ABSTRACT

Feature tracking algorithms usually rely on operators for identifying regions of interest. One of these commonly used operators is to identify parallel vectors introduced by Peikert and Roth. In this paper, we propose a new and improved method for finding parallel vectors in 3D vector fields. Our method uses a two-stage approach where in the first stage we extract solution points from 2D faces using Newton-Raphson method, and in the second stage, we use analytical tangents to trace solution lines. The distinct advantage of our method over the previous method lies in the fact that our algorithm does not require a very fine grid to find all the important topological features. As a consequence, the extraction phase does not have to be at the same resolution as the original dataset. More importantly, the feature lines extracted are topologically consistent. We demonstrate the tracing algorithm with results from several datasets.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques;

Keywords: vector fields, vortex cores, feature extraction, tracing, pseudo inverse

1. INTRODUCTION

Vortices are among the most important features occurring in CFD field data. Feature extraction is ideal for determining areas with strongly swirling flows, which could have positive or negative impact. In the turbomachinery design vortices reduce efficiency. In aircraft design, vortices over an aircraft wing, for example, might greatly increase its lift. But, they could also create strong turbulence leading to structural fatigue. Vortices generated by an automobile can increase its drag, create noise, and possibly cause decreased visibility in the rain. There are a several methods used to automatically perform vortex core extraction. There are many numerical algorithms that have been proposed to solve this problem. The algorithms vary in accuracy and stability. In this paper we propose a stable numerical algorithm to extract parallel features.

The contribution of our method is two-fold. First, our method allows one to preserve the feature topology. Second, our algorithm can operate on a much coarser grid and therefore save a lot of time and effort compared to the previous
methods. In the following section, we review related work, followed by Section 3 to provide the description of our method, which consists of two stages and both stages are explained in detail. In section 4, we discuss the implementation issues. Section 5 describes the results collected when we applied our algorithm to several data sets, and finally Section 6 contains conclusions and plans for a future work.

2. RELATED WORK

The majority of vortex-core extraction research lately has focused on finding methods for extraction line type features that are regarded as centers of vortices. Mira and Kida describe the method based on pressure-like scalar quantity $p$ that is computed from the velocity field. Saujudí and Haimes base their method on critical points theory as they search for the center of swirling flows.

Singer and Banks proposed another method for finding parallel features by using a predictor/corrector algorithm. Their method uses vorticity vector to trace a direction to the next solution point. But since the vorticity vector is not an analytical tangent its direction must be corrected with another extraction phase after every step. This algorithm also needs a suitable starting point that already lie on the solution line. Also, the size of the correction step must be closely bounded so that the method does not diverge from the correct feature line. The main advantage of the curve following tracing method is that it allows one to trace lines that are very close to the real feature line, and it does not require a very fine grid to accomplish that.

Just recently Theisel et al. have introduced new method for tracking vortex core lines in time-dependent 3D flow fields. Their method method is based on the concept of feature flow fields that they have introduced in. They derive appropriate vector fields in such a way that the search vortex core lines are stream lines on them. The extraction and tracking of vortex core lines becomes a stream line/surface integration of vector fields.

Peikert and Roth were first to introduce parallel vector operator and present several methods for feature line extractions. Their algorithms consist of two stages. In the first stage, datasets are viewed as a collection of 3D cells, and their algorithm then extracts solution points from each 2D face of each cell using Newton-Raphson iteration. One weakness of this extraction method is that at most only one solution point can be extracted from each 2D face. That happens because the algorithm terminates as soon as it finds the first solution point.

In the second stage, the goal is to connect as many solution points as possible using a certain heuristic. Peikert and Roth presented several methods for connecting solution points. One method makes use of the knowledge of grid topology in addition to several connection rules. Connection rules are very simple. If there are only two points in a cell, they simply connect these two points. If there are four intersections in a cell, they choose the two connections with maximal distances from each other out of three possible pairs of connections (where no pair can have shared vertices).

If there are an odd number of intersection points in a cell, their algorithm does not assume any connections; likewise their algorithm does not handle cases with six and more points in a cell. If odd number of points is found inside a 3D cell than they subdivide the cell into smaller cells until they get cells with even number of points or no points at all. Their method was designed to work with a set of points labeled with additional Boolean flags that would indicate points of interest with respect to some criteria. The implementation of their algorithm posted on the web indicates that they do not handle efficiently of cases where the intersection with the solution line lies in a corner of the cell. Furthermore their method does not handle cases where solution line lies directly on a 2D face and not just passing through.

Peikert and Roth also suggested another method for creating solution lines. Their method is considered to be state of the art. This method is described as follows: Rule 1: always connect points to existing lines before adding new lines; Rule 2: always connect closest points first. The algorithm starts with a set of intersection points, and maintains a list of connected lines and free points.

In our view, both of these approaches have several drawbacks. For one, these algorithms do not necessarily produce topologically correct feature lines. Also, in order to produce more accurate representation of the existing features these methods require a finer grid of cells. Therefore, there is a significant cost associated with the processing during the extraction phase. In addition, we found that their methods are very much reliant on the correct information returned by the extraction phase. Unfortunately, While Newton-Raphson is one of the most popular iterative root finding method, it does not always converge. If the extraction method does not find a solution point on the 2D face of a given cell, then the
methods would probably lose a feature or some of their lines would break to differ from the real solution lines. Another problem associated with the extraction method has to do with the nature of the Newton-Raphson method. Newton-Raphson would terminate after it finds the first solution, but it is possible to have more than one solution on the same 2D face. Again features would be lost and the data topology would be altered.

3. METHODS

3.1. Brief description
Our method as do most of the previous methods\textsuperscript{10} is based on an underlying grid and a simple trilinear interpolation inside the grid cells. Our solution first finds intersection points of the vortex core lines on grid faces and then proceeds tracing the solution lines until the lines intersect other grid faces. We keep on tracing every solution point until no more solution points remain to be found.

Since features from parallel vectors form lines in most cases as shown by,\textsuperscript{6} the feature extraction algorithm can be conveniently divided into two stages. In the first stage, we find the intersection of these feature lines with 2D cell faces. In the second stage, we connect the extracted intersection points into lines. There are some potential problems in each of these stages. In the first stage, we must account for the possibility that there are multiple intersections on any given cell face. In fact, it is also possible that a feature lies on the cell face. Also, unless the cells are subdivided, we cannot capture sub-cell resolution feature lines. In the second stage, we must account for ambiguities in connecting the extracted points to form consistent and continuous lines.

Most of the improvement that we introduce in this paper come in the second stage of the feature extraction process. The point extraction phase is quite standard. Still, it does merit a few words. A commonly used method to check whether two 3D vectors $V$ and $W$ are parallel is to take the cross product: $c = v \times w$. Many extraction algorithms are based on solving the equations that sets all three components of $c$ as zero. But these methods are not very suitable for numerical reasons. This is because the three components of $v \times w$ are linearly dependent (which follows from $(v \times w) \cdot v = 0$). As a consequence, it is a bad idea to treat $v \times w$ as a derived field and to apply standard techniques for finding its zeros. There are many other algorithms that can be classified as isosurface-based approaches. These algorithms arbitrarily choose two of the three components and generate isosurfaces for them and find their intersection by using "Marching Lines" algorithm.\textsuperscript{11} Then on the intersection one needs to check whether the third component is within an acceptable threshold.

There is yet another set of algorithms that tries to minimize the square of the magnitude of $C$. This quantity is defined as $D = c_x^2 + c_y^2 + c_z^2$. It is clear that: (i) $D \geq 0$, and (ii) $D = 0$ if and only if $D_x = 0, D_y = 0, D_z = 0$. This approach can find the minimum using standard numerical methods for minimization such as conjugate gradient or Newton-Raphson methods. If $D$ is within an acceptable threshold, then it can be claimed that a solution is found. The weakness of this method is that it could be trapped in a local minimum and therefore misses the real feature. In this paper, we propose two methods to solve these problems of feature extraction using parallel vectors.

3.2. Extraction phase
The first method that we propose and use for extraction in our implementation is a Newton-Raphson algorithm that considers all three equations as an over-specified systems. (This method was first proposed in M. Roth’s PhD thesis). The Newton-Raphson step in 1D estimates the zero crossing of $f$ as:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{1}$$

So, the correction step is

$$f_i' \Delta x = -f_i. \tag{2}$$

In our case we are interested in using Newton-Raphson in two dimensions. We call the two parameters describing the grid face $\ell_1$ and $\ell_2$ which are initially preset to 0.5 (the center of the face). At this position we evaluate (using trilinear
interpolation) the two vectors $v$ and $w$, and their partial derivatives with respect to the parameters $\frac{\partial v}{\partial \ell_1}$ and $\frac{\partial w}{\partial \ell_2}$. We want to minimize vector

$$c = v \times w$$  \hspace{1cm} (3)$$

Using chain rule we get partial derivatives with respect to $\ell_i$.

$$\frac{\partial c}{\partial \ell_i} = \frac{\partial v}{\partial \ell_i} \times w + v \times \frac{\partial w}{\partial \ell_i}$$  \hspace{1cm} (4)$$

$\frac{\partial c}{\partial \ell_i}$ forms a $2 \times 3$ matrix that we will call $N_c$.

$$N_c \Delta \ell = -c$$  \hspace{1cm} (5)$$

The equation that gives us least square solution would be:

$$N_c^T N_c \Delta \ell = -N_c^T c$$  \hspace{1cm} (6)$$

$$\Delta \ell = -N_c^T c(N_c^T N_c)^{-1}$$  \hspace{1cm} (7)$$

The solution results in the correction step that gets progressively close to the solution. The algorithm terminates if anyone of the following conditions gets satisfied:

- if the square of $c$ gets smaller than certain fixed number (that is close to 0)
- if number of correction steps exceeds certain number. (15 in our implementation)

This method relies on all three vector components in calculating the solution point. Hence this technique avoids numerical problems associated with methods that select only two components to do their calculations. This is especially evident in cases where the third vector component is parallel to one of the major axes.

There is another approach that we propose in this paper to address the extraction problem. A vector $V$ is parallel to $W$ if and only if $\exists s$ that satisfies $V = sW$. We specify this system with three equations:

$$V_x(u, v) = sW_x(u, v)$$  \hspace{1cm} (8)$$

$$V_y(u, v) = sW_y(u, v)$$  \hspace{1cm} (9)$$

$$V_z(u, v) = sW_z(u, v)$$  \hspace{1cm} (10)$$

where $u$ and $v$ are the two parameters that specify the grid faces. This system has three equations with three parameters, and therefore can be easily solved with standard numerical algorithm such as the Newton-Raphson algorithm.
3.3. Tracing phase

The algorithm described in this section is a core innovation that we introduce in this paper. In this stage, we need to connect all the extracted points to form a smooth feature line. However, if there are more than two points in one cell, ambiguities arise when connecting them. We propose the use of analytical tangents to resolve this ambiguity. Some algorithms arbitrarily throw away one equation and estimate the tangent of the feature line based on the two remaining equations. But as we mentioned in the previous section, this introduces large numerical errors. A new form of analytical tangent that preserves all three equations is therefore proposed as follows. Given a feature point \( X \) where \( V(X) \parallel W(X) \), and the gradients of \( V \) and \( W \) at \( X \), \( \nabla V \) and \( \nabla W \) respectively, we would like to find an analytical form of tangent of the feature line. We denote \( V(X) \) and \( W(X) \) as \( V^* \) and \( W^* \) respectively.

Since \( V^* \parallel W^* \), there must exist an \( s \) that satisfies \( V^* = sW^* \), where \( s = \pm \frac{\|V^*\|}{\|W^*\|} \). The sign of \( s \) depends on whether these two vectors point in the same direction or not. After scaling with \( s \), \( V^* \) and \( sW^* \) are the same, so we denote them as \( \overline{OA} \). See Figure 1. After moving a little distance away from \( X^* \), \( V \) changes from \( A \) to \( B \) and \( sW \) changes from \( A \) to \( C \). We present the following observation: if \( V \) and \( W \) must always be parallel, then \( \overline{BC} \parallel \overline{OA} \) as \( \theta \to 0 \), where \( \theta \) is an angle between \( \overline{OA} \) and \( \overline{OC} \). This is because if \( \overline{OC} \parallel \overline{OB} \), then \( O, C \) and \( B \) must be collinear. Hence, we have \( \angle ABC = \angle OAC + \theta \). When we let \( \theta \to 0 \), we get \( \angle ABC = \angle OAC \), and therefore \( \overline{OA} \parallel \overline{BC} \). Because

\[
\overline{BC} = \overline{OB} - \overline{OC} = (V^* + sV) - s(W^* + sW) = \nabla V \cdot \delta X - s\nabla W \cdot \delta X = (\nabla V - s\nabla W) \cdot \delta X \tag{11}
\]

and \( \overline{OA} \parallel \overline{BC} \), there must exist an \( s_1 \) that satisfies \( (\nabla V - s\nabla W) \cdot \delta X = s_1V^* \). Assuming \( (\nabla V - s\nabla W) \) has a full rank, this implies \( \delta X = s_1(\nabla V - s\nabla W)^{-1} \cdot V^* \). It means the tangent of the feature line at \( X^* \)

\[
(\nabla V - s\nabla W)^{-1} \cdot V^* \tag{12}
\]

A whole feature line can be traced from just one point using this analytical form of the tangent by relying on the differential properties of the vector fields at \( X^* \).

![Figure 1. Rationale for tangent formulation.](image-url)

Solving \( (\nabla V - s\nabla W)^{-1} \cdot V^* \) turns out to be not so simple. If we would just use the regular inversion technique, we would quickly run into problems in cases where the feature line would approach the matrix \( A = (\nabla V - s\nabla W) \) singularity points. These are points where \( det(A) \) gets very close to zero, and our tracing soon goes off track. But since we are only interested in the direction of the feature line and we do not care about the magnitude of the vector we do not have to multiply the result by \( \frac{1}{det(A)} \) and therefore we can use the following method to solve out problem:

Given two vector fields \( V \) and \( W \), and a feature point \( P \) where \( V \parallel W \). Let the gradients at \( P \) be \( \nabla V \) and \( \nabla W \). Denote a matrix \( A = \nabla V - \frac{\|V\|}{\|W\|} \cdot \nabla W \), and \( N \) as the tangent of the parallel vector feature line at \( P \). The three components of \( N = < N_x, N_y, N_z > \) can be obtained as follows using Cramer’s rule:
\[ N_x = \begin{vmatrix} V_x & A_{xy} & A_{xz} \\ V_y & A_{yx} & A_{yz} \\ V_z & A_{zx} & A_{zz} \end{vmatrix} \]  

(13)

\[ N_y = \begin{vmatrix} A_{xx} & V_x & A_{xz} \\ A_{yx} & V_y & A_{yz} \\ A_{zx} & V_z & A_{zz} \end{vmatrix} \]  

(14)

\[ N_z = \begin{vmatrix} A_{xx} & A_{xy} & V_x \\ A_{yx} & A_{yy} & V_y \\ A_{zx} & A_{zy} & V_z \end{vmatrix} \]  

(15)

Note that in the algorithm, \( N \) is usually normalized for later application. After the tangent is calculated, an inexpensive Euler’s method with small steps, can be used to trace the solution line inside a 3D cell. At each step, the value of the tangent at the new location is evaluated and the tracing continues.

4. IMPLEMENTATION ISSUES

We have implemented our algorithm partly in Matlab and partly in C. For visualization of feature lines, we have created a volume rendering of the magnitude of the cross product \( C = V \times W \). The region where \( ||C|| \) approaches zero are colored blue and indicate the location of the real feature lines. In Matlab, we have implemented both extraction and tracing phases. The resulting solution points were then displayed together in the volume rendering so that we can see how accurate the traced lines really are. In the extraction phase, we use bilinear interpolation to find the vector values at any given point. In our tracing phase, we rely on trilinear interpolation to find the gradients in \( x, y, \) and \( z \) directions. Our algorithm was implemented on Windows XP platform using Matlab 6.0, Visual Studio .NET 2003 C++ compiler and VTK (visualization toolkit).

5. RESULTS

We experimented with several synthetic data sets to test out our algorithm. We had two goals in mind when we started working on this project. First was to find an algorithm that would be topologically consistent and would not miss features. Of course since there is no way for us mathematically to prove the completeness. One obvious problem is that it can never capture anything that is smaller than the resolution of the grid. Nonetheless, we feel that our method captures solution lines better than previous methods did and we have an empirical evidence to back up our claim. Our second goal was to try inventing the technique that would be faster and more efficient than previous approaches.

5.1. Feature correctness

The first set is a 3D \( 1 \times 1 \times 1 \) cell with vectors at each corner that have been set randomly. Figure 2 shows an interesting case where we have 4 points and where our algorithm produces a trace that matches closely to the real solution line while Peikert and Roth’s algorithm would have connected the wrong points and therefore would produce a topologically incorrect solution lines. We can verify that our tracing lines are indeed the correct lines by looking at volume rendering shown in Figure 5. On this figure our tracing lines are colored in black and real solution areas are captured by blue clouds. Every point of the volume rendering represents a magnitude of a cross product of two vectors. The blue regions indicate places where the magnitude of the cross product gets close to zero. Thus, our solution lines can be seen to correctly pass through the middle of dark-blue regions.

The second data set shown in Figures 3, 6, 4, 7 was also created by randomly generating vector values at each corner of a \( 1 \times 1 \times 1 \) cell and then resampled into a \( 7 \times 7 \times 7 \) cells in order to generate smoother feature lines. Again, our tracing algorithm is able to capture all real feature lines while Peikert and Roth’s algorithm does not fare as well. In Figure 3 where feature lines are generated with Peikert and Roth’s algorithm, look at the cell pointed to by the black arrow. Points are not connected correctly at this cell and therefore feature two lines go off track. The proof that in the cell under consideration, our lines are correct and their lines are not can be found in Figure 8 that represents a volume rendering of the cell pointed to by black arrow. In addition to problems with connecting solution points, the lines produced by their algorithm are not smooth. The only way for their method to produce smoother lines is to do more subdivisions. Now we can compare that figure with the same dataset traced with our method in Figure 4 and there all the lines are smooth and continuous.
5.2. Algorithm efficiency

In Figure 6 we see large number of smaller cubical cells. These cells are the result of the subdivision performed by Peikert and Roth’s algorithm when extraction phase reports odd number of solution points on the faces of a cell. The quality of feature lines produced by their method depends in a large part on the resolution of the data grid. The smaller the cubical cells, the better the solution lines. While this is not a huge problem for small data sets that we have tested the algorithm on, it will become a serious issue for very large datasets. Our algorithm, on the other hand, does not need to have a high resolution of the data grid to produce high quality feature lines. All it needs is several solution points that lie on the solution lines. Then all the solution lines can be traced from these starting points. The speed of our tracing depends in a large part on the step that we choose for our initial value algorithm (Euler’s algorithm in our case).

Figure 2. Volume rendering of a 3D cell with 4 solution points. Red crosses are the solution points (locations where feature lines intersect the 2D faces of the cell); red lines are the connection lines produced by Peikert and Roth’s method; blue lines produced by our tracing method.

Figure 3. Data grid consists of 7x7x7 cells. View from the top. Red crosses indicate feature points; red lines indicate feature lines produced by Peikert and Roth’s method.

Figure 4. Data grid consists of 7x7x7 cells. View from the top. Red crosses indicate feature points; blue lines indicate feature lines produced by our tracing method.
Figure 5. 3D cell with 4 solution points. Red balls are the solution points (locations where feature lines intersect the 2D faces of the cell); black lines are the tracing lines produced by our algorithm; blue clouds represent the area where cross products of two vectors get closer to zero.

Figure 6. Data grid consists of 7x7x7 cells. View from the side. Red crosses indicate feature points; red lines indicate feature lines produced by Peikert and Roth’s method.

Figure 7. Data grid consists of 7x7x7 cells. View from the side. Red crosses indicate feature points; blue lines indicate feature lines produced by our tracing method.
6. CONCLUSIONS

We have introduced a new method for tracing parallel feature lines in 3D vector fields. The method has the following characteristics:

- The method elaborates on a well-known extraction techniques to find initial solution points.
- The method uses new technique for finding analytical tangent for tracing solution lines starting from the initial solution points.
- Our method does not require a large number of solution points to capture a majority of important feature lines, and leads to a speed up in algorithm execution in terms of number of cells that need processing.
- Our experiments showed that the resulting feature lines produced by our tracing method align tightly with the real solution lines and that makes our algorithm produce lines that are topologically correct.
- The extraction phase prove to be not as critical for the accuracy of our method as it is for other parallel vector extraction methods, since our tracing method would still be able to accurately trace the solution lines.

One of the underlying weakness of the original approach, and which is also present in our current extraction phase, is the fact that only one intersection point can be captured per face. The current best practice is to subdivide the cell face to see if there are other intersections on the face. However, this is an expensive proposition if it has to be done for all the faces. Our tracing phase partially addresses this limitation if the other intersection points on the same face belong to feature lines where at least one intersection point has been found. Two possible scenarios exist. One, starting with the first intersection point, we trace the feature line which then turns around and intersects the same cell face at another intersection point. Second, another feature line can find the second intersection point as that line is being traced. Our tracing phase also handles the case where the feature line intersects the face at a corner.

6.1. Future direction

There are a number of issues that remain to be addressed in this project. We need to try our method on diverse data sets. We might still improve upon current extraction scheme that uses Newton-Raphson by using hybrid approach Conjugate-Gradient in combination with Newton-Raphson. This should allow us to converge in case where typical Newton-Raphson fails. Also it could prove worthwhile to have several starting points for Newton-Raphson and not only one. Choosing several starting points would allow us to capture more initial points on 2D faces.
ACKNOWLEDGEMENTS
We would like to thank Professor Allen Van Gelder for his insightful comments.

REFERENCES