Illustrating Dynamics of Time-Varying Volume Datasets in Static Images

Jennis Meyer-Spradow, Timo Ropinski, Jan Vahrenhold, Klaus Hinrichs

Institut für Informatik, Westfälische Wilhelms-Universität Münster
Einsteinstraße 62, 48149 Münster, Germany
Email: {spradow,ropinski,jan,khh}@math.uni-muenster.de

Abstract

We present visualization techniques for time-varying volume datasets, i.e., time series of 3D data volumes. Instead of sequentially displaying the consecutive 3D volumes contained in such a time series, our method allows to depict the dynamics in a single static image. We extract the motion dynamics inherent to a time-varying volume dataset automatically during a preprocessing step by using three-dimensional block matching. The final rendering of a 3D volume associated with a specific point in time incorporates the extracted information to depict dynamics in a single static image by non-photorealistic overlays in the form of glyphs and other graphical elements. In addition the preceding and following volumes of the time series may be rendered and combined with the final image.

1 Introduction

Volume rendering has become a mature field of interactive 3D computer graphics. It supports professionals from different domains when exploring volume datasets, representing for example medical or meteorologic structures and processes. Current medical scanners produce datasets having a high spatial resolution. Especially Computer Tomography (CT) datasets are highly suitable for showing anatomical structures. However, when dynamic processes have to be explored a single 3D volume dataset is often insufficient. For example Positron Emission Tomography (PET) imaging techniques are used for exploring the dynamics of metabolism. Successive scans produce a time series of images each showing the distribution of molecules at a certain point in time. Time-varying volume data is also produced when applying functional Magnetic Resonance (MR) imaging techniques to evaluate neurological reaction to certain stimuli.

In the subsequent sections we will use the term "time-varying (volume) dataset" or just "dataset" for a time series of 3D data volumes and "3D volume" for a single data volume at a specific point in time.

Although 3D volumes can be displayed separately to view the data for specific points in time, the dynamics contained in a time-varying dataset can only be extracted when considering all of them. Therefore time-varying volume datasets are often visualized by displaying the 3D volumes sequentially in the order they have been acquired. When rendering these 3D volumes at a high frame rate, the viewer is able to construct a mental image of the dynamics. Current graphics hardware supports rendering of time-varying datasets having moderate dimensions with appropriately high frame rates. However, often this rendering technique is unappropriate, since it is preferable in some domains to view still images rather than dynamic image sequences. The following two problems arise when dealing with image sequences of 3D volumes. When several people watch an image sequence, to exchange findings with the group a viewer usually points at a specific feature within an image. Since the images are dynamic, it is hard to pinpoint such an item of interest, especially if it is moving, and the animation needs to be paused. Another problem is the exchange of visualization results. In medical departments diagnostic findings are often exchanged on *static media* such as film or paper. However, it is not possible to exchange time-varying volume datasets in this manner. Although the sequential viewing process could be simulated by showing all 3D volumes next to each other, this could lead to registration problems, i.e., it would be more difficult to identify a reference item across the 3D volumes.

This paper proposes visualization techniques which allow to represent motion dynamics extracted from a time series of 3D volumes within a single static image. To this end we extract motion information between successive 3D volumes in a pre-processing step. Our visualization techniques depict motion by composing the successive 3D volumes and additional visualization elements. These visualization elements are used to augment the rendering of data extracted from the time-varying volume dataset at a specific point in time. For achieving effects adapted from cartoons in order display motion we combine polygonal elements with the volume rendering of the source datasets to obtain the final image.

In the next section we discuss related work. Section 3 briefly introduces the used motion estimation technique. Section 4 describes the visualization techniques used to depict the extracted motion information especially the usage of transparency, *speedlines* and motion glyphs. After discussing the application of our techniques to datasets from different domains in Section 5, the paper concludes in Section 6.

2 Related Work

Mostly, non-photorealistic rendering techniques are used for depicting motion in a single image. Masuch et al. [13] present an approach to visualize motion extracted from polygonal data by speedlines, repeatedly drawn contours and arrows; different line styles and other stylization give the viewer the impression of a hand-drawing.

Nienhaus and Döllner [15] present a technique to extract motion information from a behavior graph, which represents events and animation processes. Their system automatically creates cartoon-like graphical representations.

Joshi and Rheingans [9] use speedlines and flow ribbons to visualize motion in volume datasets. However, their approach handles only feature extraction data and is mainly demonstrated on experimental datasets.

A lot of research in the area of dynamic volume visualization has been dedicated to flow visualization. Motion arrows are widely used [10], but are often difficult to understand because information is lost when performing a 2D projection to the screen. Boring and Pang [4] tried to tackle this problem by highlighting arrows, which point in direction specified by the user. Texture based approaches compute dense representations of flow and visualize it [14]. First introduced for 2D flow, it has also been used for 3D flow visualization, for example with volume line integral convolution used by Interrante and Grosch [7]. Svakhine et al. [16] emulate traditional flow illustration techniques and interactive simulation of Schlieren photography.

Hanson and Cross [6] present an approach to visualize surfaces and volumes embedded in four-dimensional space. Therefore they use 4D illuminated surface rendering with 4D shading and occlusion coding. Woodring et al. [18] interpret time-varying datasets as four-dimensional data fields and provide an intuitive user interface to specify 4D hyperplanes, which are rendered with different techniques. Ji et al. [8] extract time-varying isosurfaces and interval volumes considering 4D data directly.

Our work is based partially on a block matching algorithm to extract motion information from time-varying volume datasets presented by de Leeuw and van Liere [11]. They rendered the estimated motion using polygonal models which are combined with volume rendering results. In contrast to their approach we detect and visualize object motion instead of flow introduced by single voxels.

3 Extracting Dynamics from Time-Varying Volume Datasets

To visualize dynamics expressively, motion information needs to be retrieved from the underlying data. Usually motion is not accessible directly and has to be extracted by applying appropriate algorithms. For our visualization we use an algorithm introduced by de Leeuw and van Liere [11]. Here we will give only a brief overview of the underlying concepts, while a complete explanation can be found in the original paper.

The goal is to extract dynamic motion from time-varying volume datasets. For each voxel in one 3D volume the algorithm searches for a corresponding voxel with a similar intensity in the successive 3D volume and interprets the spatial difference as motion. However, considering only one voxel for the comparison would be insufficient. Hence, all voxels in a block around the two candidates are used, which works similar to a block matching algorithm as used in 2D video compression [1].

Since comparing a voxel block with each voxel block in the next 3D volume is not very efficient, only those blocks of the successive 3D volume are tested, which lie in the neighborhood region of the current block. This test region is called the search window and its size is dataset dependent.

Comparing the blocks is done by using a matching operator $\ominus$. The result of $\ominus$ is greater when comparing less similar blocks and smaller for more similar ones. We calculate the absolute differences for each corresponding voxel and scale it with the result of a Gaussian weighting function, centered in the appropriate block matching region. This factor expresses that we are interested in the motion at a particular position—so nearby voxels are more important than those farther away.

We have adapted the 3D block matching algorithm to be better suitable for our visualization techniques, because a meaningful estimation was not possible for all blocks. Hence we discard blocks having only sparse content, i.e., having only a small number of voxels with an intensity not equal to zero. This elimination avoids that these sparse blocks might match to arbitrary destination blocks. For example a block containing only a single voxel lying on the border of an object may match to several other blocks, containing only one voxel with a similar intensity value.

Furthermore in cases in which more than one candidate with the same best matching value exist, we decided to choose the candidate with the minimal distance.

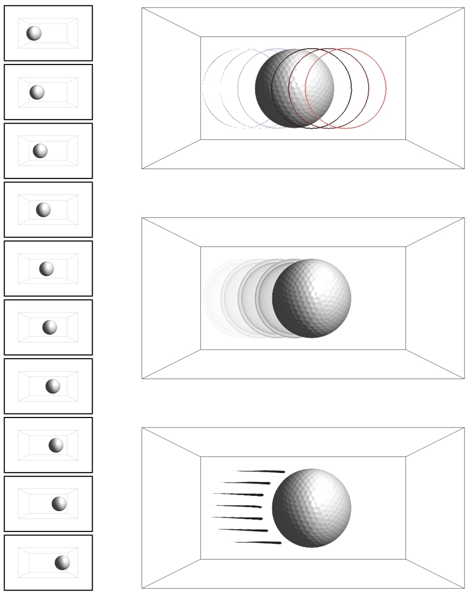

Figure 1: Three visualization techniques to depict the motion of the golfball volume dataset. The current 3D volume is rendered using isosurface shading. Edges of preceding and successive 3D volumes are shown (*top*), preceding 3D volumes are rendered semi-transparent (*center*), dynamic motion is depicted with speedlines (*bottom*). Renderings of ten time steps extracted from the dataset are shown (*left*).

4 Illustrating Dynamics

When visualizing time-varying volume datasets we do this by using the current 3D volume—which would be rendered when rendering the 3D volumes sequentially—as a frame of reference. This frame is visualized by applying direct volume rendering techniques and contributes the most to the image. The information obtained from preceding and successive 3D volumes is used in order to augment the image. In the following subsections we explain three different techniques for performing this augmentation. The first subsection explains how to merge the information directly contained in different 3D volumes, while the following two subsections explain how to use the meta information extracted from the dataset.

4.1 Depicting Multiple Datasets

To directly visualize the content of preceding or successive 3D volumes while rendering the current frame with direct volume rendering techniques we propose a combination of transparency and edge detection. In contrast to motion blur these techniques avoid blurring the information contained in the preceding resp. successive 3D volume. Both techniques can be parameterized based on the difference in time between the preceding/successive 3D volume and the point in time given by the current 3D volume.

The edge detection we apply is an image-based technique which allows us to emphasize the silhouette of objects contained in a 3D volume by applying appropriate filter kernels. To select which

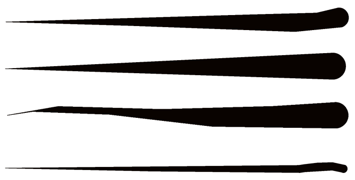

Figure 2: Hand-drawn speedlines are used. To enhance the desired cartoon-like impression they are disturbed during rendering.

edges should be emphasized we introduce an edge threshold. This threshold ensures that only edges exceeding a certain thickness end up in the final image. The color of the edges is parameterized depending on the point in time of the 3D volume to be processed. In Figure 1 (*top*) cold colors are used to depict the silhouettes of preceding 3D volumes, while warm colors are used to emphasize the silhouettes of object contained in successive 3D volumes. For the preceding 3D volumes the edge threshold is set depending on the difference in time, such that 3D volumes with a greater difference in time are depicted by using thinner edges.

Inspired by an advertisement illustration (see Figure 9) we have also applied transparency techniques in order to show the preceding 3D volumes. In these cases the transparency used to show the objects contained in the 3D volumes varies depending on their corresponding points in time. An image which has been generated with this technique is shown in Figure 1 (*center*).

Once the current 3D volume as well as the desired preceding and successive 3D volumes are rendered, the images need to be composed in order to obtain a single image as the final result. For this composition we can either blend the results or mask them by considering the background color.

4.2 Speedlines

Speedlines are a very common approach to depict past motion in cartoons. We have developed an algorithm to automatically generate speedlines for 3D volumes by using motion information extracted in a pre-processing step.

4.2.1 Direction and Shape Estimation

Our algorithm starts by first estimating the motions' preeminent direction $\vec{m}$. If all voxels move with approximately the same speed, we can simply use the motion vector with median slope among all vectors. For vector fields that exhibit a significant variance with respect to the absolute speed of the objects, we compute the *weighted* median of the slopes where each slope is weighted with the (normalized) length of the corresponding motion vector. Both variants of median-finding can be implemented by first sorting the slopes according to their polar angle followed by no more than a single scan over the sorted set of slopes [5, Ch. 9].

We then compute the convex hull of the motion vectors; the convex hull will be used as a conservative approximation of the vector field's shape. Based upon the estimated direction and shape, we compute the line ℓ that is orthogonal to $\vec{m}$ and tangent to the convex hull such that the extension of $\vec{m}$ to a ray does not intersect ℓ. Finally, we project the convex hull along $\vec{m}$ onto ℓ. For reasons that become evident later, the interval obtained by this projection is shrunk to 85% of its original length, resulting in an interval $\bar{\ell}$—see Figure 3.

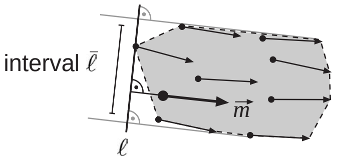

Figure 3: Projection of the convex hull along the median slope $\vec{m}$ (fat) and shrunken interval $\bar{\ell}$.

4.2.2 Speedline Positioning

The number k and actual position of the speedlines can be controlled by the user. The algorithm first subdivides $\bar{\ell}$ into k equal-sized subintervals and constructs for each of the $k+1$ interval end points a support line parallel to $\vec{m}$ (dotted lines in Figure 4).

Finally, a speedline is positioned on each of the $k + 1$ support lines such that it is at the distance ε selected by the user from the convex hull. The position is slightly blurred and a randomly selected, hand-drawn model is used as speedline to

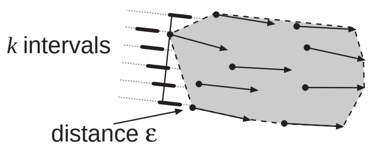

Figure 4: Influence of parameters k and ε.

enhance the visual impression (see Figure 2). An application of the speedlines technique is shown in Figure 1 (*bottom*).

If we had not shrunk the interval $\bar{\ell}$, an extreme, yet not unlikely, configuration where the slope of the top- or bottommost edge of the convex hull is close to $\overrightarrow{m}$ would result in a visually irritating effect: the top- or bottommost speedline would be positioned relatively far away from the other speedlines since their support lines would run almost parallel to the top- or bottommost edges of the convex hull—see, e.g., the gray lines in Figure 3.

4.2.3 Implementation Details

Our implementation of the convex hull algorithm is based upon Andrew's algorithm [2], a convex hull algorithm that is especially efficient if the given point set is presorted in some lexicographically consistent way. We obtain such an order by computing the projection of the points, i.e., the origins and the tips of the vectors, on the line orthogonal to $\overrightarrow{m}$. Furthermore, since the speedlines will be drawn on only one "side", say on the left side, of the convex hull, it is sufficient to actually compute the corresponding left hull of the motion vectors. Thus, we can prune all points to the right of the line through the minimal and maximal points with respect to the above projection and simply compute the left hull of the remaining points—see Figure 5.

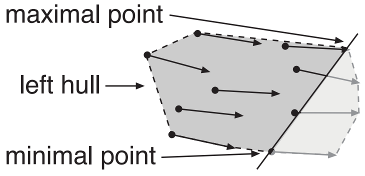

Figure 5: Computation of a partial convex hull.

To efficiently sort the vectors according to their slopes, we use the "orientation" predicate [3] which provides an elegant way of implicitly dealing with "infinite" slopes. However, since the "polar angle" order is a circular order, we have to define a zero direction such that for any two directions the circular order yields an unambiguous result. For the rendering of the first frame, we set this direction to be the direction of the negative x-axis, and for each consecutive frame, we assume a temporal coherence and set it to be the inverse direction of the median $\overrightarrow{m}$ used in the previous frame.

4.3 Motion Arrows

Since speedlines require to have a main direction of motion within the 3D volume, different visualization techniques are needed for cases where arbitrary directions are present. Motion arrows provide a good alternative for these cases. Furthermore they can be used to show multiple arbitrary motion directions overlaying the main direction of movement.

Arrows are appropriate because they can visualize the direction of motion and—encoded as arrow length—its velocity. However, to avoid visual clutter only a subset of the arrows should be visualized. Another important aspect is the three dimensional character of the arrows. This is especially hard to communicate when the arrow size falls below a certain minimum value, as it is often the case when combining arrows with volume renderings in order to avoid occlusions. Thus it has to be ensured that the depth information of arrows being almost collinear with the view vector is communicated.

4.3.1 Arrow Selection

First we address the problem of selecting a subset of the arrows for rendering. We use a two step approach. In the first step position dependent information, in the second step motion and orientation dependent information is considered.

To classify the arrows based on their position, the probably simplest idea is to use a three dimensional raster, selecting only each n-th arrow in each dimension of the 3D volume. However, this may result in several arrows occluding each other and thus confusing the observer. Nevertheless this approach is sometimes adequate to obtain an overview of the motion in the dataset—especially when the motion direction is almost uniform.

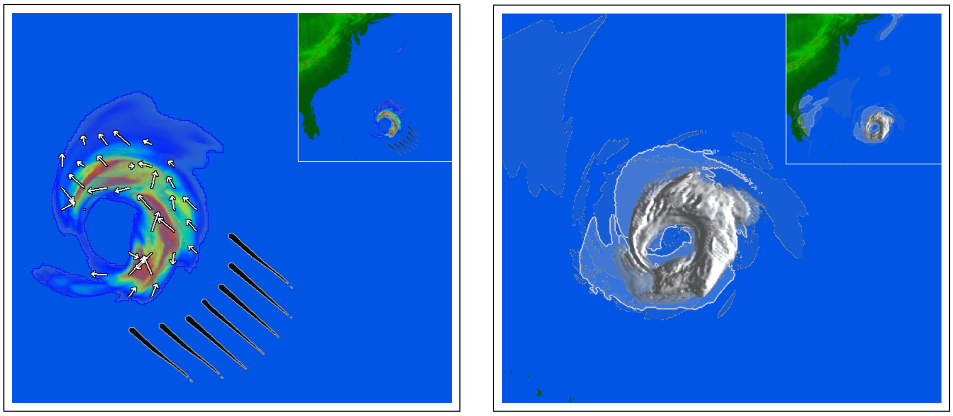

Figure 6: Visualization of the Hurricane Isabel. Using speedlines to show simultaneously the motion of the hurricane and the distribution of rain (*left*). Volume Rendering of the clouds modality by visualizing the silhouettes of the next two steps using edge detection (*right*).

An alternative approach is to select only the arrows, lying on an isosurface given by the visualized object(s). Note that this surface cannot be computed in a pre-processing step because it is highly dependent on various rendering parameters such as thresholding and the applied transfer function. Thus before ray-casting the 3D volume, we compute entry points at the corresponding isosurface of the volume. Since we are using GPU-based ray-casting we obtain these parameters in image space. Further to avoid cluttering we use a two dimensional raster to select only every n-th motion information. Since we are working in image space, we thus obtain appropriately placed arrows.

In the second step of the selection process we take into account motion velocity or the motion direction to discard certain arrows. Meaningful parameters are minimal and maximal velocity for the first case, and angle and variance for the second case. The user has to keep in mind, that there are probably ”unwanted” motion information. To give a hint we can render these arrows in another color.

4.3.2 Arrow Rendering

We first render the 3D volume and then overlay it with the arrows. As mentioned above rendering 3D arrows may incorporate some problems for the user when recognizing the arrows direction. This recognition problem can be reduced by rendering only a few, relative large arrows having an orientation that is easier to perceive. In cases where these larger arrows are not sufficient due to size restrictions, we decided to render 2D arrows aligned in image space.

5 Use Cases

5.1 Hurricane Isabel

Hurricane Isabel was the only Category 5 hurricane in the 2003 Atlantic hurricane season. A simulation of different variables of this hurricane—as rain, snow, or wind speed—has been done by the National Center for Atmospheric Research in the United States. In 2004 this simulation has been used in the IEEE Visualization Contest. The data for each variable consists of 48 time steps, each containing $500 \times 500 \times 100$ voxels with float precision. Almost all of the participating teams used a combination of volume and polygonal rendering. Only one team rendered voxels as semi-transparent quads and spheres in different colors. Another team tried to use vortex detection algorithms but they were not convinced by the results of this approach. However, their automatic detection of the central region of the hurricane succeeded. The winning team extracted data with complex attributes (“high velocity *and* clouds”) and rendered various properties of the hurricane.

For testing of our methods we applied an appro-

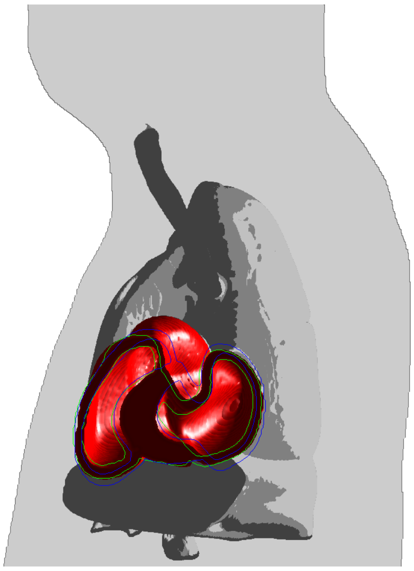

Figure 7: Cut through a beating human heart. The green and blue edge depict the next two timesteps (see color plate).

priate scaling to obtain a datasize of one byte per voxel, but we did not change the resolution.

Our first aim was to visualize the main motion direction of the hurricane with speedlines while simultaneously visualizing rain. We applied our motion estimation to the “rain dataset”. To filter out disturbances, we selected the motion arrows by length and then used our speedline algorithm. For visualizing the rain distribution we used direct volume rendering by applying an appropriate transfer function combined with edge enhancement. Additionally the motion within the hurricane was visualized with 2D arrows. To ease the spatial comprehension, a map of the region is displayed below the volume rendering (see Figure 6 (*left*)).

We have also visualized three time steps of the “cloud dataset”. The first time step is rendered using direct volume rendering, the second and third time step are rendered by displaying the silhouette and a semi-transparent filling using different shades of gray. Again we use the map layer to improve spatial comprehension. This method produces good results from a top view (see Figure 6 (*right*)).

Viewing the dataset from south northward allows interesting insights into the temporal cloud distribution. As can be seen in Figure 8, the current cloud will expand to the shape depicted by the red edges and then further expand as indicated by the blue edges.

The examination of the dataset can further be enhanced by using a clipping plane aligned to the viewing plane, in such a way that only one slice of the hurricane is visualized.

5.2 Heart Motion

In the area of medical visualization we have experimented with our technique in order to visualize heart motion. We have used a segmented time-varying volume dataset, that we have cut with a clipping plane to show motion of only the left ventricle. Again we used the edge detection technique to display one time step while we used silhouettes for the other time steps. In this case we also give some context information by rendering the organs behind the clipping plane using toon shading. As it can be seen in Figure 7, heart motion is clearly visible. To achieve an impression similar to visualizations used in medical illustrations we added a rendering of the body’s silhouette behind the volume rendering.

6 Conclusions and Future Work

In this paper we have presented visualization techniques for depicting dynamics in still renderings of time-varying volume datasets. To extract motion information from these datasets, we have applied a slightly modified version of the three dimensional block matching algorithm.

Three different techniques have been proposed to visualize motion: (1) Applying edge enhancement and transparency techniques when rendering multiple 3D volumes, (2) overlaying speedlines and (3) rendering of motion arrows. All these techniques are applied automatically and do not need any user input. In order to obtain interactive frame rates they have been implemented as extensions to GPU-based ray-casting. To demonstrate the usability of our visualization techniques we have described the application to real world datasets.

With the presented techniques it is possible to present a reasonable subset of the motion information contained in a time-varying dataset in a single image. Thus this information can be communicated more easily since it can also be visualized on static media.

Our approach has the drawback that using speedlines with volume data containing different objects moving in several directions will result in wrong speedline positioning. We will address this problem by extending the motion detection to support the use of clustering. Thus it will be possible to track only specific regions of interest within the dataset to obtain better results when dealing with more *chaotic* data or to show different speedlines with distinct directions for the specific objects.

7 Acknowledgments

This work was partly supported by grants from the Deutsche Forschungsgemeinschaft (DFG), SFB 656 MoBil Münster, Germany (project Z1). The golfball dataset is available at `voreen.uni-muenster.de`. It has been generated with the voxelization library *vxt* developed by Milos Sramek. The Hurricane Isabel dataset is courtesy of the National Center for Atmospheric Research in the United States. The medical dataset has been generated using the 4D NCAT phantom.

References

[1] A. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989

[2] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9(5):216–219, 1979

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* 2nd ed., Springer, 2000

[4] Ed Boring and Alex Pang. Directional Flow Visualization of Vector Fields. In *IEEE Visualization*, 1996, 389–392

[5] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* 2nd ed., MIT Press, 2001

[6] A. J. Hanson, R. A. Cross. Interactive visualization methods for four dimensions. In *IEEE Visualization*, 1993, 196–203

[7] V. Interrante and C. Grosch. Strategies for Effectively Visualizing 3D Flow with Volume LIC. In *IEEE Visualization*, 1997, 421–424

[8] G. Ji, H.-W. Shen, and R. Wenger. Volume Tracking using Higher Dimensional Isocontouring. In *IEEE Visualization*, 2003, 201–208

[9] A. Joshi and P. Rheingans. Illustration-inspired techniques for visualizing time-varying data. In *IEEE Visualization*, 2005, 679–686

[10] Robert S. Laramee, Helwig Hauser, Helmut Doleisch, Benjamin Vrolijk, Frits H. Post, and Daniel Weiskopf. The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. In *Computer Graphics Forum*, 23(2):203–221, 2004

[11] W. de Leeuw and R. van Liere. BM3D: Motion estimation in time dependent volume data. In *IEEE Visualization*, 2002, pp. 427–434.

[12] A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. In *IEEE Visualization*, 2002, 211–218.

[13] M. Masuch, S. Schlechtweg, and R. Schulz. Speedlines – Depicting Motion in Motionless Pictures. In *SIGGRAPH '99 Conference Abstracts and Applications*, 1999, 277.

[14] N. Max, R. Crawfis, and D. Williams. Visualizing Wind Velocities by Advecting Cloud Textures. In *IEEE Visualization*, 1992, 179–184

[15] M. Nienhaus and J. Döllner. Depicting Dynamics Using Principles of Visual Art and Narrations. *IEEE Computer Graphics & Applications*, 25(3):40–51, 2005

[16] N. Svakhine, Y. Jang, D. Ebert, and K. Gaither. Illustration and Photography Inspired Visualization of Flows and Volumes. In *IEEE Visualization*, 2005, 687–694

[17] C. Tietjen, T. Isenberg, and B. Preim. Combining Silhouettes, Surface and Volume Rendering for Surgery Education Planning. In *IEEE VGTC Symposium on Visualization*, 2005, 303–310

[18] J. Woodring, C. Wang, and H.-W. Shen. High Dimensional Direct Rendering of Time-Varying Volumetric Data. In *IEEE Visualization*, 2003, 417–424

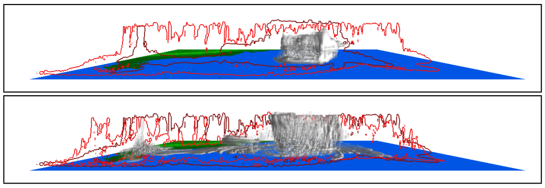

Figure 8: Sideview of the cloud dataset superimposed with edges. In each subfigure the first volume is rendered using direct volume rendering. Of the second and third volumes only the silhouette is shown in dark resp. bright red.

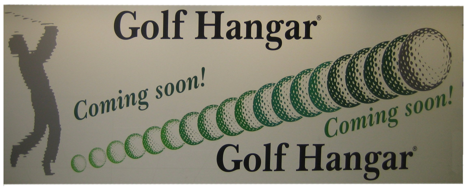

Figure 9: A golf hangar advertisement depicting motion by showing preceding golfballs. (Image courtesy of Golf Hangar at the Düsseldorf Airport, Germany.)

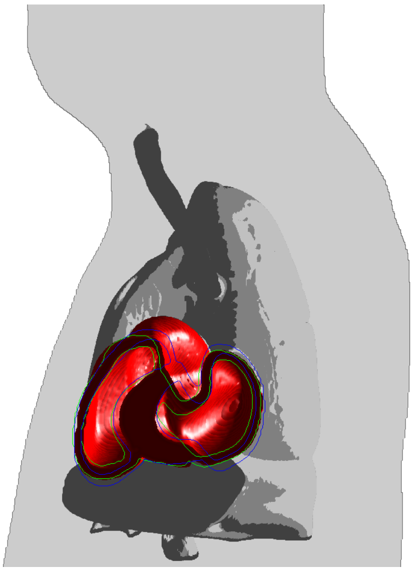

Figure 10: Color version of the cut through a beating human heart. The green and blue edge depict the next two timesteps.

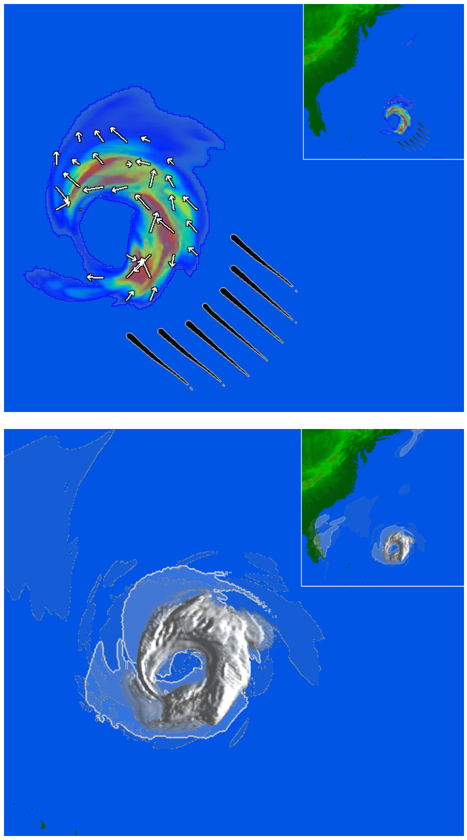

Figure 11: Color version of the visualization of the Hurricane Isabel. Using speedlines to show simultaneously the motion of the hurricane and the distribution of rain (*top*). Volume Rendering of the clouds modality by visualizing the silhouettes of the next two steps using edge detection (*bottom*).