Interactive Feature Extraction and Tracking By Utilizing Region Coherency

Chris Muelder* Kwan-Liu Ma†
University of California, Davis

Abstract

The ability to extract and follow time-varying flow features in volume data generated from large-scale numerical simulations enables scientists to effectively see and validate modeled phenomena and processes. Extracted features often take much less storage space and computing resources to visualize. Most feature extraction and tracking methods first identify features of interest in each time step independently, then correspond these features in consecutive time steps of the data. Since these methods handle each time step separately, they do not use the coherency of the feature along the time dimension in the extraction process. In this paper, we present a prediction-correction method that uses a prediction step to make the best guess of the feature region in the subsequent time step, followed by growing and shrinking the border of the predicted region to coherently extract the actual feature of interest. This method makes use of the temporal-space coherency of the data to accelerate the extraction process while implicitly solving the tedious correspondence problem that previous methods focus on. Our method is low cost with very little storage overhead, and thus facilitates interactive or runtime extraction and visualization, unlike previous methods which were largely suited for batch-mode processing due to high computational cost.

Index Terms: I.4.6 [Computer Graphics]: Segmentation—Region growing, partitioning; I.4.7 [Computer Graphics]: Feature Measurement—Feature representation

1 Introduction

The growing power and capability of supercomputers enables scientists to simulate sophisticated physical phenomena and chemical processes with increasing details and fidelity, leading to important advances in their respective areas of study. A typical simulation can generate time-varying 3D volume data with a size ranging from a few to hundreds of terabytes, which is too large to visualize directly and store completely. The situation is becoming worse as the data size continues to increase in a rapid rate, making it difficult or mostly impossible for the scientists to examine the full extent of the data. Since the value of the output data is determined by scientists' ability to extract knowledge from it, the data must be either reduced to some degree or distributed for processing to make it manageable and visualizable. Furthermore, even when it is possible to visualize the whole data set, the complexity of the data could result a visualization that is either too cluttered or confusing to interpret. Feature extraction is a data reduction solution based on scientists' knowledge about their data. It allows scientists to not only more compactly store the essential aspects of their data, but also more economically examine these aspects and more concisely communicate with others their findings.

With feature extraction, instead of visualizing the entire data set at once, features of interest are isolated (i.e., segmented) from the overall data so they can be more easily transferred, stored, examined, and presented. For example, extracted features can simply be colored differently in order to make them more distinct. Or, they can be shown individually to better illustrate specific properties. Most often though, a set of features is used to simplify the data set, since after the features are extracted, the rest of the data can be discarded. Doing so greatly reduces the size of the data, and can make exploring such a data set more interactive.

*muelder@cs.ucdavis.edu
†ma@cs.ucdavis.edu

When analyzing features of time-varying data, not only do they need to be extracted, but they need to be tracked through time. In order to follow the evolution of individual features, the features must be consistently classified from time step to time step. Otherwise the feature information can not be coherently presented in the visualization. For example, if color is being used to distinguish different features, then a single feature must maintain the same color throughout time for the color to be helpful. This task is frequently referred to as the *correspondence problem* [11].

Existing feature extraction techniques tend to fall into one of two categories: correspondence based approaches and spatio-temporal extraction approaches. The majority of feature tracking techniques are correspondence based approaches, which extract features in each time step separately then correspond the extracted features in consecutive time steps based on criteria such as position, size, orientation, or region overlap [12, 13, 14, 16, 18, 17]. These approaches generally are fairly robust, but do not reuse previous extraction information in the extraction process. On the other hand, spatio-temporal extraction approaches treat the time axis just like a spatial axis, then apply a standard feature extraction technique such as region growing or isosurfacing to the entire spatio-temporal domain at once [2, 25]. These make better use of temporal coherency, but they require that all time steps be available at once, and they still do full extraction in each time step. The approach presented here does not follow either of these paradigms.

This paper presents a new method to track individual features through time more efficiently than correspondence based methods and more reliably than spatio-temporal methods, by exploiting the incremental feature changes that occur between time steps. This is achieved by utilizing information from previous time steps to predict a feature's region in the current time step, then correcting the predicted region to extract the feature's actual region through growing and shrinking processes. We demonstrate the effectiveness of our method using two turbulent flow data sets. We also show its efficiency by presenting several timing tests.

1.1 Related work

Feature extraction and tracking is a well established technique for analyzing time-varying data in a variety of fields, such as video analysis [4, 24], computer vision [19, 5, 23], and flow visualization [11]. In all of these fields, feature tracking is applied by first extracting features, and then following them through time. The extraction techniques vary between applications and are often data dependent. Many applications use methods based on image processing techniques, such as region growing, thresholding, and edge detection [11]. These methods effectively segment the data, as is often done with medical data [7]. Work has also been done to extend or accelerate them [15]. Other techniques employ a neural net to determine

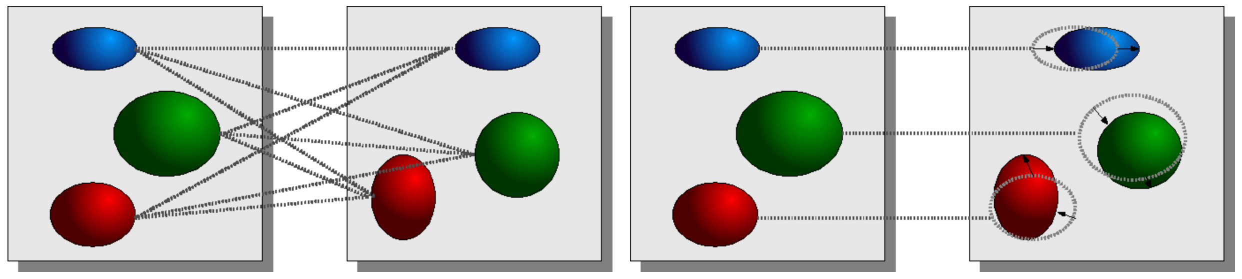

(a) The traditional approach: Extract features from each time step and then correspond afterwards (b) Our approach: Predict feature's region based on previous time steps then reshape it to the correct region

Figure 1: *Feature tracking approaches.* Unlike traditional approaches, our approach utilizes coherency between time steps in the extraction process.

region boundaries [22]. These techniques can all be directly applied to spatio-temporal data [11], but doing so would not fully use special properties of the time axis.

Feature tracking for video analysis and computer vision tasks operate in a 2D image space, so they are often based on image processing techniques such as region growing. In video analysis, the whole time range is often available, so some approaches incorporate the time axis and operate on a 3D spatio-temporal domain. The work of Botchen et al. [4], for instance, presents features in video data to the user in a 3D volume, and the work of Wang et al. [24], uses 3D segmentation in order to apply non-photorealistic techniques to video data. Many computer vision techniques extract regions from each time step independently then correspond them, just as is often done for flow visualization [19]. However, there are some that take a more efficient approach, through prediction and reuse of feature information across time steps. For instance, Galvin et al. [5] use an energy function minimization to move corner features from one time step to the next, and hence the cost is proportional to the distance the features have moved. Our approach is most closely related to Hundelshausen's and Rojas' work [23], which uses the previous time step's regions as a starting point, then it grows and shrinks these regions to extract the actual regions in the current time step. However, they focus on 2D time varying image data, while we extend the concept to 3D time varying volumetric data.

Most three-dimensional, time-varying volume tracking systems extract features for each time step separately and then associate them through time. This association process is the primary focus of most feature tracking algorithms, particularly in flow visualization [11]. Many of the original algorithms correspond features based on whether or not their regions overlap in adjacent time steps [14]. Others are based on attribute correspondence, which correspond features based on attributes of the region, such as position, size, shape, orientation, or topology [12, 13, 21, 20, 6]. Improvements to the robustness of these approaches have also been researched. For instance, in several papers, [16, 18, 17], Silver et al. require regions to have a threshold of overlap or similar attributes, such as region volume, before they get associated. Another approach done by Ji and Shen deals with the problem of robustness, by using a global optimization correspondence algorithm [8]. Others aim to improve speed, such as a work done by Wu et al. [27], which uses bitmap indexing to more rapidly and efficiently extract and subsequently correspond regions. Several tracking techniques only consider the surface of the feature [1, 9, 3, 10]. These are advantageous because surfaces can be calculated quite rapidly, but they do not apply to general volume visualization since they only consider surfaces. But all of these systems calculate the regions in each time step independently before doing any correspondence.

1.2 Our Approach

Figure 1 shows the fundamental difference between traditional approaches and our predictive region coherency based approach. The traditional approaches [10, 16, 11], depicted in Figure 1(a), perform feature extraction on each time step separately and subsequently correspond them by comparing them across timesteps. As shown in Figure 1(b), our approach does not follow this paradigm. Rather, it works by predicting a feature's region in subsequent time steps then extracting the actual region starting with this predicted region. By doing this it effectively reuses the information from previous time steps to reduce the amount of work needed in each time step. Also, it implicitly solves the correspondence problem since it utilizes correspondence information to extract regions. Finally, it can be used to extract individual features without the need to do a global segmentation where all features are extracted at once. This makes it ideal for an interactive system, where the user can select which feature or set of features to extract and dynamically refine the parameters of the segmentation algorithm.

2 A New Feature Tracking Technique

As mentioned, the technique used in this paper is a combination of two methods: predicting a feature's region, then adjusting the surface of this region to extract the correct feature. Prediction is the process of guessing a feature's region in space based on its location in previous time steps. Once this prediction is made, then the actual region is extracted by adjusting the surface of the predicted region through the use of a technique based on region growing.

2.1 Prediction

When dealing with time-varying data, features often follow fairly predictable paths. So, when tracking a feature, it is often possible to predict where a feature will move to. By taking this predictions into account, the feature tracking can be made more robust. In the datasets we use, the time steps were spaced out evenly, so we assume a constant amount of time between each time step. However, the prediction methods we use can be extended to non-evenly spaced time steps. While more complex prediction functions than the ones presented here are surely possible, they are beyond the scope of this paper. We implemented three different prediction methods: direct prediction, linear prediction, and quadratic prediction. Figure 2 shows a diagram of all three of these predictive techniques.

The simplest prediction method is direct prediction, where the predicted region is a direct projection of the same region in the previous time step. This is the most naive prediction, but also the simplest and fastest to calculate - it just starts with a copy of the

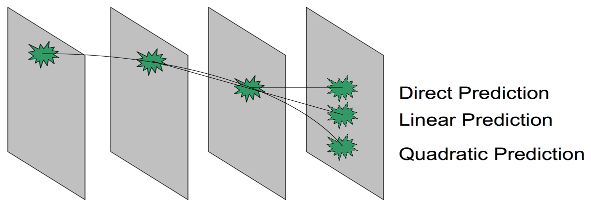


Figure 2: The three predictive methods. Direct prediction depends only on the previous time step, linear on the previous two, and quadratic on the previous three.

previous time step. That is, the position $\vec{v}_t$ of each voxel is simply:

$$\vec{v}_t = \vec{v}_{t-1}$$

where $\vec{v}_{t-1}$ is the position of that voxel in time step $t-1$

The next method used is linear prediction, where the predicted region is also a projection of the previous time step, but is offset according to the line through the centers of the region in the previous two time steps. This is done to better track moving objects. In order to perform this calculation, the centers of the previous two regions are calculated during their extraction. The centers are calculated by taking an unweighted average of the positions of the voxels of the region. This is very straightforward to do and adds minimal overhead. Then we create a new predicted surface where the center is on the line between the centers in the previous two time steps, and each surface voxel's position relative to the new center is the same as in the previous time step. That is, each voxel's new position $\vec{v}_t$ must satisfy:

$$(\vec{c}_t - \vec{c}_{t-1}) = (\vec{c}_{t-1} - \vec{c}_{t-2}) \text{ and } (\vec{v}_t - \vec{c}_t) = (\vec{v}_{t-1} - \vec{c}_{t-1})$$

Where $\vec{c}_{t-k}$ is the center of the region in time step $t-k$ and $\vec{v}_{t-1}$ is the position of that voxel in time step $t-1$. Then, solving for $\vec{v}_t$, the new position is:

$$\vec{v}_t = \vec{v}_{t-1} + (\vec{c}_{t-1} - \vec{c}_{t-2})$$

The final predictive method used here is a quadratic prediction, where the predicted region is a projection of the previous time step, offset according to a parabola that goes through the centers of the region in the previous three time steps. This way, the predictions can even follow accelerating objects. The quadratic prediction can produce even better results with almost no additional overhead over the linear prediction, since it depends only on the same set of region centers calculated for the linear prediction. The only difference is that one more center must be saved ($\vec{c}_{t-3}$) and the new center and the new voxels' positions must satisfy

$$\vec{c}_t + 3 * \vec{c}_{t-1} - 3 * \vec{c}_{t-2} + \vec{c}_{t-3} = 0 \text{ and } (\vec{v}_t - \vec{c}_t) = (\vec{v}_{t-1} - \vec{c}_{t-1})$$

solving for $\vec{v}_i$ now yields

$$\vec{v}_t = \vec{v}_{t-1} + (2 * \vec{c}_{t-1} - 3 * \vec{c}_{t-2} + \vec{c}_{t-3})$$

Where t is the current time step and $\vec{c}_{t-k}$ denotes the center of the region for each time $t-k$.

These prediction methods depend on previous time steps, and so the first couple time steps are special cases. In the first time step, no prediction can be made, so a standard region growing algorithm must be used to create the initial region. In the second time step, there is not enough information to define the linear or a parabolic predictions, so these simply fall back to direct prediction. Finally, in the third time step, a parabola can not be defined, but the line can, so the quadratic prediction must fall back to linear prediction.

Pseudo-Code 1 Pseudo-Code for the coherent region extraction algorithm

```
Predict current feature's region
Initialize queue with surface of predicted region
while (queue not empty)
    pop a voxel v off of the queue
    if (v's value above threshold value)
        mark v as being in the region
        if(v on the boundary of the surface)
          save v as part of new surface
        for each neighbor n of v
          if (n not marked as inside region)
            if(n not in queue)
              enqueue n
    else if (v's value below threshold value)
        mark v as being not in the region
        if(v is on the boundary of the surface)
          save v as part of new surface
        for each neighbor n of v
          if (n in the current region)
            if(n not in queue)
              enqueue n
```

The amount of work required to go from the predicted region to the correct region is proportional to the quality of the initial prediction for each time step. If the prediction is pathologically poor, it could actually take up to twice as much work as a direct spatio-temporal region growing algorithm (once to shrink, and once to grow). However, in real world data, where the region moves smoothly, such a pathological case is unlikely. Most often, regions in real data move fairly smoothly and predictably over time.

2.2 Coherent Region Extraction

When predicting a region for a time step, the prediction will almost never line up exactly with the correct region. Thus, it is necessary to correct the region by adjusting its boundary as necessary. As depicted in Figure 3, this adjustment is actually a combination of two processes: boundary growing and boundary shrinking. Both boundary growing and shrinking are based on region growing [7].

The region growing algorithm works by doing a breadth-first search on voxels starting from a seed voxel or set of voxels [7]. Like any breadth-first search, this is done with a queue of voxels to visit. This queue is primed with the seed voxel(s), then each voxel in the queue is tested to see if it belongs to the region. If it is, it is marked as being part of the region. Then, any of its neighbors that have not yet been visited are enqueued. This process is repeated until the queue is empty, at which point every voxel in the region has been visited. The region growing algorithm is very flexible, since the test to see whether a voxel should be in the region or not can be based on several different properties, such as a threshold value or gradient magnitude. In our examples, we are considering regions of high vorticity, so a threshold value is used, but the technique would work with other region growing variants as well.

The boundary growing process works just like traditional region growing, starting with the border of the object and expanding out. This is done with a breadth-first search, just as in region growing. As is common in region growing, we use a threshold value to define the feature, but another boundary property could be used as easily. The primary difference is that the initial queue for the boundary growing process is the surface of the predicted region. The boundary shrinking process is done in a similar manner, but in reverse. That is, it essentially performs a region growing on the background space, growing into the predicted region. Just like in boundary growing, it also starts by simply initializing the region growing queue with the current region's surface. Pseudo-code for

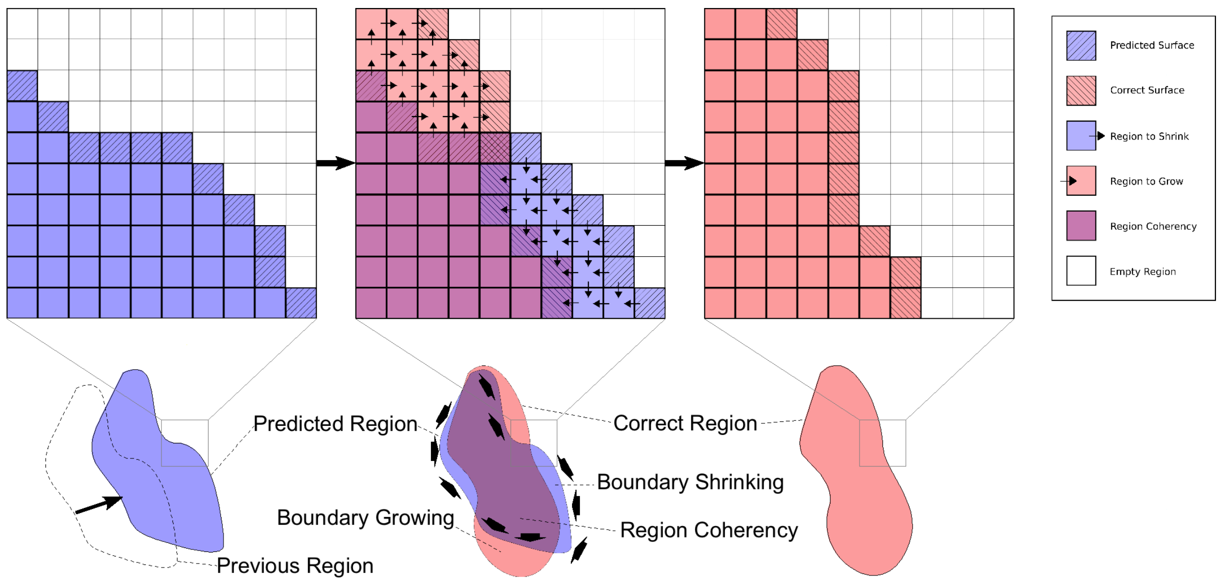


Figure 3: *Coherent extraction.* First, the feature in the previous time steps is used to predict the feature's region in the current timestep. However, the predicted region of the feature is not the correct region. In order to actually extract the correct region, some parts of it must be grown into (indicated by red voxels), and some parts of the predicted region must be shrunk away (indicated by blue voxels). By proceeding in this manner the coherent part of the feature (purple voxels) is captured in the overlap between the prediction and the actual region, and does not need to be reevaluated.

these processes is given in Pseudo-Code 1.

One advantage of this approach over traditional region growing is that the amount of work necessary is proportional only to the difference between the predicted region and the actual region. When the prediction is good, the amount of work required is much less than the work required to do a full region growing. If the prediction is exact, then the total amount of work required is directly proportional to the surface area of the region, as opposed to its volume which is guaranteed to be larger. Also, as the sampling rate on the time axis increases, the amount of change between time steps decreases, and so the prediction functions are automatically more accurate. So the amount of work per time step required by our approach decreases as well, whereas the amount of time required per time step by traditional methods is constant.

There are some limitations to this approach. For instance, a feature's region must be sufficiently close in sequential time steps to match up. That is, the sampling along the time axis has to be sufficiently fine or the prediction has to be good enough that the predicted regions overlap with the correct regions and no other regions. This is a fairly common assumption among feature tracking works [14], since otherwise the feature would not be trackable. Our approach is even more flexible than many classic approaches because even in the case where there is no overlap, it is possible for a different prediction function to create an overlap. Unlike many existing approaches, such as topology based correspondence methods [21, 20, 6], our technique works with just the surface of the regions, and it assumes that the feature is solid and that hollow regions do not spontaneously appear in the center of the region. However, this is also a fairly reasonable assumption for most physically-based phenomena. In the case that such hollow regions do appear and are important, they can be cleared by a simple scan-line algorithm. This approach also does not aid in the initial detection of new features. That is, it does not automatically detect birth events. In our implementation, this can be performed by the user in an interactive manner or through an automated initial feature extraction step.

2.3 Feature Event Handling

As features evolve over time, they can undergo several events, starting with birth, followed by splitting and/or merging, and finally death. Our work has mainly focused on how to efficiently track individual features based on user selection. For a more comprehensive view of the fluid flow, however, these events are often of interest [11, 13]. We have thus considered how to couple our tracking algorithm with basic event handling capabilities.

Some events are handled inherently by our approach, while others take extra processing. Birth events, in particular, would not be detected at all without extra computation. So we added the capability to iterate through all the unsegmented voxels of any given timestep to detect any new features. Obviously, this adds a fairly substantial complexity to the tracking process. However, applying it to each timestep after all preexisting features are already tracked can fully automate detection of birth events. When a feature undergoes a splitting event, it is potentially ambiguous which of the subregions should be the current feature, and which one(s) should be new features. So we currently treat the divided regions as one discontinuous feature. This is simple because it requires no extra computation, as would be induced by more complicated options. While this choice can induce problems if the regions drift far apart, this is often not the case. It would be possible to detect such a discontinuity by performing a breadth-first search on the surface of the feature, and subsequently dividing the feature accordingly, but this adds complexity. Merge events between two or more features are currently resolved implicitly by our approach. When a set of features converge in a given timestep, the first one to get evaluated will simply expand into the entire region, and the rest will be dropped and noted as a merge due to their collision with the first feature. While this is very easy to implement, it can lead to problems since it throws off the prediction methods. It also does not account for which of the merging regions is larger. It could be improved, for example, by first taking the union of the predicted regions and extracting it as being the largest of the input regions. Finally, death events are the most simple, since the region simply shrinks to nothing, at which point the surface queue is empty and the tracking of

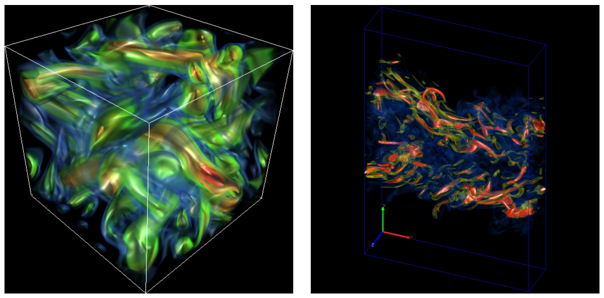
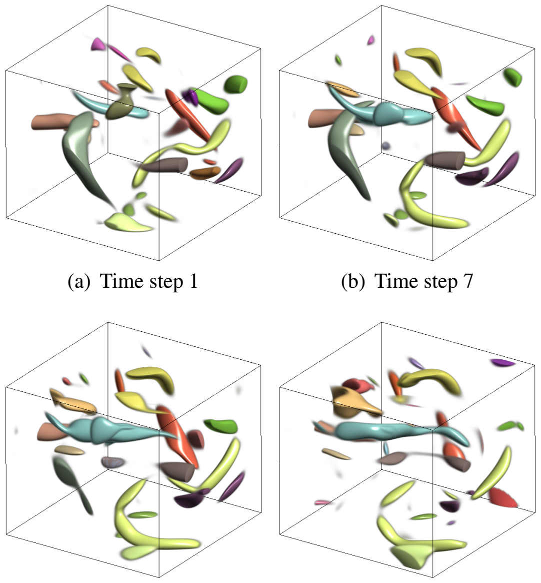

(a) Turbulent Vortex Data Set (b) Combustion Vorticity Data Set

Figure 4: The two data sets used in this study. In both data sets, the features of interest are the regions of high vorticity, which correspond to vortices. As can be seen, direct volume renderings of this data can be busy and confusing, which makes them ideal for feature extraction approaches.


(a) Time step 1 (b) Time step 7

(c) Time step 12 (d) Time step 18

Figure 5: *The turbulent vortex dataset*: All vortices in the dataset extracted and tracked through time with event handling. Several time steps are shown and the features are highlighted with different colors. The extracted surfaces take an average of 31,698 voxels per time step, which is 1.5% the size of the original data.

that feature ends.

Explicit event handling increases the complexity of the process, making it less attractive for interactive systems. It is possible to reduce the amount of work necessary in a few ways. For instance, detecting birth events does not necessarily need to be done every time step - extracting new events could be done sparingly and then tracked backwards in time as well as forwards to find their actual birth time. Even though event handling increases complexity, it is necessary to enable a fully automated system, such as simulation time feature tracking.

3 Results

To show the effectiveness and efficiency of this approach, it was implemented and tested on several datasets. We show here the turbulent vortex data set (used by Silver, Shen, and others [8, 16, 18, 17, 26]),which is 128x128x128, and a turbulent combustion data set, which is 480x720x120. Volume renderings of each are shown in Figure 4. Both datasets are time-varying scalar volumes of the vorticity of fluid flow simulations, and the features of interest we focus on are regions of high vorticity. All tests were run on one core of a Mac Pro with an 2.66Ghz Intel Xeon processor and 8GB RAM.

3.1 Effectiveness

In Figure 5, all the vortices have been extracted from the turbulent vortex data set and tracked through time. Each vortex was extracted and tracked individually. As can be seen, these features were correlated appropriately through time, just as in traditional methods. Each of the previously mentioned feature events (births, deaths, splits, and merges) can also be seen. Since vortices in this data set do not move greatly in each time step, the final results are the same regardless of which prediction method was used. Thus, the robustness of the algorithm was not greatly tested in this data, but it is still useful for comparing against other works that use this particular data set. It also demonstrates the space savings of feature extraction, as the extracted surfaces are on average only 1.5% of the total number of voxels.

We can see in Figure 6 that the combustion data set is much more complex than the turbulent vortex data set. The primary challenge to this data set are that features are very close to each other, due to the complexity of the data. Furthermore, there is a general trend of linear motion along the x axis, which makes it difficult to track features. The features often move enough that direct correspondence and overlap-based methods not only fail to correspond with the correct feature, but they also often correlate the wrong features together. By using linear prediction, this constant velocity motion is countered, resulting in a more accurate set of tracked features. The quadratic prediction algorithm would also work just as well, but since features in this data set were not accelerating much, the improvement would be minimal. In Figure 6, a single vortex has been extracted and tracked for analysis. Due to the size of the data set, and the large number of features, performing this task with a standard feature correspondence approach would require a much larger amount of calculation. However, our technique extracts this feature very quickly. If the user is only interested in the evolution of a single feature or a small set of features over time, then it is unnescessary to extract and track the rest of the features in the data, and our approach is good for such cases.

Our technique is based on region growing, which is a segmentation technique. Thus, the set of extracted features form a segmentation for any given time step. This enables the use of many techniques that depend on segmentation. Figure 7 shows one application of this segmentation information to direct volume rendering. Figure 7(a) is rendering of just the extracted features, but there is no context of the rest of the data. Figure 7(b) adds this context by using ray traced volume rendering to render the background data semi-transparently. But the extracted features are still kept colored and opaque, so that they can still be followed.

3.2 Timing tests

To show the efficiency of these approaches, several tests were run to measure the time to extract and track individual features from the data. Since traditional correspondence approaches do not handle the task of tracking individual features well, they would not be directly comparable to our approach. Instead, we compare our approach to a 4D region growing approach [22], which is the most similar existing technique.

One timing test was run on a set of four features over the first 32 time steps of the turbulent vortex data set, and the results are in

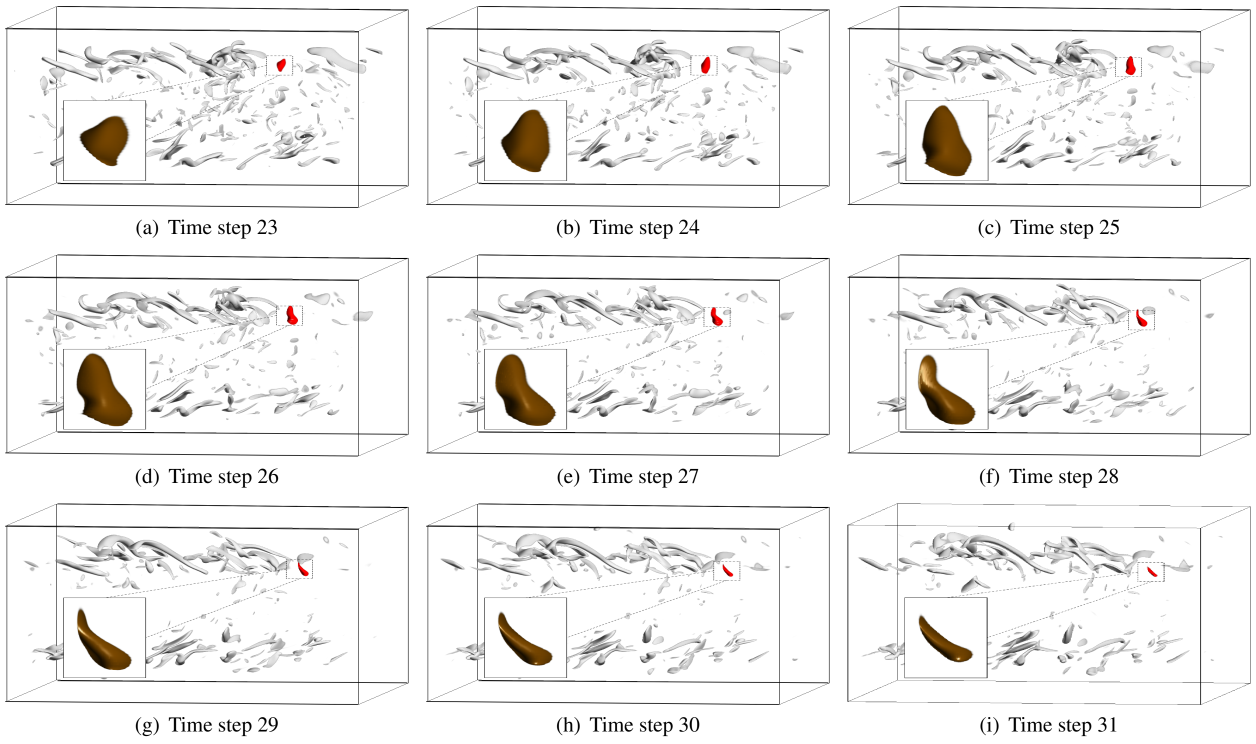

(a) Time step 23 (b) Time step 24 (c) Time step 25
(d) Time step 26 (e) Time step 27 (f) Time step 28
(g) Time step 29 (h) Time step 30 (i) Time step 31

Figure 6: *The combustion vorticity dataset*: One vortex, highlighted in red, has been interactively extracted and tracked through the data. Traditional correspondence approaches would have to extract and correlate all features, even if user is only interested in one feature.

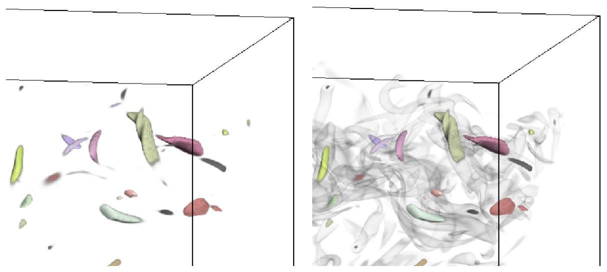
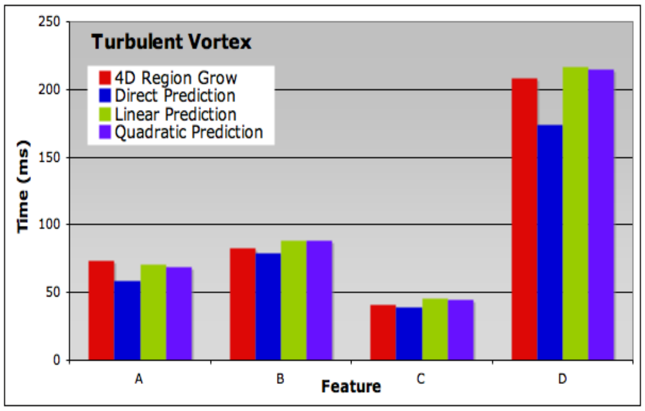

(a) Without context. (b) With context.

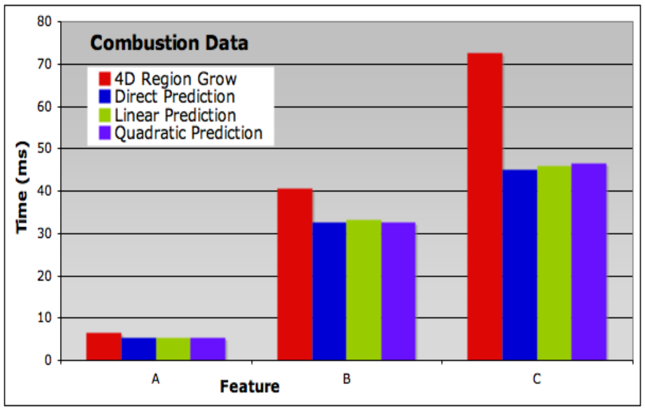
Figure 7: *The combustion vorticity dataset, time step 64*: Since the our technique is based on segmentation, volume rendering can use this segmentation information to clearly present the features while maintaining context.

Figure 8(a). Of interest here is that all of the techniques took about the same amount of time. That is, there was no major advantage or disadvantage of the predictive measures compared to the 4D region growing in terms of performance. The direct prediction faired slightly better than the others, but not by much. The lack of a speed improvement in this particular example can be explained by the fact that the selected regions did not move dramatically, but they were shrinking. Thus, while the traditional method only had to grow into the smaller region in each consecutive time step, the predictive methods had to do as much or more work on average just to shrink the larger predicted region. This could probably be solved with a prediction function that takes into account differences in size.

Another timing test was done on three features in 12 time steps of the combustion vorticity data set. In this dataset, many features move substantially. In fact, many of them move enough to create incorrect overlap. Because of this, many features can not be accurately tracked by the direct prediction, but are trackable by linear or quadratic prediction, demonstrating their improved robustness. The worst performing was the 4D region growing algorithm, since it moves both backwards and forwards through time from every point in the region, which often lets it spread to all the features in the data set at once, unless the threshold value for the features is quite high. Due to this difficulty, few features were trackable by all four methods. Three such features of varying sizes were extracted. Their times are shown in Figure 8(b). In these features, all three prediction functions gave nearly equivalent performances, but they all outperformed 4D region growing. The equivalence of the performance of the 3 predictive methods indicates that the overhead of calculating the linear or quadratic prediction is equivalent to the reduced amount of calculation in region adjustment.



(a) Timing on the turbulent vortex data set. 4 vortices were extracted and tracked over the first 32 time steps.



(b) Timing on the combustion vortex data set. 3 vortices were extracted and tracked over 12 consecutive time steps.

Figure 8: *Timing tests.* Each algorithm was run on several different features in each data set.

4 Discussion

Since the algorithm presented here is similar to 4D region growing, it shares many of the same advantages and disadvantages as the 4D region growing method. One such advantage is that it connects regions in different time steps during the extraction step, so it implicitly solves the correspondence problem. The primary disadvantage of these approaches is that regions which should not be associated in different time steps can end up overlapping. This leads to potentially conflicting and incorrect correspondences. However, this problem can be alleviated by better prediction algorithms; as long as each predicted region overlaps only the feature of interest and no other features, the results will be correct. This correspondence problem has also been addressed in previous works by requiring a threshold of overlap to reduce the error [11], and a similar technique could be applied here.

4.1 Complexity and Scalability

With regard to complexity, the theoretical cost per time step is equal to the difference in volume between the predicted regions and the actual regions plus the region's surface area. That is, the cost is

$$O((V_{predicted} + V_{actual} - V_{predicted \cap actual}) + A)$$

The amount of work required is dependent on how much a region changes per time step and on the quality of the prediction function. As the sampling rate in the temporal domain increases, the region will change less, so our technique will have to do less work per time step. Thus, in well behaved or highly sampled data, the difference between the predicted and actual volumes will nearly vanish, so the total work can be expected to be generally on the order of the size of the surface of the region, which is always smaller than the volume of the region. This property would make this approach ideal for simulation time feature extraction, because the temporal resolution during simulation is often much higher than in the output data. In the worst case, where regions in adjacent time steps are just barely overlapping, our approach would have to do twice as much work as direct region growing in 4D space. However, this occurs where direct region growing or overlap methods would be about to fail, since the regions are barely touching each other. The efficiency can also be improved with a more accurate prediction function, because the more accurate the prediction, the smaller the volume of the difference will be, and hence the less extraction work that is needed.

4.2 Extracting Individual Features

Another advantage of an approach like ours is that an individual feature can be extracted and tracked interactively, without the need to perform a global extraction of all features. Most approaches [8, 14] extract every feature from every time step and then correlate them together. If the user is only interested in the evolution of a single feature or a small set of features over time, then it is unnescessary to extract and track the rest of the features. Since our coherent region extraction method works by tracking individual features, it is innately useful for this task, and can perform quite efficiently. Also, as features are tracked independently, the user could interactively alter the parameters (such as threshold values) used to define the features. That is, it could be possible for the user to interactively adjust these parameters independently per feature and then propagate these adjustments through time.

4.3 Simulation Time Feature Extraction

During a simulation, scientists often only store 1 out of every 100-300 time steps according to the storage space they can afford. These temporal gaps compound the problem of feature tracking, since longer time between time steps means that features move farther. Therefore, it is beneficial to perform feature extraction and tracking during the simulation itself, so that the tracking algorithm can utilize the time steps that would otherwise be skipped. In order to do so, the feature tracking algorithm used must be efficient and fast, so that it does not greatly hinder the simulation itself. The feature tracking algorithm presented here would be ideal for such applications, since it actually becomes more efficient and more reliable at higher temporal resolution. Also, it should be computationally easy to parallelize this technique, as all operations are spatially coherent. Finally, while our feature tracking and extraction approach is currently aimed towards interactive feature extraction, automating processes such as birth event detection can be refined for simulation time processing situations by only checking for birth events when a time step is to be output for instance. This way, extra birth event detection would be minimized.

4.4 Other Future Work Directions

While we have shown the effectiveness of our approach, there are still improvements and extensions that can be made. The prediction functions described here have been found to be effective, but it should be possible to implement better prediction functions. In particular, the prediction functions we used consider only the motion of the objects with respect to their centers, not other attributes such as angular motion, change in volume, or discontinuities upon regions that merge or split in subsequent time steps. A prediction function that utilizes properties of the actual data in the volume would be particularly effective. For example, in flow data, a prediction could be derived from the actual flow vectors to create a much more accurate prediction function. Heuristic or artificial intelligence based prediction functions would likely produce good results as well. Another improvement of the prediction methods would be to incorporate more time steps and do a curve fit rather than a direct calculation. It would be useful for the prediction functions to detect when they are wrong, due to factors such as discontinuities or numerical instability. This could perhaps be done by averaging several predictive region centers while dropping outliers. Any of these would potentially slow down the prediction step, but would be very helpful in detecting and countering noise or errors. There are many possible ways the event handling could be improved as well, such as handling splitting more explicitly or adding a visualization of a timeline of events. It would also be useful to apply our technique to higher resolution data, particularly data with better temporal resolution, in order to show how the algorithm scales to larger data sets. It would be interesting to perform a more thorough study on performance versus robustness with respect to temporal resolution for our approach compared to various other feature tracking algorithms. Finally, as we propose that our method would be effective for simulation time feature extraction, as it could be parallelized and subsequently run from within an actual simulation.

5 Conclusion

When we are still struggling with data at terascale, petascale computing is around the corner. The development of adequate data analysis and visualization methods is absolutely needed to timely address the growing challenges created by data intensive supercomputing. Our prediction-correction feature tracking method is appealing because it is more efficient than the correspondence-based methods while being more reliable than spatiotemporal techniques. While the prediction functions we implemented were fairly limited in scope, we did find that they improved the robustness of the approach, even when they did not improve the performance. Even though feature extraction can significantly lower the amount of data that must be stored and looked at, it has not been widely used due to its heavy reliance on domain knowledge. Our method is easy to implement and low cost, and when deployed for a new fluid flow application it is possible to decouple the domain knowledge dependency from the tracking process. We anticipate the growing use and integration of feature extraction in scientific supercomputing.

Acknowledgements

This research was supported in part by the U.S. National Science Foundation through grants CNS- 0716691, OCI-0325934, OCI-0749217, CNS-0551727, CCF-0811422, CCF-0808896, OCI-0749227 and OCI-049321, and the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777 and DE-FG02-08ER54956.

References

[1] C. Bajaj, A. Shamir, and B.-S. Sohn, 2002. "Progressive Tracking of Isosurfaces in Time-Varying Scalar Fields," Technical Report, University of Texas at Austin, 2002.

[2] D. Bauer and R. Peikert. Vortex tracking in scale-space. In *In Data Visualization 2002. Proc. VisSym '02*, pages 233–240, 2002.

[3] P. Bhaniramka, R. Wenger, and R. Crawfis. Isosurfacing in higher dimensions. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 267–273, 2000.

[4] R. P. Botchen, M. Chen, D. Weiskopf, and T. Ertl. GPU-assisted Multi-field Video Volume Visualization. In *Proceedings of Volume Graphics '06*, pages 47–54, 2006.

[5] B. Galvin, B. McCane, and K. Novins. Robust feature tracking. In *Fifth International Conference on Digital Image Computing, Techniques, and Applications (DICTA), December 6-8, 1999, Perth, Australia.*, 1999.

[6] C. Garth, X. Tricoche, and G. Scheuermann. Tracking of vector field singularities in unstructured 3d time-dependent datasets. In *VIS '04: Proceedings of the conference on Visualization '04*.

[7] R. Huang and K.-L. Ma. Rgvis: Region growing based techniques for volume visualization. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 355, Washington, DC, USA, 2003. IEEE Computer Society.

[8] G. Ji and H.-W. Shen, 2006. "Feature Tracking Using Earth Mover's Distance and Global Optimization," Pacific Graphics 2006.

[9] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 28, Washington, DC, USA, 2003. IEEE Computer Society.

[10] R. J. Moorehead and Z. Zhu. Feature extraction for oceanographic data using a 3d edge operator. In *IEEE Visualization*, pages 402–405, 1993.

[11] F. H. Post, B. Vrolijka, H. Hauser, R. S. Laramee, and H. Doleisch, 2003. "The State of the Art in Flow Visualisation: Feature Extraction and Tracking," Eurographics Volume 22 (2003), Number 4 pp. 117.

[12] F. Reinders, F. H. Post, and H. J. Spoelder. Attribute based feature tracking. In *In Data Visualization '99. Proc. VisSym '99*, pages 63–72, 1999.

[13] F. Reinders, F. H. Post, and H. J. W. Spoelder. Visualization of time-dependent data using feature tracking and event detection. In *The Visual Computer*, pages 17(1):55–71, Feb. 2001.

[14] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *Computer*, 27(7):20–27, 1994.

[15] A. Sherbondy, M. Houston, and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 23, Washington, DC, USA, 2003. IEEE Computer Society.

[16] D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. In *IEEE TVCG, 3(2)*, June 1997.

[17] D. Silver and X. Wang. Tracking scalar features in unstructured datasets. In *VIS '98: Proceedings of the conference on Visualization '98*, 1998.

[18] D. Silver and X. Wang. Visualizing evolving scalar phenomena. In *Future Generation Computer Systems, 15(1):99-108*, Feb. 1999.

[19] 1995. S.M. Smith and J.M. Brady. Asset-2: Real-time motion segmentation and shape tracking. Transactions of the IEEE on Pattern Matching and Machine Intelligence, vol. 17, number 8, pp 814-820, 1995.

[20] H. Theisel and H.-P. Seidel. Feature flow fields. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, 2003.

[21] X. Tricoche, T. Wischgoll, G. Scheuermann, and H. Hagen. Topology tracking for the visualization of time-dependent two-dimensional flows. *Computers & Graphics*, 26(2):249–257, Apr. 2002.

[22] F.-Y. Tzeng and K.-L. Ma. Intelligent feature extraction and tracking for large-scale 4d flow simulations. In *Proceedings of Supercomputing 2005 Conference.*, 2005.

[23] F. von Hundelshausen and R. Rojas. Tracking regions and edges by shrinking and growing. In *In Proceedings of the RoboCup 2003 International Symposium, Padova, Italy, 2003.*, 2003.

[24] J. Wang, Y. Xu, H. Shum, and M. Cohen, 2004. Video Tooning. In Proceedings of ACM SIGGRAPH (August 2004), pp. 574–583.

[25] C. Weigle and D. C. Banks. Extracting iso-valued features in 4-dimensional scalar fields. In *In Proc. Symposium on Volume Visualization*, pages 103–110, 1998.

[26] J. Woodring, C. Wang, and H.-W. Shen. High dimensional direct rendering of time-varying volumetric data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 55, Washington, DC, USA, 2003. IEEE Computer Society.

[27] K. Wu, W. Koegler, J. Chen, and A. Shoshani. Using bitmap index for interactive exploration of large datasets, 2003.