

Feature Tracking with Skeleton Graphs

Benjamin Vrolijk Freek Reinders

Frits H. Post

email: {b.vrolijk, k.f.j.reinders, f.h.post}@its.tudelft.nl

6th November 2000

Abstract

One way to visualize large time-dependent data sets, is by visualization of the evolution of features in these data. The process consists of four steps: feature extraction, feature tracking, event detection, and visualization.

In earlier work, we described the execution of the tracking process by means of basic attributes like position and size, gathered in ellipsoid feature descriptions. Although these basic attributes are accurate and provide good tracking results, they provide little shape information. In other work, we presented a way to describe the shape of the features by skeleton attributes.

In this paper, we investigate the role that the skeleton graphs can play in feature tracking and event detection. The extra shape information allows detection of certain events much more accurately, and also allows detection of new types of events: changes in the topology of the feature.

Keywords: Data Visualization, Feature Extraction, Feature Tracking, Skeleton, Skeleton Graph, Graph Matching, Event Detection.

1 Introduction

The problem of analysing very large scientific data sets originated the field of scientific visualization in the 1980s. The size of data sets has grown rapidly in the past decade, in particular with data sets generated from simulations of highly dynamic phenomena, such as time-dependent flows. Yet many visualization techniques, especially global field visualization techniques such as volume rendering or iso-surfaces, do not scale easily to very large data sets, and thus are not very well suited to the analysis of time-dependent data sets.

One solution to this problem is the approach called *feature extraction*, in which interesting phenomena (features) are automatically detected in large fields, and quantitatively described by computing attribute sets [15]. This results in a quantitative description of the features, which can be used for visualization and further investigation. Examples of features in flow data are vortices, shock waves, and recirculation zones. An advantage of this approach is that we concentrate on the relevant phenomena and the amount of data to be handled is reduced by a factor of 1000 or more.

Time-dependent data sets are often represented by a number of data fields (*frames*), one for each time step. If features are extracted from each frame separately, a correspondence between features in consecutive frames can be established, using the attribute sets. We call this process *time tracking* [11, 13, 8]. It allows us to visualize the development of these corresponding features over the total time interval of a simulation: motion paths, growth and shape changes can be shown. In these time histories, we can also detect interesting temporal changes (*events*), such as the birth of a new feature, the exit of a feature from the domain, or interactions of multiple features, such

as splitting into two or more features, or merging of multiple features into one [9]. Again, these events can be quantitatively described.

For time tracking, we have used primary attribute sets of the features: centre position, volume, size, and orientation [8]. The determination of these attributes is usually based on volume integrals of feature objects, and the accuracy and robustness of these attributes has been demonstrated [10]. The shape of a feature is described by a spatial distribution function, which gives a good estimate of the overall dimensions of a feature, but does not describe the feature's shape in any detail.

For some purposes, the feature shape is important and must be quantified in more detail. In a previous paper [7], we have proposed the use of skeletons and distance transforms for a better quantification of the shape of a class of features. A simplified skeleton can be represented as a graph, which can be used in combination with distance transforms to reconstruct the basic 3D shape of a feature, and can also be used as an iconic object for visualization.

In this paper we will investigate the role that these skeleton graph models can play in feature tracking and event detection. Generally, the tracking process will be guided by the primary attributes determined by volume integrals, but the skeleton graphs will permit detection of certain events with much higher accuracy. Also, the temporal development of feature shape is quantified in detail, and new types of events, such as changes in the topology of a feature, can be detected.

The structure of the paper is as follows. In Section 2, the feature extraction process is summarized. In Sections 3 and 4, we describe feature tracking and event detection methods using skeleton graphs. In Section 5 some related work is mentioned. We present some results in Section 6, including a comparison of integral properties and skeleton-based properties. We will then summarize and give future research directions in Section 7.

2 Feature extraction

The process of feature extraction can typically be described by the following four steps:

- segmentation or data selection: the parts of the data set are selected that are of interest. This can be done in many different ways, depending on the type of features to be detected. One general technique selects nodes in a grid by applying multiple thresholds to the data defined at each grid point, or quantities derived from the data [15]. In principle however, any segmentation technique that produces a set of spatially located or grouped data items can be used.
- clustering: from the data items, coherent groups can be formed by building coherent clusters from spatially related items. This results in a number of clusters, which can be treated as distinct objects.
- attribute calculation: for each cluster, a number of descriptive attributes are calculated, such as centre position, volume or orientation.
- iconic mapping: the attributes of the clusters are mapped to the parameters of parametric icons, which can be easily visualized.

We have often used volume integrals to compute the attributes of the features. This gives us the volume and the centroid of the features. By computing the eigenvalues and eigenvectors of the covariance matrix of the point positions, we can get a first-order estimation of the size and orientation. These attributes can be used to create an ellipsoid representation of the features, for visualization. In the rest of this paper, we will also use the terms ellipsoid volume and ellipsoid position, when we mean these attributes, computed by a volume integral.

Because the ellipsoid representation does not give a real description of the shape and orientation of the feature, we have created another attribute set, to describe the shape in more detail.

We have developed an algorithm to create a *skeleton graph representation* of a feature [7]. The algorithm first computes the skeleton of an object. This skeleton consists of the central points of

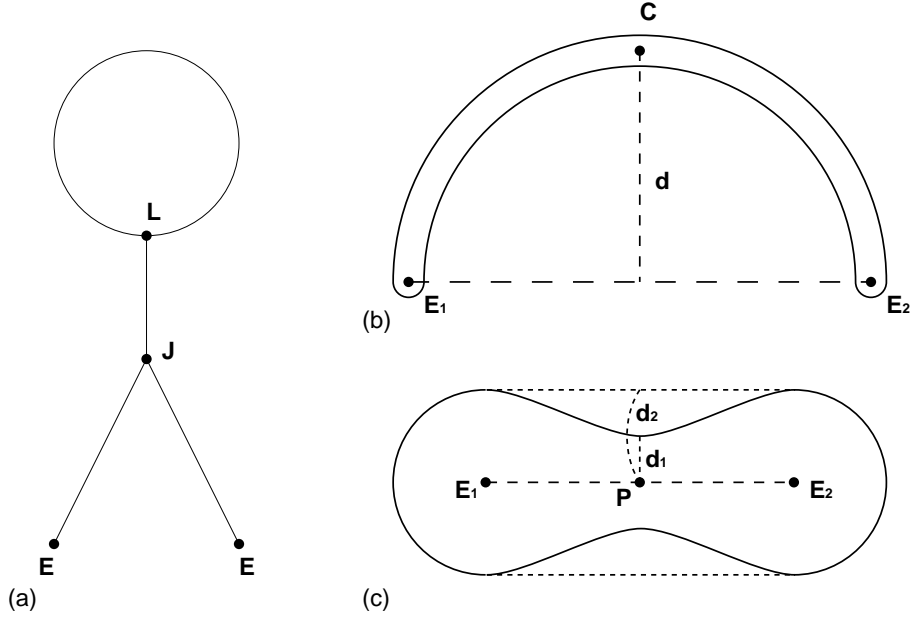


Figure 1: a) The topological node types. b) Adding a curve node. c) Adding a profile node.

the object, and is obtained by successively removing the outer layers of object points, as long as the topology of the object does not change.

Also, a distance transformation is computed, which stores for each point the minimal distance to the surface of the object.

From these data sets, we create a skeleton graph. The points of the skeleton become the nodes of the graph and the neighbouring nodes are connected by edges. Each node gets two attributes: the position of the node and the distance transform (DT) at that point.

This graph can then be simplified. To describe the topology of the object, we only need a set of special points. We distinguish three different types of topological nodes (see Figure 1a): *end* (E), *junction* (J) and *loop* (L) nodes. End nodes have one edge, junction nodes have more than two edges and loop nodes have a topological edge to themselves.

Besides these topological nodes, we introduce two types of geometric nodes for describing the shape more accurately. The first type is the *curve node* (see Figure 1b). Here, the skeleton is strongly curved. The topology of the feature consists of only one edge (E_1E_2). However, to be able to reconstruct the path of the skeleton more accurately, we have to add at least one node (C) in between. We call this node a curve node. The number of curve nodes added is dependent on a user provided tolerance. This tolerance gives the maximum allowed distance (d) of the skeleton points from the skeleton graph edges.

The second type is the *profile node* (see Figure 1c). Here, the skeleton of the feature is a straight line, but the object surface is curved. Thus, the distance transform varies along the topological edge E_1E_2 . Therefore, to be able to reconstruct the surface of the feature more accurately, we have to add a profile node (P) halfway. The number of profile nodes added is dependent on a user provided tolerance, which determines the maximum allowed distance of the interpolated DT (d_2) from the real DT (d_1).

We now have a graph, which exactly describes the topology of the object, and approximately describes the shape. We can reconstruct the surface of the object by using the DTs of the nodes. By representing the nodes by spheres with radius DT, and the edges by conical segments, we get an approximate reconstruction of the original shape. This reconstructed shape can be used for visualization and for attribute calculation.

The reconstructed shape is only an inscribed object of the original shape, because the distance used is the minimal distance to the surface. Therefore, the attribute calculation will not be as accurate as the volume integral calculation, but when the latter is not available, the former can be used instead. We have done a few tests to determine the accuracy of these calculations; the results of these tests will be described in Section 6.

3 Feature tracking

When features have been extracted in every frame of a time-dependent data set, still no motion information is available. To obtain this, we have to establish a frame-to-frame correspondence between features in successive frames. This is called the *correspondence problem* [2]. There are several ways to solve this problem [11, 13, 9]. Our algorithm is based on the assumption that features evolve predictably. Once a part of a path of a feature has been found, we assume we can make a prediction of the path into the next frame. We then compare the prediction with the real features in that next frame and search for a match. If one or more matches are found, we add the best match to the end of the path and continue into the next frame. For the prediction we simply use linear extrapolation. We assume for example, that if a feature moves a distance d from frame $t - 1$ to frame t , it will move the same distance from frame t to frame $t + 1$. A similar assumption is made for volume, orientation, etc. For the matching of a prediction with a feature, we need a way to determine a correspondence between two features.

We use functions that compute a correspondence factor for each type of attribute. For the volume integral attributes, for instance, we have correspondence functions for the position and the volume (see Table 1).

Correspondence criterion	Correspondence function	Correspondence factor
Position	$\ P_1 - P_2\ \leq T_P$	$C_P = 1 - \frac{\ P_1 - P_2\ }{T_P}$
Volume	$\frac{ V_1 - V_2 }{\max(V_1, V_2)} \leq T_V$	$C_V = 1 - \frac{ V_1 - V_2 }{\max(V_1, V_2) T_V}$

Table 1: Examples of correspondence functions for feature tracking. P is position, V is volume, T is tolerance.

When using the skeleton graph representations of features for tracking, we can use correspondence functions based on the topology of the skeleton graphs. For this, we could use, for example, a tolerant graph matching algorithm, such as described by B.T. Messmer in [5]. However, because of the high time and/or space complexity of these algorithms, we have created our own, more efficient special-purpose comparison algorithm.

This algorithm chooses one node in the first graph and compares that node with all nodes of the same type in the second graph. In the recursive function that compares both nodes, two arrays are used to keep track of the marked nodes in both graphs. If the number of edges from both nodes is equal, and the number of neighbouring marked nodes is equal, then both nodes are marked. Next, an unmarked neighbouring node in the first graph is chosen and compared with all unmarked neighbouring nodes in the second graph. This process is repeated until either all nodes in both graphs are marked or no more correspondences are found. In the first case, the two graphs are topologically equivalent, in the second case, the topologies differ.

To make the whole comparison algorithm more efficient, we have created two more functions that are run before the recursive algorithm.

The first function simply counts the number of edges and topological nodes of each type in both graphs. If these numbers do not correspond, the graphs will certainly not be topologically

equivalent. However, if the numbers are equal, further testing is necessary.

The second function counts the types of the nodes each node is connected to, or equivalently, counts the types of connections (end–end, end–junction, etc.) in both graphs. Again, if the numbers do not correspond, the two graphs have different topologies. Otherwise, the recursive comparison algorithm is run to give the definitive answer.

Note that we only test the topology of the skeleton graphs, not the geometry. Because the geometry of the skeleton graphs is based on tolerances provided by the user during the feature extraction process, it is possible that, for example, the number of curve nodes changes, while the shape of the feature hardly changes. The topology of a skeleton graph is more stable than the geometry, therefore we have decided to use only the topological information of the skeleton graphs in these correspondence functions.

Note also, that it is possible that, although the topology of a feature stays the same, the shape of the feature changes very much. For instance, when a junction disappears on one side and at the same time a junction appears on the other side. We do not detect these events, because for that we need a one-to-one mapping of the nodes and edges of two skeleton graphs. Because features evolve (grow/shrink and move) anyway, it is very hard to compute such a correspondence.

We have made one topology correspondence function that returns a boolean, that is, there is no correspondence if the topology changes. A second correspondence function allows a certain number of topological changes, such as the addition of loops or edges.

Because in our test application¹ sometimes the topology of the features changes more than once within a few frames, we have decided not to use the topology information for feature tracking, although it is, of course, still possible.

We have found that it works very well to search for continuing paths, using only the global attributes of the features. Therefore, we have decided to use the position and volume correspondence functions from Table 1 also when using skeleton graph descriptions of features for time tracking. Therefore, we have to compute the position and the volume of a skeleton graph.

Furthermore, we can compute another global attribute of a skeleton graph, namely its length. We compute the total length of a skeleton by adding the individual lengths of all edges of the graph. The correspondence function for the length of a skeleton graph is comparable to the function for the volume.

The volume of a skeleton graph is computed by adding the volumes of all conical segments. The volume of one segment is computed by using the formula for the volume of a conical frustum:

$$V = \frac{1}{3}\pi h(r_1^2 + r_1r_2 + r_2^2) \quad (1)$$

with h the length of the edge and r_1 and r_2 the DTs of the end nodes of the edge. To make the computation of the total volume more accurate, we add the volume of half a sphere for each end node of the skeleton.

The position of a skeleton graph is defined to be the centre of gravity (COG) of the object, assuming a uniform density throughout the object. We can compute this by calculating the weighted average of the centres of gravity of the individual segments, using the segment volumes as weights. The COG of a conical frustum is located at

$$COG = p_1 + h(p_2 - p_1), \quad (2)$$

with p_1 and p_2 the positions of the end nodes and

$$h = \frac{\frac{1}{2}\sqrt[3]{4r_1^3 + 4r_2^3} - r_1}{r_2 - r_1}, \quad (3)$$

the relative position of the COG on the edge from p_1 . For an edge with $r_1 = r_2$, that is a cylinder-shaped edge, the COG is located exactly halfway, so $h = 0.5$.

¹We use a 91 frame data set with turbulent vortex structures, obtained from a fluid dynamics simulation. Data courtesy D. Silver and X. Wang of Rutgers University.

4 Event detection

When the continuing paths have been found, we can search for events in the evolution of the features. For instance, when a path ends, we would like to know why this happens. Examples of events are the birth or death of a feature, entry into or exit from the domain, and the splitting of one feature or merging of multiple features into one [9]. All these events come in pairs, which are each other's opposites in time. That means that if one event can be found in forward time direction, the other can be found by using the same algorithm in backward time direction. With the detailed shape information that is available with skeletons it is now also possible to detect changes in topology. We call this topological events.

4.1 Terminal events

When a feature decreases in size and disappears, we say a *death event* has occurred. We can detect such an event by comparing the volume of the features in the last frames of the path. When the volume is decreasing and the volume of the prediction is very small or negative, we call it a death event. A *birth event* is the same, only in negative time direction.

When a feature moves to an open boundary of the domain, it is possible that the feature leaves the domain through that boundary, thus ending the path. We can detect such an event by watching the position of the feature in the last frames of the path. When a feature moves towards the boundary and the position of the prediction is close to or beyond the boundary, we say we have found an *exit event*. When the same happens in opposite time direction, we call it an *entry event*.

4.2 Interaction events

When multiple features merge into one, we call this a *merge event*. In the opposite time direction, it is called a *split event*. We search for merge events when there are two or more paths ending in one frame and the ends are unresolved. We then try to merge them into a candidate feature in the next frame, for example the first feature of another path. When we have a number of end nodes in one frame and a candidate feature in the next frame, we first limit the number of end nodes by applying a neighbourhood criterion: the prediction of the end node must be within a certain search space defined around the candidate feature. When two or more end nodes remain after this selection, we try each possible combination of these end nodes and compute a merged feature from these end nodes. Next, we compare the computed merged feature with the candidate feature in the next frame. For this final comparison, the usual continuation criteria can be used. Thus, to be able to search for split and merge events, we need two extra functions: a neighbourhood function and a merge function. For ellipsoid representations, the neighbourhood function computes the distance between the centres of the ellipsoids. The merge function creates a new hypothetical feature, with a volume equal to the sum of the volumes of the merged features, and a position equal to the weighted average of the individual ellipsoid positions. When using skeleton graph representations, we can compute a much more accurate neighbourhood distance (see Figure 2). The most accurate measure is the edge-to-edge distance, that is, the shortest distance between any two edges of the two skeletons. Two simpler but less accurate measures are the node-to-edge and the node-to-node distances. A fourth distance measure for skeletons is comparable to the measure used for ellipsoids and uses the distance between the centres of gravity of both skeletons. The merge function for two skeletons creates a hypothetical skeleton consisting of all the nodes and edges of the separate skeletons, plus an extra edge connecting the two nearest nodes.

4.3 Topological event

Besides position and size, also the topology of a feature can change during the evolution. When this happens, we call this a *topological event*. We detect this event by using the topology comparison algorithm described earlier. When one of the three tests fails, we know the topology has changed.

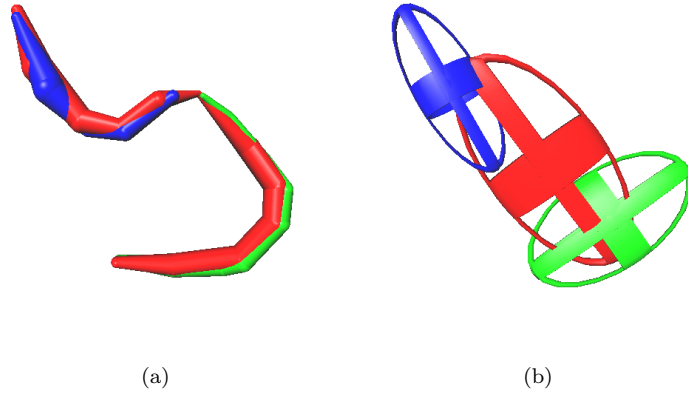


Figure 2: The same split event, using different feature representations. a) skeleton representation, b) ellipsoid representation. The red feature is before split, blue and green are after split.

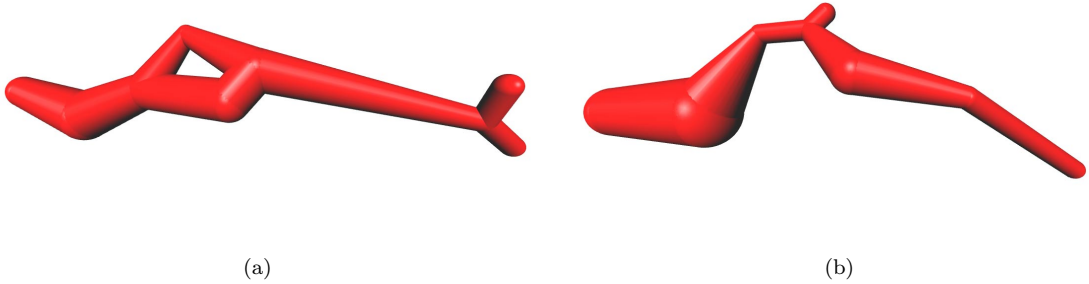


Figure 3: A loop event has occurred. In figure a) the skeleton contains a loop, in figure b), the next frame, the loop has disappeared.

Next, we try to determine what kind of event has happened. We make a distinction between a *loop event* and a *junction event*. Using Euler's formula

$$V - E + F = C - H \quad (4)$$

it is easy to decide what has happened. In formula (4), V is the number of vertices, E the number of edges and F the number of faces. $C - H$ is called the Euler number, with H the number of holes and C the number of connected objects. Because we are only dealing with connected graphs, F is always 0, and C always 1 [7]. This way formula (4) reduces to:

$$H = 1 - V + E \quad (5)$$

When the value of H changes, we call it a loop event (see Figure 3). If not, but the number of (topological) edges is different, a junction has originated or disappeared, therefore we then call it a junction event. If neither of these is true, but the topology has changed in any other way, we simply call it a topological event.

5 Related Work

Solving the correspondence problem is an issue not only in scientific visualization, but also, for example, in image processing and computer vision. For 3D field data, the matching can be achieved by two methods: region correspondence or attribute correspondence [9].

- With methods based on region correspondence, the matching can be achieved based on spatial overlap [1, 13, 14], or by finding the minimum affine transformation that transforms an object from one frame to the next [3].
- The methods based on attribute correspondence use attributes of the features, such as position and volume, for matching. Examples are the tracking of tokens by using motion smoothness [12], and the tracking of feature evolution [11] for event detection.

Our approach to solving the correspondence problem uses the second method. In the feature extraction process, we compute the attributes of the extracted objects. These attributes are then used for computing the correspondence. The comparison is based on high-level attribute data, not on the original data. Therefore, the comparison is simple and efficient. Furthermore, the correspondence functions can be physics-based, and different, depending on the feature type.

For comparison of graphs, Messmer and Bunke have investigated error-tolerant and error-correcting subgraph isomorphism detection, for example using decision trees [5, 4, 6].

In our application, the basic feature tracking is done using the integral attributes, without detailed comparison of the graphs. Changes in the topology of a feature are detected by the algorithm, described in Section 3. This algorithm is less complex than the ones described by Messmer and Bunke, but suffices for our purposes. Complex graph isomorphism analysis is not necessary for detecting loop and junction events, but it may be useful for more detailed analysis of topological events.

6 Results

Because we can use the reconstructed volume of the skeleton graphs for time tracking, we have performed a number of tests to determine how good the approximation is. We use the volume computed by a volume integral for comparison.

We have tested the influence of the shape of a feature on the volume reconstruction of the skeleton graph description of the feature. For this test, we have generated a number of ellipsoid-shaped features, with different axis ratios. We have three extreme cases:

1. a 'zeppelin-shaped' ellipsoid (axis $R_1 == R_2 \ll R_3$)
2. a 'sphere-shaped' ellipsoid (axis $R_1 == R_2 == R_3$)
3. a 'disc-shaped' ellipsoid (axis $R_1 == R_2 \gg R_3$)

The skeleton of a zeppelin is one topological edge, so the accuracy of the volume reconstruction is dependent on the number of profile nodes in between. The skeleton of a sphere is a single node, so the sphere should be represented very well. The skeleton of a disc is also a single node, but because the DT is very small, the volume estimation of the skeleton graph representation will be very bad.

Figure 4 shows the results of our tests. On the y-axis is the ratio of the skeleton volume and the integral volume, which should ideally be 1. On the x-axis is the ratio of the two equal axes to the third axis. In the middle is the sphere, with ratio 1. To the right the ratio is > 1 , that is the disc shape. To the left the ratio is < 1 , the zeppelin shape. It is easily seen, as expected, that flat objects are not represented very well; the volume ratio decreases very quickly on the right side of the graph. It is also clear from the left side of the graph, that the profile nodes are very important to get a good shape and volume approximation.

As a final test to check the accuracy of the skeleton volume, we have computed the average volume ratio for all features in the complete 91 frame data set of our test application. This average ratio is about 42%.

When comparing the skeleton graph centre of gravity with the volume integral position, we find an average distance of 1.40 voxels in a data set of 128^3 voxels.

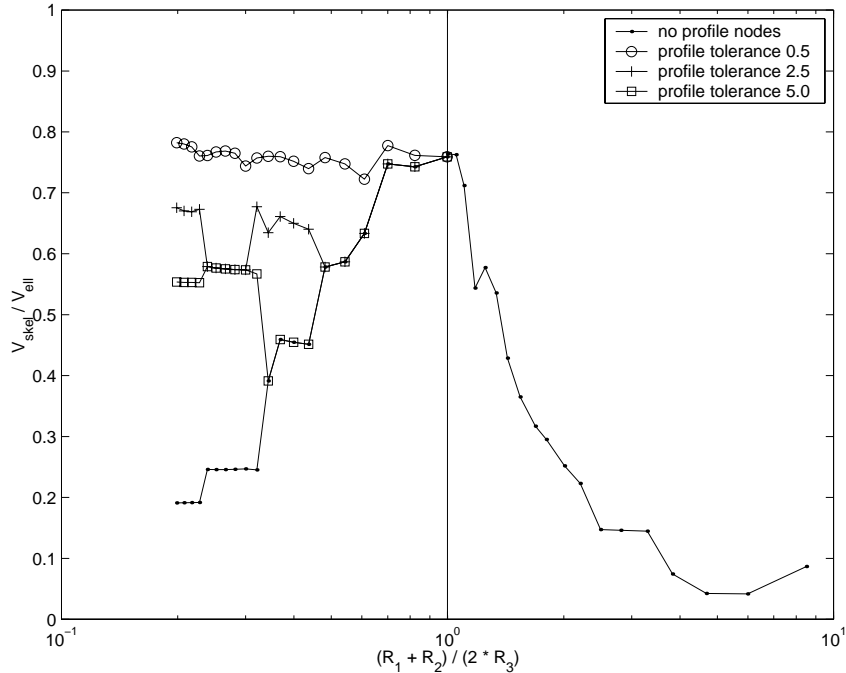


Figure 4: The skeleton volume ratio for different axis ratios.

Because the volume estimation of the skeletons is not very accurate, tracking features using this attribute will probably not work as well as when using the integral volume. We have performed an experiment tracking the same data set using different sets of correspondence functions. The first set uses the ellipsoid position and ellipsoid volume attributes. Both of these are computed using a volume integral, so both are accurate attributes of the original feature. The second set of functions uses the skeleton graph position (COG) and volume attributes. Again, especially the skeleton graph volume is not very accurate. The third set is similar to the second, except we are also using the skeleton graph length. The fourth set is a combination of the first and second, that is, it uses both the ellipsoid and skeleton position and volume attributes. Finally the fifth set also uses these attributes, plus the skeleton length.

Using these sets of correspondence functions and with the same parameters, we have tracked one data set with a number of iterations and computed what percentage of the total number of features was resolved. During these iterations the tolerances are subsequently raised until the final maximum allowed tolerance. For example, with a tolerance of 10, in 4 iterations, the tolerances used would be respectively 2.5, 5, 7.5 and finally 10. The maximum tolerances we have used are 15 voxels for position and 0.5 (50%) for volume and length. These values have been reached in 10 iterations. The minimal path length allowed is 4 frames. In a final iteration we searched for remaining paths of length 3. The results of this experiment are shown in Figure 5.

In the graph there is an extra line, called 'ell OR skel', which is a special case. For this test, we have alternately used the ellipsoid and skeleton functions, instead of simultaneously. In this case correspondences are found when predictions comply with either the ellipsoid or the skeleton functions. The results are therefore better than the rest. The number of iterations, however, is doubled. The maximum percentage of solved features, achieved in this final test, is 92.7%. The rest of the cases are all very close at the end, ranging from 88.9% to 91.5%. After the first iterations there is a large variation, with the ellipsoid functions scoring the highest percentage. At the end, the skeleton functions score slightly higher than the ellipsoid functions, and the combination of both ('ell+skel') is even better. We have also compared the paths resulting from the ellipsoid and skeleton function sets. They match for 95.70%, that is, in the data set of 20 frames with a total of

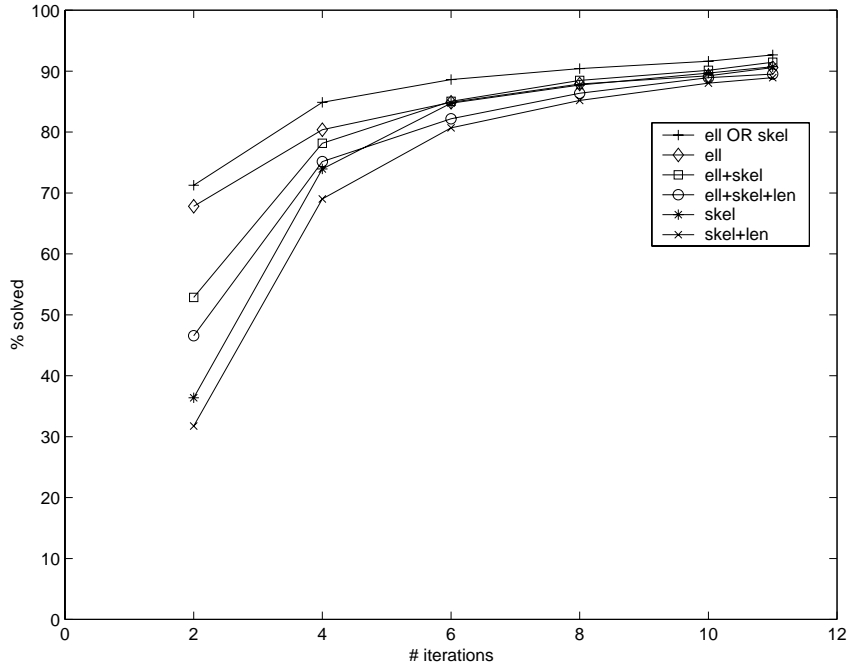


Figure 5: The percentage resolved features for different sets of correspondence functions.

668 features, 557 correspondences were found in both cases, 13 correspondences found only using the skeleton set, and 12 correspondences found only by the ellipsoid set.

As seen in Figure 2, the skeleton neighbourhood functions will probably work much better for finding split and merge events, than those based on ellipsoid attributes. To test this hypothesis, we have performed event detection on one data set with four different neighbourhood functions. We have used the skeleton edge-to-edge and the simpler node-to-edge neighbourhood functions. Furthermore, we have used the skeleton COG and the ellipsoid position neighbourhood functions. Figure 6 shows the relation of the number of split/merge events found and the tolerance needed to detect them, for the four different neighbourhood functions. To find all events, we need a four times larger tolerance with ellipsoids, than with skeletons. This means that a much more restrictive test can be used for the skeletons, and thus the reliability of the results is higher.

Of course, it is important to measure the performance of our algorithms. In a 91 frame data set with 3864 features and 261 paths we searched for split and merge events. On a SGI Octane with a 225 MHz processor and 256 Mb of RAM, it took 19.8 seconds to find 39 events using the edge-to-edge neighbourhood function with a tolerance of 15. When using the ellipsoid position neighbourhood function instead, with the same tolerance, it took 5.6 seconds. However, only 23 events were found. To find 39 events, we needed a tolerance of 24 and it took 25.8 seconds. Thus, the edge-to-edge neighbourhood function is 3 to 4 times slower than the ellipsoid neighbourhood function. However, because a smaller tolerance suffices to find the same number of events, eventually the use of the skeleton neighbourhood function is faster.

In the same data set we searched for topological events, to test the performance of our topology comparison algorithm. It took 0.3 seconds to search the whole data set and find 199 junction and 4 loop events.

As an example, in Figure 7 are a number of frames in the evolution of a single feature. A number of topological and interaction events have been detected, for example, in frame $t = 72$, a merge event has occurred, and in frame $t = 89$, the large feature has split in three.

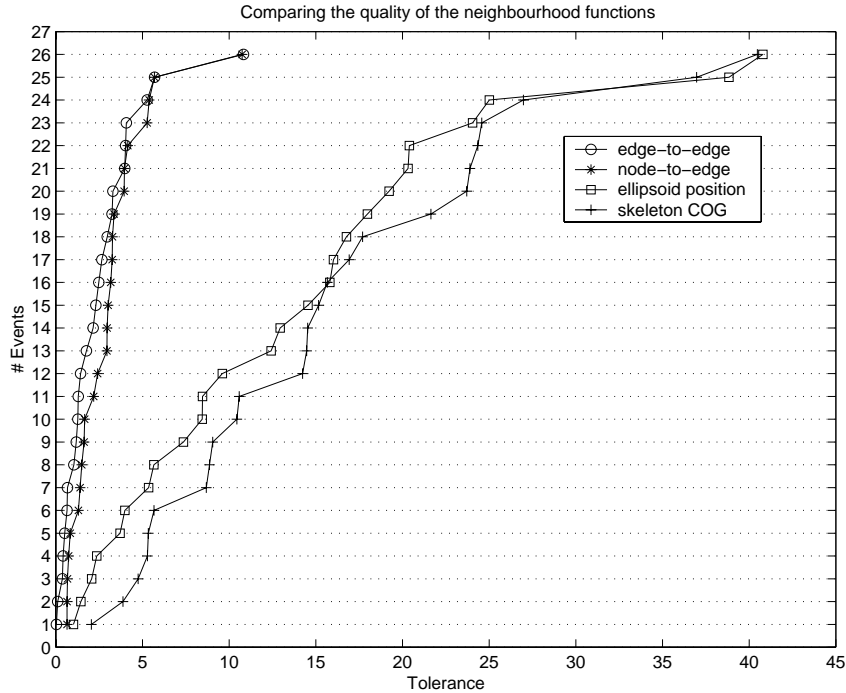


Figure 6: The number of split/merge events found against the tolerance for different skeleton and ellipsoid neighbourhood functions.

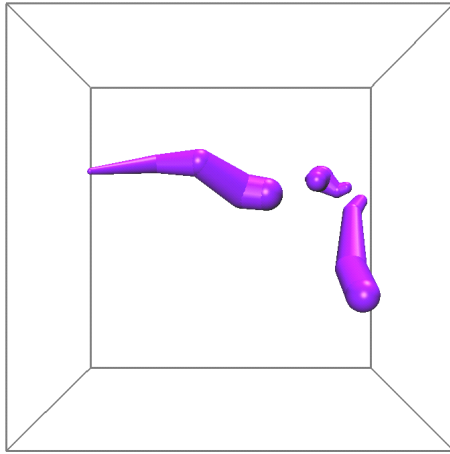
7 Conclusions and Further Research

In Figure 5 it is shown that using ellipsoid attributes for tracking continuations gives better results with less iterations. However, when tolerances are increased, the results draw nearer to each other and can even become better when using skeleton attributes or both types of attributes. Furthermore, as shown in Figure 6, the skeleton neighbourhood functions are much better for finding split and merge events, than those based on ellipsoid data. And finally, the skeleton attribute sets give much better shape information, allowing us to look for a completely new type of events: the topological events.

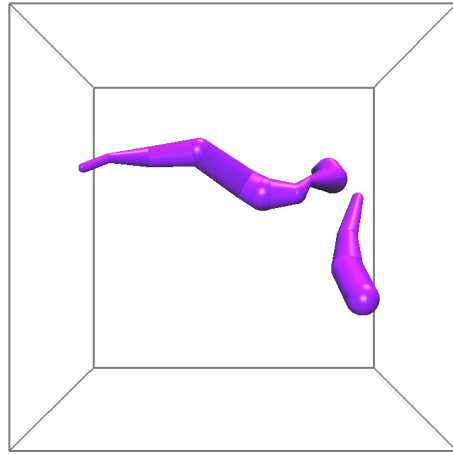
More research could be done to create a more accurate line skeleton representation, in which the contour of the skeleton is represented more accurately. In our skeletons, nodes are represented by spheres, with a radius equal to the nearest distance to the surface, and edges by conical segments connecting these nodes. Therefore, a cross-section of the skeleton, perpendicular to an edge, is always a circle. A more accurate representation could be to compute a complete contour of the feature, more or less simplified. This is, of course, computationally expensive. Much simpler, but still providing a much better volume estimation of the feature than our current skeletons, would be to compute an ellipse fitting of the cross-section, or to use a circle with radius equal to the average distance instead of the minimal distance to the surface.

A next step would be to develop a new representation for surface skeletons. Our line skeletons work well for the test application we have used, with mostly tube-like features. But for applications with features such as shock waves or separation surfaces, this method does not work very well.

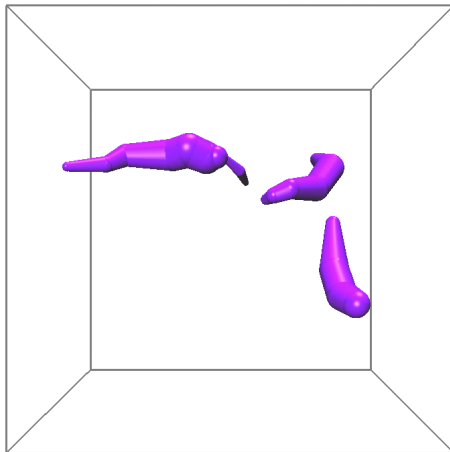
Future work could include the development of geometrical events, for detecting all significant shape changes. Also, more precise descriptions of events could be developed, for example, describing not only the fact that two features merge, but also exactly how the original features are connected.



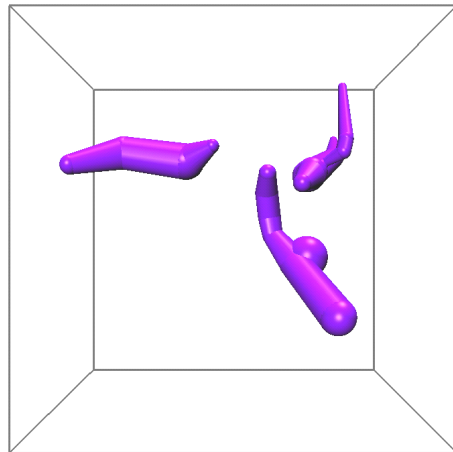
(a) $t = 70$



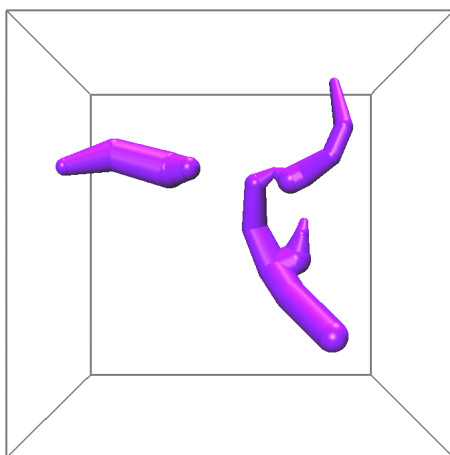
(b) $t = 72$



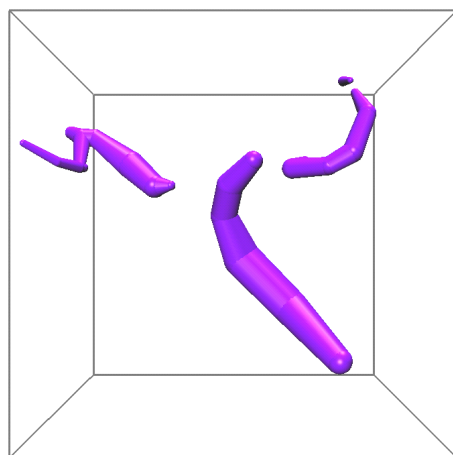
(c) $t = 75$



(d) $t = 81$



(e) $t = 84$



(f) $t = 89$

Figure 7: Six frames in the evolution of an object.

References

- [1] 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.
- [2] D.H. Ballard and C.M. Brown. *Computer Vision*. Prentice-Hall, 1982.
- [3] D.S. Kalivas and A.A. Sawchuk. A Region Matching Motion Estimation Algorithm. *CVGIP: Image Understanding*, 54(2):275–288, Sep 1991.
- [4] B.T. Messmer and H. Bunke. Error-correcting graph isomorphism using decision trees. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(6):721–742, 1998.
- [5] B.T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–505, May 1998.
- [6] B.T. Messmer and H. Bunke. A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recognition*, 32(12):1979–1998, 1999.
- [7] F. Reinders, M.E.D. Jacobson, and F.H. Post. Skeleton Graph Generation for Feature Shape Description. In W. de Leeuw and R. van Liere, editors, *Data Visualization 2000*, pages 73–82. Springer Verlag, 2000.
- [8] F. Reinders, F.H. Post, and H.J.W. Spoelder. Attribute-Based Feature Tracking. In E. Gröller, H. Löffelmann, and W. Ribarsky, editors, *Data Visualization '99*, pages 63–72. Springer Verlag, 1999.
- [9] F. Reinders, F.H. Post, and H.J.W. Spoelder. Visualization of Time-Dependent Data using Feature Tracking and Event Detection. *The Visual Computer*, 2000. Accepted for publication.
- [10] 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, April 1998.
- [11] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing Features and Tracking Their Evolution. *Computer*, 27(7):20–27, July 1994.
- [12] I.K. Sethi, N.V. Patel, and J.H. Yoo. A General Approach for Token Correspondence. *Pattern Recognition*, 27(12):1775–1786, Dec 1994.
- [13] D. Silver and X. Wang. Volume Tracking. In R. Yagel and G.M. Nielson, editors, *Proc. Visualization '96*, pages 157–164. IEEE Computer Society Press, 1996.
- [14] D. Silver and X. Wang. Tracking Scalar Features in Unstructured DataSets. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proc. Visualization '98*, pages 79–86. IEEE Computer Society Press, 1998.
- [15] 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.