamlines placed at a given level are frozen and used as additional constraints for the finer levels. This approach produces short streamlines added to fill the gaps in-between the streamlines placed at coarser levels. Our method improves over the quality and smoothness of transitions between levels by *elongating all previously placed streamlines* after each increase of the density and before each new placement of streamlines (see Fig.8). The pseudo-code for the multiresolution version of our algorithm is as follows:

```
MULTIRESOLUTION(number_of_levels)

begin
    placement=PLACEMENT(field,density,saturation)
    For_each_level_do
        increase density
        placement.lengthen_streamlines()
        placement.place_new_streamlines()
        return placement
    end_do
end
```

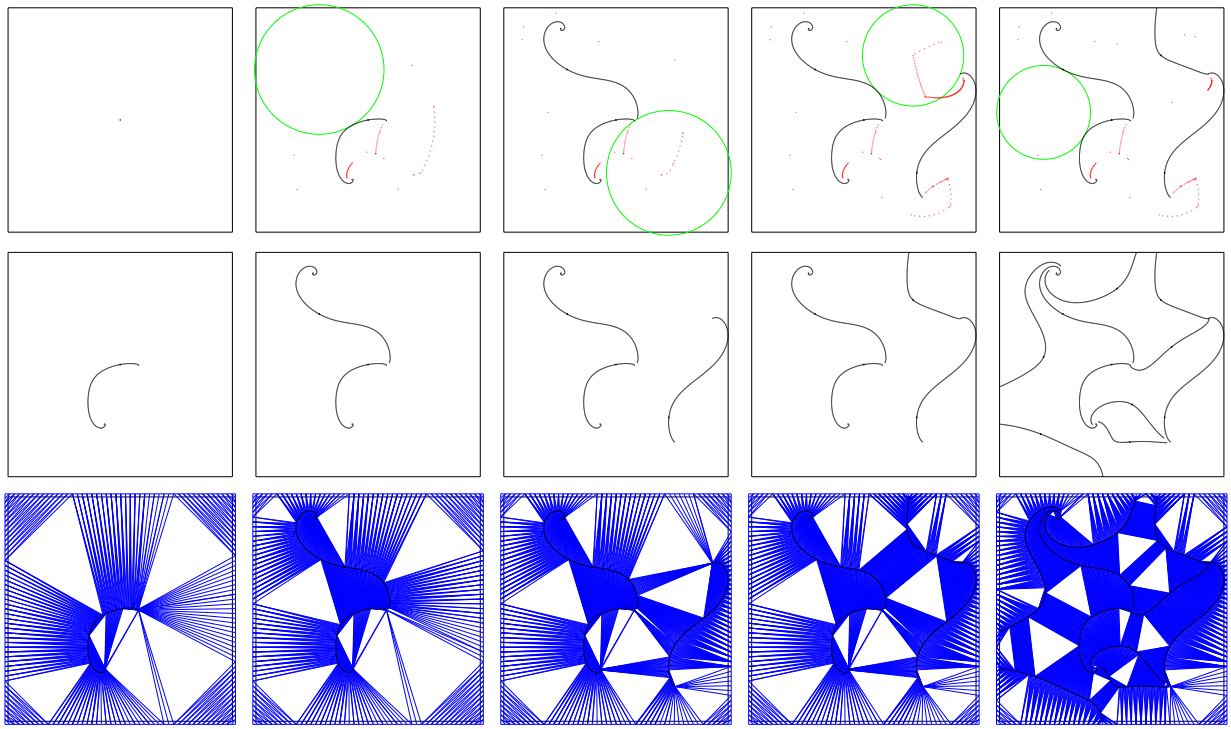


Figure 4: The intermediate stages of our streamline placement algorithm. The biggest cavities are modeled by the green circles. The triangulation corresponding to each placement is depicted in blue, and the candidate seed points are depicted in red. The final result is shown in Fig.1.

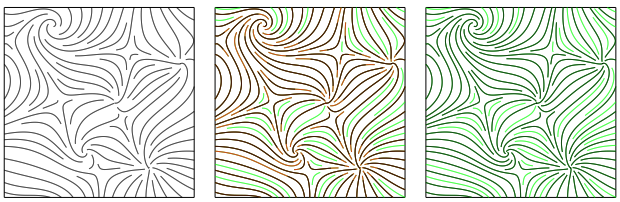


Figure 8: Multiresolution placement of streamlines: From a given initial low density, the following sequence is applied: (a) place streamlines (black parts), increase density, (b) lengthen existing streamlines (red parts), (c) place streamlines (green streamlines), etc..

4.6 Implementation

We have implemented our algorithm in C++. The 2D Delaunay triangulation and other components for geometric computing are provided by the Computational Geometry Algorithms Library [17]. The priority queue is taken from the Standard Template Library [21]. Part of the code provided by Greg Turk has been reused to read the vector fields from files as discrete sets of 2D vectors sampled on a regular grid. Overall our implementation takes no more than 2000 lines of code, with 1600 for the placement algorithm itself (remaining part is devoted to file input/output, debugging and display). For comparison, we have also implemented the algorithm from Jobard and Lefer [12] using our data structure for efficient evaluation of distance queries. Our algorithm is currently being prepared to be submitted as a CGAL package.

5 Results

Fig.9 depicts a streamline placement computed from a real dataset of the wind in Japan (provided by Weathernews Inc., Japan). The 2D vectors of the input dataset are non uniformly sampled and correspond to 522 wind sensors widespread on a large area surrounding the country (few sensors in the sea, lots of them on land). The domain boundary is delimited by the convex hull of the sensors. The vector field is obtained by natural neighbor interpolation over the sensor point locations. This result illustrates the behavior of our algorithm both on turbulent and on laminar areas. The close-ups are taken from turbulent regions and illustrate placements with an increased density of streamlines.

Fig.10 depicts comparative results between Turk-Banks [1], Jobard-Lefer [12] and our algorithm. All pictures from the first two columns are reproduced from the original papers. We now list the noticeable differences for several criteria.

Domain saturation Our algorithm saturates the domain by construction. The two other placements also saturate the domain, except for Jobard and Leffer's placements where some free spaces remain, in particular nearby singularities in the densest placement.

Flow coherency A high quality placement minimizes the number of discontinuities to better emphasize the flow coherency. Turk-Banks and our algorithm exhibit similar results for both sparse and dense cases. The denser, the more discontinuities appear in the results produced by Jobard-Lefer. Since our streamlines are started at the farthest point from all other streamlines, our algorithm tends to generate streamlines which are mostly stopped by critical points or by domain boundaries at the begining, then the remaining streamlines are stopped mostly by other streamlines when the overall density increases. This partially explains the behaviour of our placement algorithm which nicely wraps the singularities (see Figure 1).

Uniform density Some residual spaces remain in Jobard-Lefer algorithm by placing new seeds at the local ideal spacing distance. Although unavoidable, these spaces are more equally distributed in Turk-Banks and our placements. In essence our algo-

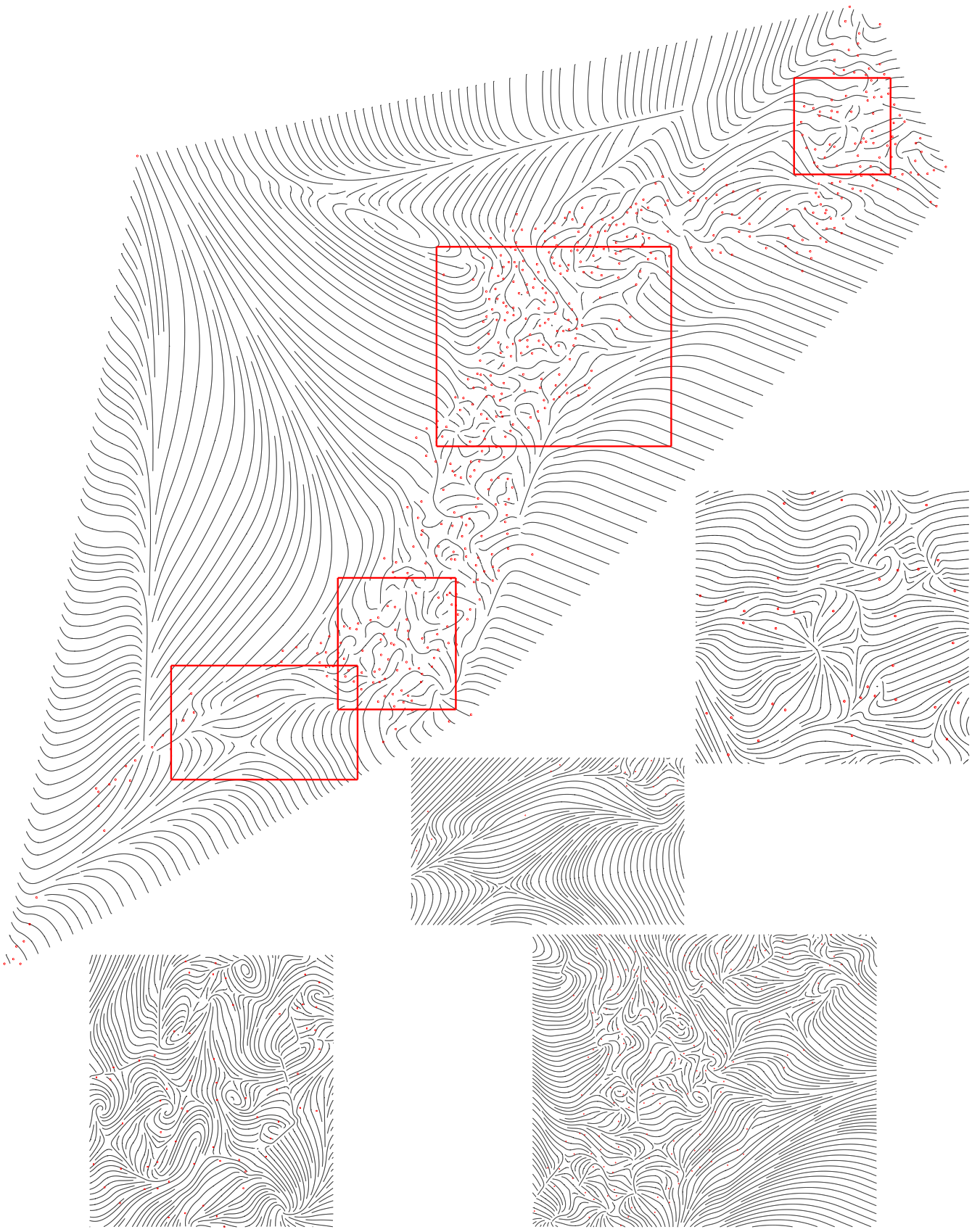
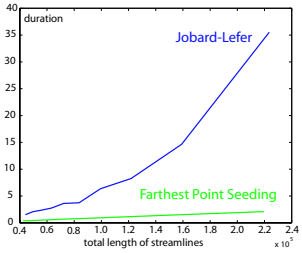


Figure 9: Visualization of wind in Japan estimated at 522 non-uniform wind sensors locations (sensor positions are depicted as red dots). The close-ups are taken from the most turbulent areas and have been generated using an increased density of streamlines. Notice that the laminar areas are explained by the sparsity of wind sensors on the sea.

rithm does not intend to match the *desired* local density perfectly, and produces more locally uniform placements instead.

Timings and memory Table 1 summarizes the timings measured on a 2GHz Pentium IV. As already mentioned in Section 4.6 we have implemented the algorithm from Jobard and Lefer [12] to obtain fair measures on a modern computer. For Turk and Banks algorithm, we have executed their software available for download [1]. Our algorithm improves over Jobard-Lefer algorithm by the quality of placements, within shorter times. It produces comparable results with Turk-Banks algorithm, while being two orders of magnitude (200 times) faster. Notice that our implementation of Jobard-Lefer algorithm with a Delaunay triangulation data structure is approximately twice as fast as an implementation similar to the original.

Scalability. We compare the scalability of our algorithm with Jobard-Lefer greedy algorithm (see inset). To quantify scalability we measure the duration (in s) with respect to the total length of streamlines already placed (expressed in number of integration steps). Our algorithm exhibits a linear behavior whereas our implementation of Jobard-Lefer algorithm does not, mainly due to the quickly increasing number of seed points generated (our method uses a sparse priority queue instead).



Separating dist. (% width)	Our algorithm	Jobard-Lefer [12]	Turk-Banks [1]
0.84	240 ms	790 ms	47 s
1.68	140 ms	290 ms	36 s
3.36	80 ms	130 ms	20 s

Table 1: Timings measured on a 2GHz Pentium IV. For each density our algorithm is respectively 195, 257 and 250 times faster than Turk-Banks for a comparable quality.

6 Conclusion

A novel greedy algorithm for high quality placement of streamlines is proposed. Our algorithm favors the generation of long streamlines by using a farthest point seeding strategy, and ensures the domain saturation by construction. It is simple, efficient, deterministic and involves only two basic algorithmic components which are a priority queue and a Delaunay triangulation both available in widespread libraries [21, 17]. The main strength of our method lies into the balance between efficiency and placement quality; our greedy algorithm produces results of comparable quality with respect to the state-of-the-art optimization-based method from Turk-Banks, while being on average 200 times faster (timings are measured on the same computer). Most functions that trigger the algorithm sequence such as farthest point localizations and distance queries are built upon the Delaunay triangulation. Once parameterized with the appropriate arithmetic, the latter provides us with robustness and scalability.

As future work we plan to add a preliminary step to our algorithm by detecting all closed streamlines as recently proposed by [22], and inserting them to the initial placement. We also wish to extend our algorithm to 3D vector fields. One possible improvement would be to add one final optimization stage to our algorithm, so as to trade off speed for placement quality. Finally, a stimulating challenge would be to elaborate upon a streamline placement algorithm for time-varying flows.

[1] http://www.cc.gatech.edu/~turk/streamlines/streamlines.html

Acknowledgments

The authors thank Greg Turk for the streamline package available for download, Bruno Jobard for providing us a copy of his Ph.D. thesis and a set of vector fields, Xavier Tricoche for discussion, and David Cohen-Steiner for help with the distance approximation.

References

[1] G. Turk and D. Banks. Image-Guided Streamline Placement. In *ACM SIGGRAPH Conference Proceedings*, pages 453–460, 1996.

[2] J. J. van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 309–318, 1991.

[3] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270, 1993.

[4] RS. Laramee, H. Hauser, H. Doleisch B. Vrolijk, FH. Post, and D. Weiskopf. The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2004.

[5] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. Feature extraction and visualization of flow fields. In *Eurographics 2002 State-of-the-Art Reports*, pages 69–100. European Association for Computer Graphics, The Eurographics Association, 2002.

[6] C. Rössl and L. Kobbelt. Line-art rendering of 3d-models. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, 2000.

[7] J. Zander, T. Isenberg, S. Schlechtweg, and T. Strothotte. High Quality Hatching. *Computer Graphics Forum (Proceedings of Eurographics)*, 23(3):421–430, 2004.

[8] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Levy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics. Special issue for SIGGRAPH*, pages 485–493, 2003.

[9] M. Marinov and L. Kobbelt. Direct anisotropic quad-dominant remeshing. In *Proceeding of Pacific Graphics, Seoul*, 2004.

[10] B. Jobard. *Visualisation de champs de vecteurs bidimensionnels à base de streamlines*. PhD thesis, Univ. du Littoral Côte d'Opale, 2000.

[11] FJ. Bossen and PS. Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, pages 63–74, 1996.

[12] B. Jobard and W. Lefer. Creating Evenly-Spaced Streamlines of Arbitrary Density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55, 1997.

[13] V. Verma, D. T. Kao, and A. Pang. A Flow-guided Streamline Seeding Strategy. In *IEEE Visualization*, pages 163–170, 2000.

[14] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, 1997.

[15] J.-D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *Proc. 1st Symp. on Geom. Proc.*, pages 9–18, 2003.

[16] M. Lindenbaum, M. Porat, Y. Y. Zeevi, and Y. Eldar. The farthest point strategy for progressive image sampling, 1996.

[17] CGAL: Computational Geometry Algorithms Library. www.cgal.org.

[18] J. Flötotto. CGAL interpolation package. www.cgal.org.

[19] W. H. Press, W. T. Vetterling, S. A. Teukolsky, and B. P. Flannery. *Numerical Recipes in C++: the art of scientific computing*. Cambridge Univ. press, 2 edition, 2002.

[20] B. Jobard and W. Lefer. Multiresolution flow visualization. In *WSCG 2001 Conference Proceedings*, 2001.

[21] M. H. Austern. *Generic Programming and the STL*. Addison-Wesley, 1998.

[22] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Grid-independent detection of closed stream lines in 2d vector fields. In *VMV 2004*, 2004.

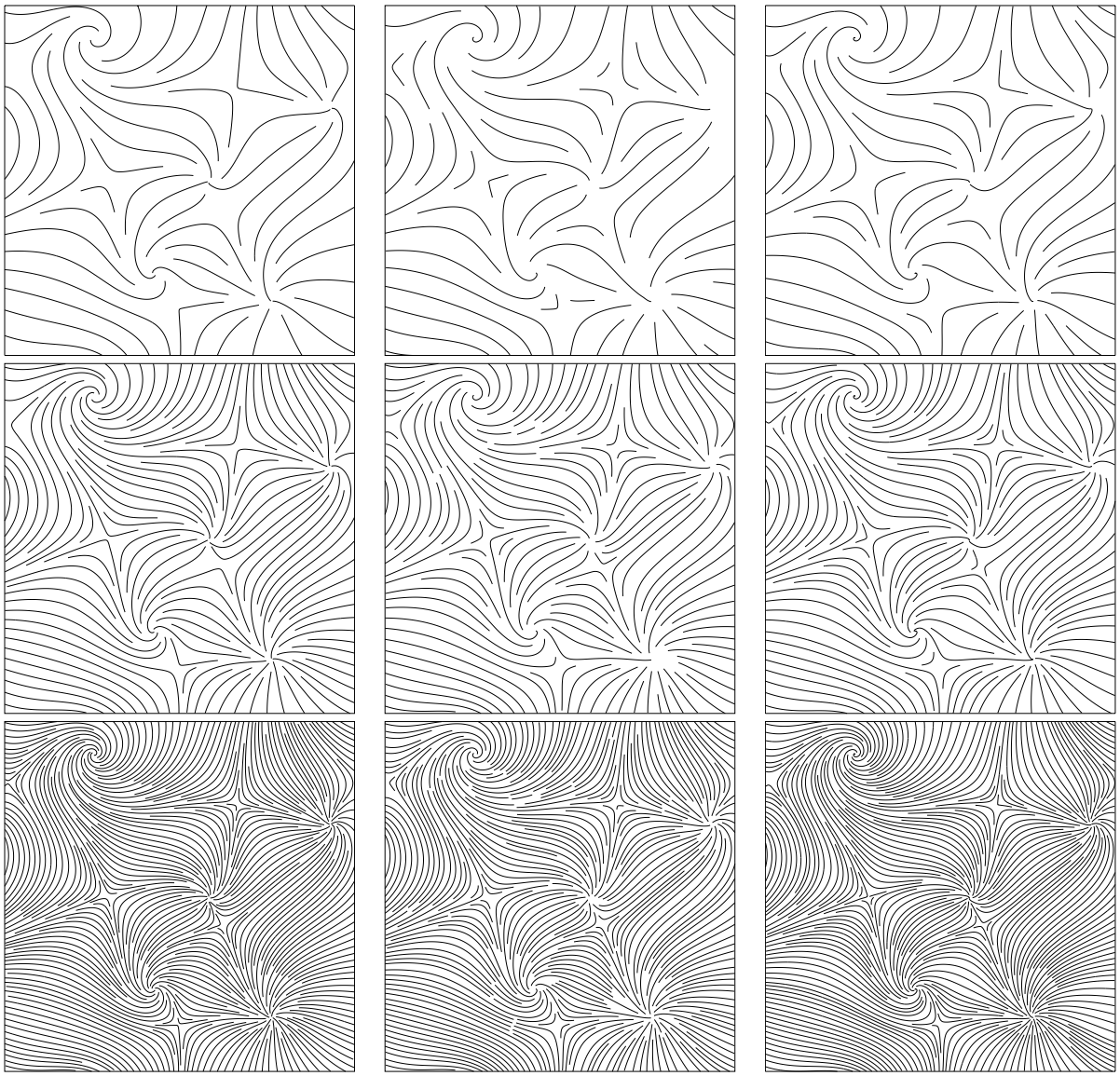


Figure 10: Streamline placements generated by Turk-Banks [1], Jobard-Lefer [12], and by our algorithm for three increasing densities (top to bottom, 3.36%, 1.68%, 0.84% of the flow width requested as separating distance).