Attribute-Based Feature Tracking

Freek Reinders[1], Frits H. Post[1], and Hans J.W. Spoelder[2]

[1] Dept. of Computer Science, Delft University of Technology
email: {k.f.j.reinders, f.h.post}@cs.tudelft.nl
[2] Fac. of Sciences, Div. of Physics and Astronomy, Vrije Universiteit
email: hs@nat.vu.nl

Abstract. Visualization of time-dependent data is an enormous task because of the vast amount of data involved. However, most of the time the scientist is mainly interested in the evolution of certain features. Therefore, it suffices to show the evolution of these features. The task of the visualization system is to extract the features from all frames, to track the features, i.e. to determine the correspondences between features in successive frames, and finally to visualize the tracking results.
This paper describes a tracking system that uses feature data to track the features and to determine their evolution in time. The feature data consists of basic attributes such as position, size, and mass. For each set of attributes a number of correspondence functions can be tested which results in a correspondence factor. This factor makes it possible to quantify the goodness of the match between two features in successive time frames. Since the algorithm uses only the feature data instead of the grid data, it is feasible to perform an extensive multi-pass search for continuing paths.

Keywords: Time-Dependent Data, Feature Extraction, Attribute-Based Feature Tracking.

1 Introduction

Numerical simulations are increasingly focusing on the investigation of time-dependent phenomena. In general this results in a vast amount of data that is hard to process, interpret, and visualize. Off-line generation of images (using a global visualization technique) and creation of a playback animation, can give an overview of the evolution of the data [4]. However, this has a number of deficiencies: it cannot be performed interactively since the amount of data is too large to handle, it leaves the extraction of apparent features to the visual inspection by the scientist, and it does not give a quantitative description of the evolving phenomena. In order to overcome these flaws one should use an automatic way of tracking coherent structures (features) in the data. This requires a matching criterion between features in successive time steps (frames). The problem of matching two features is also cited as *the correspondence problem* [3].

Once the correspondence problem is solved, the sequence as a whole can be described. For each feature a description can be made of its lifespan, describing it's motion, growth, interaction with other features, etc. This allows us to select one feature and visualize the evolution of it, or to detect certain events like unusual changes or a specific interaction. The tracking process results in entirely new ways of visualizing evolving phenomena.

The correspondence problem is an issue in several scientific disciplines such as image processing, computer vision, and scientific visualization. Although each discipline has its own perspective to the problem, many approaches are similar to each other. Roughly, the methods can be classified in two categories: image-based methods and feature-based methods.

In image-based methods, the displacement of coherent structures is determined on a pixel-to-pixel basis. Examples of image-based techniques are (from image processing) the maximization of the cross-correlation between two images [8], and methods (from computer vision) using optical flow [1]. Since these methods involve an optimization process using the original data, they are suitable for 2D-images, but are inefficient for 3D fields.

In feature-based methods, first feature extraction takes place for each frame. The resulting features are then matched to the features in subsequent frames. Features may include points, lines, surfaces, or volumes, and can be extracted in numerous ways depending on the application domain and the type of feature. After the extraction, the correspondence problem can be solved using two mechanisms: region-matching, or attribute-matching. Region-matching is achieved by matching the regions of interest, for instance by overlap [2, 11, 12], or by a linear affine model [5]. These methods process the (binary) grid-data and consequently are memory consuming. Contrary to this, attribute-matching [9, 10] only works with certain attributes describing the features, which is a small set of numbers. Hence, attribute-matching costs less memory, one can even afford multi-pass searching for the optimal correspondence between features. However, since the attributes are a reduced model of a feature, the problem is finding a suitable set of correspondence criteria.

This paper describes an attribute-based feature tracking system that solves the correspondence problem based on primary attributes of features, such as position, size, and mass. The key of the algorithm is the use of a prediction scheme and the use of a multi-pass search for continuing paths. The process is highly interactive; the scientist can guide the tracking process by changing criteria and parameters, resulting in different tracking solutions.

The paper is organized as follows, Section 2 summarizes our feature extraction procedure, Section 3 defines a number of components of the tracking algorithm, and Section 4 describes the algorithm itself. Then, two applications are described in Section 5, and finally, some conclusions and topics for future research are given.

2 Feature Extraction

The first step in feature tracking is the extraction of features from each frame. Feature extraction is a set of techniques in scientific visualization aiming at algorithmic, automated extraction of relevant features from data sets [13]. The tracking algorithm described here works on features that are 3D amorphous 'objects' of a size (much) smaller than the data set domain, but larger than a grid cell. The two essential attributes of such features are position and size. In principle, these can be obtained with any feature extraction method, but in this work we have used a method based on selective visualization, which is described in more detail in [13] and [7].

The result of the feature extraction is a number of features, each quantitatively described by a set of attributes, which can be visualized by an iconic representation. The collection of the attribute sets of all features is called *feature data.* The feature data lifts the original grid data field to a higher level of abstraction; data describing the features in contrast to data containing interesting features. We developed an object-oriented data structure in C^{++} which stores the feature data and which allows operations on the features. The possibility of manipulating the features as individual entities has important consequences: the features can be quantitatively compared, and for our tracking purpose the grid data is not needed anymore.

The transformation from grid data to feature data signifies a data reduction in the order of 1000. The feature data is a drastically reduced model of the original data. Therefore, one has to verify the accuracy of the calculated attributes before they can be used for tracking. In [7] we showed that the determination of position and size is very accurate and robust even for noisy data. Thus, the feature data is reliable and can be used for time tracking.

3 Tracking Components

3.1 Prediction

Our tracking algorithm is based on a simple assumption: **features evolve consistently**, i.e. their behavior is predictable. This implies that once a path of an object is found, we can make a prediction to the next frame and search for features in that frame that correspond to the prediction. A prediction can be made for the next frame at the end of the path, but also for the preceding frame at the beginning of the path. This means we can search forward and backward in time. Figure 1 shows six matched features that form the path of an object (dark objects), it shows the prediction at the end of the path (transparent object), and it shows a candidate feature (light object). Clearly, the candidate feature corresponds very well to the prediction and should be added to the path.

The concept of making a prediction and searching for candidate features corresponding to this prediction has two major advantages. Firstly, it allows tracking of fast moving small objects that do not overlap. Secondly, it allows tracking in two directions of time: forward and backward. Still, a correspondence must be detected between the prediction and the candidates in the next frame.

3.2 Correspondence functions

In order to detect a correspondence between the prediction and a candidate feature, we evaluate a number of *correspondence functions.* These correspondence

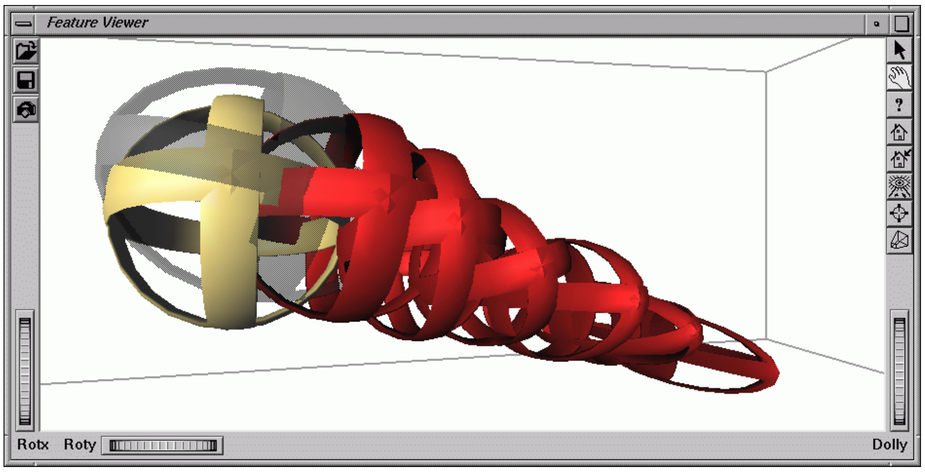

Fig. 1. A visualization of a path (dark objects), its prediction (transparent object) and one of the candidates (light object).

functions C_{func} are based on certain consistency rules like consistent growth, speed, rotation, etcetera. Each function is accompanied by a tolerance T_i and a weight W_i. The tolerance allows a deviation from the prediction and the weight indicates the importance of the function ($W_i \leq 0.0$ means no evaluation of the function at all). The correspondence factor returned by a function is $0 \leq C_{func} \leq 1.0$ if the candidate deviates from the prediction within the tolerance, and it is $C_{func} < 0.0$ if the deviation is larger then the tolerance (equation 1). The total correspondence between a prediction and a candidate is expressed as the weighted sum over all contributing functions (equation 2). The result has similar behavior as equation 1, hence a positive match between a prediction and a candidate is found when $Corr \geq 0.0$.

$$C_{func} = \begin{cases} 1 & \text{Exact match} \\ 0 & \text{Limit tolerance} \\ < 0 & \text{No match} \end{cases} \tag{1}$$

$$Corr = \frac{\sum_{i=1}^{N_{func}} C_i * W_i}{\sum_{i=1}^{N_{func}} W_i} \tag{2}$$

The number of possible correspondence functions depends on the attribute sets in a feature. Each attribute set yields a number of correspondence functions. For instance an ellipsoid fit contains information about size, position, and orientation. Therefore, the accompanying correspondence functions (table 1) are based on the consistency of growth, speed, and rotational speed. Consistency of growth is tested by the two volumes V, consistency of speed is tested by the distance between the two centroids $dist(p, c)$, and the rotational speed is tested by the dot scalar product of the main-axes of the two ellipsoids $= cos(\angle(\overline{p}, \overline{c}))$. It should be noted that the semantics of the correspondence functions depend on the physical phenomena underlying the dynamics of the features.

Correspondence function	Correspondence factor	consistency rule
$\frac{\lVert V_p - V_c \rVert}{\max(V_p, V_c)} <= T_{vol}$	$C_{vol} = 1 - \frac{\frac{\lVert V_p - V_c \rVert}{\max(V_p, V_c)}}{T_{vol}}$	consistent growth
$dist(p, c) <= T_{pos}$	$C_{pos} = 1 - \frac{dist(p, c)}{T_{pos}}$	consistent speed
$1 - \lVert cos(\angle(\overline{p}, \overline{c})) \rVert <= T_{angle}$	$C_{angle} = 1 - \frac{1 - \lVert cos(\angle(\overline{p}, \overline{c})) \rVert}{T_{angle}}$	consistent rotational speed

Table 1. Correspondence functions linked to the ellipsoid fit attribute set, p = prediction, and c = candidate.

The same correspondence factors are used to determine a *confidence index* for a complete path. Each connection (edge) between two features in this path gives two correspondence factors (forward and backward), except for the connection to end-nodes and from starting-nodes of the path. These only have one factor since only a prediction can be made in one direction. The confidence index of a complete path is calculated as follows:

$$Conf(\text{path}) = 1 - e^{-\frac{t}{\tau}} \qquad (3)$$
$$\text{with } t = \sum_{i=1}^{edges} C_i$$

where τ is a growth factor (it can be taken equal to the minimal path length discussed in section 4.1). The confidence factor increases as the length of the path increases, which is convenient since our confidence in a path increases for longer paths. The confidence index is used when a choice has to be made between two paths sharing the same feature.

4 Tracking Algorithm

With the prediction, correspondence factor and confidence index we have all the components needed for a successful tracking algorithm. In order to obtain a prediction, we still have to initialize a path before we are able to continue it with corresponding candidates in consecutive frames.

4.1 Initialization

The initialization of a path is achieved by assuming a correspondence between two features in two successive frames. This assumption leads to a prediction that can be compared to candidates in the third frame. If there is a candidate in the third frame that corresponds to the prediction, a new path is created and the path is continued into subsequent frames. Since a match in the third frame may be found coincidentally, an additional test on the resulting path is performed to ensure genuineness: the path length must be greater than a minimal path length (normally taken 4 or 5 frames).

Algorithm 1 Initialization, starting a new path

```
StartPaths()
{
    for_all (frame[i])
        for_all (unmatched features in frame[i])
            for_all (candidates in Frame[i+1]) {
                assume connection between $\text{feature}_i$ and $\text{candidate}_{i+1}$
                calculate $\text{prediction}_{i+1}$
                for_all (candidates in Frame[i+2]) {
                    if (Correspondence($\text{prediction}_{i+1}$, $\text{candidate}_{i+2}$))
                        create new path
                        ContinuePath(path, frame[i+3])
                    if (path->length >= MinPathLength)
                        add path to graph
                }
            }
}
```

Furthermore, two options are possible for the candidates: 1) all features in the next frame are candidates, and 2) only unmatched features in the next frame are candidates. The first option allows multiple solutions for one feature, but also requires an exhaustive search. The second choice limits the search because the number of candidates decreases when more paths are found, but it also removes a feature as possibility once a feature has been added to a path, i.e. the tracking solution depends on the order in which the features are tested. The pseudo code of the initialization is shown in Algorithm 1 (forward tracking). The worst case complexity of the initialization is of $O(nm^2)$, with n the number of frames and m the number of features per frame which is usually much less than 100. It should be noted that the number of unmatched features from which a path is started, decreases rapidly once a number of paths are found. So in practice the complexity will be much lower.

Once a path has been initialized, it is continued recursively until the path ends or until the last frame is reached. Obviously, this scheme may lead to multiple paths sharing the same feature: e.g. multiple candidates satisfy the prediction, or a candidate was already added to another path. This happens especially if the tolerances are relaxed. In case of multiple correspondences the path with the best confidence index gets the advantage.

4.2 Multiple Passes

The initialization can be performed forward and backward in time. This will yield different tracking results because the prediction is different in each direction. Correspondences may be found in one direction, but not in the other direction. Therefore, it is useful to go back and forth through the frames in several passes. If a pass is started with an existing tracking solution, the existing paths are first tested for extension before new paths are initialized. There are two ways to extend a path: first paths may be joined; if two paths start and end in successive frames, the two paths may be each other's continuation. Second, a path may be extended with unmatched features in the next frame.

Thus, multiple passes can be conducted on an existing correspondence solution. This also allows multiple passes with changed tracking parameters, for instance with different tolerances. Experience shows that good tracking results are obtained when the tracking is started with strict tolerances and for each successive pass relax the tolerances a bit (with only unmatched features as candidates). First, the obvious paths are found, and then more indistinct paths are found, thus the order of the tested features has less influences.

4.3 Feature Graph

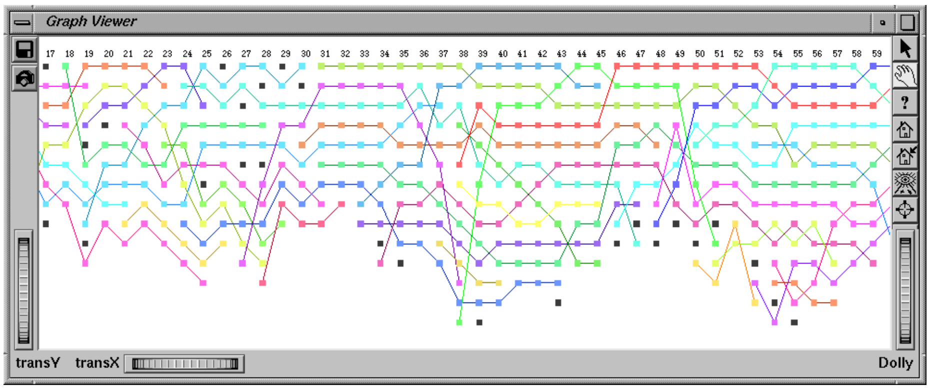

Fig. 2. The feature graph: a node represents a feature in a certain frame and an edge between two nodes represents a positive correspondence between the two features.

The result of the tracking algorithm can be visualized in a directed graph (Figure 2). The frames are plotted horizontally and the features are plotted vertically. Each frame is represented as a level in the graph and all features in that frame are depicted as a node at that level. An edge between two nodes visualizes an established correspondence between two features. Once two features are matched they belong to the same object, at two different instances of time. Therefore, all connecting nodes identify the path of the same object and the features in this path can be given the same object-id (and color). The path describing the motion of an object is stored in the graph as a collection of nodes and edges;

also, information is stored about the paths passing through each frame. The graph data structure holds both path-information and frame-information, i.e. it provides possibilities to walk through the graph in both longitudinal and transverse direction.

5 Applications

5.1 Flow Past a Tapered Cylinder

The first application is a flow past a tapered cylinder[1] consisting of 400 frames of 2.6 Mb each (Plot3D data, grid size 64x64x32, total data size $>$ 1Gb). Features were extracted from the enthalpy (enthalpy $\geq$ -0.6997), and for each feature an ellipsoid fit was calculated and stored in a feature data file (total size 476Kb). The resulting features are highly interacting regions in the wake of the cylinder. After executing six passes with increasing tolerances for the position and size functions, 280 paths were found and only 329 of the original 4121 features remained unmatched (more than 92% matched). These unmatched features can be explained by the fact that the continuation rules do not apply for feature interaction events such as split/merge. Figure 2 shows a small part of the complete graph, the features are vertically sorted by their size. Black nodes indicate unmatched features, and colors indicate the different paths. In a second viewer (the player), the 3D icons of the features can be viewed in a loop over the frames, or one frame can be selected and viewed (Figure 3). An animation can be found on the web [6].

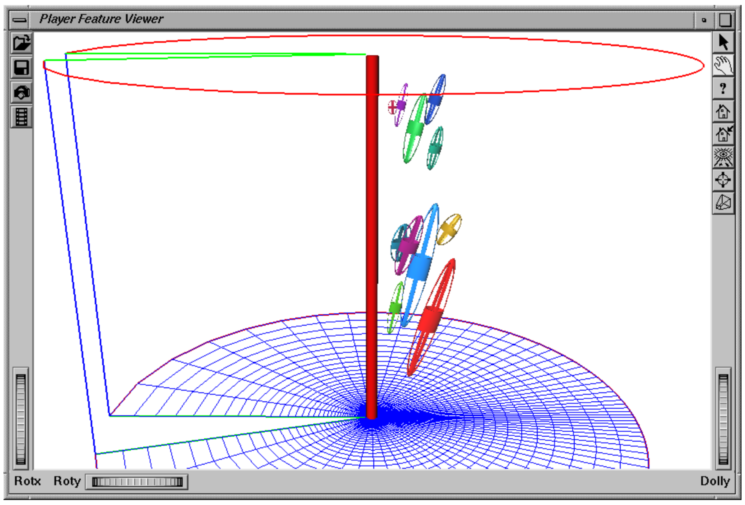

Fig. 3. Flow past a tapered cylinder.

[1] Data courtesy NASA Ames Research Center, http://science.nas.nasa.gov/Software/DataSets/

5.2 Turbulent Vortex Structures

The second application is a CFD simulation with turbulent vortex structures. We obtained the feature data as described in [11]. The feature data consisted of 100 frames with for each vortex structure its position, volume, and mass attributes. The 100 frames contain a total of 4903 features (an average of almost 50 per frame). After the tracking 277 paths were found and only 314 features remained unmatched. This example was a good test case for our tracking algorithm, because of the high average and strongly varying number of features per frame. We found that the algorithm had no problems in processing the data (each pass took less than a minute on an SGI Onyx with one 75 MHz R8000 processor and 256 Mb of memory). An animation of this example can also be found on the web [6].

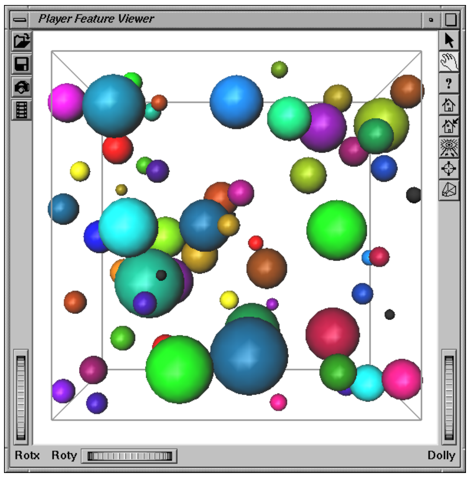

Fig. 4. Turbulent vortex structures.

6 Conclusions and Future Research

Time-dependent data can be analyzed and visualized by the feature tracking system discussed in this paper. The system first extracts interesting features from the data, then calculates basic attributes such as position and size for each feature, and solves the correspondence problem using these attributes. The solving-algorithm is based on the assumption that features evolve consistently and therefore their behavior is predictable. Using simple behavioral rules, we predicte the position and other attributes of each feature in the next frame. This prediction is used to find matching features in the next frame; all candidates in that frame are tested for correspondence with the prediction. Correspondence is found when a number of correspondence functions are satisfied. Each attribute set is associated with a number of correspondence functions that is tested with a certain tolerance and weight.

The initialization of a path is achieved by testing all possible connections until a match is found in the third frame. The resulting path is extended until no more connections can be found, or the path reaches the last frame. When the path satisfies a minimal path length it is added to the feature graph and all the features in the path are given an object-id. This process of finding continuing paths can be conducted forward and backward in time, thus allowing multiple-pass searching. A convenient scheme is to start with strict tolerances (finding obvious paths) and relax the tolerances each pass (finding less obvious paths). The applications show that this tracking system works fine, even with chaotic data with many interacting features.

Although the algorithm is currently limited to finding continuing paths, we believe that it provides a good starting point for finding events like split/merge, and birth/death (see [9]). Events may be detected by applying a new set of correspondence functions. In the future, also effort will be put in more sophisticated prediction schemes, other visualization and interaction techniques for user guided tracking, and tracking based on other attributes, such as skeleton descriptions.

Acknowledgements

The authors wish to thank dr. Jarke J. van Wijk of TU Eindhoven for the stimulating discussions, and prof. Deborah Silver and Xin Wang of Rutgers University for the use of their turbulent vortex data.
This work is supported by the Neth erlands Computer Science Research Foundation (SION), with financial support of the Netherlands Organization for Scientific Research (NWO).

References

1. G. Adiv. Determining 3D Motion and Structure from Optical Flows Generated by several Moving Objects. *IEEE Trans. on PAMI*, 7:384–401, 1985.
2. Y. Arnaud, M. Debois, and J. Maizi. Automatic Tracking and Characterization of African Convective Systems on Meteosat Pictures. *J. of Appl. Meteorology*, 31:443–453, May 1992.
3. D.H. Ballard and C.M. Brown. *Computer Vision*. Prentice-Hall, 1982. Chapter 7.
4. J. Becker and M. Rumpf. Visualization of Time-Dependent Velocity Fields by Texture Transport. In D. Bartz, editor, *Visualization in Scientific Computing '98*, pages 91–101. Springer Verlag, 1998.
5. D.S. Kalivas and A.A. Sawchuk. A Region Matching Motion Estimation Algorithm. *CVGIP: Image Understanding*, 54(2):275–288, Sep 1991.
6. F. Reinders. http://wwwcg.twi.tudelft.nl/~freek/Tracking. Web page with feature tracking examples.
7. F. Reinders, H.J.W. Spoelder, and F.H. Post. Experiments on the Accuracy of Feature Extraction. In D. Bartz, editor, *Visualization in Scientific Computing '98*, pages 49–58. Springer Verlag, 1998.
8. W.B. Rossow, A.D. Del Genio, and T. Eichler. Cloud-Tracked Winds from Pioneer Venus OCPP Images. *J. of Atmos. Sci.*, 47(17):2053–2082, Sep 1990.
9. R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing Features and Tracking Their Evolution. *IEEE Computer*, 27(7):20–27, July 1994.
10. I.K. Sethi, N.V. Patel, and J.H. Yoo. A General Approach for Token Correspondence. *Pattern Recognition*, 27(12):1775–1786, Dec 1994.
11. D. Silver and X. Wang. Volume Tracking. In R. Yagel and G.M. Nielson, editors, *IEEE Proc. Visualization '96*, pages 157–164. Computer Society Press, 1996.
12. D. Silver and X. Wang. Tracking Scalar Features in Unstructured DataSets. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *IEEE Proc. Visualization '98*, pages 79–86. Computer Society Press, 1998.
13. T. van Walsum, F.H. Post, D. Silver, and F.J. Post. Feature Extraction and Iconic Visualization. *IEEE Trans. on Visualization and Computer Graphics*, 2(2):111–119, 1996.