Simplified Representation of Vector Fields

Alexandru Telea* Jarke J. van Wijk*

Eindhoven University of Technology

Abstract

Vector field visualization remains a difficult task. Many local and global visualization methods for vector fields such as flow data exist, but they usually require extensive user experience on setting the visualization parameters in order to produce images communicating the desired insight. We present a visualization method that produces simplified but suggestive images of the vector field automatically, based on a hierarchical clustering of the input data. The resulting clusters are then visualized with straight or curved arrow icons. The presented method has a few parameters with which users can produce various simplified vector field visualizations that communicate different insights on the vector data.

Keywords: Flow Visualization, Simplification, Clustering

1 INTRODUCTION

Visualization of vector data produced from application areas such as computational fluid dynamics (CFD) simulations, environmental sciences, and material engineering is a challenging task. Even though many vector data visualization methods have been developed and are in widespread use, gaining insight in complex vector fields is still difficult. In comparison with scalar data, vector fields have an inherently higher complexity and therefore pose understanding and representation problems which are more difficult to address.

As vector datasets obtained from simulations grow increasingly larger and as they often have to be presented to non-specialized audiences, there is a demand for methods that display in an effective and compact manner, such that insight in the global behavior and also a good understanding of local effects is quickly obtained. A good inspiration for such visualizations is commercial imagery displaying fluid flow or weather forecasts, which convey the flow information in an intuitively understandable way using a few simple but suggestive icons, combined with a well chosen background. It would be desirable to have a method to produce such visualizations automatically from flow datasets. However, the most usual actual procedure requires a strong input from human users that have to construct the simplified visualization by tuning a multitude of parameters in a visualization package or manually place icons or probes such as streamline starting positions.

*Eindhoven University of Technology, Dept. of Mathematics and Computing Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: {alext, vanwijk}@win.tue.nl

We present a compact visualization method for vector fields which produces simplified flow displays automatically. The user intervention is limited to a few parameters which allow an intuitive control of the look or complexity of the resulting image. Our method combines several ideas from existing visualization techniques, outlined in Section 2. Section 3 presents the method. The behavior of the presented algorithm under various input is outlined in Section 4, and its implementation and integration in an end-user environment is presented in Section 5. We conclude the paper with further research directions.

2 BACKGROUND

Although effective, existing methods for flow visualization still have limitations. There are several problems involved with them:

1. Although such methods generate richly detailed and shaded pictures, the process of generating them can be time-consuming and intricate to set-up and control. Considerable trial and error and knowledge of the techniques' implementation is required to produce images that communicates the desired insight. Methods that generate such images automatically or with minimal user intervention would thus be highly useful.

2. Users must take subjective decisions while producing a visualization, e.g. which streamline start positions to select, how fine to sample a field plotted by a hedgehog, what colormap to use, etc. Visualizations in which users can and have to control all these parameters are prone to miss important aspects in the data due to uninspired tuning of these parameters.

3. Dataset aspects can be missed also when observing a visualization if too much or poorly structured information is presented. Arrow plots show for example all the dataset information but are often hard to interpret visually.

4. Datasets such as those produced from CFD simulations grow larger and larger, exceeding the visualization software capabilities and the the observation power of the users viewing them.

Numerous techniques have been developed for flow visualization. However, the above problems are usually only partially addressed.

The most ubiquitous flow visualization method uses hedgehogs or similar glyphs. Hedgehog plots are intuitive but impractical for large 2D or 3D datasets due to the visual difficulty to perceive dense arrow renderings. Although subsampling can be used to reduce the arrow count, this often introduces visual artifacts and may not provide the best arrow distribution to convey insight in the dataset. For example, the hedgehog visualization in Fig. 1 a) is unclear in the region where the flow field exhibits a bifurcation. To avoid such problems, flow visualizations such as commercial advertisements or weather forecasts that must convey insight to a more diverse and often untrained audience are produced manually by a placing a few glyphs (e.g. curved arrows) over the flow regions where something 'interesting' happens. Other global techniques include

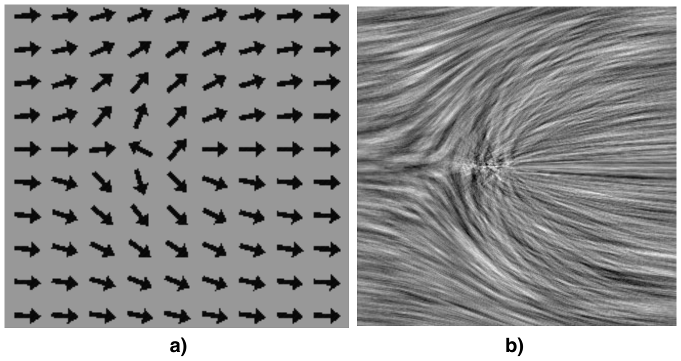


Figure 1: Flow visualized with hedgehog (a) and spot noise (b)

texture-based methods such as spot noise and line integral convolution [3, 2]. Although these give good local insight and global coverage, they don't show the flow direction (see Fig. 1 b) and thus may be difficult to interpret for spatially rapid changing flows.

Advection based techiques such as streamlines, streamtubes, particles, or flow ribbons [13, 8] give good local insight, but require strong input from the user (e.g. the positions to advect from), so are less adequate for getting a global impresion of a flow.

Vortex and feature extraction techniques [12, 1] simplify large flow datasets efficiently into a few features that are eventually tracked in the time dependent cases [7] and displayed using various iconification methods. Although compact, such techniques often require user knowledge to e.g. set various parameters to control feature detection and tracking. Since feature extraction methods convey insight in a particular flow feature, which is identified and then specifically tracked, they may be ineffective in producing a global flow picture. The above can be said also about visualization tools based on selection expressions [11].

Topological field analysis [4] is an excellent, ideally automatic way to produce compact and mathematically insightful flow representations. The produced visualizations are however sometimes too abstract to be directly interpretable by non-specialists and can be unstable for fields with many critical points.

As the quality of user-steered visualizations may depend on a good guess of the steerable parameters, some work was targeted at automated visualization. However, the most widespread metaphor remains the interactively steerable visualization system in which the user experiments with different visualization techniques and parameter settings to visually fine tune the rendering until the desired images are obtained.

Visualization methods can be applied either directly on the datasets, or on simplified datasets. Simplified datasets contain less data points, so visualizing them generates less visual problems such as cluttering. The flow features of interest must however be recognizable in the simplified datasets. Multiresolution techniques such as Fourier or wavelet based methods [5, 6, 14] represent datasets as hierarchies with different levels of detail. However useful, such methods simplify the entire domain uniformly so they can not produce visualizations containing *both* global and detailed information in various parts of the *same* image.

The challenge we identify is to generate insightful flow visualizations as automatically as possible offering a good local insight, global coverage, directional insight, and overall a simple and intuitive perception of the flow.

3 SIMPLIFICATION

Our aim to produce intuitive flow visualizations automatically from a flow dataset exploits the observation that (curved) arrow plots, like the ones used by weather forecasts or commercial imagery are easy to interpret. An arrow carries a visually unambiguous representation of the direction and curvature of the flow over the region covered by and immediately around its drawing. A second observation is that most of the visualizations of the above mentioned types use a few large arrows to indicate the main directions of the flow and optionally small, detail arrows to depict local behaviour. The large arrows have a stronger visual impact and thus communicate the important flow attributes quickly to the spectator by visually filling in a larger space on the illustration, while the smaller arrows cover only smaller illustration areas, and thus have a limited and well confined visual impact. The same separation of information visualization in larger, structurally more important elements and smaller detail elements can be also seen in the construction of urban maps, for example. In our case, our goal can be summarized as 'represent a given vector field with a given arrow count in the most suggestive manner'.

3.1 Algorithm Principle

To model the above we use the *cluster* concept. We define a cluster as a connected subdomain of a flow dataset on which the flow is approximated for visualization purposes by a single (curved) arrow, called the cluster's representative. We visualize a flow dataset on a given domain by covering that domain with a (small) number of clusters such that their representatives are as close as possible to the given field's vectors. For the time being, we shall assume that the representatives are straight arrows, so every cluster carries actually a vector.

```
ClusterSet s;
for (all cells $cell_i$ in dataset)
{
    c = makeCluster($cell_i$);
    set level of c to 0;
    add c to s;
}

for (all clusters $c_i$ in s)
  for (all clusters $c_j$ neighbours of $c_i$)
  {
     e = clustering_error($c_i$,$c_j$);
     insert pair ($c_i$,$c_j$) in increasing order
     of error e in a hash-table;
     mark $c_i$ and $c_j$ as NOT_CLUSTERED;
  }

int l=0;
for (all pairs ($c_i$,$c_j$) in increasing order of
     error in the hash-table)
  if (both $c_i$ and $c_j$ are NOT_CLUSTERED)
  {
     c = mergeClusters($c_i$,$c_j$);
     set level of c to l++;
     mark $c_i$ and $c_j$ as CLUSTERED;
     for (all neighbours $n_i$ of c)
     {
       e = clustering_error(c,$n_i$)
       insert pair (c,$n_i$) in order in hash-table;
     }
  }

return c as root of tree;
```

Figure 2: Clustering algorithm

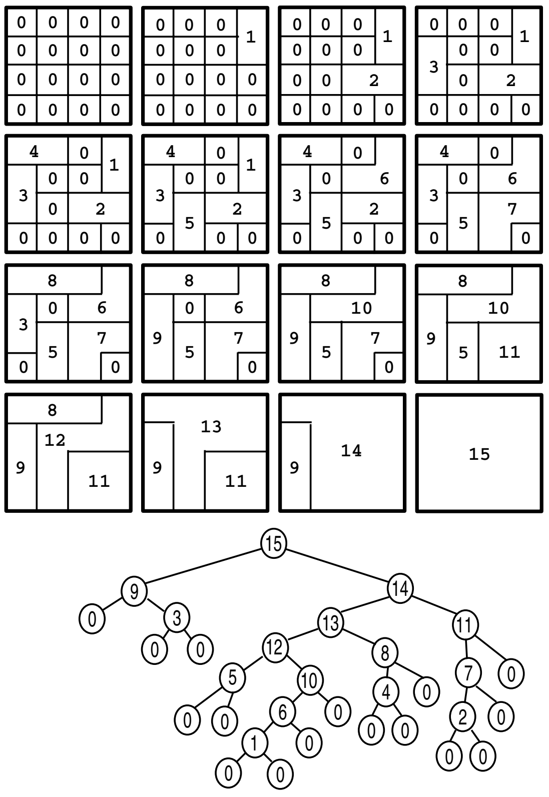


Figure 3: Clustering algorithm example

Our input is a vector field given as a mesh of cells with one vector per cell. Datasets with node vector data can be easily converted to cell data by averaging the node data over cells. The clusters are in this case connected sets of cells, so two clusters are neighbors if they contain cells that are neighbors, i.e. share an edge in 2D or face in 3D. We start creating one initial cluster from every dataset cell, having the cell's vector data as its representative, with the origin in the cell's center. Next, we perform a bottom-up clustering algorithm by repeatedly selecting the two most resembling neighboring clusters and merging them to form a larger cluster until a single cluster emerges which covers the whole given domain and represents it by a single arrow. The algorithm produces a binary cluster tree with the initial clusters as leaves and the root as the final cluster covering the whole domain. Merging two clusters produces a new cluster covering the union of its parents and having a representative computed by averaging its parents' representatives. Initial clusters represent thus the dataset exactly, while clusters higher in the tree have higher representation errors.

To remember at which stage of the clustering a given cluster was produced, every cluster in the tree has a level attribute. This information will be used for the visualization of the clustered data described in Section 4. Leaves have level 0, while the level of other nodes is computed from an integer incremented during the clustering algorithm. Since we use binary clustering, the algorithm will do $N-1$ steps and the root's level will thus be $N-1$, where N is the number of cells in the initial dataset. The pseudocode for the clustering algorithm is shown in Fig. 2. After the clustering is completed, we can easily examine the dataset at a simplification level l by displaying only the clusters with levels smaller or equal to l that have parents with levels greater than l.

Figure 3 shows the 16 steps taken by the algorithm on a hypothetical 4x4 2D regular grid and the produced cluster tree.

The algorithm's key is the evaluation of the similarity of two clusters (function *clustering_error*() in the pseudocode) and the merging procedure *mergeClusters*(). The fact that neighbor clusters are merged in the increasing order of this error function, i.e. in decreasing order of similarity, ensures that the obtained clustering minimizes the representation error, which is thus known at all levels. These operations are presented in the next Section.

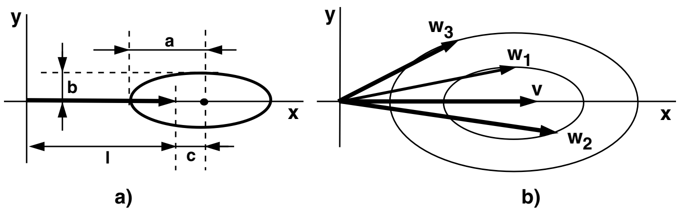


Figure 4: Elliptic similarity function

3.2 Similarity Evaluation and Cluster Merging

The clustering algorithm is driven by two main elements: the cluster similarity function, i.e. how we find the best clusters to merge, and the merging operation itself, i.e. how a new cluster is made by merging two existing ones. We present first the similarity function and then the merging procedure. The similarity function evaluates how similar two clusters are, and thus gives implicitly an estimate of the representation error one would get by merging them. This is so since two identical clusters, i.e. with equal representatives and sizes, can be obviously merged with no error into a larger cluster with the same representative. We compare clusters by comparing their representative vectors so we do not explicitly use the clusters' shapes. We split the comparison of these vectors into two parts: the direction and magnitude comparison and the position comparison.

To estimate the direction and magnitude similarity, we introduce first the notion of iso-error contours. Given a 2D vector $\mathbf{v}$ of length l taken along the x axis for simplicity, we define an iso-error contour of a given value as being the locus of the apexes of all vectors $\mathbf{w}$ with the same position as $\mathbf{v}$ that are equally similar to $\mathbf{v}$. The position difference will be treated separately. Based on the remarks starting this section, we would like the similarity function to acount for similar directions, so we model the iso-error contours as ellipses with the grand axis aligned with $\mathbf{v}$ and centered along $\mathbf{v}$ at a distance c from its apex (Fig. 4 a). The ellipse's size (half-axes a, b) and center-to-vector apex distance are functions of the error magnitude such that larger ellipses having the same aspect ratio correspond to larger error magnitudes. For example, in Fig. 4 b $\mathbf{w}_1$ and $\mathbf{w}_2$ are equally similar to $\mathbf{v}$, but more similar than $\mathbf{w}_3$, since $\mathbf{w}_3$'s apex is on a larger iso-error elliptic contour.

As outlined, the parameters a, b, c of an elliptic contour are increasing functions of the contour's error t. We can model such a contour by an equation $f(x, y, t) = 0$ with

$$f(x,y,t) = [\frac{x-c(t)}{a(t)}]^2 + [\frac{y}{b(t)}]^2 - 1 \qquad (1)$$

and the parameters a, b, c by linear functions of t

$$\begin{aligned} a(t) &= \alpha t \\ b(t) &= \beta t \\ c(t) &= l + \gamma t \end{aligned} \qquad (2)$$

Solving $f(x, y, t) = 0$ for t leads to a second-order equation in t. We impose the condition $t > 0$ since we are interested in positive error values and we find t as a function of x, y, the vector length l

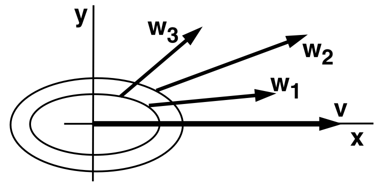


Figure 5: Vector position similarity function

and the coefficients α, β, γ.

$$t = \frac{\sqrt{((x-l)^2 + \frac{\alpha^2}{\beta^2}y^2)(\alpha^2-\gamma^2) + \gamma^2(x-l)^2}}{\alpha^2-\gamma^2} - \frac{\gamma(x-l)}{\alpha^2-\gamma^2} \tag{3}$$

Given a vector **w**=$[x, y]$, the dissimilarity as compared to the x axis-aligned vector **v**=$[0, l]$ is thus t computed with the above expression. Suitable values for the coefficients α, β, γ are based on a heuristics saying how the error iso-contours change as the error increases. A possible setting that relates these coefficients to the vector's length l and creates ellipses with an aspect ratio of 1:2 is given by:

$$\begin{aligned} \alpha &= 2l \\ \beta &= l \\ \gamma &= l \end{aligned} \tag{4}$$

Note that α must be greater than γ, otherwise the ellipses intersect each other and thus do not represent iso-error contours of a well-defined error function.

The above compared only the vectors' sizes and directions. To compare the vectors' positions, we use the same idea of error iso-contours, but this time around the vector's position. For example, $\mathbf{w}_1$ and $\mathbf{w}_3$ in Fig. 5 are equally similar to **v** from their positions' viewpoint as these are on the same iso-error contour, while $\mathbf{w}_2$ is less similar, its position being situated on a larger iso-error contour.

If we denote **w**'s position by x_s, y_s, the iso-error contours around the reference vector **v**'s position are defined as ellipses described by $s(x_s, y_s) = 0$, where s is given by:

$$s(x_s, y_s) = \frac{x_s^2}{d^2} + \frac{y_s^2}{e^2} - 1 \tag{5}$$

The value of s gives thus the dissimilarity between the positions of the two involved vectors. The coefficients d and e control the ellipse's aspect ratio and size. To model position similarity, it is useful to let the elliptic iso-error contours' aspect ratio under the user's control. For instance, an aspect ratio d/e of 3:1 favorizes clustering *along* the vectors' directions, one of 1:3 tends to cluster orthogonally to this direction, while a 1:1 aspect ratio favorizes all clustering directions equally, all other parameters being the same.

The functions $t(x, y)$ and $s(x_s, y_s)$ give the distance in our error space between two vectors' directions and magnitudes, and positions respectively. We can define the similarity of two vectors as any monothonically increasing function of t and s since these are independent quantities. A good candidate is a linear combination $l(s, t)$ of s and t:

$$l(s,t) = As + (1-A)t \tag{6}$$

where the coefficient A can be used to favor clustering on similar directions and magnitudes, repectively similar positions. The similarity function l is not a distance in the strict mathematical sense (it is not symmetric since the ellipses used are sized on just one of the two vectors v_1, v_2 that we compare). To make l (seen now as a function of v_1, v_2) symmetric, we simply replace it by $l'(v_1, v_2) = l(v_1, v_2) + l(v_2, v_1)$.

When the algorithm decides to merge two clusters using the presented similarity function, it creates a new cluster containing the union of the merged clusters' cells. Its representative vector is computed as an area-weighted (in 2D) or volume-weighted (in 3D) average of the merged clusters' representative vectors. The same is done for the representative vector's position, which coincide thus always with the cluster's gravity center. In most cases, the similarity function will generate convex clusters. However, we don't impose explicit geometric constraints on the cluster shape, so concave clusters having the gravity center outside their perimeter may appear.

4 EFFECT OF PARAMETERS

The original challenge was to generate insightful flow visualizations automatically by displaying a simplified flow. However, since different users may be interested in different aspects of the same flow, it is conceptually hard to generate a unique flow simplification method satisfying all users. In this sense, our algorithm has three control parameters. Varying these parameters one can produce a large range of visualizations emphasizing on various aspects of the flow data. These parameters are described in the following.

4.1 Level of Detail

The presented algorithm produces a hierarchical cluster structure which describes the flow dataset at different detail levels. For a level l, the dataset is represented with $N-l$ clusters, where N is the initial number of dataset cells. This hierarchy allows us to easily create visualizations that answer the question "display the flow by showing its F most prominent features", by selecting the $N - F^{th}$ hierarchy level and displaying one flow icon per cluster. Since the flow over a cluster is approximated by the cluster's representative vector, we can directly visualize clusters by rendering their representative vectors in a hedgehog style, using the cluster size to control their magnitude. Figure 6 and Fig. 13 a on the color plate show the same flow presented in Fig. 1 and a circular vortex flow, visualized with different numbers of vectors. Figures 13 b-e on the color plate show simplifications of a 3D field similar to the 2D one in Fig. 1 rendered from several viewpoints and with several numbers of arrows and arrow thicknesses.

Visualization of the clusters is however independent on their construction. We can obtain better results if instead of plain arrows we display curved arrows, by computing streamlines from every cluster's center up and downstream and capping them with arrow hats. Using curved arrows (e.g. represented by splines) as cluster representatives would be a step further, as the clustering process itself and not just the final visualization would be driven by the curved arrow model. A second observation is that the cluster visualization, similarly to iconic visualization, is effective when only a few icons (e.g. curved or straight arrows) are displayed to show the main structure of the flow. On the other hand, texture-based visualizations as spot noise are effective in showing flow local details with an equal emphasis over the entire domain. We combined the two visualization methods by superimposing the curved arrows generated by the clustering over a spot noise texture (see Fig. 7 for a couple of examples). Combining curved arrows with spot noise textures can increase the overall clarity of the visualization, especially for more complex flows for which the generated textures tend to be noisy and thus visually unclear (Fig. 7 d). Spot noise textures can be also effectively combined with 3D curved arrow visualizations, as shown in Fig. 13 e where a semi-transparent spot noise textured 2D slice adds a spatial clue to the otherwise hard to perceive 3D curved arrow rendering.

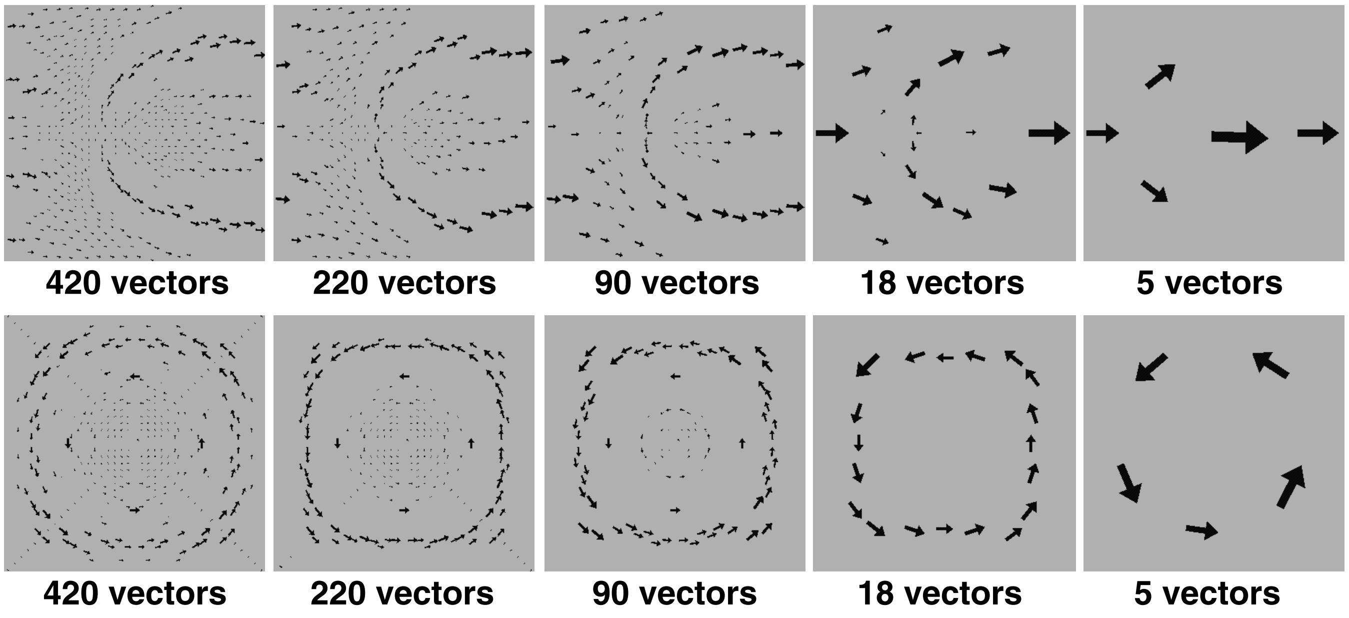


Figure 6: Visualization of two flow datasets at various simplification levels

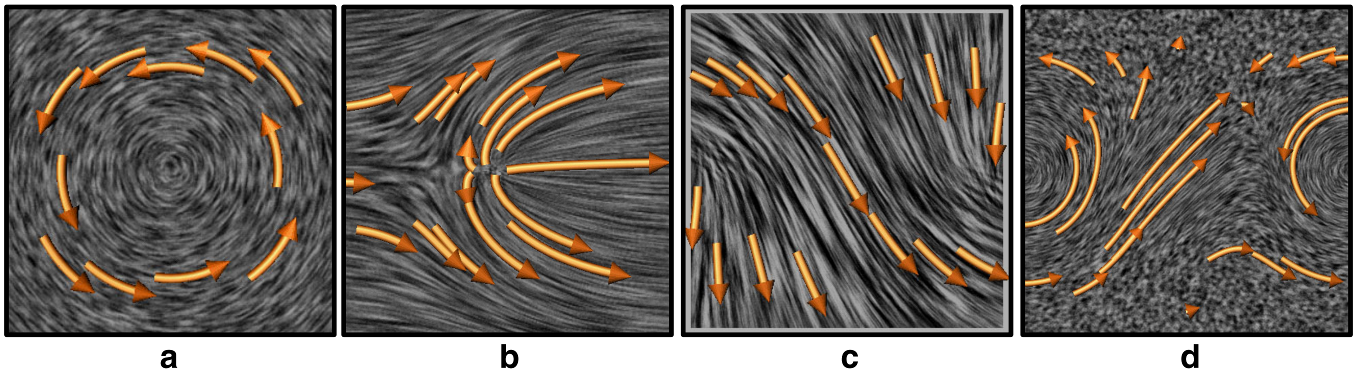


Figure 7: Flow visualizations using curved arrows on spot noise textured backgrouds

4.2 Clustering Parameters

At the core of the clustering algorithm, the decision to merge clusters is taken by the vector similarity function presented in Section 3.2. The clustering process can be controlled by changing several parameters. For example, α, β, γ (Equation 4) influence the way we compare the vectors' directions and magnitudes; d, e (Equation 5) influence the way we compare the vectors' positions, while A (Equation 5) controls the overall interest we have in comparing directions and magnitudes versus comparing the vectors' positions.

Two parameters have an important influence on the clustering, namely the parameter A and a new parameter B that determines d and e by the equations $d = B$, $e = 1 - B$, and the normalization $d + e = 1$. By changing the A and B parameters, we can produce several visualizations communicating different insights on the input flow data. We found it useful to leave these parameters under the direct control of the end-user, as different users or the same user with different datasets may wish to perform different simplifications. Figure 8 shows several clusterings of the fork flow, for several values of the B and A parameters varying from 0 to 1 on horizontal, respectively on vertical, and displaying the same number of vectors. To get a better impresion of the clustering, we displayed also the cluster of each vector using different colors.

This figure shows several aspects. Small values of A favor clustering of vectors with similar directions (table upper rows), and thus produce a subdivision of the original square domain into pie-like sectors, as the vectors' direction varies the least along a radius of the vortex. Small values of B favor clustering orthogonally to the vectors' directions (table left columns). The two effects strengthen each other in the upper-left table corner, where the images show clusters having borders almost perfectly aligned orthogonally on the vector field. Large values of A favor clustering of vectors with similar positions (table lower rows), and thus produce a domain subdivision into clusters with similar size. Large values of B favor clustering along the vectors' direction (table right columns). The two previous effects strengthen each other in the lower-right table corner, where the clustering resembles an almost regular subdivision. The extreme combinations of the A and B parameters are also illustrated for the vortex flow simplification in Fig. 9.

The effects of the parameter B on the clustering direction can be clearly seed in Fig. 10. Figure 10 a shows a simplification of a nearly constant vector field done for a large value of B which favors clustering along the vector field's direction. In contrast to this, Fig. 10 b shows the same vector field simplified with a small B value which

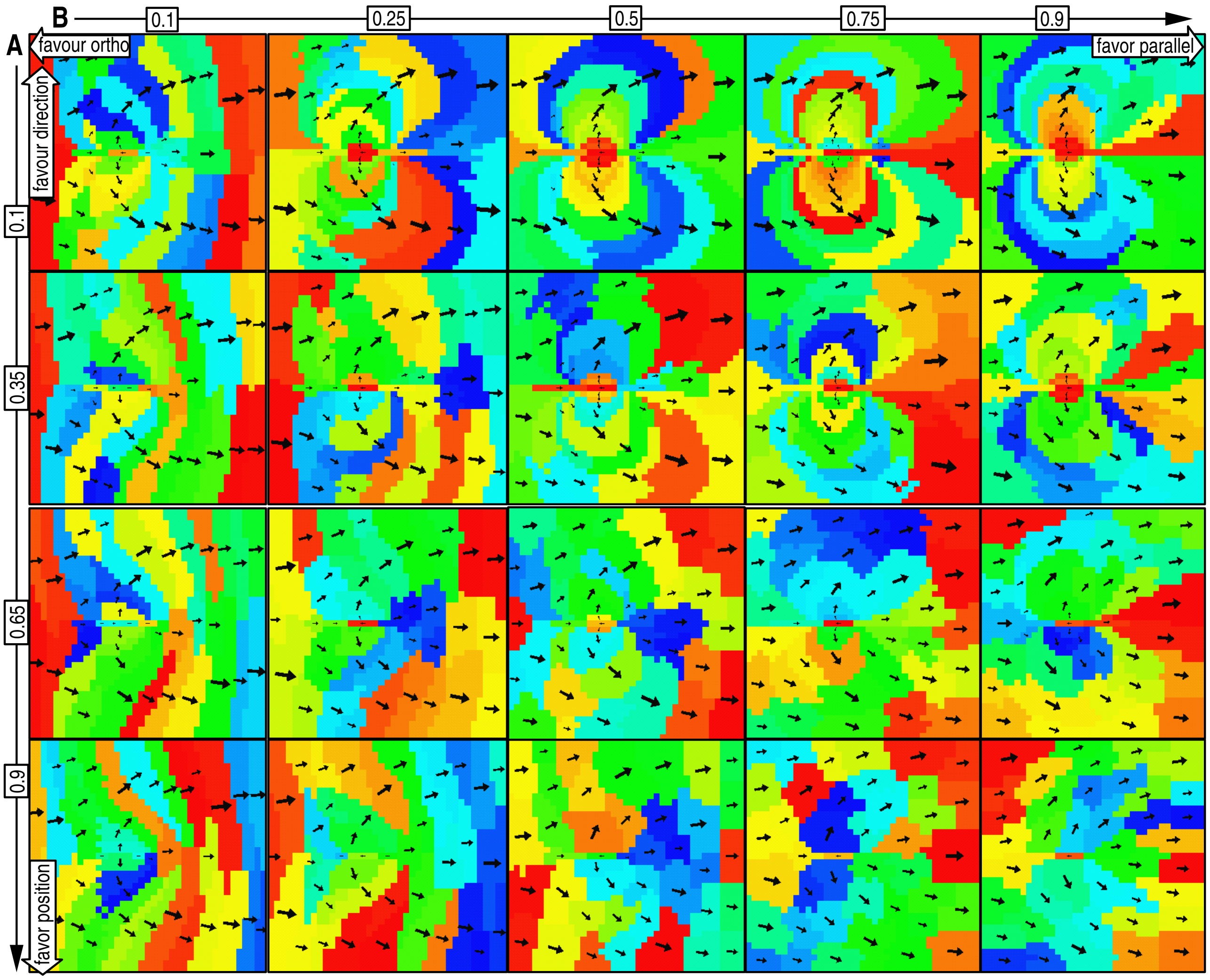


Figure 8: Fork flow simplification for several A and B values

favors clustering orthogonally to the vector field's direction.

Users interested in the main flow directions would employ small values of A and B, as the two strengthen each other in producing clusters over which vectors having similar directions. For datasets exhibiting both laminar and high vorticity flow, this setting generates clusterings that may contain very large, respectively very small clusters in the same image. Although this conveys both global and detail information in the same image, displaying both very small and large arrow icons may sometimes be undesirable due to their high visual non-uniformity. If this is not desired, the A parameter can be increased to favor more uniformly-sized clusters. At the other extreme, users comfortable with hedgehog-like visualizations that uniformly sample the whole domain can obtain similar images by using high values of A and B. We see this ability to easily select the type of visualization by continuously interpolating between the mentioned extremes as an important advantage of the presented method. Similarly, our method can be seen as combining the advantages of fully interactive visualization systems where the user is responsible for all the parameter settings and gets no system assistance and fully automated ones, where the user can not influence the system's heuristics.

5 IMPLEMENTATION

The clustering algorithm has two main phases (see (Fig. 2). First, a list of all the possible cluster pairs (c_i, c_j) that can be merged has to be produced. Since primary clusters are created from the dataset's cells and since clusters are allowed to merge only with neighbor clusters, we maintain for every cluster a list of its neighbor clusters throughout the algorithm to efficiently find the merge candidates of a cluster by scanning only its neighbors. Initially, the primary clusters get their neighbors using the cell neighboring information available from the input dataset in $O(N)$ time, where N is the number of cells in the dataset. The input dataset can be either a structured or unstructured mesh since cell neighbouring information can be easily computed for any mesh type.

The second phase of the algorithm involves an iteration-merge pass over the cluster-pairs list created initially, until this list contains a single cluster. Using a hash-table of pairs which keeps its contents in increasing order of the clustering error allows us to find the next pair to merge in $O(1)$ time. As insertion of new cluster pairs in the hash-table is $O(logN)$, we can show that the second phase has a cost of maximum $O(NlogN)$. The clustering has thus an overall cost of $O(NlogN)$.

The cluster-pair list often contains several cluster-pairs with the

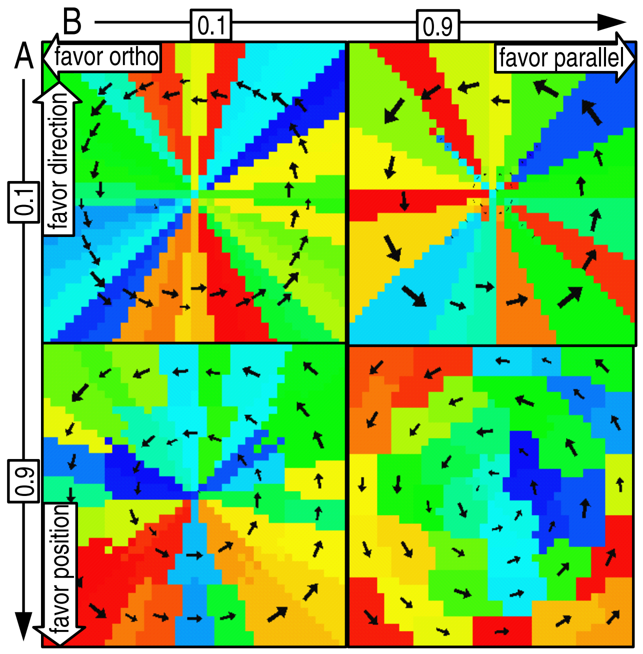


Figure 9: Vortex flow simplification for several *A* and *B* values

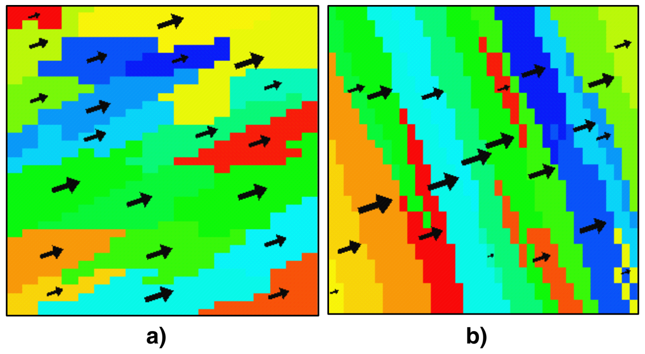


Figure 10: Constant flow simplification that favors clustering along the vector field direction (a) and across the vector field direction (b)

same clustering error. Since the domain to be clustered consists of a finite grid of finite size cells, the clustering is sensitive to which of the above pairs it picks first, since, as two clusters are merged, this influences their neighbors' chances to merge, and so on. This can be sometimes observed in the final image as regular cluster patterns which have nothing to do with the data but emerge from the clustering order. To remove this unpleasant effect, we included an option to permute the initial cluster-pair list randomly (shuffle) which maximizes the chances that clustering starts from several points randomly distributed over the whole domain.

We implemented the presented algorithm as a collection of classes written in the C++ language and integrated them with the Visualization Toolkit (vtk) package. Vtk is a powerful, comprehensive scientific visualization library which conveniently provided the data structures needed for flow data representation and manipulation. Next, we integrated our classes in the VISSION interactive visualization environment [9]. VISSION's open object-oriented architecture allowed us to easily couple our clustering pipeline's output to its available 3D rendering and direct manipulation modules, its input to various vtk flow data sets, and to the spot noise texture generation pipeline. End users can freely configure the visualization pipelines with new modules in a visual editor (Fig. 12 a) similar to systems like AVS [10] or Data Explorer, steer the clustering parameters by means of GUIs (Fig. 12 b), and monitor the results interactively in 3D cameras (Fig. 12 c). The clustering itself takes a few seconds for datasets around 10000 cells like the ones shown in this paper's illustrations, while selecting the level of detail to display is done in real time on a SGI O2 R5000 machine. Simplification of 3D flow fields as those presented in Fig. 11 and Fig. 13 b-e is however considerably slower, as these contain more cells, each having a higher neighbor count.

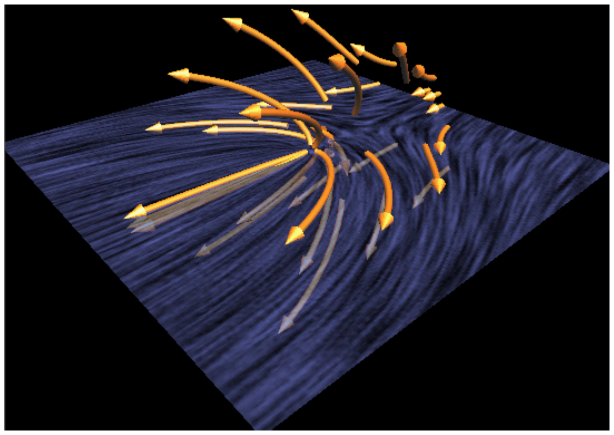
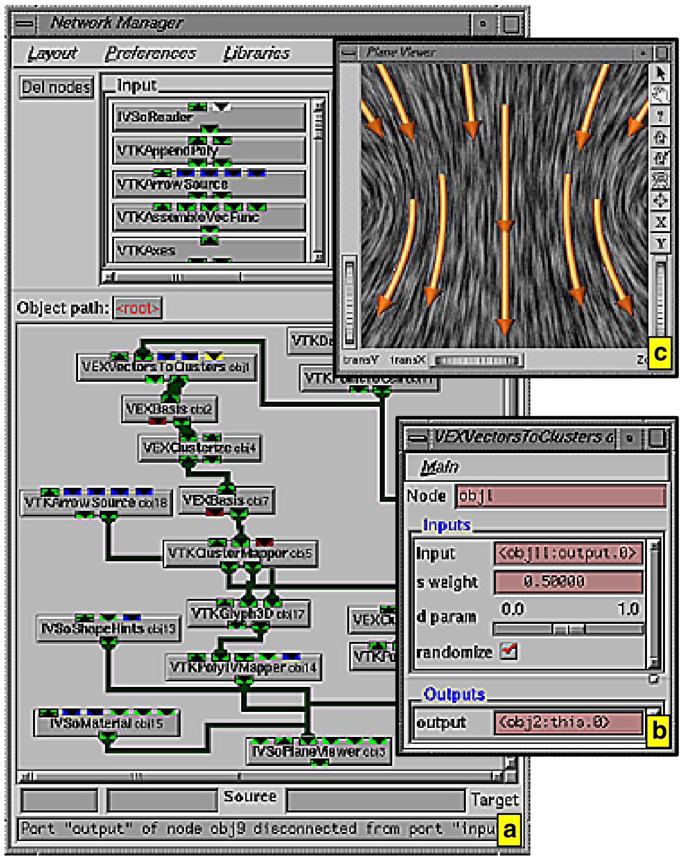


Figure 11: Simplification of 3D flow field

By connecting the 'level of detail' input port of the simplification pipeline to an iterator which dynamically varies an integer value and its output to a MPEG production module, one can easily produce a simplification MPEG animation displaying a progressively simplified view of a given flow dataset. Animated visualizations of the progressively simplified data are however not always an appropriate manner to convey insight in a vector field. This is caused by the fact that, as clusters get progressively merged, their representative arrows may be replaced by new arrows having sensibly different sizes and positions. This is mainly due to the fact that the clustering algorithm minimizes the representation error *at each individual hierarchy level* but does not explicitly try to preserve a visual coherency between the visual representations (e.g. arrow icons) of consecutive hierarchy levels. Doing this would involve a different definition of the cluster similarity function, which should encompass both a comparison with spatial neighbor clusters and with parent and child clusters in the hierarchy. We believe that the complexity for devising and efficiently implementing such a similarity function would not pay off as compared to its possible advantages.

6 CONCLUSION

This paper presented a new method for producing compact vector field visualizations using a hierarchical simplification of the vector data. The simplification is driven by a vector similarity function whose parameters can easily be tuned to produce visualizations which stress different aspects of the vector field. After the simplification, one can visualize the vector data at different levels of detail by interactively choosing the simplification level to be displayed. For the display phase, we used straight or curved arrows for the simplified data over a spot noise textured background that fills the visual gaps between the arrows with local detail. We have integrated our method in an interactive dataflow visualization system where one can change the parameters influencing simplification and monitor the resulting images interactively.



Port "output" of node obj9 disconnected from port "inpu

Node obj1

Input <obj1:output.0>

s weight 0.50000

d param 0.0 1.0

randomize ✓

output <obj2:this.0>

Figure 12: Clustering pipeline and visualization in the Vission system

Put in mathematical terms, we try to approximate a vector field by a (small) hierarchical set of constant basis functions of finite supports whose union equals the field's support. The basis functions' supports are the clusters and their values are the clusters' representative vectors. The shapes of the basis functions' supports are determined such that the representation error is minimized (this is actually the clustering algorithm). The main difference between our approach and other representation techniques reducible to basis functions such as Fourier analysis or wavelets is that we do not prescribe a priori the shape of the basis functions' supports, but determine it by a minimization process. In this respect, the presented technique does not use a given set of basis functions (like the sine and cosine functions used by the Fourier analysis or the polynomial basis functions used by spline techniques) but computes a different base for each new given input dataset.

To completely automate our technique we plan to devise a method of setting the simplification parameters that are now still under the end-user's control, based on a model of the user's perception of a vector field. Such a method should be able to determine how to produce the most insightful or most easily perceivable visualizations and act as a feed-forward control on the presented simplification. In the same time we plan to devise better models for the clusters which should approximate the underlying field more accurately, e.g. by using a curved arrow model based on splines. On one hand, the introduction of such techniques would simplify the task of the user considerably by producing perceptually meaningful visualizations with a minimal or no direct user intervention. On the other hand, animations that show progressively simplified views of the same dataset on the outline of the idea sketched in Section 5 would be easier to produce, as a perceptually based simplification could guarantee a certain coherency among several simplification levels of the same dataset.

References

[1] D. C. Banks and C. A. Singer. Vortex tubes in turbulent flows: Identification, representation, reconstruction. *Proc. Visualization 94, IEEE Computer Society Press, pp. 132-139*, 1994.

[2] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. *Computer Graphics (Proc. SIGGRAPH '93), pp. 263-279*, 1993.

[3] W. de Leeuw and J. J. van Wijk. Enhanced spot noise for vector field visualization. *Proc. Visualization 95, IEEE Computer Society Press, pp. 233-239*, 1995.

[4] J. Helman and L. Hesselink. Representation and display of vector field topology in fluid flow data sets. *Computers, vol. 22, no. 8, pp. 27-36*, 1989.

[5] P. Burrel J. Rossignac. Multiresolution 3d approximations for rendering complex scenes. *Modelling in Computer Graphics, E. B. Falcidieno, T. L. Kunn, eds, Springer Verlag, pp 455-465*, 1993.

[6] S. Mallat. A theory for the multiresolution signal decomposition: The wavelet representation. *IEEE Pattern Analysis and Machine Intelligence, vol. 11, no. 7, pp. 676-693*, 1989.

[7] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *IEEE Computer Graphics and Applications, vol. 27, no. 7, pp. 20-27*, 1994.

[8] W. J. Schroeder, C. R. Volpe, and W. E. Lorensen. The stream polygon: A technique for 3d vector field visualization. *Proc. Visualization 91, IEEE Computer Society Press, pp. 126-132*, 1991.

[9] A. C. Telea and J. J. van Wijk. VISSION: An object oriented dataflow system for simulation and visualization. *Proceedings of the 10th IEEE/Eurographics Visualization Workshop, Vienna, Austria*, 1999.

[10] C. Upson, T.Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, and J. Vroom. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications, July 1989, pp30-42*, 1989.

[11] T. van Walsum. Selective visualization on curvilinear grids. *PhD thesis, Delft University of Technology, the Netherlands*, 1995.

[12] T. van Walsum, F. H. Post, D. Silver, and F. J. Post. Feature extraction and iconic visualization. *IEEE Computer Graphics and Applications, vol. 2, no. 2, pp. 111-119*, 1996.

[13] J. J. van Wijk. Flow visualization with surface particles. *IEEE Computer Graphics and Applications, vol. 13, no. 4, pp. 18-24*, 1993.

[14] P. Wong and D. Bergeron. Hierarchical representation of very large data sets for visualization using wavelets. *Scientific Visualization, eds. G. Nielson, H. Hagen, H. Mueller, IEEE Computer Society Press, p. 415-429*, 1997.

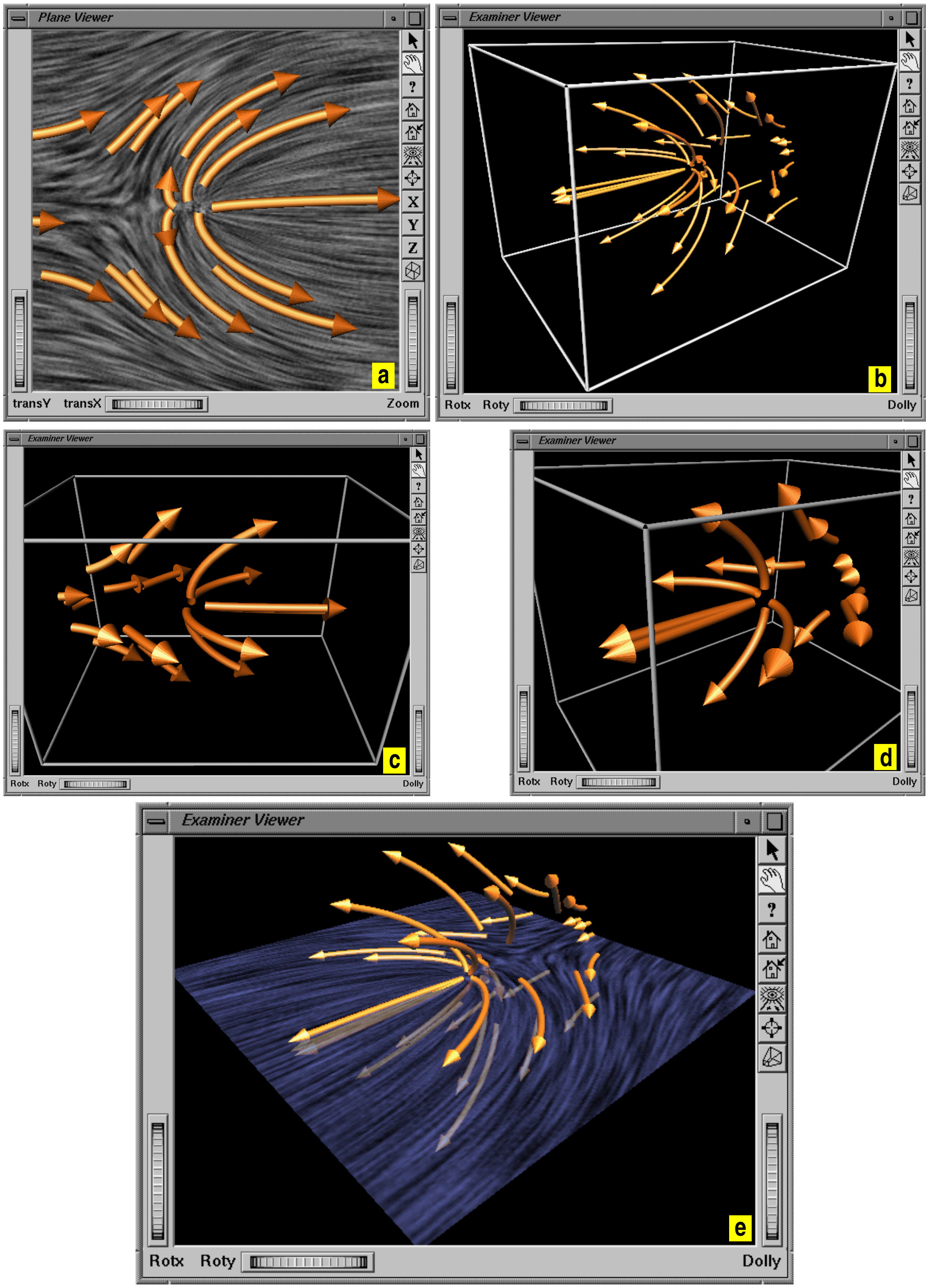


Figure 13: Examples of 2D (a) and 3D (b-e) flow field simplifications