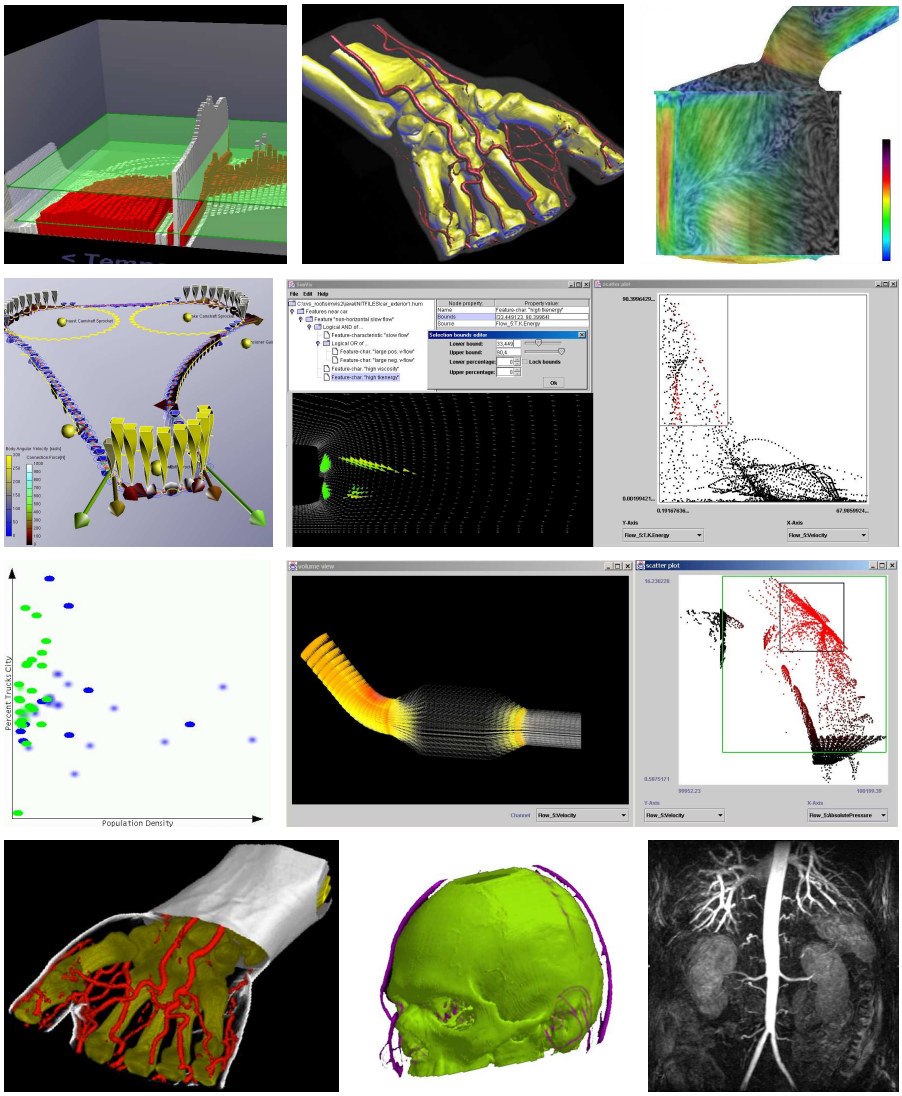
Selected Papers by Helwig Hauser (2000–2004)



Ten selected papers, all peer-reviewed and published by Helwig Hauser (together with others) in 2000–2004, as an addendum to Helwig Hauser's Habilitationsschrift from December 2003, which furthermore contains seven other selected papers from the same research period. Above, every one image represents one of the following papers (in chronological order, most recent first). See the preface for publication information on the ten papers.

Preface

The following text is a collection of ten selected papers which all have been published by Helwig Hauser (together with others) in the years 2000-2004 (after having passed through a peer-review process). This collection is an addendum to the Habilitationsschrift (i.e., the thesis which is required for the Habilitation; short: Habil) of Helwig Hauser which contains seven further selected papers from the same time-span. In the Habilitationsschrift relations to other papers by Helwig Hauser (not themselves contained in the Habil) are discussed – most of these papers referred to are contained in this collection. Below the publication information is given for all ten papers.

The first paper ("TimeHistograms for Large, Time-Dependent Data" by Robert Kosara, Fabian Bendix, and Helwig Hauser) is currently (April 2004) accepted for publication in the Proceedings of the 6th Joint IEEE TCVG – EUROGRAPHICS Symposium on Visualization (VisSym 2004), May 19-21, 2004, in Konstanz, Germany. The paper within this collection is the paper version which will show up in the proceedings.

The second paper ("High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware" by Markus Hadwiger, Christoph Berger, and Helwig Hauser) is published in the Proceedings of IEEE Visualization 2003 (Vis 2003), Oct. 19-24, 2003, in Seattle, Washington, pp. 301-308. The paper is related to Helwig Hauser's Habil and especially to "Two-Level Volume Rendering" (Helwig Hauser et al., TVCG 2001).

The third paper ("Image Space Based Visualization of Unsteady Flow on Surfaces" by Robert S. Laramee, Bruno Jobard, Helwig Hauser) is also published in the Proceedings of IEEE Visualization 2003 (Vis 2003), Oct. 19-24, 2003, in Seattle, Washington, pp. 131-138. A follow-up paper (together with Jack van Wijk) is accepted for publication in TVCG 2004.

The fourth paper ("Interactive 3D Visualization Of Rigid Body Systems" by Zoltan Konyha, Krešimir Matković, and Helwig Hauser) is also published in the Proceedings of IEEE Visualization 2003 (Vis 2003), Oct. 19-24, 2003, in Seattle, Washington, pp. 539-546.

The fifth paper ("Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data" by Helmut Doleisch, Martin Gasser, and Helwig Hauser) is published in the Proceedings of the 5th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003), May 26-28, 2003, in Grenoble, France, pp. 239-248. This paper is strongly related to Helwig Hauser's Habil.

The sixth paper ("Useful Properties of Semantic Depth of Field for Better F+C Visualization" by Robert Kosara, Silvia Miksch, Helwig Hauser, Johann Schrammel, Verena Giller, and Manfred Tscheligi) is published in the Proceedings of the 4th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2002), May 27-29, 2002, in Barcelona, Spain, pp. 205-210. This paper is also related to Helwig Hauser's Habil and especially to "Semantic Depth of Field" (Kosara, Miksch, Hauser; InfoVis 2001).

The seventh paper ("Smooth Brushing for Focus+Context Visualization of Simulation Data in 3D" by Helmut Doleisch and Helwig Hauser) is published in the Proceedings of The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media 2002 (WSCG 2002), Feb. 4-8, 2002, in Plzen, Czech Republic, pp. 147-154. This paper is strongly related to Helwig Hauser's Habil.

The eighth paper ("RTVR – a Flexible Java Library for Interactive Volume Rendering" by Lukas Mroz and Helwig Hauser) is published in the Proceedings of IEEE Visualization 2001 (Vis 2001), Oct. 21-26, 2001, in San Diego, CA, pp. 279-286. This paper is strongly related to Helwig Hauser's Habil and especially to "Two-Level Volume Rendering" (Helwig Hauser et al., TVCG 2001).

The nineth paper ("Space-Efficient Boundary Representation of Volumetric Objects" by Lukas Mroz and Helwig Hauser) is published in the Proceedings of the 3rd Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2001), May 28-30, 2001, in Ascona, Switzerland, pp. 193-202. This paper is also related to the Helwig Hauser's Habil.

The tenth paper ("Interactive High-Quality Maximum Intensity Projection" by Lukas Mroz, Helwig Hauser, and Eduard Gröller) is published in the Proceedings of EUROGRAPHICS 2000 (EG 2000), Aug. 20-25, in Interlaken, Switzerland, pp. C-341-C-350. This paper is also related to the Habil.

Table of Contents

TimeHistograms for Large, Time-Dependent Data

Robert Kosara, Fabian Bendix, Helwig Hauser

VRVis Research Center, Vienna, Austria
http://www.VRVis.at/vis/
{Kosara, Bendix, Hauser}@VRVis.at

Abstract

Histograms are a very useful tool for data analysis, because they show the distribution of values over a data dimension. Many data sets in engineering (like computational fluid dynamics, CFD), however, are time-dependent. While standard histograms can certainly show such data sets, they do not account for the special role time plays in physical processes and our perception of the world.

We present TimeHistograms, which are an extension to standard histograms that take time into account. In several 2D and 3D views, the data is presented in different ways that allow the user to understand different aspects of the temporal development of a dimension. A number of interaction techniques are also provided to make best use of the display, and to allow the user to brush in the histograms.

1. Introduction

While time is just another dimension in many data sets in terms of data organization, the way we perceive it and also its influence on physical phenomena is quite unique. Therefore, special methods are needed for time-dependent data.

Getting an overview of data – the major trends and the outliers – is very important in trying to understand the data. This is even more important for data that changes over time. One very popular method for getting such an overview are histograms. Standard histograms, however, provide little interaction and are also of little use for time-varying data.

The context of this work is computational fluid dynamics (CFD). Typical data sets in our experience are usually in the order of magnitude of about 100,000 cells for 100 time steps, or 1,000,000 cells for 10 time steps. Each cell has about 15 to 20 data attributes for each time step.

For dealing with such amounts of data, SimVis [2] was developed (see also Section 5). SimVis provides a number of different views, most of which use methods that are traditionally used in information visualization (InfoVis). Examples for such methods are scatterplots, parallel coordinates, and histograms. These views not only show the data, they also provide the user with the means to interact with the data and specify interesting features by means of brushing (i.e., selecting parts of the data and showing the selection in all views, thus making it possible to see connections).

1.1. Histograms and Time

A histogram is a bar chart, where the number of bars is usually selected by the user. Each bar corresponds to a *bin*, which is a value range on the selected axis. All values in the data set are compared to the bin boundaries, the values that fall into each bin are counted. The height of each bar then represents the number of values in each bin (Figure 1).

We distinguish two types of dimension in the discussion of time-dependent histograms: the data dimension and the display dimension (Figure 1). The traditional histogram is one-dimensional in its data (the bins are defined along one dimension) and drawn in two dimensions. It is of course also possible to draw a standard histogram using 3D graphics. But the more interesting case is that of two data dimensions, one of which is the time. Such a two-dimensional histogram (in terms of data) can be quite easily understood if drawn in 3D, because the user can imagine it being made up of a number of one-data-dimension histograms, one for each time step. Such a two-dimensional histogram can also be drawn in 2D, however, by projecting it into the plane along the count axis (Sections 4.2 and 4.3).

Time is handled in SimVis by means of time steps, which do not have to be equally spaced. Data for each point is only available for these time steps – therefore, they can be considered *event time* (meaning that the time axis is shaped by the occuring data, not in a continuous way). A time-related

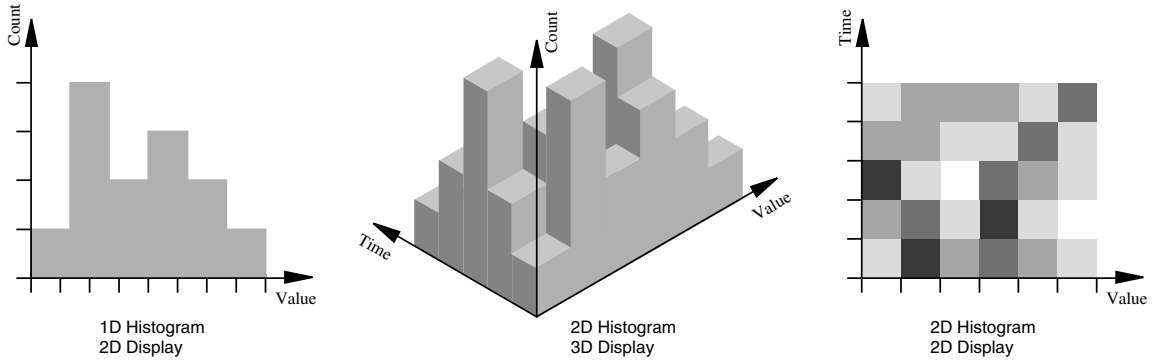


Figure 1: *2D and 3D visualizations of 1D and 2D histograms.*

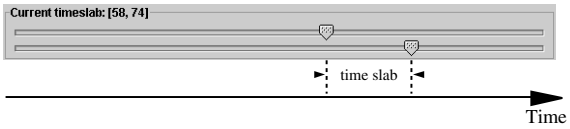


Figure 2: *A time slab. The sliders define a time slab from time step 58 to 74 (both are included in the slab).*

concept that plays a big role in SimVis is the definition of a time range, called a *time slab* (Figure 2). The current time slab can include one or more time steps, and is important because actions such as brushing apply only to these time steps. Time slabs can be defined as any interval in the total time range of the data set, there is no hierarhical time structure (e.g., combining seconds into minutes). Time slabs are always defined in terms of full time steps in the data set.

1.2. Motivation for Improvements, Goals

For the data described above, standard histograms were of little use. While it was possible to change the time step displayed by the standard histogram, the static image always only showed one time step. This proved to be too little information to be useful in practice.

At the same time, the standard histogram provided a good basis to start from because of its visual simplicity, but also because of its familiarity and ubiquity.

TimeHistograms are an extension of standard histograms which take time into account. Our goals in the design of TimeHistograms were the following:

- Give an overview over complex, time-varying data. Histograms are very useful for looking at all the data at once, but doing this in a way that is abstract enough not to overwhelm the user with details.
- Show temporal information in static images. While interaction is a very important part of TimeHistograms, the user needs to be able to look at static images to get a better impression of what is going on. This is also important for documentation, where static images are needed.
- Retain the easy readability of the standard histogram. Make the additional information as easy to read as possible, and always provide the user with simpler information within the same image to fall back on.
- Support linking and brushing. Besides view-related interaction, the histogram also has to show brushed data, and also allow brushing in both 2D and 3D views.

1.3. Data Set

The data set used in the illustrations in this paper comes from the simulation of gas flow through a T-junction section joining two pipes. There is cold air flowing through the junction from one pipe, when a stream of hot air starts to enter from the other. The two air streams mix until an equilibrium state is reached. The data set consists of about 33,000 cells with 16 dimensions at 100 time steps.

2. Related Work

Histograms are used in both information and scientific visualization applications. This section gives a brief overview over a few uses of histograms that provide some interactivity (other than the selection of the axis). While there are certainly many more examples of histograms implemented in different applications, very little seems to get published about them – at least in the visualization and information visualization literature.

The influence explorer [10] allows the user to explore dependencies in situations where input as well as output values are multi-dimensional. It uses histograms for both the specification of input variables and the display of the results. The user can change the input value ranges on one set of histograms to see the results displayed on the other.

In a similar way, histograms can be brushed to answer

questions in a geographical data application [6], where the histograms act purely as a visualization of input data, which can be brushed to see the output on a map.

Parallel bargrams [11] are histograms with bars rotated by 90 degrees, and put end to end. This way, bargrams show the distribution of values on axes with few different values, and allow the user to select values by simply clicking on them.

A different kind of histogram are adjacency matrices that are used for the display of telecom call information [1]. Each cell in such a matrix is colored by the number of items in it, thus effectively creating a 2D histogram. Similar 2D histograms can be found in fields such as sound processing (time-frequency charts).

Histograms are not restricted to information visualization, however, but are also used for transfer function specification in volume rendering [4]. They show first and second derivatives of the data, and this way allow the user to directly select boundaries between materials – which usually are the most interesting features.

3. Early Approaches to TimeHistograms

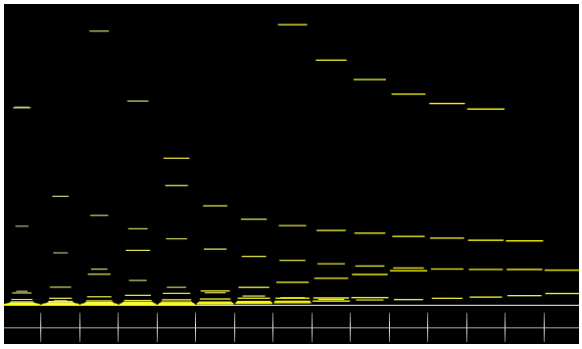
Designing good histograms proved to be much more difficult than initially anticipated. This section describes two early attempts that did not lead to useful time-dependent histograms.

One of the developed techniques was to simply exchange the normal bars with History Bars [8], which is shown in Figure 3a. There is an outline of the current histogram and for each bin, multiple lines are drawn at various heights. These lines represent bars at different time steps and they are discriminated by their width, opacity and color.

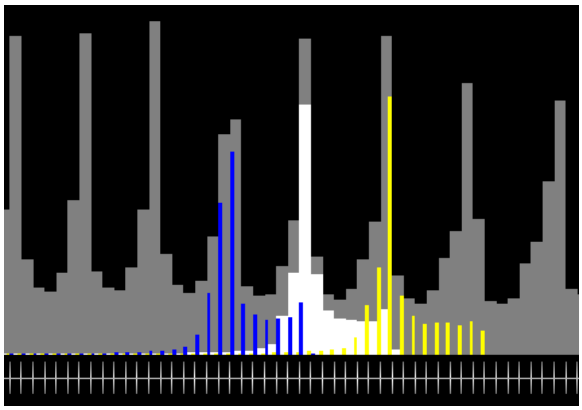
History Bars had good user feedback when used in process visualization, but proved to be problematic in our context. Due to the visual similarity and proximity of the bars, it is very easy to perceive bars from different bins and time steps as one unit (appearing as slightly curved lines in Figure 3a). However, this impression of trends is totally wrong.

Because of that there was the need for a shape that is more self-contained and that forms a better contrast to the rectangular histogram. The resulting mode which is described below in detail is called the Point Mode (Section 4.3.2).

Another unsuccessful idea was to display the next and the previous time step in a normal histogram fashion, but as a smaller "sub-chart" (Figure 3b). Visually separating the two levels proved to be impossible, however, and when moving the time slider, it was impossible to understand the changes. The line mode (Section 4.3.3) is a better solution following a similar idea.



a) History Bars – different time steps per histogram bar are encoded as the length of the line. Proximity, however, is a stronger cue, so curved lines appear that do not represent real data.



b) Inner History – the sub-charts cannot be separated easily, therefore the bars blend into one confusing plot.

Figure 3: *Early unsuccessful, abandoned approaches.*

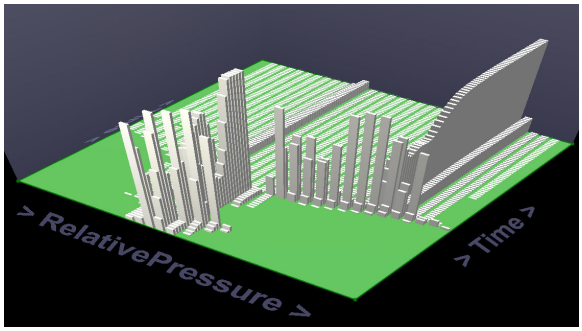
4. TimeHistograms

One of the goals in the development of TimeHistograms was to retain the simplicity and ease of understanding of the standard histogram as much as possible (Section 1.2). The traditional histogram is still visible in all the views, and is distinguished visually from the additional information. So the user can always look at the simple histogram of the currently selected time step and ignore all other information.

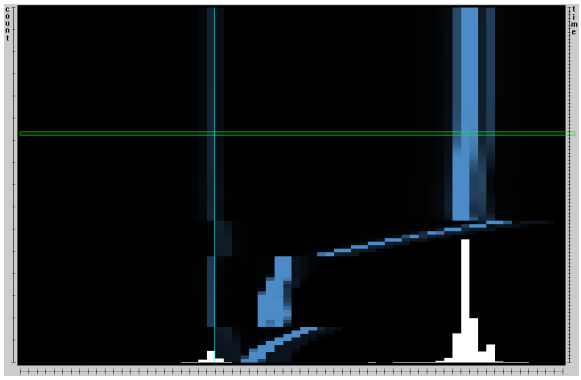
The discussion in this section is structured by the display dimensionality, not the dimensionality of the data (see the discussion in Section 1.1). The 3D histogram is described first, because it provides a good basis for the 2D histograms that are explained later. The 2D time context also is practically a projection of the 3D histogram (Figure 4).

4.1. TimeHistograms in 3D

While a 3D depiction is generally more complicated and requires more interaction, we have found it to be easier to un-



a) TimeHistogram in 3D, showing the development of pressure over time. One can see where the values change a lot and where a stable condition is reached.



b) 2D time context, showing the same data. This is a projection of the 3D histogram onto the background of the 2D histogram. The green frame marks the current time step on the time axis of the time context display (right).

Figure 4: *TimeHistograms in 3D and 2D time context (see also Colorplate 1).*

derstand for time-dependent histograms, and also to provide a better overview than the 2D views.

The concept of the 3D Timehistogram is quite simple: draw the histogram for each time step as a row of cuboids rather than rectangles, and arrange them one behind the other along the time dimension (Figures 1, 4, and 7).

The 3D display is made up of shaded cuboids that give a very good 3D impression. The user can very easily navigate around the histogram. Rotation is done in an object-centered way, and restricted to two axes: a vertical one for rotating the histogram to see it from different sides, and a horizontal one to change the height of the virtual camera over the ground plane. When the height is changed, the axis labels are rotated so that they face the user as much as possible. To see the direction of the axes, a simple indication is added to the labels that shows in which direction the values get larger (they follow the direction indicated by the '>' signs).

It is also possible to zoom in and out (this is done using the mouse wheel), and to pan the display (by grabbing the histogram with the right mouse button and moving it around).

The described interaction is very powerful, yet easy to learn for the user. In addition, this view also implements the common interactions described in Section 4.4.

Another important interaction method is the control of the global height of the histogram bars. This control is simply a magnification of the height dimension, and can be changed in a rather large range. This is important in 2D as well as 3D displays, because there is no way for the program to tell which information the user is interested in. If the user is interested in the large bars, their height might lead to smaller bars not being visible anymore due to the dynamic range of the data.

4.2. Projected 3D Histograms as 2D Context

In addition to the standard histogram, a time context display can be put onto the background of the histogram (Figure 4b). This context overview is conceptually a projection of the 3D histogram along the height dimension: its value axis is parallel to the value axis of the histogram, and the time axis is parallel to the height axis of the 2D histogram.

The height values of the 3D histogram projection in the background are mapped to the opacity of rectangles representing the histogram bins. This mapping is linear and can be influenced by the user with a simple slider, which changes the slope of the mapping. This way, the user can select if he or she wants to see more details in the small or in the large values. Providing more parameters (e.g., which mapping, an offset) was considered but not done in order not to overwhelm the user with too many degrees of freedom.

The goal of the 2D context display is not to show exact values, but trends. It is possible to see if the values increase or decrease over time, or if they stay (approximately) constant. This view shows this information without the need for interaction and without any occlusion.

On the time axis (which is displayed on the right of the display, see Figure 4b), the current time step or time slab is marked with a green rectangle. This connects the time slider with the depiction and also allows the user to compare histogram bar heights with shades of blue, and thus makes it possible to tell which approximate heights bars have that are represented only in the overview (from other time steps). Additionally, the user can select the time step or slab to be displayed directly by clicking on this time axis. This provides a more direct means for navigating to interesting time steps than using the standard time control (Figure 2) would.

The 2D overview is not a mode of the display, but rather

additional information that can be combined with any of the other 2D modes. It is most useful with standard and point mode, however.

The data dimension shown in Figure 4b is the relative pressure, which shows an interesting pattern. After a relatively steady increase, it very quickly settles down into an almost constant pattern after about half of the total time. An interesting "overshooting" can also be seen right before the settling down begins.

4.3. TimeHistograms in 2D

Another way of showing changes over time in a standard histogram is the integration of the time axis into the normal 2D histogram. Here the screen space limits the possibilities of displaying the temporal data distribution.

All extensions presented in this section are based on the standard histogram display. The user can always look at the display and see the original histogram for the current time step. The remaining time steps are then visualized according to the used mode. Each mode augments the visualization in a different way and provides different kinds of information.

4.3.1. Standard Mode

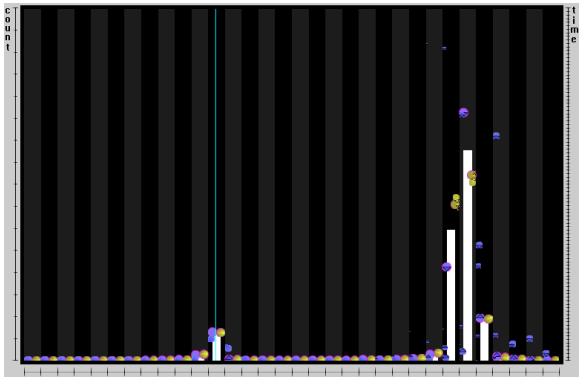
In addition to simply displaying the histogram for an axis, the user can move the currently displayed time step, and this way see the differences. It is also possible to switch on additional context, which is the sum of all histograms in the current time slab. This provides information about a range of time steps and is comparable to the arithmetic mean, because the height of the bars can be scaled arbitrarily and remains proportional (i.e., the display acts as if the sum was then divided by a number close to the number of time steps in the time slab).

Getting information about the temporal aspects of the histogram requires interaction in this mode. With a slider, the user can move the time slab or time step and this way traverse all time steps. If the differences between two successive displays are too big, this traditional histogram will not provide enough temporal context to make it possible for the user to envision the temporal data distribution – which is where the other modes come in.

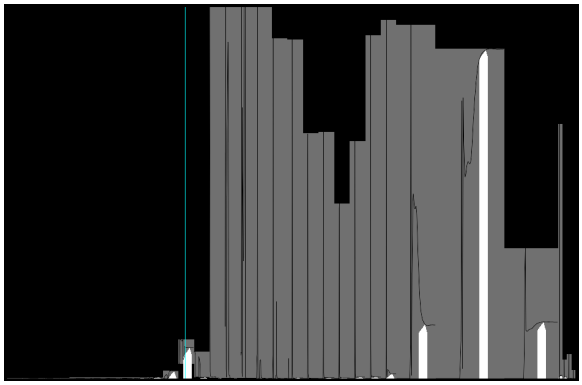
One very simple but quite important feature of the 2D view in TimeHistograms is that the zero on the value axis is marked with a turquoise line (e.g., Figure 4b and Figure 5c, right). This provides very important context to the engineer that is also much easier to read than the numbers giving the limits on the value scale.

4.3.2. Point Mode

In addition to the bars showing the histogram, this mode shows the immediate temporal neighborhood for each bar using small disks or "points" (Figure 5a). The points are



a) Point Mode. The colored disks/points represent the past and future of each bar (see below).



b) Line Mode. Every line chart shows the temporal development of the counts in one bin. The white bars show the current histogram, their tips point to the current values in the graphs.

c) Details of point and line mode. Left: The larger the point, the closer it is to the current time step. Blue points represent the past, yellow points the future. Right: The grey box contains the whole graph, therefore a vertical offset of the box tells the user that the count in this bin is never zero.

Figure 5: *The different modes of 2D TimeHistograms.*

drawn next to each bar, and represent past and future values for a small number of time steps. Points representing the past are colored blue and appear to the left of the bar, points showing future values are yellow and are drawn to the right of the bar (consistent with the time axis on the time slider). Their size also differs as a function of their distance from the current time step. The closest points are the largest, and they get smaller the farther back or ahead in time they are.

Because it is not easy to compare sizes of points that are far away from each other, small triangles are drawn into the points if they are farther than two diameters away from each other. They point to the next point, telling the user where to look for the next point, while retaining a smoother image when the points are close to each other (the sizes are then easy to compare and the sequence quite obvious). In addition, the points representing the time steps immediately preceding and following the current one, respectively, are also drawn with a purple outline, so that the user can find them quickly. What sounds complicated in the textual description is visually quite simple and relatively easy to follow. Usually, the size change (which is immediately visible) is also sufficient to see the temporal development – the additional information is only there for very complex cases.

The design for this mode built on the lessons learned from the failed attempt (Section 3) to make use of an idea from process visualization of real-time data [8], where values are displayed so that they leave a 'trail' that shows their history. The round points form a nice contrast to the rectangular bars, and are therefore easy to distinguish. It is also possible to 'switch' between looking at the bars or looking at the points easily.

In order to directly compare histograms from two different time steps, the user can activate a *tooltip* mode (Figure 6d). In this mode, if the user hits a point with the mouse, a polyline is drawn for the histogram of the time step that this point belongs to. This display is more effective to compare entire histograms, while the points are sufficient for telling the direction for a single bar or a small number of bars.

When using the F+C distortion (Section 4.4.1) to zoom into parts of the histogram, bars are drawn for the immediate temporal neighborhood, and points are then drawn for some more time steps ahead and back (Figure 6c). This adds to the information present in the display, without using much more space. At the same time, the points and bars' heights are visually so similar, that there is almost no discontinuity between them, and it is very easy to tell the shape of the bars that are scrolled in when moving the time slider from the points.

The same dimension (relative pressure) is used in Figure 5a as in the description of the 2D overview (Section 4.2). The little overshooting can also be told in this mode, even though it is not that obvious. The histogram for the current time step can be read much more precisely, though.

4.3.3. Line Mode

A different 2D representation of the time-dependent histogram uses a more familiar metaphor. A simple line graph is drawn inside each box, showing the temporal development for that bin (Figure 5b). Inside the box, the time axis is parallel to the value axis of the histogram (this is similar to the idea of dimensional stacking [5]). A small 'pointer' is drawn on top of each bar of the histogram which points to the current position on the graph.

The whole line graph for each bar is contained in a gray box. This box shows the user the minimum and maximum values for each of the bars. If the value count in the bin is never zero, the user will notice this because the box will not touch the base line in this case.

Since there is usually not enough screen space to properly see the time line, Focus+Context distortion (Section 4.4) makes it possible to see a small set of bars in detail.

But even without interaction, this mode provides two levels of information. When looking at the global structure (white on dark), one sees the histogram for the current time step. When looking at the details (black on grey), one can read the development of each bar over time.

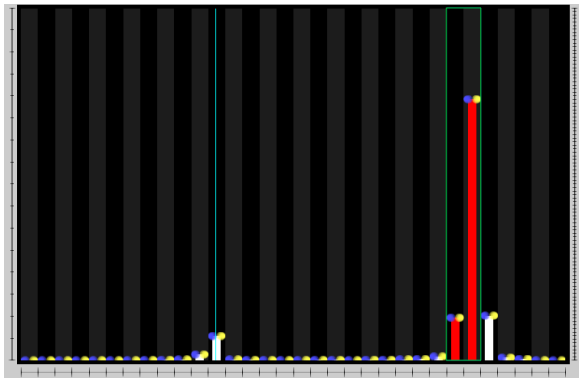
4.4. Common Interactions

Independently of the view, there are a few interactions that are always possible. They are implemented in slightly different ways, but are conceptually the same.

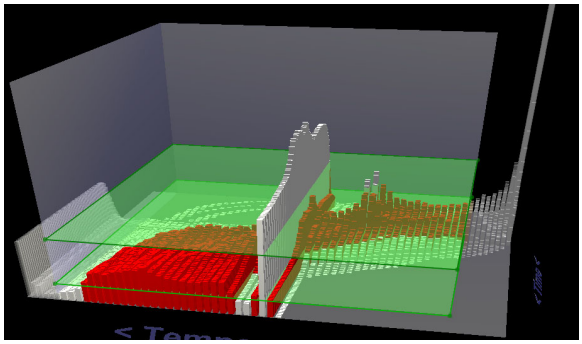
Perhaps the most important one is the zooming into the data. The user can specify ranges, in both the value and the time dimensions (the time slab). In the 2D views, only the data within these regions are displayed. Time is treated a bit differently here, because the user can not only change the upper and lower limit of the time slab to be displayed, but can also move the complete slab. This has proved to be useful in the other SimVis views, and seems to be a rather natural way of working with time for complex data.

4.4.1. Focus+Context (F+C)

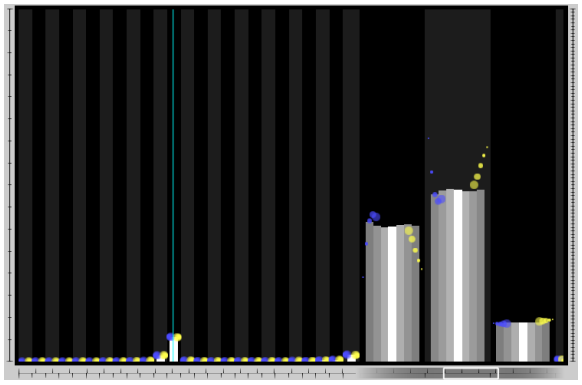
In the 2D views, a classical focus+context method is implemented that is similar to the perspective wall [7]. The user selects a region that can then be enlarged, while the rest of the display is compressed (Figure 6c). This distortion is not binary like the TableLens [9], but changes from the center of interest to the outer parts. The same distortion is applied to a whole bin, however. The distortion is not only visible in the bars and the background, but there are also two scales at the bottom of the display: one shows the undistorted scale, and one the distorted one. This provides the user not only with the possibility to get more space for the display, but also different information can be displayed in the additional space (e.g., in point mode, bars are shown for the immediate temporal neighborhood). The enlarged region can also be moved, to explore the data with the current settings.



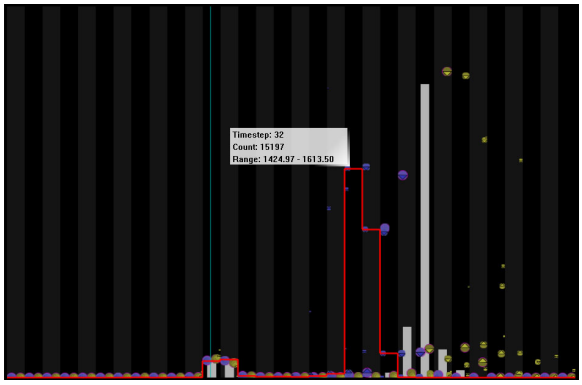
a) Brushing in 2D – the user has selected the values between the two vertical green lines. Brushed bins are now drawn in red. The turquoise line shows where zero is on the value axis.



b) Brushing in 3D – all bins whose counts lie between the two horizontal planes are brushed (displayed in red).



c) Table lens-like Focus+Context (F+C) – parts of the display can be zoomed in to show more information, like more time-steps as a bar chart.



d) Tooltip mode – a tooltip is shown that gives the user information about the bin (current time step, data point count, value range) as well as a second histogram (in red) for the point the mouse is currently pointing at. Differences (right) as well as similarities (left) are visible.

Figure 6: *Interaction in TimeHistograms (see also Colorplate 2)*

In 3D, the F+C display is different. The bars representing the values are not only pushed out of the cube representing the current value range, but their footprint is also made smaller, and they are drawn translucently. This provides a very clear metaphor for the F+C display (the user can see what happens while moving the sliders), and also makes it possible to see through large occluding structures (Figure 7). Such large structures are often found at the border of the value range, because they represent values from start or boundary conditions.

4.4.2. Brushing

Brushing in the histogram is done in terms of bins, both in the 2D and 3D views. When the user makes a selection in 2D (Figure 6a), all bins that are touched by the selection are completely selected (and the selection is expanded). Because the user does not know the distribution of values inside the bin, partially brushing bins does not appear to be sensible. In the temporal dimension, only the points in the current time slab are brushed.

In 3D, brushing works in a different dimension than in 2D, namely in terms of the value count in the bins. The user can drag two horizontal planes inside the box of the 3D histogram in the height direction, so that the planes intersect with some or all of the cuboids. Each of the planes represents a certain count value. All data values are selected that belong to bins which have a height in the range defined by the two selection planes (Figure 6b). This selection only applies to the bins in the focus, i.e., those that have not been pushed out into the context region.

Arguably, brushing by bin count rather than by bin could work in a very similar way in 2D, but has not yet been implemented. This is planned for the future, though.

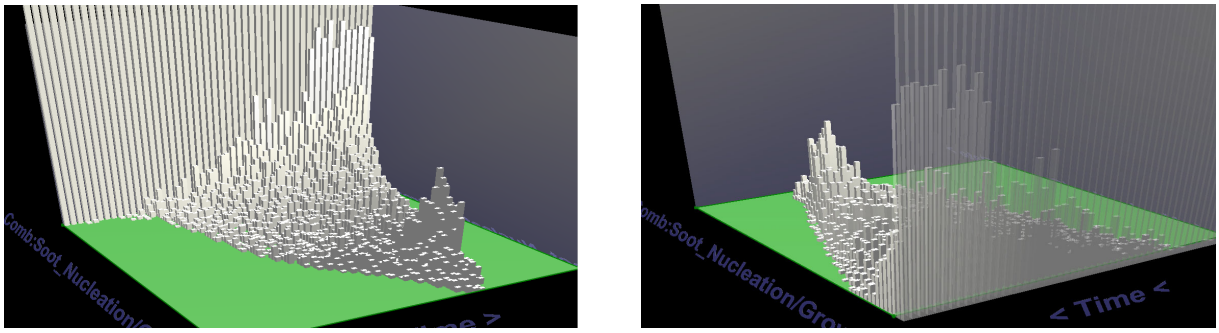


Figure 7: *Getting rid of large occluders near the borders. The occluder at the far end of the left image would block the view from the other side. Pushing a few rows of the histogram into the context allows the user to see the data from another perspective, and even to get an idea of the context data.*

4.4.3. Scaling

The program also provides the possibility to automatically scale the value and the height axis. The user can request each of the axes separately to be either scaled to the current time slab or the complete data set. Scaling the value range such that empty bins at the lower and upper end of the scale are left out can cause problems with understanding change. When the time slab is moved and the data mostly moves along the value axis, the user must keep track of the numeric values that give the locations of the boundaries. But for quickly zooming into the currently relevant values, this mode is very useful.

Scaling in the height dimension, as mundane as it may sound, is a very important interaction in TimeHistograms. Automatic scaling of this axis is very difficult, because of the extreme dynamic range of the values (one bar might represent 200,000 data items, while another might only have a count of 50). Therefore, the user must be able to change the scaling to change the focus to large or small bars. In the 2D view, if a bar gets taller than can be displayed in the window, a small arrow is drawn onto it, showing the user that the bar actually extends beyond that border. In 3D, this is not necessary, because the user better understands that he/she is only looking through a "window" at the data, and it is also easier to see the full extent of the bar (by changing the viewpoint).

4.4.4. Number of Bins, Tooltip

Another very simple interaction is the definition of the number of bins. At the moment, the user can select between 16, 32, 64, 128, 256, 512, and 1024 bins. The reason for using powers of two is an implementation detail (calculations for display distortions are easier this way), but it would be very easy to change this to allow arbitrary numbers. From our experience, the numbers 32 and 64 are the most common choices, and there seems to be no need for an intermediate number of bins. Choosing the right bin size is usually considered a very important question for histograms, but for this application domain, we have found it to not be very critical. The data is relatively smoothly distributed, and the values are never precisely the same, so they tend to "leak" to neighbouring bins. The histograms at different resolutions therefore look very similar and hold few surprises for the user.

The tooltip, while available in all modes, is especially useful in point context mode. For each bar, the tooltip shows the time step, the number of values in the bin, and the precise bin boundaries. The time step is interesting because in point context mode, the tooltip can also display an alternative histogram in addition to the current one (Section 4.3.2).

4.4.5. Using A Different Second Dimension

Instead of time, a different dimension can be selected as the second dimension (Figure 8). The resulting histogram shows the distribution of values into bins defined by the two dimensions. There is a very simple duality between this display and a scatterplot: Seen from above, the histogram looks like a scatterplot (if a sufficient number of bins is selected). This is similar to the 2D overview (Section 4.2). But the scatterplot does not show how many values were plotted onto the same pixel – this information can only be seen in the histogram.

5. Integration in SimVis

SimVis (described briefly in the introduction) is a framework for interactive specification of features in simulation data. The definition of such features is modeled as a tree of brushes, which are selections of data. These selections can be combined using logical operations to define more complex features (e.g., regions where the flow is slow, the temperature is high, and where the rate of change was small in a certain time period).

The framework makes use of multiple, linked views from information as well as scientific visualization. The result of the feature specification is linked to all views and can be seen in various views. The TimeHistogram is one of these views.

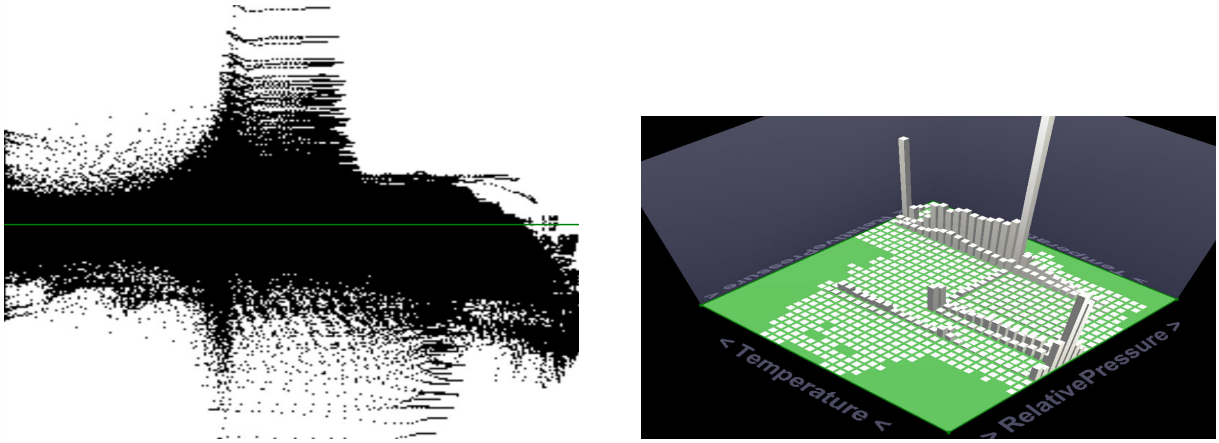


Figure 8: *Using the TimeHistogram to plot two dimensions against each other – both plots show the same dimensions. In contrast to the scatterplot, the histogram shows the extreme clustering of a lot of values around very distinct structures.*

TimeHistograms can both show brushed data and allow the user to brush directly in the histogram. In SimVis, two different levels of brushing are distinguished: brushes that were done in the current view, and the globally defined brush (which is the combination of all the brushes, so the points in this brush might not contain any of the points brushed in a particular view). The color of the bars changes gradually from white to red (or yellow) the more brushed data points they contain.

All modes of TimeHistograms run at interactive frame rates for the data sets described in the introduction.

6. Discussion and Experiences

TimeHistograms have not been the subject of a formal user study yet, but they have been presented to CFD experts who are working with the SimVis application. The results of this rather informal evaluation are described here, along with a few examples of how interaction works in practice with TimeHistograms.

We found to our surprise that the 3D display is much more intuitive and accessible than the 2D modes. Users could relate to it better and also understand the meaning of the graphical display more easily. The occlusion problems that we normally associate (occlusion, difficult navigation, etc.) were no real issue.

At this point, we also found a problem in the initial point mode design, which did not have the marks for the first time step and the small arrows pointing to the next or previous time step. We found that we needed this additional information so that users would not overlook points or not be able to tell which came first (the size difference can be hard to see when the points are far apart).

TimeHistograms were also used in a recent case study on a Diesel exhaust system [3], where they mainly served to tell when the chemical processes (burning of the soot) had settled down. To do this, one needs to look at the histogram and to tell when the changes between time steps become so small that they can be ignored. The TimeHistogram proved to be very valuable for this purpose.

When implementing a 2D and a 3D view of essentially the same data, the question of which performs better comes to mind. While we found that 3D provided a much better introduction to time-dependent histograms for the novice users, we also found some problems with it. One is certainly occlusion, which is a problem inherent in 3D. It is much easier to miss a feature in the 3D histogram than when using the 2D overview, for example (Figure 9). But the 2D modes also bear the possibility of "temporal occlusiont't', due to the limited amount of time steps they can display at once.

Concluding, the combination of both 2D and 3D views on the same data offers a lot of ways to gain insight into the data by depicting its different aspects.

7. Conclusions

TimeHistograms are an extension of the well-known histograms for time-dependent data. They are a useful addition that provides additional insight, particularly for large and complex data sets. The presented techniques provide different views on the data in 2D and 3D, which require interaction, but also provide a lot of additional information.

While the 3D view was originally anticipated to be problematic due to the usual perception problems, we found it to be very useful and easy to understand for our audience, and even to serve as an introduction to the more demanding 2D views.

TimeHistograms were demonstrated with scientific data

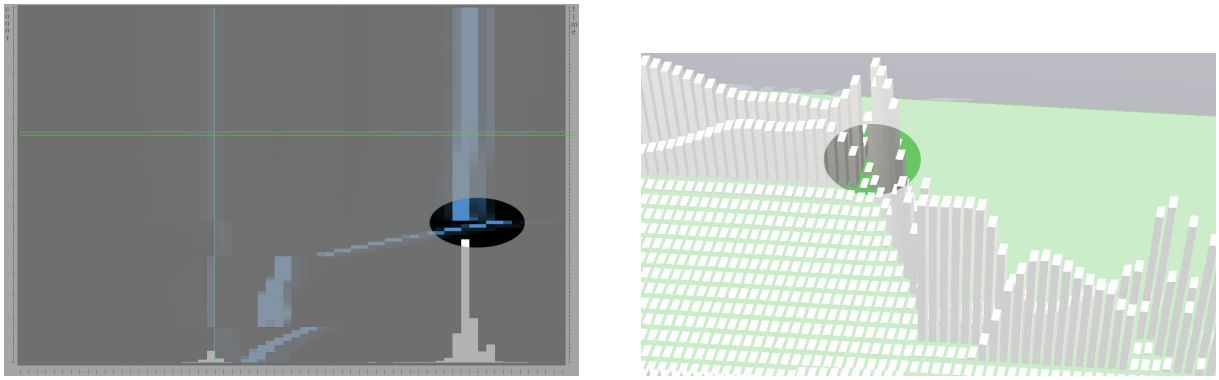


Figure 9: *Working around occlusion using interaction: Seeing the small interesting detail marked in the left image requires quite some interaction to be clearly visible in the 3D view on the right. In such a case, the 2D view reveals details at one glance that could have easily been missed in 3D.*

in this paper, but they are certainly also useful for time-varying data that is not bound to a physical object.

Future work includes more in-depth evaluation of the technique with users, as well as more work on the 3D view. Brushing in that view should be made more flexible, and also the separation of different time steps and labeling of histograms in 3D requires some more work. More flexible brushing in both 2D and 3D modes will also be another interesting topic to work on.

Acknowledgements

This work has been done in the scope of the basic research on visualization at the VRVis Research Center in Vienna, Austria (`http://www.VRVis.at/`), which is funded by the Austrian research program Kplus. The data set used in the images is courtesy of AVL List GmbH, Graz, Austria. We would like to thank the reviewers for their very insightful and detailed remarks.

References

[1] James Abello and Jeffrey Korn. MGV: A system for visualizing massive multidigraphs. *Transactions on Visualization and Computer Graphics*, 8(1):21–38, 2002.

[2] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the 5th Joint IEEE TCVG - EUROGRAPHICS Symposium on Visualization (VisSym 2003)*. ACM Press, 2003.

[3] Helmut Doleisch, Michael Mayer, Martin Gasser, Roland Wanker, and Helwig Hauser. Case study: Visual analysis of complex, time-dependent simulation results of a diesel exhaust system. In *Proceedings of the Joint Eurographics – IEEE TCVG Symposium on Visualization (VisSym 2004)*, 2004.

[4] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings IEEE Visualization 2001 (Vis'01)*, pages 255–262. IEEE Computer Society Press, 2001.

[5] J. LeBlanc, M. O. Ward, and N. Wittels. Exploring N-dimensional databases. In *Proceedings of IEEE Visualization 1990 (Vis'90)*, pages 230–239. IEEE Computer Society, 1990.

[6] Qing Li and Chris North. Empirical comparison of dynamic query sliders and brushing histograms. In *Proceedings of the IEEE Syposium on Information Visualization (InfoVis'03)*, pages 147–154. IEEE Computer Society Press, 2003.

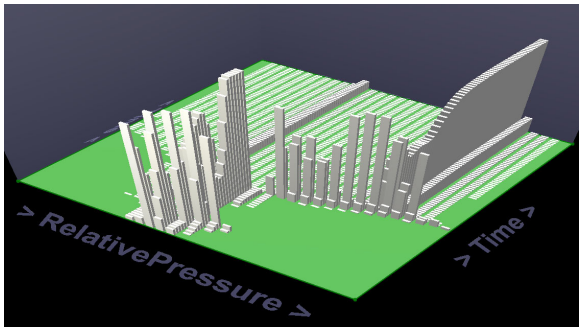
[7] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: Detail and context smoothly integrated. In *Proceedings of ACM CHI '91 Conference on Human Factors in Computing Systems*, pages 173–179, 1991.

[8] Krešimir Matković, Helwig Hauser, Reinhard Sainitzer, and Eduard Gröller. Process visualization with levels of detail. In *IEEE Symposium on Information Visualization 20002 (InfoVis 2002)*, pages 67–70. IEEE Computer Society Press, 2002.

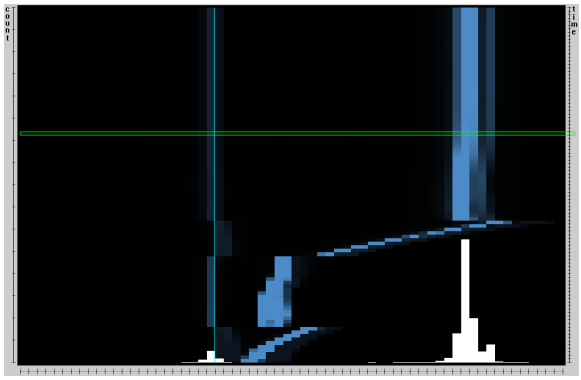
[9] Ramana Rao and Stuart K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proceedings of ACM CHI '94 Conference on Human Factors in Computing Systems*, pages 318–322, 1994. Color plates on pages 481-482.

[10] Lisa Tweedie, Robert Spence, Huw Dawkes, and Hua Su. Externalising abstract mathematical models. In *Proceedings on Human Factors in Computing Systems*, page 406ff. ACM Press, 1996.

[11] Kent Wittenburg, Tom Lanning, Michael Heinrichs, and Michael Stanton. Parallel bargrams for consumer-based information exploration and choice. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 51–60. ACM Press, 2001.

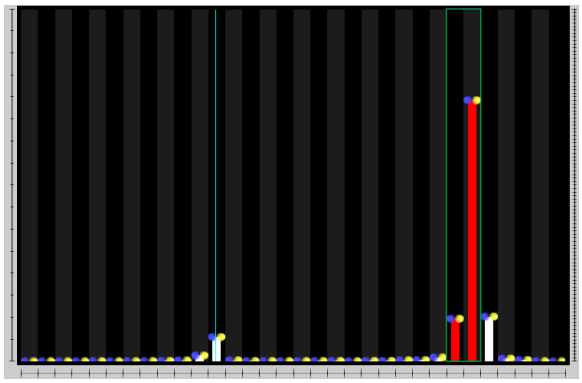


a) TimeHistogram in 3D, showing the development of pressure over time. One can see where the values change a lot and where a stable condition is reached.

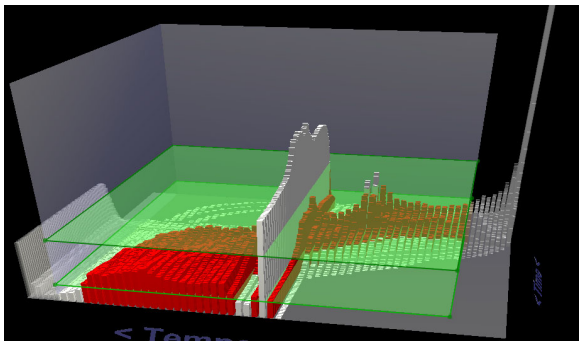


b) 2D time context, showing the same data. This is a projection of the 3D histogram onto the background of the 2D histogram. The green frame marks the current time step on the time axis of the time context display (right).

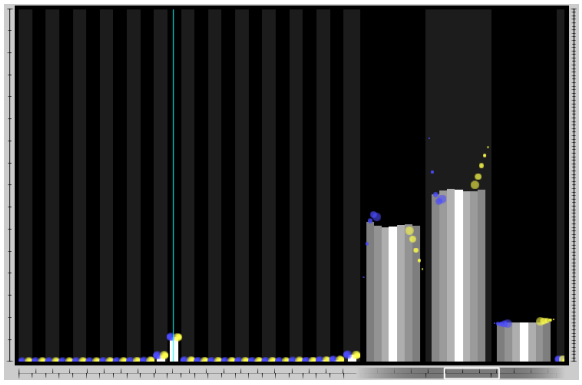
Colorplate 1: TimeHistograms in 3D and 2D time context.



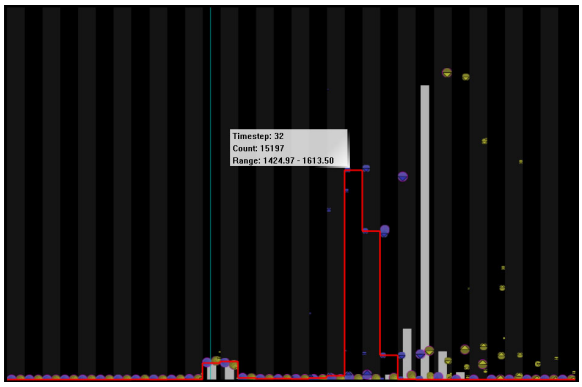
a) Brushing in 2D – the user has selected the values between the two vertical green lines. Brushed bins are now drawn in red. The turquoise line shows where zero is on the value axis.



b) Brushing in 3D – all bins whose counts lie between the two horizontal planes are brushed (displayed in red).



c) Table lens-like Focus+Context (F+C) – parts of the display can be zoomed in to show more information, like more time-steps as a bar chart.



d) Tooltip mode – a tooltip is shown that gives the user information about the bin (current time step, data point count, value range) as well as a second histogram (in red) for the point the mouse is currently pointing at. Differences (right) as well as similarities (left) are visible.

Colorplate 2: Interaction in TimeHistograms.

High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware

Markus Hadwiger Christoph Berger Helwig Hauser *

VRVis Research Center, Austria

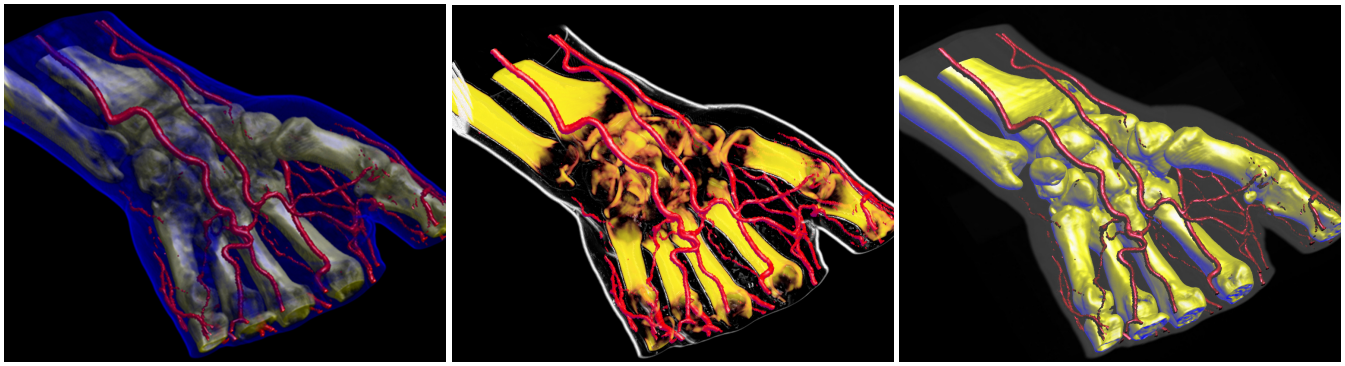


Figure 1: Segmented hand data set (256x128x256) with three objects: skin, blood vessels, and bone. Two-level volume rendering integrates different transfer functions, rendering and compositing modes: (left) all objects rendered with shaded DVR; the skin partially obscures the bone; (center) skin rendered with non-photorealistic contour rendering and MIP compositing, bones rendered with DVR, vessels with tone shading; (right) skin rendered with MIP, bones with tone shading, and vessels with shaded iso-surfacing; the skin merely provides context.

Abstract

One of the most important goals in volume rendering is to be able to visually separate and selectively enable specific objects of interest contained in a single volumetric data set, which can be approached by using explicit segmentation information. We show how segmented data sets can be rendered interactively on current consumer graphics hardware with high image quality and pixel-resolution filtering of object boundaries. In order to enhance object perception, we employ different levels of object distinction. First, each object can be assigned an individual transfer function, multiple of which can be applied in a single rendering pass. Second, different rendering modes such as direct volume rendering, iso-surfacing, and non-photorealistic techniques can be selected for each object. A minimal number of rendering passes is achieved by processing sets of objects that share the same rendering mode in a single pass. Third, local compositing modes such as alpha blending and MIP can be selected for each object in addition to a single global mode, thus enabling high-quality two-level volume rendering on GPUs.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: volume rendering, segmentation, non-photorealistic rendering, consumer graphics hardware

*mailto:{Hadwiger|Berger|Hauser}@VRVis.at, http://www.VRVis.at/vis/

1 Introduction

In many volume rendering methods, all voxels contained in a volumetric data set are treated in an identical manner, i.e., without using any a priori information that specifies object membership on a per-voxel basis. In that case, visual distinction of objects is usually achieved by either using multiple semi-transparent iso-surfaces or, more commonly, with direct volume rendering and an appropriate transfer function. In the latter case, multi-dimensional transfer functions [Kindlmann and Durkin 1998; Kniss et al. 2001] have proven to be especially powerful in facilitating the perception of different objects. In recent years, non-photorealistic volume rendering approaches [Ebert and Rheingans 2000; Csebfalvi et al. 2001; Lu et al. 2002] have also been used successfully for improving the perception of distinct objects embedded in a single volume.

However, it is also often the case that a single rendering method or transfer function does not suffice in order to distinguish multiple objects of interest according to a user's specific needs. A very powerful approach to tackling this problem is to create explicit object membership information via segmentation [Udupa and Herman 1999], which usually yields one binary segmentation mask for each object of interest, or an object ID for each of the volume's voxels.

Unfortunately, integrating segmentation information and multiple rendering modes with different sets of parameters into a fast high-quality volume renderer is not a trivial problem, especially in the case of consumer hardware volume rendering, which tends to only be fast when all or most voxels can be treated identically. On such hardware, one would also like to use a single segmentation mask volume in order to use a minimal amount of texture memory. Graphics hardware cannot easily interpolate between voxels belonging to different objects, however, and using the segmentation mask without filtering gives rise to artifacts. Thus, one of the major obstacles in such a scenario is filtering object boundaries in order to attain high quality in conjunction with consistent fragment assignment and without introducing non-existent object IDs.

In this paper, we show how segmented volumetric data sets can be rendered efficiently and with high quality on current consumer graphics hardware. The segmentation information for object dis-

tinction can be used at multiple levels of sophistication, and we describe how all of these different possibilities can be integrated into a single coherent hardware volume rendering framework.

First, different objects can be rendered with the same rendering technique (e.g., DVR), but with different transfer functions. Separate per-object transfer functions can be applied in a single rendering pass even when object boundaries are filtered during rendering. On an ATI Radeon 9700, up to eight transfer functions can be folded into a single rendering pass with linear boundary filtering. If boundaries are only point-sampled, e.g., during interaction, an arbitrary number of transfer functions can be used in a single pass. However, the number of transfer functions with boundary filtering in a single pass is no conceptual limitation and increases trivially on architectures that allow more instructions in the fragment shader.

Second, different objects can be rendered using different hardware fragment shaders. This allows easy integration of methods as diverse as non-photorealistic and direct volume rendering, for instance. Although each distinct fragment shader requires a separate rendering pass, multiple objects using the same fragment shader with different rendering parameters can effectively be combined into a single pass. When multiple passes cannot be avoided, the cost of individual passes is reduced drastically by executing expensive fragment shaders only for those fragments active in a given pass. These two properties allow highly interactive rendering of segmented data sets, since even for data sets with many objects usually only a couple of different rendering modes are employed. We have implemented direct volume rendering with post-classification, pre-integrated classification [Engel et al. 2001], different shading modes, non-polygonal iso-surfaces, and maximum intensity projection. See figures 1 and 2 for example images. In addition to non-photorealistic contour enhancement [Csebfalvi et al. 2001] (figure 1, center; figure 2, skull), we have also used a volumetric adaptation of tone shading [Gooch et al. 1998] (figure 1, right), which improves depth perception in contrast to standard shading.

Finally, different objects can also be rendered with different compositing modes, e.g., alpha blending and maximum intensity projection (MIP), for their contribution to a given pixel. These per-object compositing modes are object-local and can be specified independently for each object. The individual contributions of different objects to a single pixel can be combined via a separate global compositing mode. This two-level approach to object compositing [Hauser et al. 2001] has proven to be very useful in order to improve perception of individual objects.

In summary, the major novel contributions of this paper are:

- A systematic approach to minimizing both the number of rendering passes and the performance cost of individual passes when rendering segmented volume data with high quality on current GPUs. Both filtering of object boundaries and the use of different rendering parameters such as transfer functions do not prevent using a single rendering pass for multiple objects. Even so, each pass avoids execution of the corresponding potentially expensive fragment shader for irrelevant fragments by exploiting the early z-test. This reduces the performance impact of the number of rendering passes drastically.

- An efficient method for mapping a single object ID volume to and from a domain where filtering produces correct results even when three or more objects are present in the volume. The method is based on simple 1D texture lookups and able to map and filter blocks of four objects simultaneously.

- An efficient object-order algorithm based on simple depth and stencil buffer operations that achieves correct compositing of objects with different per-object compositing modes and an additional global compositing mode. The result is conceptually identical to being able to switch compositing modes for any given group of samples along the ray for any given pixel.

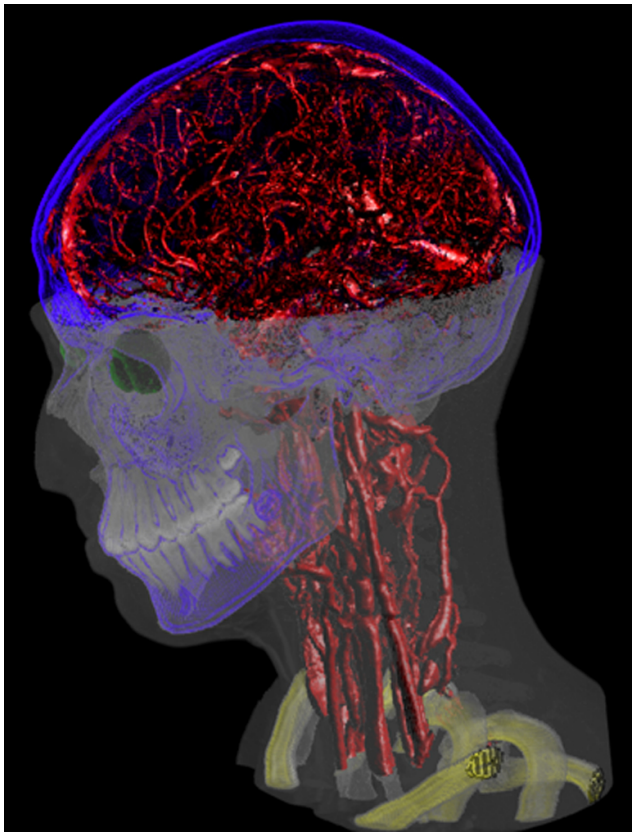


Figure 2: Segmented head and neck data set (256x256x333) with six different enabled objects. The skin and teeth are rendered as MIP with different intensity ramps, the blood vessels and eyes are rendered as shaded DVR, the skull uses contour rendering, and the vertebrae use a gradient magnitude-weighted transfer function with shaded DVR. A clipping plane has been applied to the skin object.

Related work

The framework presented in this paper is based on re-sampling and rendering a volume via a stack of textured slices that are blended on top of each other in either back-to-front or front-to-back order. For this purpose, either view-aligned slices through 3D textures [Cullip and Neumann 1993; Cabral et al. 1994; Westermann and Ertl 1998], or object-aligned slices with 2D textures can be used. In the latter case, intermediate slices can also be interpolated on-the-fly during rendering in order to attain tri-linear interpolation [Rezk-Salama et al. 2000]. The number of slices necessary for high-quality results can be reduced drastically by considering two adjacent slices as constituting a single slab and compensating for non-linear transfer function changes within each slab via a lookup into a pre-integrated transfer function table [Engel et al. 2001]. Multi-dimensional transfer functions [Kindlmann and Durkin 1998] are a very important tool for distinguishing different objects contained in a volume, especially when combined with an intuitive and interactive interface for specifying them [Kniss et al. 2001]. In recent years, there also has been a remarkable interest in non-photorealistic rendering techniques such as tone shading [Gooch et al. 1998] that are increasingly being applied to volumes [Ebert and Rheingans 2000; Csebfalvi et al. 2001; Lu et al. 2002]. If no segmentation information is present, multiple rendering passes with one transfer function each and non-photorealistic shading can be used in order to enhance perception of individual objects [Lum and Ma 2002].

The idea of using two conceptual levels for compositing volumetric objects has first been described in the context of a fast soft-

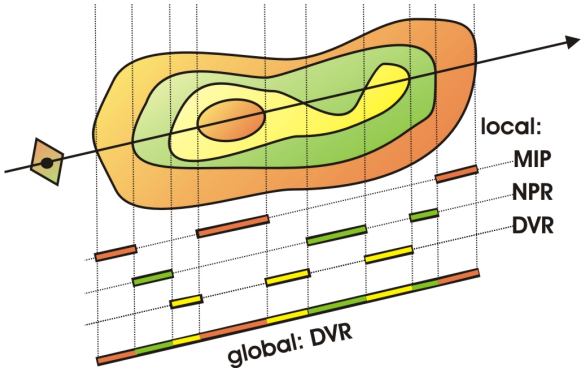


Figure 3: A single ray corresponding to a given image pixel is allowed to pierce objects that use their own object-local compositing mode. The contributions of different objects along a ray are combined with a single global compositing mode. Rendering a segmented data set with these two conceptual levels of compositing (local and global) is known as *two-level volume rendering* .

ware two-level volume renderer [Hauser et al. 2001], and we refer to this earlier work for detailed suggestions on when and how different rendering and compositing modes together with appropriately specified transfer functions can facilitate object perception.

The topic of image and volume segmentation is a huge area on its own [Udupa and Herman 1999], and we simply treat the segmentation information as additional a priori input data that are already available for a given data set. In order to achieve high rendering quality, it is necessary to distinguish individual objects with subvoxel precision [Tiede et al. 1998], i.e., what we refer to as pixel-resolution boundary filtering. Even linear filtering of segmentation data is not directly possible on graphics hardware when more than two objects have been segmented, since object IDs cannot be interpolated directly. Ultimately, rendering segmented data sets can be viewed as being composed of multiple individual volumetric clipping problems. Recent work has shown how to achieve high-quality clipping in graphics hardware [Weiskopf et al. 2003], which can also be combined with pre-integrated classification by adjusting the lookup into the pre-integration table accordingly [Röttger et al. 2003]. However, it is not trivial to apply clipping approaches to the rendering of segmented data as soon as the volume contains more than two objects and high quality results and a minimal number of rendering passes are desired. Excluding individual fragments from processing by an expensive fragment shader via the early z-test is also crucial in the context of GPU-based ray casting in order to be able to terminate rays individually [Krüger and Westermann 2003].

2 Rendering segmented data sets

For rendering purposes, we simply assume that in addition to the usual data such as a density and an optional gradient volume, a *segmentation mask volume* is also available. If embedded objects are represented as separate masks, we combine all of these masks into a single volume that contains a single object ID for each voxel. Hence we will also be calling this segmentation mask volume the *object ID volume*. IDs are simply enumerated consecutively starting with one, i.e., we do not assign individual bits to specific objects. ID zero is reserved (see later sections). The object ID volume consumes one byte per voxel and is either stored in its own 3D texture in the case of view-aligned slicing, or in additional 2D slice textures for all three slice stacks in the case of object-aligned slicing. With respect to resolution, we have used the same resolution as the original volume data, but all of the approaches we describe could easily be used for volume and segmentation data of different resolutions.

In order to render a segmented data set, we determine object membership of individual fragments by filtering object boundaries

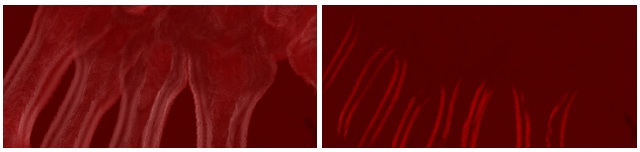


Figure 4: Detecting changes in compositing mode for each individual sample along a ray can be done exactly using two rendering buffers (left), or approximately using only a single buffer (right).

in the hardware fragment shader (section 3). Object membership determines which transfer function, rendering, and compositing modes should be used for a given fragment. We render the volume in a number of rendering passes that is basically independent of the number of contained objects. It most of all depends on the required number of different hardware configurations that cannot be changed during a single pass, i.e., the fragment shader and compositing mode. Objects that can share a given configuration can be rendered in a single pass. This also extends to the application of multiple per-object transfer functions (section 4) and thus the actual number of rendering passes is usually much lower than the number of objects or transfer functions. It depends on several major factors:

Enabled objects. If all the objects rendered in a given pass have been disabled by the user, the entire rendering pass can be skipped. If only some of the objects are disabled, the number of passes stays the same, independent of the order of object IDs. Objects are disabled by changing a single entry of a 1D lookup texture. Additionally, per-object clipping planes can be enabled. In this case, all objects rendered in the same pass are clipped identically, however.

Rendering modes. The rendering mode, implemented as an actual hardware fragment shader, determines what and how volume data is re-sampled and shaded. Since it cannot be changed during a single rendering pass, another pass must be used if a different fragment shader is required. However, many objects often use the same basic rendering mode and thus fragment shader, e.g., DVR and iso-surfacing are usually used for a large number of objects.

Transfer functions. Much more often than the basic rendering mode, a change of the transfer function is required. For instance, all objects rendered with DVR usually have their own individual transfer functions. In order to avoid an excessive number of rendering passes due to simple transfer function changes, we apply multiple transfer functions to different objects in a single rendering pass while still retaining adequate filtering quality (section 4).

Compositing modes. Although usually considered a part of the rendering mode, compositing is a totally separate operation in graphics hardware. Where the basic rendering mode is determined by the fragment shader, the compositing mode is specified as blend function and equation in OpenGL, for instance. Changing the compositing mode happens even more infrequently than changing the basic rendering mode, e.g., alpha blending is used in conjunction with both DVR and tone shading.

Different compositing modes per object also imply that the (conceptual) ray corresponding to a single pixel must be able to combine the contribution of these different modes (figure 3). Especially in the context of texture-based hardware volume rendering, where no actual rays exist and we want to obtain the same result with an object-order approach instead, we have to use special care when compositing. In order to ensure correct compositing, we are using two render buffers and track the current compositing mode for each pixel. Whenever the compositing mode changes for a given pixel, the already composited part is transferred from the *local compositing buffer* into the *global compositing buffer*. Section 5 shows that this can actually be done very efficiently without explicitly considering individual pixels, while still achieving the same compositing behavior as a ray-oriented image-order approach, which is crucial for achieving high quality. For faster rendering we allow falling back to single-buffer compositing during interaction (figure 4).

2.1 Basic rendering loop

We will now outline the basic rendering loop that we are using for each frame. Table 1 gives a high-level overview.

Although the user is dealing with individual objects, we automatically collect all objects that can be processed in the same rendering pass into an *object set* at the beginning of each frame. For each object set, we generate an *object set membership texture*, which is a 1D lookup table that determines the objects belonging to the set. In order to further distinguish different transfer functions in a single object set, we also generate 1D *transfer function assignment textures*. Both of these types of textures are shown in figure 5 and described in sections 2.3, 3, and 4. After this setup, the entire slice stack is rendered. Each slice must be rendered for every object set containing an object that intersects the slice, which is determined in a pre-process. If there is more than a single object set for the current slice, we optionally render all object set IDs of the slice into the depth buffer before rendering any actual slice data. This enables us to exploit the early z-test during all subsequent passes for each object set, see below. For performance reasons, we never use object ID filtering in this pass, which allows only conservative fragment culling via the depth test. Exact fragment rejection is done in the fragment shader. Before a slice can be rendered for any object set, the fragment shader and compositing mode corresponding to this set must be activated. Using the two types of textures mentioned above, the fragment shader filters boundaries, rejects fragments not corresponding to the current pass, and applies the correct transfer function. In order to attain two compositing levels, slices are rendered into a local buffer, as already outlined above. Before rendering the current slice, those pixels where the local compositing mode differs from the previous slice are transferred from the local into the global buffer using the global compositing mode. After this transfer, the transferred pixels are cleared in the local buffer to ensure correct local compositing for subsequent pixels. In the case when only a single compositing buffer is used for approximate compositing, the local to global buffer transfer and clear are not executed.

2.2 Conservative fragment culling via early z-test

On current graphics hardware, it is possible to avoid execution of the fragment shader for fragments where the depth test fails as long as the shader does not modify the depth value of the fragment. This early z-test is crucial to improving performance when multiple rendering passes have to be performed for each slice. If the current slice's object set IDs have been written into the depth buffer before, see above, we conservatively reject fragments not belonging to the current object set even before the corresponding fragment shader is started. In order to do this, we use a depth test of `GL_EQUAL` and configure the vertex shader to generate a constant depth value for each fragment that exactly matches the current object set ID.

```
DetermineObjectSets();
CreateObjectSetMembershipTextures();
CreateTFAssignmentTextures();
FOR each slice DO
  TransferLocalBufferIntoGlobalBuffer();
  ClearTransferredPixelsInLocalBuffer();
  RenderObjectIdDepthImageForEarlyZTest();
  FOR each object set with an object in slice DO
    SetupObjectSetFragmentRejection();
    SetupObjectSetTFAssignment();
    ActivateObjectSetFragmentShader();
    ActivateObjectSetCompositingMode();
    RenderSliceIntoLocalBuffer();
```

Table 1: The basic rendering loop that we are using. Object set membership can change every time an object's rendering or compositing mode is changed, or an object is enabled or disabled.

2.3 Fragment shader operations

Most of the work in volume renderers for consumer graphics hardware is done in the fragment shader, i.e., at the granularity of individual fragments and, ultimately, pixels. In contrast to approaches using lookup tables, i.e., paletted textures, we are performing all shading operations procedurally in the fragment shader. Section 6 contains details about the actual volume shading models we are using. However, we are most of all interested in the operations that are required for rendering segmented data. The two basic operations in the fragment shader with respect to the segmentation mask are fragment rejection and per-fragment application of transfer functions:

Fragment rejection. Fragments corresponding to object IDs that cannot be rendered in the current rendering pass, e.g., because they need a different fragment shader or compositing mode, have to be rejected. They, in turn, will be rendered in another pass, which uses an appropriately adjusted rejection comparison. For fragment rejection, we do not compare object IDs individually, but use 1D lookup textures that contain a binary membership status for each object (figure 5, left). All objects that can be rendered in the same pass belong to the same object set, and the corresponding object set membership texture contains ones at exactly those texture coordinates corresponding to the IDs of these objects, and zeros everywhere else. The re-generation of these textures at the beginning of each frame, which is negligible in terms of performance, also makes turning individual objects on and off trivial. Exactly one object set membership texture is active for a given rendering pass and makes the task of fragment rejection trivial if the object ID volume is point-sampled. When object IDs are filtered, it is also crucial to map individual IDs to zero or one before actually filtering them. Details are given in section 3, but basically we are using object set membership textures to do a binary classification of input IDs to the filter, and interpolate after this mapping. The result can then be mapped back to zero or one for fragment rejection.

Per-fragment transfer function application. Since we apply different transfer functions to multiple objects in a single rendering pass, the transfer function must be applied to individual fragments based on their density value and corresponding object ID. Instead of sampling multiple one-dimensional transfer function textures, we sample a single global two-dimensional transfer function texture (figure 6). This texture is not only shared between all objects of an object set, but also between all object sets. It is indexed with one texture coordinate corresponding to the object ID, the other one to the actual density. Because we would like to filter linearly along the axis of the actual transfer function, but use point-sampling along the axis of object IDs, we store each transfer function twice at adjacent locations in order to guarantee point-sampling for IDs, while we are using linear interpolation for the entire texture. We have applied

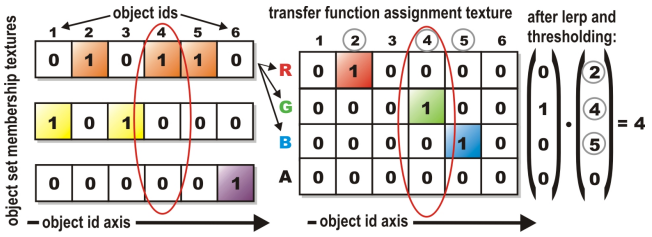


Figure 5: Object set membership textures (left; three 1D intensity textures for three sets containing three, two, and one object, respectively) contain a binary membership status for each object in a set that can be used for filtering object IDs and culling fragments. Transfer function assignment textures (right; one 1D RGBA texture for distinction of four transfer functions) are used to filter four object boundaries simultaneously and determine the corresponding transfer function via a simple dot product.

this scheme only to 1D transfer functions, but general 2D transfer functions could also be implemented via 3D textures of just a few layers in depth, i.e., the number of different transfer functions.

We are using an extended version of the pixel-resolution filter that we employ for fragment rejection in order to determine which of multiple transfer functions in the same rendering pass a fragment should actually use. Basically, the fragment shader uses multiple RGBA transfer function assignment textures (figure 5, right) for both determining the transfer function and rejecting fragments, instead of a single object set membership texture with only a single color channel. Each one of these textures allows filtering the object ID volume with respect to four object boundaries simultaneously. A single lookup yields binary membership classification of a fragment with respect to four objects. The resulting RGBA membership vectors can then be interpolated directly. The main operation for mapping back the result to an object ID is a simple dot product with a constant vector of object IDs. If the result is the non-existent object ID of zero, the fragment needs to be rejected. The details are described in section 4. This concept can be extended trivially to objects sharing transfer functions by using transfer function IDs instead of object IDs. The following two sections will now describe filtering of object boundaries at sub-voxel precision in more detail.

3 Pixel-resolution boundaries

One of the most crucial parts of rendering segmented volumes with high quality is that the object boundaries must be calculated during rendering at the pixel resolution of the output image, instead of the voxel resolution of the segmentation volume. Figure 7 (left) shows that simply point-sampling the object ID texture leads to object boundaries that are easily discernible as individual voxels. That is, simply retrieving the object ID for a given fragment from the segmentation volume is trivial, but causes artifacts. Instead, the object ID must be determined via filtering for each fragment individually, thus achieving pixel-resolution boundaries.

Unfortunately, filtering of object boundaries cannot be done directly using the hardware-native linear interpolation, since direct interpolation of numerical object IDs leads to incorrectly interpolated intermediate values when more than two different objects are present. When filtering object IDs, a threshold value s_t must be chosen that determines which object a given fragment belongs to, which is essentially an iso-surfacing problem. However, this cannot be done if three or more objects are contained in the volume, which is illustrated in the top row of figure 8. In that case, it is not possible to choose a single s_t for the entire volume. The crucial observation to make in order to solve this problem is that the segmentation volume must be filtered as a successive series of binary volumes in order to achieve proper filtering [Tiede et al. 1998], which is shown in the second row of figure 8. Mapping all object IDs of the current object set to 1.0 and all other IDs to 0.0 allows using a global threshold value s_t of 0.5. We of course do not want to store these binary volumes explicitly, but perform this mapping on-the-fly in the fragment shader by indexing the *object set membership texture* that is active in the current rendering pass. Filtering

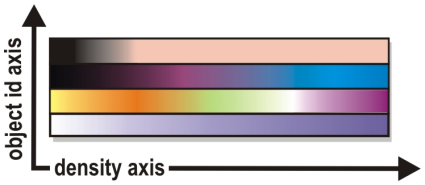


Figure 6: Instead of multiple one-dimensional transfer functions for different objects, we are using a single global two-dimensional transfer function texture. After determining the object ID for the current fragment via filtering, the fragment shader appropriately samples this texture with $(density, object\_id)$ texture coordinates.

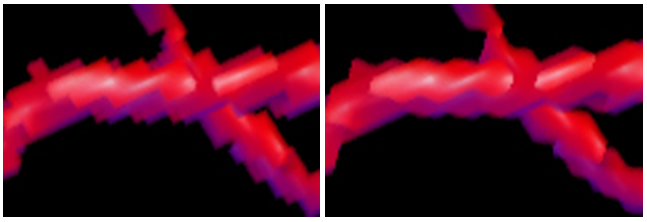


Figure 7: Object boundaries with voxel resolution (left) vs. object boundaries determined per-fragment with linear filtering (right).

in the other passes simply uses an alternate binary mapping, i.e., other object set membership textures. One problem with respect to a hardware implementation of this approach is that texture filtering happens before the sampled values can be altered in the fragment shader. Therefore, we perform filtering of object IDs directly in the fragment shader. Note that our approach could in part also be implemented using texture palettes and hardware-native linear interpolation, with the restriction that not more than four transfer functions can be applied in a single rendering pass (section 4). However, we have chosen to perform all filtering in the fragment shader in order to create a coherent framework with a potentially unlimited number of transfer functions in a single rendering pass and prepare for the possible use of cubic boundary filtering in the future.

After filtering yields values in the range $[0.0, 1.0]$, we once again come to a binary decision whether a given fragment belongs to the current object set by comparing with a threshold value of 0.5 and rejecting fragments with an interpolated value below this threshold (figure 8, third row). Actual rejection of fragments is done using the `KIL` instruction of the hardware fragment shader.

Linear boundary filtering. For object-aligned volume slices, bi-linear interpolation is done by setting the hardware filtering mode for the object ID texture to nearest-neighbor and sampling it four times with offsets of whole texels in order to get access to the four ID values needed for interpolation. Before actual interpolation takes place, the four object IDs are individually mapped to 0.0 or 1.0, respectively, using the current object set membership texture. We perform the actual interpolation using a variant of texture-based filtering [Hadwiger et al. 2001], which proved to be both faster and use fewer instructions than using `LRP` instructions. With this approach, bi-linear weight calculation and interpolation can be reduced to just one texture fetch and one dot product. When intermediate slices are interpolated on-the-fly [Rezk-Salama et al. 2000], or view-aligned slices are used, eight instead of four input IDs have to be used in order to perform tri-linear interpolation.

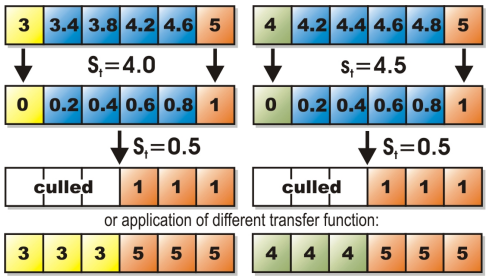


Figure 8: Each fragment must be assigned an exactly defined object ID after filtering. Here, IDs 3, 4, and 5 are interpolated, yielding the values shown in blue. Top row: choosing a single threshold value s_t that works everywhere is not possible for three or more objects. Second row: object IDs must be converted to 0.0 or 1.0 in the fragment shader before interpolation, which allows using a global s_t of 0.5. After thresholding, fragments can be culled accordingly (third row; see section 3), or mapped back to an object ID in order to apply the corresponding transfer function (fourth row; see section 4).

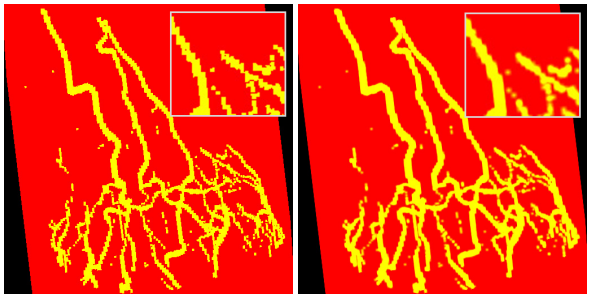


Figure 9: Selecting the transfer function on a per-fragment basis. In the left image, point-sampling of the object ID volume has been used, whereas in the right image procedural linear interpolation in the fragment shader achieves results of much better quality.

Combination with pre-integration. The combination of pre-integration [Engel et al. 2001] and high-quality clipping has been described recently [Röttger et al. 2003]. Since our filtering method effectively reduces the segmentation problem to a clipping problem on-the-fly, we are using the same approach after we have mapped object IDs to 0.0 or 1.0, respectively. In this case, the interpolated binary values must be used for adjusting the pre-integration lookup.

4 Multiple per-object transfer functions in a single rendering pass

In addition to simply determining whether a given fragment belongs to a currently active object or not, which has been described in the previous section, this filtering approach can be extended to the application of multiple transfer functions in a single rendering pass without sacrificing filtering quality. Figure 9 shows the difference in quality for two objects with different transfer functions (one entirely red, the other entirely yellow for illustration purposes).

In this case, we perform several almost identical filtering steps in the fragment shader, where each of these steps simultaneously filters the object boundaries of four different objects. After the fragment's object ID has been determined via filtering, it can be used to access the global transfer function table as described in section 2.3 and illustrated in figure 6. For multiple simultaneous transfer functions, we do not use object set membership textures but the similar extended concept of *transfer function assignment textures*, which is illustrated in the right image of figure 5. Each of these textures can be used for filtering the object ID volume with respect to four different object IDs at the same time by using the four channels of an RGBA texture in order to perform four simultaneous binary classification operations. In order to create these textures, each object set membership texture is converted into $\lceil \#objects/4 \rceil$ transfer function assignment textures, where $\#objects$ denotes the number of objects with different transfer functions in a given object set. All values of 1.0 corresponding to the first transfer function are stored into the red channel of this texture, those corresponding to the second transfer function into the green channel, and so on.

In the fragment shader, bi-linear interpolation must index this texture at four different locations given by the object IDs of the four input values to interpolate. This classifies the four input object IDs with respect to four objects with just four 1D texture sampling operations. A single linear interpolation step yields the linear interpolation of these four object classifications, which can then be compared against a threshold of $(0.5, 0.5, 0.5, 0.5)$, also requiring only a single operation for four objects. Interpolation and thresholding yields a vector with at most one component of 1.0, the other components set to 0.0. In order for this to be true, we require that interpolated and thresholded repeated binary classifications never overlap, which is not guaranteed for all types of filter kernels. In the case of bi-linear or tri-linear interpolation, however, overlaps can never occur [Tiede et al. 1998]. The final step that has to be performed is mapping the binary classification to the desired object ID. We do this via a single dot product with a vector containing the four object IDs corresponding to the four channels of the transfer function assignment texture (figure 5, right). By calculating this dot product, we multiply exactly the object ID that should be assigned to the final fragment by 1.0. The other object IDs are multiplied by 0.0 and thus do not change the result. If the result of the dot product is 0.0, the fragment does not belong to any of the objects under consideration and can be culled. Note that exactly for this reason, we do not use object IDs of zero. For the application of more than four transfer functions in a single rendering pass, the steps outlined above can be executed multiple times in the fragment shader. The results of the individual dot products are simply summed up, once again yielding the ID of the object that the current fragment belongs to. Note that the calculation of filter weights is only required once, irrespective of the number of simultaneous transfer functions, which is also true for sampling the original object ID textures.

Equation 1 gives the major fragment shader resource requirements of our filtering and binary classification approach for the case of bi-linear interpolation with `LRP` instructions:

$$4\textbf{TEX\_2D} + 4\left\lceil \frac{\#objects}{4} \right\rceil \textbf{TEX\_1D} + 3\left\lceil \frac{\#objects}{4} \right\rceil \textbf{LRP}, \quad (1)$$

in addition to one dot product and one thresholding operation (e.g., `DP4` and `SGE` instructions, respectively) for every $\lceil \#objects/4 \rceil$ transfer functions evaluated in a single pass. Similarly to the alternative linear interpolation using texture-based filtering that we have outlined in section 3, procedural weight calculation and the `LRP` instructions can once again also be substituted by texture fetches and a few cheaper ALU instructions. On the Radeon 9700, we are currently able to combine high-quality shading with up to eight transfer functions in the same fragment shader, i.e., we are using up to two transfer function assignment textures in a single rendering pass.

5 Separation of compositing modes

The final component of our framework with respect to the separation of different objects is the possibility to use individual object-local compositing modes, as well as a single global compositing mode. The local compositing modes that can currently be selected are alpha blending (e.g., for DVR or tone shading), maximum intensity projection (e.g., for MIP or contour enhancement), and iso-surface rendering. Global compositing can either be done by alpha blending, MIP, or a simple add of all contributions.

Although the basic concept is best explained using an image-order approach, i.e., individual rays (figure 3), in the context of texture-based volume rendering we have to implement it in object-order. As described in section 2, we are using two separate rendering buffers, a local and a global compositing buffer, respectively. Actual volume slices are only rendered into the local buffer, using

```
TransferLocalBufferIntoGlobalBuffer() {
  ActivateContextGlobalBuffer();
  DepthTest( NOT_EQUAL );
  StencilTest( RENDER_ALWAYS, SET_ONE );
  RenderSliceCompositingIds( DEPTH_BUFFER );
  DepthTest( DISABLE );
  StencilTest( RENDER_WHERE_ONE, SET_ZERO );
  RenderLocalBufferImage( COLOR_BUFFER );
}
```

Table 2: Detecting for all pixels simultaneously where the compositing mode changes from one slice to the next, and transferring those pixels from the local into the global compositing buffer.

the appropriate local compositing mode. When a new fragment has a different local compositing mode than the pixel that is currently stored in the local buffer, that pixel has to be transferred into the global buffer using the global compositing mode. Afterward, these transferred pixels have to be cleared in the local buffer before the corresponding new fragment is rendered. Naturally, it is important that both the detection of a change in compositing mode and the transfer and clear of pixels is done for all pixels simultaneously.

In order to do this, we are using the depth buffer of both the local and the global compositing buffer to track the current local compositing mode of each pixel, and the stencil buffer to selectively enable pixels where the mode changes from one slice to the next. Before actually rendering a slice (see table 1), we render IDs corresponding to the local compositing mode into both the local and the global buffer's depth buffer. During these passes, the stencil buffer is set to one where the ID already stored in the depth buffer (from previous passes) differs from the ID that is currently being rendered. This gives us both an updated ID image in the depth buffer, and a stencil buffer that identifies exactly those pixels where a change in compositing mode has been detected. We then render the image of the local buffer into the global buffer. Due to the stencil test, pixels will only be rendered where the compositing mode has actually changed. Table 2 gives pseudo code for what is happening in the global buffer. Clearing the just transferred pixels in the local buffer works almost identically. The only difference is that in this case we do not render the image of another buffer, but simply a quad with all pixels set to zero. Due to the stencil test, pixels will only be cleared where the compositing mode has actually changed.

Note that all these additional rendering passes are much faster than the passes actually rendering and shading volume slices. They are independent of the number of objects and use extremely simple fragment shaders. However, the buffer/context switching overhead is quite noticeable, and thus correct separation of compositing modes can be turned off during interaction. Figure 4 shows a comparison between approximate and correct compositing with one and two compositing buffers, respectively. Performance numbers can be found in table 3. When only a single buffer is used, the compositing mode is simply switched according to each new fragment without avoiding interference with the previous contents of the frame buffer. The visual difference depends highly on the combination of compositing modes and spatial locations of objects. The example in figure 4 uses MIP and DVR compositing in order to highlight the potential differences. However, using approximate compositing is very useful for faster rendering, and often exhibits little or no loss in quality. Also, it is possible to get an almost seamless performance/quality trade-off between the two, by performing the buffer transfer only every n slices instead of every slice.

6 Rendering modes, performance

This section provides details on the actual rendering modes we are supporting, as well as some performance figures. We can use object-aligned slices with 2D textures, possibly with slice interpolation [Rezk-Salama et al. 2000], view-aligned slices with 3D textures, and slab instead of slice rendering for pre-integration [Engel et al. 2001]. Gradients can be pre-computed either via central differencing or a 3x3x3 Sobel operator, and are stored into a RGB texture in normalized form for sampling by the fragment shader.

Direct volume rendering. We have implemented both post-classification and pre-integrated classification [Engel et al. 2001]. Both of these modes can either be unshaded or shaded, optionally weighted with gradient magnitude [Levoy 1988].

Iso-surfacing. We support rendering of non-polygonal shaded iso-surfaces via the OpenGL alpha test [Westermann and Ertl 1998] and pre-integrated iso-surfaces [Engel et al. 2001], respectively.

Maximum intensity projection. The maximum intensity of all fragments corresponding to a given pixel can be retained in the frame buffer by using the `GL_MAX` compositing mode. Prior to this, the volume density is mapped through a monochrome transfer function and afterward multiplied by a constant color.

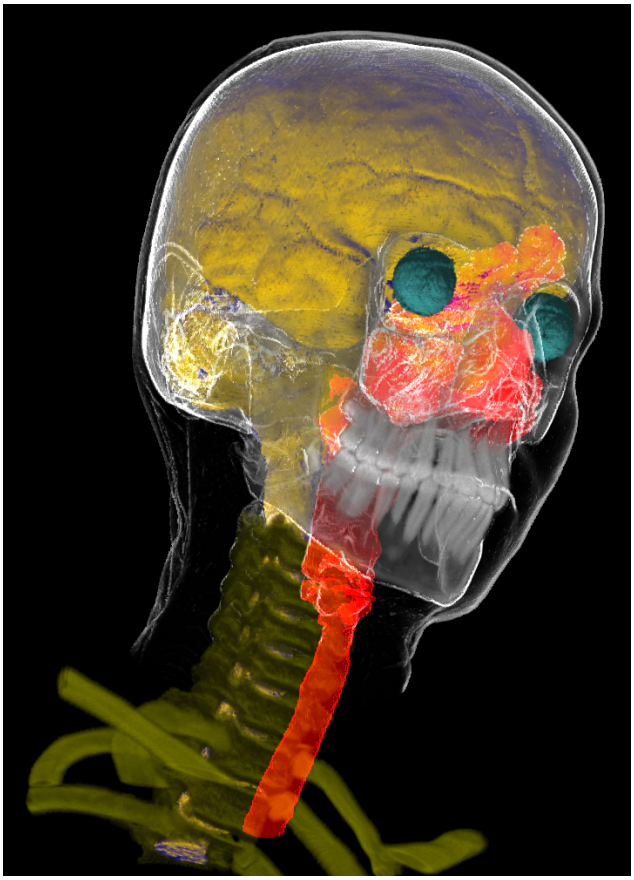


Figure 10: Segmented head and neck data set (256x256x333) with eight different enabled objects – brain: tone shading; skin: contour enhancement with clipping plane; eyes and spine: shaded DVR; skull, teeth, and vertebrae: unshaded DVR; trachea: MIP.

Contour enhancement. As one of two non-photorealistic rendering modes, we have adopted a contour shading model [Csebfalvi et al. 2001]. The intensity of a fragment is determined procedurally in the fragment shader by evaluating equation 2:

$$I = g\big(|\nabla|\big) \cdot \big(1 - |V \cdot \nabla|\big)^8, \tag{2}$$

where V is the viewing vector, ∇ denotes the gradient of a given voxel, and $g()$ is a windowing function for the gradient magnitude. We specify $g()$ through the usual transfer function interface, where the alpha component is the weighting factor for the view-dependent part, and the RGB components are simply neglected. The fragment intensity can be multiplied by a constant contour color; fragment alpha is set equal to I. The compositing mode for contours is MIP.

Tone shading. In order to enhance depth perception, we are also using tone shading [Gooch et al. 1998] adapted to volumes:

$$I = \left(\frac{1 + L \cdot \nabla}{2}\right) k_a + \left(1 - \frac{1 + L \cdot \nabla}{2}\right) k_b, \tag{3}$$

where L denotes the light vector. The two colors to interpolate, k_a and k_b, are derived from two constant colors k_{cool} and k_{warm} and the color from the transfer function k_t, using two user-specified factors α and β that determine the additive contribution of k_t:

$$k_a = k_{cool} + \alpha k_t \tag{4}$$

$$k_b = k_{warm} + \beta k_t \tag{5}$$

#slices	#obj	composit.	single	multi+ztest	*multi*
128	3	one buff.	48 (16.2)	29.2 (15.4)	*19.3 (6.8)*
128	3	two buff.	7 (3.9)	6.2 (3.2)	*5 (1.9)*
128	8	one buff.	48 (11.3)	15.5 (10)	*7 (2.1)*
128	8	two buff.	7 (3.2)	5.4 (3)	*2.5 (0.7)*
256	3	one buff.	29 (9.1)	15.6 (8.2)	*11 (3.4)*
256	3	two buff.	3.5 (2)	3.2 (1.8)	*2.5 (1.1)*
256	8	one buff.	29 (5.3)	8.2 (5.2)	*3.7 (1.1)*
256	8	two buff.	3.5 (1.7)	3.1 (1.6)	*1.2 (0.4)*

Table 3: Performance on ATI Radeon 9700; 512x512 viewport; 256x128x256 data set; three and eight enabled objects, respectively. Numbers are in frames per second. Compositing is done with either one or two buffers, respectively. The *multi* column with early z-testing turned off is only shown for comparison purposes.

Performance. Actual rendering performance depends on a lot of different factors, so table 3 shows only some example figures. In order to concentrate on performance of rendering segmented data, all rates have been measured with unshaded DVR. Slices were object-aligned; objects were rendered all in a single pass (*single*) or in one pass per object (*multi+ztest*). Compositing performance is independent of the rendering mode, i.e., can also be measured with DVR for all objects. Frame rates in parentheses are with linear boundary filtering enabled, other rates are for point-sampling during interaction. Note that in the unfiltered case with a single rendering pass for all objects, the performance is independent of the number of objects. If more complex fragment shaders than unshaded DVR are used, the relative performance speed-up of *multi+ztest* versus *multi* increases further toward *single* performance, i.e., the additional overhead of writing object set IDs into the depth buffer becomes negligible.

7 Conclusions and future work

We have shown how segmented volumes can be rendered at interactive rates with high quality on current consumer graphics hardware such as the ATI Radeon 9700. The segmentation mask is filtered on-the-fly in the fragment shader, which provides greater flexibility and facilitates using higher-order filtering in the future. In general, we are expecting a move toward programmable filtering via procedural or texture-based filter kernels for a lot of applications in the near future. This has just become possible on the most recent architectures, but as instruction count limits in the fragment shader rise or are even removed, these approaches are rapidly becoming feasible. All the algorithms we have presented are meant to take the possibilities of future hardware into account. The current restriction to eight simultaneous transfer functions with object ID filtering is solely due to the instruction count limit of our target hardware, and increases trivially with more instructions. In general, we ensure a minimal number of rendering passes by only falling back to individual passes for changes of the hardware configuration where an on-the-fly adaptation is currently impossible, i.e., the fragment shader and compositing mode. When fragment shader adaptation on a per-pixel basis becomes possible, our framework will require only minor changes. In the future, we would like to incorporate cubic boundary filtering [Hadwiger et al. 2001], and extend the combination with pre-integrated classification [Röttger et al. 2003] for application of multiple pre-integrated transfer functions in a single rendering pass. In addition or as an alternative to the early z-test, texture hulls [Li and Kaufman 2002] could be used to minimize the number of fragments that are rendered without any contribution.

Acknowledgments

We would like to thank Christof Rezk-Salama, Thomas Theußl, Klaus Engel, Matej Mlejnek, André Neubauer, Lukas Mroz, Ivan Viola, Torsten Möller, Joe Kniss, and Rüdiger Westermann for much-appreciated contributions and discussions, and Michael Doggett from ATI for providing a Radeon 9700. The data sets we have used are courtesy of Tiani Medgraph. This work has been done in the basic research on visualization at the VRVis Research Center, which is funded by the Austrian Kplus project.

References

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of IEEE Symposium on Volume Visualization*, 91–98.

CSEBFALVI, B., MROZ, L., HAUSER, H., KÖNIG, A., AND GRÖLLER, M. E. 2001. Fast visualization of object contours by non-photorealistic volume rendering. In *Proceedings of EUROGRAPHICS 2001*, 452–460.

CULLIP, T. J., AND NEUMANN, U. 1993. Accelerating volume reconstruction with 3D texture mapping hardware. Tech. Rep. TR93-027, UNC, Chapel Hill.

EBERT, D., AND RHEINGANS, P. 2000. Volume illustration: Non-photorealistic rendering of volume models. In *Proceedings of IEEE Visualization 2000*, 195–202.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of Graphics Hardware 2001*, 9–16.

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of SIGGRAPH '98*, 447–452.

HADWIGER, M., THEUSSL, T., HAUSER, H., AND GRÖLLER, E. 2001. Hardware-accelerated high-quality filtering on PC hardware. In *Proceedings of Vision, Modeling, and Visualization 2001*, 105–112.

HAUSER, H., MROZ, L., BISCHI, G.-I., AND GRÖLLER, E. 2001. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics 7*, 3, 242–252.

KINDLMANN, G., AND DURKIN, J. 1998. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of IEEE Volume Visualization '98*, 79–86.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Multi-dimensional transfer functions for interactive volume rendering. In *Proceedings of IEEE Visualization 2001*, 255–262.

KRÜGER, J., AND WESTERMANN, R. 2003. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization '03*.

LACROUTE, P., AND LEVOY, M. 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH '94*, 451–458.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (May), 29–37.

LI, W., AND KAUFMAN, A. 2002. Accelerating volume rendering with texture hulls. In *Proceedings of IEEE VolVis 2002*, 115–122.

LUM, E. B., AND MA, K.-L. 2002. Hardware-accelerated parallel non-photorealistic volume rendering. In *Proceedings of NPAR 2002*.

LU, A., MORRIS, C., EBERT, D., RHEINGANS, P., AND HANSEN, C. 2002. Non-photorealistic volume rendering using stippling techniques. In *Proceedings of IEEE Visualization 2002*, 211–218.

REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G., AND ERTL, T. 2000. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of Graphics Hardware 2000*.

RÖTTGER, S., GUTHE, S., WEISKOPF, D., ERTL, T., AND STRASSER, W. 2003. Smart hardware-accelerated volume rendering. In *Proceedings of VisSym 2003*, 231–238.

TIEDE, U., SCHIEMANN, T., AND HÖHNE, K. H. 1998. High quality rendering of attributed volume data. In *Proceedings of IEEE Visualization '98*, 255–262.

UDUPA, K. K., AND HERMAN, G. T. 1999. *3D Imaging in Medicine*. CRC Press.

WEISKOPF, D., ENGEL, K., AND ERTL, T. 2003. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics 9*, 3, 298–312.

WESTERMANN, R., AND ERTL, T. 1998. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of SIGGRAPH '98*, 169–178.

Image Space Based Visualization of Unsteady Flow on Surfaces

Robert S. Laramee† Bruno Jobard‡ Helwig Hauser†

†VRVis Research Center, Austria, www.VRVis.at, {Laramee,Hauser}@VRVis.at
‡University of Pau, France, www.univ-pau.fr, bjobard@univ-pau.fr

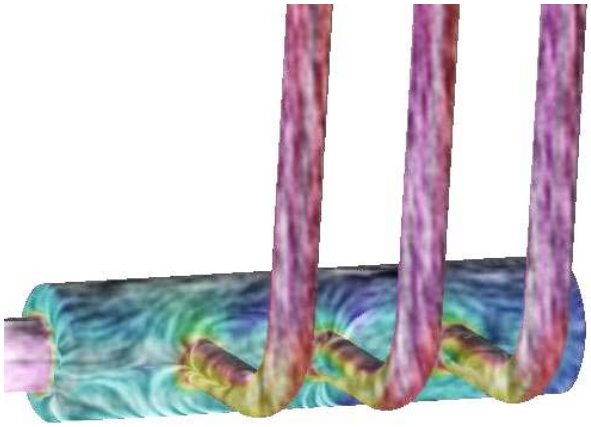


Figure 1: Visualization of flow on the surface of an intake manifold. The ideal intake manifold distributes flow evenly to the piston valves.

Abstract

We present a novel technique for direct visualization of unsteady flow on surfaces from computational fluid dynamics. The method generates dense representations of time-dependent vector fields with high spatio-temporal correlation using both Lagrangian-Eulerian Advection and Image Based Flow Visualization as its foundation. While the 3D vector fields are associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space. Frame rates of up to 20 frames per second are realized by exploiting graphics card hardware. We apply this algorithm to unsteady flow on boundary surfaces of, large, complex meshes from computational fluid dynamics composed of more than 250,000 polygons, dynamic meshes with time-dependent geometry and topology, as well as medical data.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism–Color, shading, shadowing, and texture; [Simulation and Modeling]: Simulation Output Analysis

Keywords: Unsteady flow visualization, computational fluid dynamics (CFD), surface representation, texture mapping

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

Figure 2: Visualization of flow at the complex surface of a cooling jacket -a composite of over 250,000 polygons.

1 Introduction

Dense, texture-based, unsteady flow visualization on surfaces has remained an elusive problem since the introduction of texture-based flow visualization algorithms themselves. The class of fluid flow visualization techniques that generate dense representations based on textures started with the Spot Noise [van Wijk 1991] and LIC [Cabral and Leedom 1993]. The main advantage of this class of algorithms is their *complete* depiction of the flow field while their primary drawback is, in general, the computational time required to generate the results.

Recently, two new algorithms, namely Lagrangian-Eulerian Advection (LEA) [Jobard et al. 2001] and Image Based Flow Visualization (IBFV) [van Wijk 2002], have been introduced that overcome the computation time hurdle by generating two-dimensional flow visualization at interactive frame rates, even for unsteady flow. This paves the way for the introduction of new algorithms that overcome the same problems on boundary surfaces and in three dimensions. In this paper we present a new algorithm that generates dense representations of arbitrary fluid flow on complex, non-parameterized surfaces, more specifically, surfaces from computational fluid dynamics (CFD). However, the algorithm is general enough to apply to other vector field data associated with a surface such as blood vessel flow.

Traditional visualization of boundary flow using texture mapping first maps one or more 2D textures to a surface geometry defined in 3D space. The textured geometry is then rendered to image space.

Here, we alter the classic order of operations. First we project the surface geometry to image space and then apply texturing. In other words, conceptually texture properties are advected on boundary surfaces in 3D but in fact our algorithm realizes texture advection solely in image space. The result is a versatile visualization technique with the following characteristics:

- generates a dense representation of unsteady flow on surfaces
- visualizes flow on complex surfaces composed of polygons whose number is on the order of 200,000 or more
- visualizes flow on dynamic meshes with time-dependent geometry and topology
- visualizes flow independent of the surface mesh's complexity and resolution
- supports user-interaction such as rotation, translation, and zooming always maintaining a constant, high spatial resolution
- the technique is fast, realizing up to 20 frames per second

The performance is due, among other reasons, to the exploitation of graphics hardware features and utilization of frame-to-frame coherency. The rest of the paper is organized as follows: in Section 2 we discuss related work, Section 3 details unsteady flow visualization on surfaces from CFD. Implementation details are described in Section 4 while results and conclusions are discussed in Section 5.

2 Related Work

Our work focuses on texture-based representations of unsteady flow on complex, non-parameterized surfaces. The challenge of visualizing time-dependent vector fields on surfaces at fast frame rates remains unsolved in surveys of the research literature [Post et al. 2002; Stalling 1997]. However, several techniques have been proposed to successfully resolve parts of the problem. In the next two sections we describe the two main categories of approaches for dense representations on surfaces and dense representations of unsteady 2D vector fields.

2.1 Texture-Based Flow Visualization on Surfaces

Previous research with a focus on representations of the vector field on boundary surfaces is generally restricted to steady-state flow. This is mainly due to the prohibitive computational time required. An enhanced version of Spot Noise is applied to surfaces by de Leeuw and van Wijk [de Leeuw and van Wijk 1995]. Battke et al. [Battke et al. 1997] describe an extension of LIC for arbitrary surfaces in 3D. Some approaches are limited to curvilinear surfaces, i.e., surfaces that can be parameterized using 2D coordinates. Forssell and Cohen [Forssell and Cohen 1995] extend LIC to curvilinear surfaces with animation techniques and add magnitude and directional information. Mao et al. [Mao et al. 1997] present an algorithm for convolving solid white noise on triangle meshes in 3D space and extend LIC for visualizing a vector field on arbitrary surfaces in 3D. Stalling [Stalling 1997] provides a helpful overview of LIC techniques applied to surfaces. In particular, a useful comparison of parameterized vs. non-parameterized surfaces is given.

2.2 LEA and IBFV

The algorithm in this paper is a new approach that incorporates features of both LEA and IBFV. These very effective algorithms have recently been introduced to produce dense representations of unsteady, 2D vector fields.

Jobard et al. introduced a Lagrangian-Eulerian texture advection technique for 2D vector fields at interactive frame rates [Jobard et al. 2001; Jobard et al. 2002]. The algorithm produces animations with high spatio-temporal correlation. Each still frame depicts the instantaneous structure of the flow, whereas an animated sequence of frames reveals the motion of a dense collection of particles when released into the flow. Particle paths are integrated as a function of time, referred to as the Lagrangian step, while the color distribution of the image pixels is updated in place (Eulerian step). The result represents a large step forward in bringing the visualization of unsteady flow to interactive frame rates.

Image Based Flow Visualization by van Wijk [van Wijk 2002] is the most recent algorithm for dense, 2D, unsteady vector field representations. It is based on the advection and decay of textures in image space. Each frame of the visualization is defined as a blend between the previous image, warped according to the flow direction, and a number of background images composed of filtered white noise textures. Performance times up to 50 frames per second are achieved through effective use of the graphics hardware.

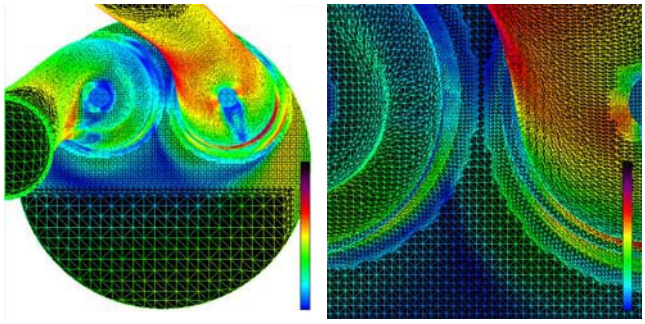


Figure 3: A wire frame view of the surface of two intake ports showing its 221,000 polygonal composition: (left) an overview from the top, note that many polygons are cover less than one pixel (right) a close-up view of the mesh between the two intake ports.

3 Unsteady Flow Visualization on Surfaces

In this section we describe our technique in detail, starting with a discussion of those factors motivating the approach.

3.1 Physical Space vs. Parameter Space vs. Image Space

One approach to advecting texture properties on surfaces is via the use of a parameterization, a topic that has been studied *ad nauseam* (e.g., Levy et al. [Lévy et al. 2002]). According to Stalling [Stalling 1997], applying LIC to surfaces becomes particularly easy when the *whole* surface can be parameterized globally in two dimensions, e.g., in the manner of Forssell and Cohen [Forssell 1994; Forssell and Cohen 1995]. However, there are drawbacks to this approach. Texture distortions are introduced by the mapping between parameter space and physical space and, more importantly, for a large number of surfaces, no global parameterization is available such as isosurfaces from marching cubes and most unstructured surface meshes resulting from CFD. Surface meshes from CFD may consist of smoothly joined parametric patches, but can have a complex topology and therefore, in general, cannot be parameterized globally. Figures 2 and 3 are examples of surfaces for which a global parameterization is not easily derived.

Another approach to advecting texture properties on surfaces would be to immerse the mesh into a 3D texture, then the texture properties could be advected directly according to the 3D vector

field. This would have the advantages of simplifying the mapping between texture and physical space and would result in no distortion of the texture. However, this visualization would be limited to the maximum resolution of the 3D texture, thus causing problems with zooming. Also, this approach would not be very efficient in that most of the texels are not used. The amount of texture memory required would also exceed that available on our graphics card, e.g., we would need approximately 500MB of texture memory if we use 4 bytes per texel and a 512^3 resolution texture.

Can the problem be reduced to two dimensions? The surface patches can be packed into texture space via a triangle packing algorithm in the manner described by Stalling [Stalling 1997]. However, the packing problem becomes complex since our CFD meshes are composed of many scalene triangles as opposed to the equilateral and isosceles triangles often found in computational geometry. The problem of packing scalene triangles has been studied by Carr et al. [Carr and Hart 2002]. For CFD meshes, triangles generally have very disparate sizes. For a given texture resolution, many triangles would have to be packed that cover less than one texel. To by-pass this, the surfaces could be divided into several patches which could be stored into a texture atlas [Lévy et al. 2002]. In any case, computation time would be spent generating texels which cover polygons hidden from the current point of view. The preceding discussion leads us to an alternative solution that, ideally, has the following characteristics: works in image space, efficiently handles large numbers of surface polygons, spends no extra computation time on occluded polygons, does not spend computation time on polygons covering less than a pixel, and supports user interaction such as zooming, translation, and rotation.

3.2 Method Overview

The algorithm presented here simplifies the problem by confining the advection of texture properties to image space. We project the surface geometry to image space and then apply a series of textures. This order of operations eliminates portions of the surface hidden from the viewer. In short, our proposed method for visualization of flow on surfaces is comprised of the following procedure:

1. associate the 3D flow data with the polygons at the boundary surface i.e., a velocity vector is stored at each polygon vertex of the surface
2. project the surface and its vector field onto the image plane
3. identify geometric discontinuities
4. advect texture properties according to the vector field in image space
5. inject and blend noise
6. apply additional blending along the geometric discontinuities previously identified
7. overlay all optional visualization cues such as showing a semi-transparent representation of the surface with shading

These stages are depicted schematically in Figure 4. Each step of the pipeline is necessary for the dynamic cases of unsteady flow, time-dependent geometry, rotation, translation, and scaling, and only a subset is needed for the static cases involving steady-state flow and no changes to the view-point. We consider each of these stages in more detail in the sections that follow.

3.3 Vector Field Projection

In order to advect texture properties in image space, we must project the vector field associated with the surface to the image plane, taking into account that the velocity vectors are stored at the polygon vertices. We chose to take advantage of the graphics hardware to

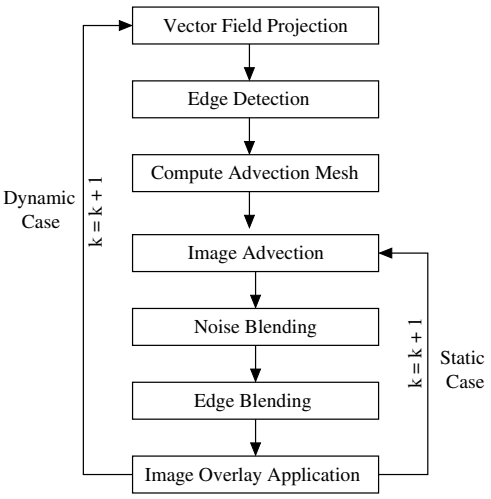


Figure 4: Flow diagram of texture-based flow visualization on complex surfaces -*k* represents time as a frame number.

project the vector field to the image plane. We assign a color whose *R*, *G*, and *B* values encode the *x*, *y*, and *z* components of the local vectors to each vertex of the boundary surface respectively. The velocity-colored geometry is rendered to the framebuffer. We use the term *velocity image* to describe the result of encoding the velocity vectors at the mesh vertices into color values. The velocity image is interpreted as the vector field and is used for the texture advection in image space. More precisely, the color assignment can be done with a simple scaling operation. For each color component, $\mathbf{h}_{rgb}$, we assign a velocity, $\mathbf{v}_{xyz}$ component according to:

$$\begin{aligned} \mathbf{h}_r &= \frac{\mathbf{v}_x - \mathbf{v}_{minx}}{\mathbf{v}_{maxx} - \mathbf{v}_{minx}} \\ \mathbf{h}_g &= \frac{\mathbf{v}_y - \mathbf{v}_{miny}}{\mathbf{v}_{maxy} - \mathbf{v}_{miny}} \\ \mathbf{h}_b &= \frac{\mathbf{v}_z - \mathbf{v}_{minz}}{\mathbf{v}_{maxz} - \mathbf{v}_{minz}} \end{aligned} \tag{1}$$

The minimum velocity component is subtracted for each color component respectively, in an effort to minimize loss of accuracy.

The use of a velocity image yields the following benefits: (1) the advection computation and noise blending is simpler in image space, thus we inherit advantages from the LEA and IBFV, (2) the vector field and polygon mesh are decoupled, thereby freeing up expensive computation time dedicated to polygons smaller than a single pixel, (3) conceptually, this is performing hardware-accelerated occlusion culling, since all polygons hidden from the viewer, are immediately eliminated from any further processing, and (4) this operation is supported by the graphics hardware. Saving the velocity image to main memory is simple, fast, and easy. A sample velocity image is shown in Figure 5 (top, left).

The construction of the velocity image allows us to take advantage of hardware-accelerated flow field reconstruction. During the construction of the velocity image, we enable Gouraud Shading, also supported by the graphics hardware. Since each velocity component is stored as hue at each polygon vertex of the surface, the smooth interpolation of hue amounts to hardware-accelerated vector field reconstruction. This is important for a minimum of two reasons. First, the polygonal primitive we choose at image advection time is independent of the original mesh polygons (more in Section 3.4). In other words, the vertices of the mesh we use to dis-

Figure 5: The 5 component images, plus a 6^{th} composite image, used for the visualization of surface flow on a ring: (top, left) the velocity image, (top, right) the geometric edge boundaries, (middle, left) the advected and blended textures, (middle, right) a sample noise image, (bottom, left) an image overlay, (bottom, right) the result of the composited images with an optional velocity color map. The geometric edge boundaries are drawn in black for illustration.

tort the image are not the same vertices where the original velocity vectors are stored. Second, interpolation is essential for flow field reconstruction. When the surface is rendered with velocity encoded as hue, the vertices of some polygons are clipped during the projection process. However, we still need to access the vector field values inside those polygons, and not just at the vertices, hence the need for reconstruction. We also note that we are not necessarily limited to linear interpolation for reconstruction. Higher order interpolation schemes can be supported by graphics hardware [Hadwiger et al. 2001].

The velocity vectors are de-coded from the velocity image according to:

$$\begin{aligned} \mathbf{v}_x &= \mathbf{h}_r \cdot (\mathbf{v}_{maxx} - \mathbf{v}_{minx}) + \mathbf{v}_{minx} \\ \mathbf{v}_y &= \mathbf{h}_g \cdot (\mathbf{v}_{maxy} - \mathbf{v}_{miny}) + \mathbf{v}_{miny} \\ \mathbf{v}_z &= \mathbf{h}_b \cdot (\mathbf{v}_{maxz} - \mathbf{v}_{minz}) + \mathbf{v}_{minz} \end{aligned} \tag{2}$$

The de-coded velocity vectors used to compute the advection mesh (Sec 3.4) are then projected onto the image plane.

The magnitude of the velocity vectors at those parts of the surface orthogonal to the image plane may be shortened as a result of perspective projection, i.e., if the dot product between the image plane normal and the 3D surface normal is zero or close to zero. This can reduce the visual clarity of the vector field's direction during animation. In our implementation, we added an option that allows the user to apply a bias to the velocity vectors that are shortened close to zero due to the projection. We can use this bias to reduce the scaling effect for curved surfaces. Conceptually it is like applying a reverse velocity clamp.

The projection of the vectors to the image plane is done after velocity image construction for 2 reasons: (1) not *all* of the vectors have to be projected (Sec. 3.4), thus saving computation time and (2) we use the original 3D vectors for the velocity mask (Sec. 4.2).

3.4 Advection Mesh Computation and Boundary Treatment

After the projection of the vector field we compute the mesh used to advect the textures similar to IBFV. We distort a regular, rectilinear mesh according to the velocity vectors stored at mesh grid intersections. The distorted mesh vertices can then be computed by advecting each mesh grid intersection according to the discretized Euler approximation of a pathline, $\mathbf{p}$, (the same as a streamline for steady flow) expressed as:

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \mathbf{v}^p(\mathbf{p}_k; t)\,\Delta t \tag{3}$$

where $\mathbf{v}^p$ represents a magnitude and direction after projection to the image plane. The texture coordinates located at the regular, rectilinear mesh vertices are then mapped to the (forward) distorted mesh positions. The distorted mesh positions are stored for fast advection of texture properties for static scenes.

Special attention must be paid in order to handle flow at geometric boundaries of the surface. Figure 6 shows an overview of the original IBFV process. During the visualization, each frame is advected, rendered, and blended in with a background image. If we look carefully at the *distort* phase of the algorithm, we notice that there is nothing to stop the image from being advected outside of the physical boundary of the geometry. While this is not a problem when the geometry covers the entire screen, this can lead to artifacts for geometries from CFD, especially in the case of boundaries with a non-zero outbound flow, e.g., flow outlets.

To address this problem we borrow a notion from LEA that treats non-rectangular flow domains, namely, the use of backward coordinate integration (also proposed by Max and Becker [Max and Becker 1999]). Using backward integration, equation 3 becomes:

$$\mathbf{p}_{k-1} = \mathbf{p}_k - \mathbf{v}^p(\mathbf{p}_{k-1}; t)\,\Delta t \tag{4}$$

In this case the texture coordinates located at the (backward) distorted mesh positions are mapped to the regular, rectilinear mesh vertices. Backward integration does not allow advection of image properties past the geometric boundaries.

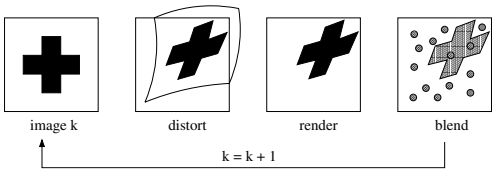


Figure 6: An overview of the original image based flow visualization

3.5 Edge Detection and Blending

While we gain many advantages by decoupling the image advection process with the 3D surface geometry, artifacts can result, especially in the case of geometries with sharp edges. If we look carefully at the result of advecting texture properties in image space, we notice that in some cases a visual flow continuity is introduced where it may be undesirable. A sample case is shown in Figure 7.

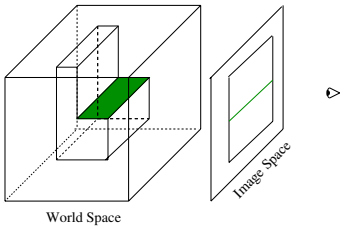


Figure 7: When a 3D surface geometry (left) is projected, continuity is created in image space (right). If the flow aligned texture properties are advected across this edge, an artificial flow continuity may result.

A portion of the 3D geometry, shown colored, is much less visible after the projection onto the image plane. If the flow texture properties are advected across this edge in image space, also shown colored, an artificial continuity results. To handle this, we incorporate a geometric edge detection process into the algorithm. During the image integration computation, we compare spatially adjacent depth values during one integration and advection step. We compare the associated depth values, z_{k-1} and z_k in world space of $\mathbf{p}_{k-1}$ and $\mathbf{p}_k$ from equation 4, respectively. If

$$|z_{k-1} - z_k| > \varepsilon \cdot |\mathbf{p}_{k-1} - \mathbf{p}_k| \tag{5}$$

where ε is a threshold value, then we identify an edge. All positions, $\mathbf{p}$, for which equation 5 is true, are classified as edge crossing start points. Special treatment must be given when advecting texture properties from these points. This process does not detect *all* geometric edges, only those edges across which flow texture properties should not be advected.

Figure 5 top, right shows one set of edges from the detection process. The geometric edges are identified and stored during the dynamic visualization case and additional blending is applied (depicted schematically in Figure 4). During the edge blending phase of the algorithm we introduce discontinuities in the texture aligned with the geometric discontinuities from the surface, i.e., gray values are blended in at the edges. This has the effect of adding a gray scale phase shift to the pixel values already blended. This could obviously be handled in different ways, e.g., choosing a random noise value to advect or inverting the noise value already present. Some results of the edge detection and blending phase are illustrated in Figure 8. In our data sets an ε of 1-2% of depth buffer is practical. However, the users may set their own value if fine tuning of the visualization is needed.

The same edge detection and blending benefits incoming boundary flow. Also an artifact of the IBFV algorithm, geometric boundaries with incoming flow may appear dimmer than the rest of the geometry. This is a result of the noise injection and blending process described in Section 3.6. In short, the background color shows through more in areas of incoming flow because not as much noise has been blended in these areas. Figure 9, top, shows a 2D slice through a 3D mesh from a CFD simulation with incoming boundary flow coming in through the narrow inlet from the right. Note that the edge of the inlet appears dim. Figure 9, bottom, shows the same slice with edge blending turned on. The boundary artifacts

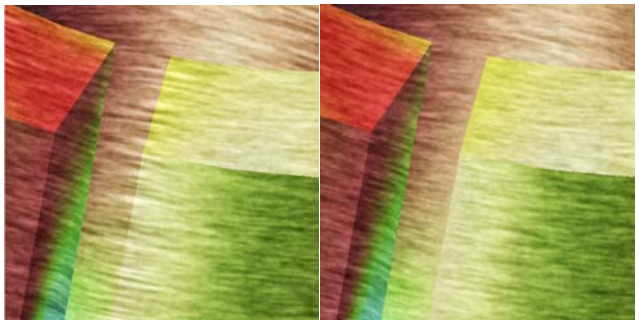


Figure 8: A close-up example of geometric edge detection: on the left side, geometric edge detection is disabled, on the right side enabled.

of the noise injection and blending process are no longer a distraction. Edge detection and blending also plays in important role while an object is rotating. Without special treatment, contours in image space become blurred when different portions of a surface geometry overlap, such as when blood vessels in Figure 12 overlap during rotation.

3.6 Noise Blending

By reducing the image generation process back to two dimensions, the noise injection and blending phase falls in line with the original IBFV technique, namely, an image, F, is related to a previous image, G, by [van Wijk 2002]:

$$F(\mathbf{p};k) = \alpha \sum_{i=0}^{k-1} (1-\alpha)^i G(\mathbf{p}_{k-i};k-i) \tag{6}$$

where $\mathbf{p}$ represents a pathline, α defines a blending coefficient, and k represents time as a frame number. Thus a point, $\mathbf{p}_k$, of an image F_k, is the result of a convolution of a series of previous images, $G(\mathbf{x};i)$, along the pathline through $\mathbf{p}_k$, with an decay filter defined by $\alpha(1-\alpha)^i$. The blended noise images have both spatial and temporal characteristics. In the spatial domain, a single noise image, $g(x)$, is described as a linearly interpolated sequence of n random values, G_i, in the range $[0, n-1]$, i.e.,

$$g(x) = \sum h_s(x - is)\, G_{i \bmod n} \tag{7}$$

where the spacing, s, between noise samples is generally greater than or equal to the distance traversed by an image property in one advection step and h_s represents a triangular black and white pulse function. Here x represents a location in the flow domain. In practice, we give the user control of s, resulting in multi-frequency texture resolutions in the spacial domain. The background textures used for blending also vary in time. In the temporal domain, each point, G_i in the background texture, periodically increases and decays according to a profile, $w(t)$, e.g.,

$$G_{i;k} = w((k/M + \phi_i) \bmod 1) \tag{8}$$

where ϕ_i represents a random phase, drawn from the interval [0,1), M is the total number of background noise images used, and where $w(t)$ is defined for all time steps. We use a square wave profile, i.e., $w(t) = 1$ if $t < 1/2$ and 0 otherwise. In our application, the user has the option of varying M. Smaller values of M result in higher frequency noise in the temporal domain whereas higher values M result in a lower temporal frequency. Figure 5 (middle, left) shows a sample blended image and Figure 5 (middle, right) shows a sample noise image.

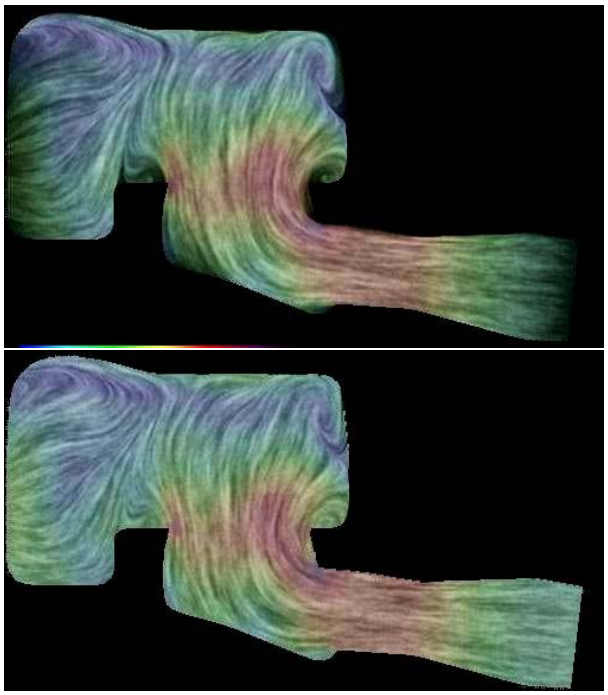


Figure 9: Here we see a 2D slice through a 3D geometry from a CFD simulation. (top) With no edge blending, the background color shows through boundary areas with incoming flow. (bottom). With edge blending, these artifacts are no longer a distraction.

3.7 Image Overlay Application

The rendering of the advected image and the noise blending may be followed by an optional image overlay. An overlay enhances the resulting texture-based representation of surface flow by applying color, shading, or any attribute mapped to color (Fig. 5, bottom, left). In implementation, we generate the image overlay following the construction of the velocity image. This overlay may render any CFD simulation attribute mapped to hue. The overlay is constructed once for each static scene and applied after the image advection, edge blending, and noise blending phases. Since the image advection exploits frame-to-frame coherency, the overlay must be applied after the advection in order to prevent the surface itself from being smeared. Also worthy of mention, is that the opacity value of the image overlay is a free parameter we provide to the user.

4 Implementation

In this section we consider some aspects of the algorithm not previously discussed which are important for implementation. Our implementation is based on the highly portable OpenGL 1.1 (`www.opengl.org`) library.

4.1 Texture Clipping

In our application, the resolution of the quadrilateral mesh used to warp the image can be specified by the user. The user may specify a coarse resolution mesh, e.g., 128×128, for faster performance or a fine resolution mesh, e.g., 512×512, for higher accuracy. However, if the resolution of the advection mesh is too coarse in image space, artifacts begin to appear. Figure 10, left, illustrates these artifacts zoomed in on the edge of a surface. In order to minimize the jagged edges created by coarse resolution texture quadrilaterals, we apply a texture clipping function. Subsets of textured quadrilateral that do not cover the surface are clipped from the visualization as shown in Figure 10, right. This can be implemented simply with the image overlay by maximizing the opacity wherever the depth buffer value is maximized, i.e., wherever there is a great depth.

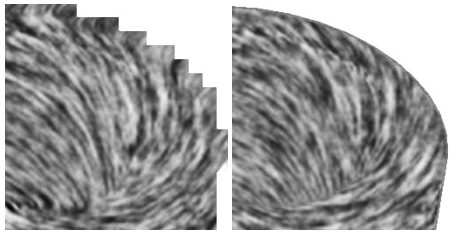


Figure 10: The result of, left, a coarse resolution advection mesh with artifacts and, right, the application of texture clipping. The resolution of the advection mesh shown on the left is 32×32 for illustration.

4.2 Velocity Mask

In order to dim high frequency noise in low velocity regions, the user also has the option of applying a velocity mask. We adopt the velocity mask of Jobard et al. [Jobard et al. 2001] for our purposes here, namely:

$$\alpha = 1 - (1 - \mathbf{v})^m \tag{9}$$

where α decreases as a function of velocity magnitude. In our case, the image overlay becomes more opaque in regions of low velocity and more transparent in areas of high velocity. With the velocity mask enabled, the viewer's attention is drawn away from areas of stagnant flow, and towards areas of high flow velocity. We note that in the context of CFD simulation data, engineers are often very concerned about areas of stagnant flow. In the case of a cooling jacket, stagnant flow may represent a region of the geometry where the temperature is too high, possibly leading to boiling conditions thus reducing the effectiveness of the cooling jacket itself. Therefore, in our case the engineers may disable the velocity mask or use the velocity mask to *highlight* areas of flow, e.g., make the hue brighter in areas of low velocity.

5 Performance and Results

Our visualization technique is applied primarily to large, highly irregular, adaptive resolution meshes commonly resulting from computational fluid dynamics simulations. [1] The ideal intake manifold (Fig. 1) supplies an equal amount of fluid flow to each piston valve. Visualizing the flow at the surface gives the engineer insight into any imbalances between the inlet pipes, in this case, the 3 long narrow pipes of the geometry. Figure 13 shows our method applied to a surface of an intake port mesh (from Fig. 3) composed of 221K polygons. The intake port mesh is composed of highly adaptive resolution surface polygons and for which no global parameterization is readily available. The method described here allows the user to zoom in at arbitrary view points always maintaining a high spatial resolution visualization. The algorithm applies equally well to meshes with time-dependent geometry and topology. Figure 11 shows the surface of a piston cylinder with the piston head (not shown) defining the bottom of the surface. The method here enables the visualization of fuel intake as the piston head slides down the cylinder. The resulting flow visualization has a smooth spatio-temporal coherency. Our algorithm also has applications in the field

[1] Supplementary video available at http://www.VRVis.at/ar3/pr2/vis03/

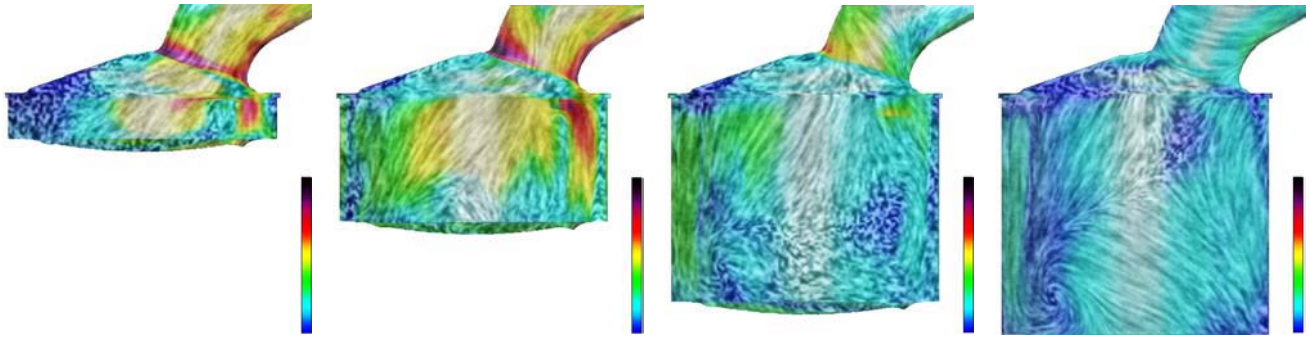


Figure 11: Snapshots from the visualization of a time-dependent surface mesh composed of a 79K polygons with dynamic geometry and topology. This intake valve and piston cylinder can also be used to analyse the formation of *wall film*, the term used to describe the liquid buildup on surfaces.

of medicine. Figure 12 shows the circulation of blood at the junction of 3 blood vessels. An abnormal cavity has developed that may hinder the optimal distribution of blood.

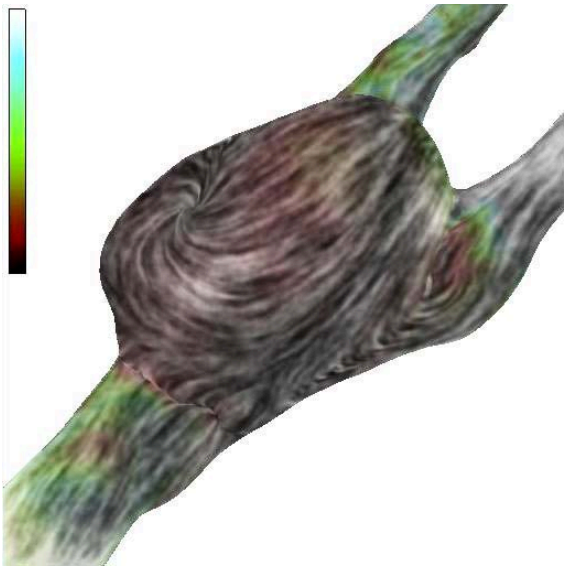


Figure 12: Visualization of blood flow at the surface of the junction of 3 blood vessels. Stagnant blood flow may occur within the abnormal pocket at the junction.

Performance was evaluated on an HP Visualize workstation with an HP *fx* graphics card, running Red Hat Linux 7.2 with a 1 GHz Pentium III dual processor and 1 GB of RAM. The performance times reported in Table 1 support interactive exploration of unsteady flow on surfaces. The first time reported in the FPS column is for the static cases of steady-state visualization and the absence of changes to the view point. The times shown in parenthesis indicate the dynamic cases of unsteady flow and interactive zooming and rotation. More specifically, the dynamic cases require the construction of a velocity image, image overlay, as well as geometric edge detection. We include geometric edge detection in the frame rates reported in Table 1. It does not introduce significant overhead since it is easily built into the advection process itself.

The performance time of our algorithm depends on the resolution of the mesh used to perform the advection and the number of polygons in the original surface mesh. In the case of steady-state flow, the algorithm no longer depends on the number of polygons in the surface mesh, but on the area covered in image space. The data set shown in Figure 1, left, does not cover as much image space, so its performance times are somewhat higher in the static case.

data set	number of polygons	advection mesh resolution	frames per second
ring (Fig 5)	10K	128 × 128	18 (5)
		256 × 256	9 (3)
		512 × 512	3 (1)
intake manifold (Fig 1)	48K	128 × 128	22 (2)
		256 × 256	14 (2)
		512 × 512	6 (1)
combustion chamber (Fig 11)	79K	128 × 128	17 (2)
		256 × 256	10 (2)
		512 × 512	4 (1)
intake port (Fig 13)	221K	128 × 128	17 (0.5)
		256 × 256	7 (0.5)
		512 × 512	2 (0.3)

Table 1: Sample frame rates for the visualization algorithm.

6 Conclusions and Future Work

We have presented a novel technique for dense representations of unsteady flow on boundary surfaces from CFD. The algorithm supports visualization of flow on arbitrary surfaces at up to 20 FPS via the careful use of graphics hardware. It supports exploration and visualization of flow on large, unstructured polygonal meshes, and on time-dependent meshes with dynamic geometry and topology. The method generates dense representations of time-dependent vector fields building on both the LEA and IBFV algorithms. It also does not waste computation time on occluded polygons or polygons covering less than one pixel. While the vector fields are defined in 3D and associated with arbitrary triangular surface meshes, the generation and advection of texture properties is confined to image space.

Future work can go in many directions including visualization of unsteady 3D flow, something we expect to see soon. Challenges will include both interactive performance time and perceptual issues. Future work also includes the application of more specialized graphics hardware features like programmable per-pixel operations in the manner of Weiskopf et al. [Weiskopf et al. 2002; Weiskopf et al. 2001] and the use of pixel textures like Heidrich et al. [Heidrich et al. 1999].

Portions of this work have been done via a cooperation between two research projects of the VRVis Research Center (www.VRVis.at) which is funded by AVL (www.avl.com) and an Austrian governmental research program called Kplus (www.kplus.at). We would also like to extend a special thanks to

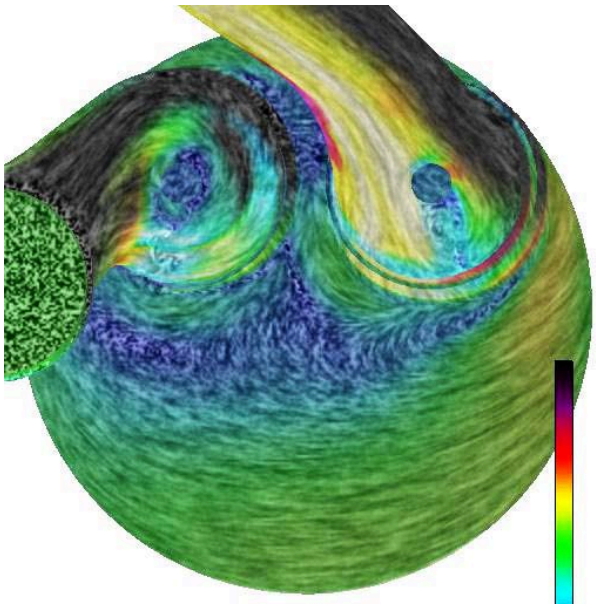


Figure 13: A view of the surface of an 221K polygonal intake port mesh show in Figure 3. Texture-based flow visualization is applied to the surface.

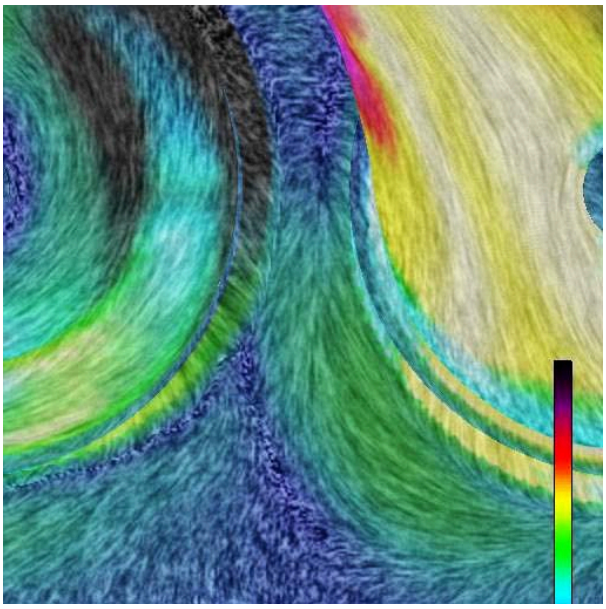


Figure 14: A close-up view of the surface of the intake port mesh show in Figure 13. Here we illustrate user-supported zooming with automatic, on the fly recalculation of the flow texture.

J. J. van Wijk for helping us to understand the IBFV algorithm and to Jürgen Schneider of AVL for his valuable insight into the CFD simulation data sets. Thanks to Jeroen van der Zijp and the FOX Windowing Toolkit (`www.fox-toolkit.org`) for help with the implementation. Thanks to Michael Mayer for medical the simulation data. We also thank Helmut Doleisch for his contributions. All CFD simulation data presented in this research is courtesy of AVL.

References

BATTKE, H., STALLING, D., AND HEGE, H. 1997. Fast Line Integral Convolution for Arbitrary Surfaces in 3D. In *Visualization and Mathematics*, Springer-Verlag, Heidelberg, 181–195.

CABRAL, B., AND LEEDOM, L. C. 1993. Imaging Vector Fields Using Line Integral Convolution. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, vol. 27, 263–272.

CARR, N. A., AND HART, J. C. 2002. Meshed Atlases for Real-Time Procedural Solid Texturing. *ACM Transactions on Graphics 21*, 2, 106–131.

DE LEEUW, W., AND VAN WIJK, J. 1995. Enhanced Spot Noise for Vector Field Visualization. In *IEEE Visualization '95 Proceedings*, IEEE Computer Society, 233–239.

FORSSELL, L. K., AND COHEN, S. D. 1995. Using Line Integral Convolution for Flow Visualization: Curvilinear Grids, Variable-Speed Animation, and Unsteady Flows. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June), 133–141.

FORSSELL, L. K. 1994. Visualizing Flow over Curvilinear Grid Surfaces Using Line Integral Convolution. In *Proceedings of the Conference on Visualization*, IEEE Computer Society Press, Los Alamitos, CA, 240–247.

HADWIGER, M., THEUSSL, T., HAUSER, H., AND GRÖLLER, E. 2001. Hardware-Accelerated High-Quality Reconstruction on PC Hardware. In *Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV-01)*, 105–112.

HEIDRICH, W., WESTERMANN, R., SEIDEL, H.-P., AND ERTL, T. 1999. Applications of Pixel Textures in Visualization and Realistic Image Synthesis. In *ACM Symposium on Interactive 3D Graphics*, ACM/Siggraph.

JOBARD, B., ERLEBACHER, G., AND HUSSAINI, M. Y. 2001. Lagrangian-Eulerian Advection for Unsteady Flow Visualization. In *IEEE Visualization*, IEEE.

JOBARD, B., ERLEBACHER, G., AND HUSSAINI, Y. 2002. Lagrangian-Eulerian Advection of Noise and Dye Textures for Unsteady Flow Visualization. In *IEEE Transactions on Visualization and Computer Graphics*, vol. 8(3), 211–222.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, 362–371.

MAO, X., KIKUKAWA, M., FUJITA, N., AND IMAMIYA, A. 1997. Line Integral Convolution for 3D Surfaces. In *Visualization in Scientific Computing '97. Proceedings of the Eurographics Workshop in Boulogne-sur-Mer, France*, Eurographics, 57–70.

MAX, N., AND BECKER, B. 1999. Flow Visualization Using Moving Textures. In *Data Visualization Techniques*, 99–105.

POST, F. H., VROLIJK, B., HAUSER, H., LARAMEE, R. S., AND DOLEISCH, H. 2002. Feature Extraction and Visualization of Flow Fields. In *Eurographics 2002 State-of-the-Art Reports*, The Eurographics Association, Saarbrücken Germany, 69–100.

STALLING, D. 1997. LIC on Surfaces. In *Texture Synthesis with Line Integral Convolution*, SIGGRAPH '97, Int. Conf. Computer Graphics and Interactive Techniques, 51–64.

VAN WIJK, J. 1991. Spot noise-texture synthesis for data visualization. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, 309–318.

VAN WIJK, J. J. 2002. Image Based Flow Visualization. In *SIGGRAPH 2002 Conference Proceedings*, Annual Conference Series, 745–754.

WEISKOPF, D., HOPF, M., AND ERTL, T. 2001. Hardware-Accelerated Visualization of Time-Varying 2D and 3D Vector Fields by Texture Advection via Programmable Per-Pixel Operations. In *Proceedings of the Vision Modeling and Visualization Conference 2001 (VMV-01)*, 439–446.

WEISKOPF, D., ERLEBACHER, G., HOPF, M., AND ERTL, T. 2002. Hardware-Accelerated Lagrangian-Eulerian Texture Advection for 2D Flow Visualizations. In *Proceedings of the Vision Modeling and Visualization Conference 2002 (VMV-01)*, 439–446.

Interactive 3D Visualization Of Rigid Body Systems

Zoltán Konyha* Krešimir Matković† Helwig Hauser‡

VRVis Research Center, Austria

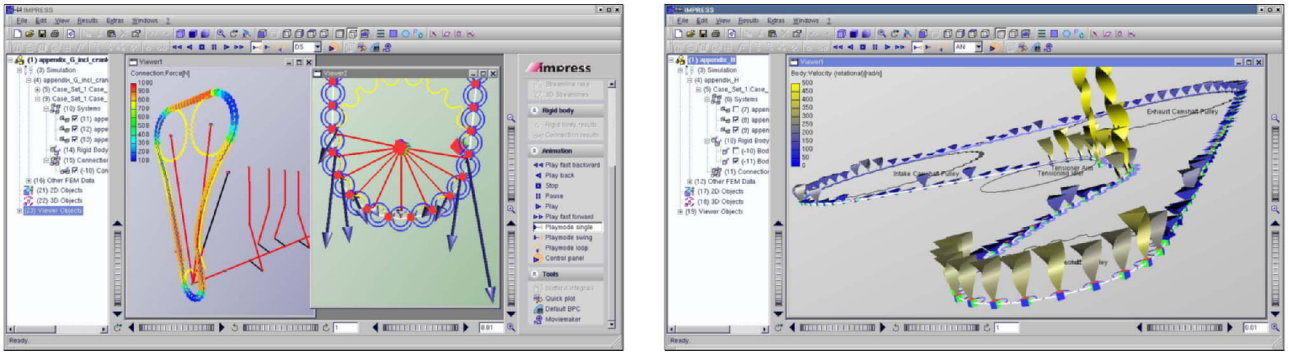


Figure 1: Interactive 3D visualization of chain and belt drives integrated into Impress.

Abstract

Simulation of rigid body dynamics has been an field of active research for quite some time. However, the presentation of simulation results has received far less attention so far. We present an interactive and intuitive 3D visualization framework for rigid body simulation data. We introduce various glyphs representing vector attributes such as force and velocity as well as angular attributes including angular velocity and torque.

We have integrated our visualization method into an application developed at one of the leading companies in automotive engine design and simulation. We apply our principles to visualization of chain and belt driven timing drives in engines.

CR Categories: I.3.8 [Computing Methodologies]: Computer Graphics—Applications; I.6.6 [Computing Methodologies]: Simulation And Modeling—Simulation Output Analysis

Keywords: rigid body dynamics, rigid body simulation, glyph based visualization, iconic visualization, automotive industry

1 Introduction

When hearing ”rigid body visualization” most of the people in the computer graphics community think of the systems for describing, simulating, and animating/visualizing rigid body systems, like skeletons used for human body animations, or similar application [Sauer and Schömer 1998; Park and Fussell 1997]. Actually, computer graphics is not the only field encompassing rigid bodies. Mechanical engineers have dealt with them on a daily basis for a quite some time. Simulation tools exist which simulate complex rigid body systems, used by mechanical engineers, which overcome the tools used in computer graphics. The main goals of the two systems are different. On the one hand, computer graphics people dealing with rigid body simulation are primarily interested in creating ”good-looking” animations with preferably real time and interactive simulation. On the other hand the main goal of mechanical engineers is to make the simulation as correct as possible which makes the simulation more complex. Our goal is to develop a visualization front-end for such complex simulation tools. The users of these system are not animators and designers, but engineers working with it on a daily basis. The data set we visualize is produced by rigid body simulation used in modern engine design.

First we focus on visualizing chain drive and belt drive simulation results. These systems are composed of many individual bodies which makes them difficult to evaluate using conventional methods. Visualization should help in the exploration and analysis (primarily) and in presentation of the simulation data. Up to now, analysis was done using a large amount of 2D charts and numbers. Such an analysis is very time-consuming and complicated. Visualization techniques proposed in this paper make it easier to interpret simulation results. The user can explore the data interactively and spot potential problems much easier than without visualization. Furthermore, some of the simulation results, like 3D vectors and relative positions in 3D space, are nearly impossible to imagine without 3D visualization.

This paper describes rigid body simulation first, since some of the readers may not be familiar with its applications in the automotive industry. We will then proceed with the description of the visualization system. Since the whole visualization is based on various glyphs [Post et al. 1995; van Walsum et al. 1996; de Leeuw and van Wijk 1993], the glyphs used will be introduced. Finally, the commercial application developed in cooperation with AVL [AVL n. d.] is presented. As far as we know this paper describes the glyph based visualization of rigid body simulation data for the first time.

*Konyha@VRVis.at
†Matkovic@VRVis.at
‡Hauser@VRVis.at

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

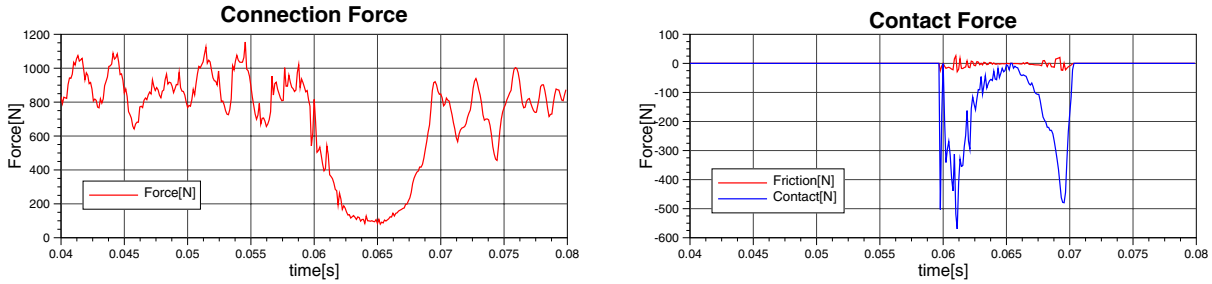


Figure 2: 2D charts showing (a) forces between two neighboring chain links and (b) forces between a chain link and sprocket.

2 Rigid Body Simulation

With increasing computational power simulation can be processed faster and with more precision than ever before. Rigid body dynamics is an established model of real world dynamics that allows reasonably accurate simulation of certain systems at an affordable computational price. In rigid body simulation [Baraff 1989; Mirtich and Canny 1995; Mirtich and Canny 1996] real world objects are modelled by physically ideal rigid bodies that can have 6 degrees of freedom (translation and rotation along all three major axes), but cannot be deformed. The bodies are connected by connection elements that are basically springs with dampers. Bodies can also come into contact occasionally by colliding with each other.

Rigid body simulation is extensively used in industrial simulation including automotive industry [Hoffman and Dowling 1999], aircraft and ship design. Engineers are interested in simulation since it is much faster and cheaper than producing a physical prototype. Prototype production and testing is still – and will be – common in engine production. However, simulation can reduce the number of prototype cycles and thereby improve development speed and cut down costs. The simulation data in our case comes from *AVL Tycon*, a rigid body based simulator developed at AVL [AVL n. d.] for the design of timing drives in engines. We focus on chain and belt drives in this paper. In simulation both chain and belt drives are handled in the same way. In chain drives each chain link is a rigid body interconnected via stiff springs that model the pins. Belt drives are simulated by dividing the belt into several rigid sections. Let us examine a typical chain drive simulation first.

A typical chain drive has 100-200 chain links. The simulator computes data for approximately 5000 time steps. For each time step and rigid body, e.g chain link a set of approximately 20 attributes is computed. The attributes include:

- displacement and orientation of bodies
- translational and angular velocity of bodies
- translational and angular acceleration of bodies
- forces and torques in connections between bodies
- relative displacement and relative velocity between endpoints of the connections

It is clear that the amount of data produced for such a relatively simple system is huge and not easy to analyze. So far the simulation results have been presented in 2D graphs that provide vague resemblance to the actual 3D models. Figure 2 shows two such graphs displaying connection forces between two neighboring chain links and contact forces between the chain link and the sprockets. Note that these two charts represent one link only, and that there are about 200 such charts describing only one attribute, e.g. force between chain links.

Our goal is to develop a more intuitive 3D visualization system for rigid body simulation data which makes the analysis faster and enables the user to explore the data in ways not possible in 2D charting.

3 Rigid Body Simulation and Visualization

The overhelming amount of data and 2D charts produced by the simulation is hardly manageable even for an experienced user. Therefore visualization should be used to assist the user in exploration, analysis and presentation of the data. Since the primary users of such visualization tool are engineers, the main goal is to support data exploration and analysis. Visualization supports data exploration in numerous ways.

The next section describes the motivation for developing an interactive 3D visualization framework for rigid body simulation data.

3.1 Motivation for Interactive 3D Visualization

An experienced engineer can certainly gather a lot of information from 2D charts, but having the visualization feedback can speed up the process significantly. It is difficult and often even impossible to imagine complex 3D geometry from numerical data only. For example, if there is a sprocket with its radius and position given in 3D space on the one hand; and a chain link with its geometry, position and orientation on the other, it is practically impossible to predict if the chain link actually touches or even intersects the sprocket. Rather all 3D vector quantities, like contact point positions or forces between sprockets and links, accelerations, etc. are difficult to imagine based on 3D coordinates only.

Furthermore, if the user wishes to examine more attributes simultaneously, the 3D visualization offers many possibilities. Interactive exploration where the user can navigate through, adjust or toggle parameters is simply not possible using conventional non-interactive means.

The behavior of the chain drive is of interest to engineers only when the chain drive is in motion. The 2D charts cannot depict the motion. On the contrary, interactive visualization can display the chain in motion and thereby enrich the static data. This feature is especially valuable to users who want to have an overview of the dynamic behavior.

If the user wants to explore a particular space in the system, e.g where chain leaves the sprocket, they can simply zoom in and explore parameters for all chain links passing by. Doing the same without interactive visualization would require scanning through numerous charts, looking for each chain link in turn. If the user is interested in more than one attribute at the same time the process can become frustrating. It is clear that visualization can improve the presentation of rigid body simulation results.

The simulation models of chain and belt drives in Tycon are identical. Identical attributes are computed thus the same visualization

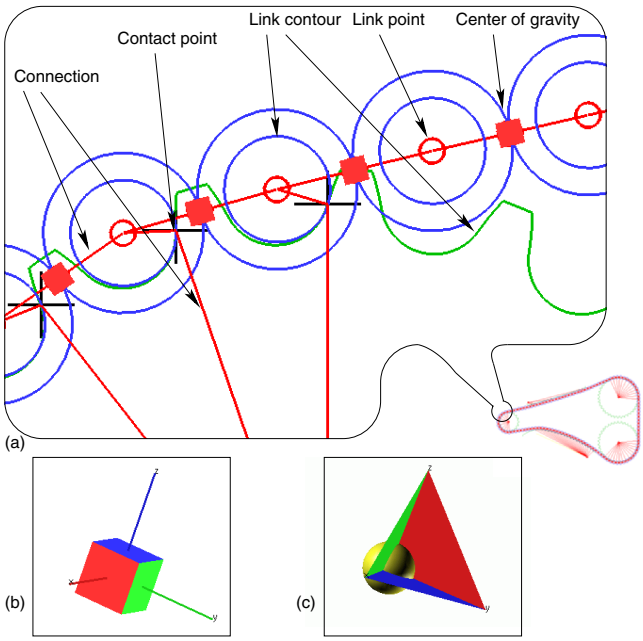


Figure 3: (a) Model of a chain drive. Overview and zoom of a part showing elements. (b) and (c) options for drawing center of gravity and local coordinate system.

methods can be applied to both drive types. In order to visualize attributes we use various glyphs. In the following section we describe the glyphs used in our visualization solution.

4 Glyphs for Rigid Body Visualization

In order to represent the rigid body system itself and related attributes various glyphs are required. The following sections introduce the glyphs we propose for visualization of bodies and connections in rigid body systems. The glyphs representing vector attributes (forces, velocities) and angular attributes are also described.

All glyphs described can depict multiple attributes simultaneously. However, correlation of attributes in rigid body simulation is quite natural. Therefore we generally map only one attribute to one glyph for simplicity.

4.1 Model of Rigid Body System

Rigid body systems in physics consist of *rigid bodies* and – generally elastic – *connections*. The bodies have a *center of gravity* and any number of *link points* defined in the body's local coordinate system. The center of gravity is by default represented by a cube which indicates the orientation of the body. The orientation can be clarified further by drawing the bodies local coordinate system as shown in Figure 3 (b) and (c). This is especially necessary when the user decides to use a sphere glyph to represent the center of gravity.

Connections run between link points of two bodies. Each link point has a *contour* assigned which is used in the simulator to calculate contact points between bodies. This modelling paradigm implies that a sprocket for example is modelled as a single rigid body with one link point at its center. The contour of this link point (shown in green in Figure 3) a) is the actual shape of the sprocket. Similarly, each chain link has link points at both of its pins that have circular contours, shown in blue. The smaller circle is the surface

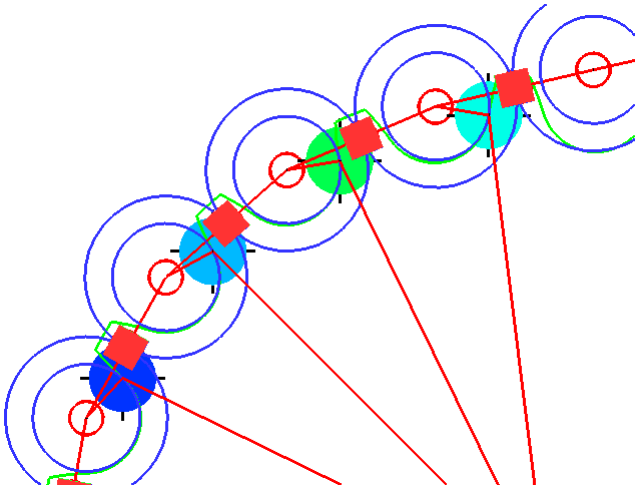


Figure 4: Color coded disks visualizing the magnitude of contact forces between a sprocket and chain links. The disks are positioned at the contact points.

that comes into contact with the sprocket while the larger circle is the outer surface of the link sliding on guides.

4.2 Colored disks

A very simple glyph that was found to be especially useful in visualizing magnitude of forces is a simple color coded disk shown in Figure 4. It has proved to be a quick way of locating the areas of extreme forces – something engineers perform very often as a first step when investigating a chain drive. The color is mapped to the visualized attribute. The sizes of all disks are the same. However, the user can specify the diameter of the disks to suit various model sizes.

4.3 Arrows

Probably the most straightforward way to visualize vector quantities is using arrows. Force, translational velocity and acceleration are often depicted using arrows in physics, thus it comes natural to use them in our visualization framework, too. The direction and the magnitude of the visualized attribute are indicated by the direction and length of the arrow. The starting point of the arrow identifies the point in space where the force acts.

In our implementation we offer 4 drawing styles of arrow glyphs. This allows the user to visualize various vector attributes simultaneously without confusing them. As shown in Figure 6 we offer flat head and cross head arrows, along with 3D cone head and pyramid-shaped ones. However, in 3D visualization only (c) and (d) should be used. (a) and (b) are more suited to 2D scenarios.

It is possible to visualize the real 3D vector or just its components in the body's local coordinate system. This offers the chance of visualizing only the normal component of the velocity of chain links for example as seen in Figure 12.

In addition, it is also possible to show any component along *any* axis of the local coordinate system. The advantage of this becomes obvious when one tries to display tangential velocity of chain links. The arrows would "pile up", occlude each other and render the image illegible as shown in Figure 5 (a). Our solution allows the user to display the tangential component *normal* to the chain itself as shown in Figure 5 (b). By doing so they have a good presentation of the magnitude, while the direction is known to be tangential

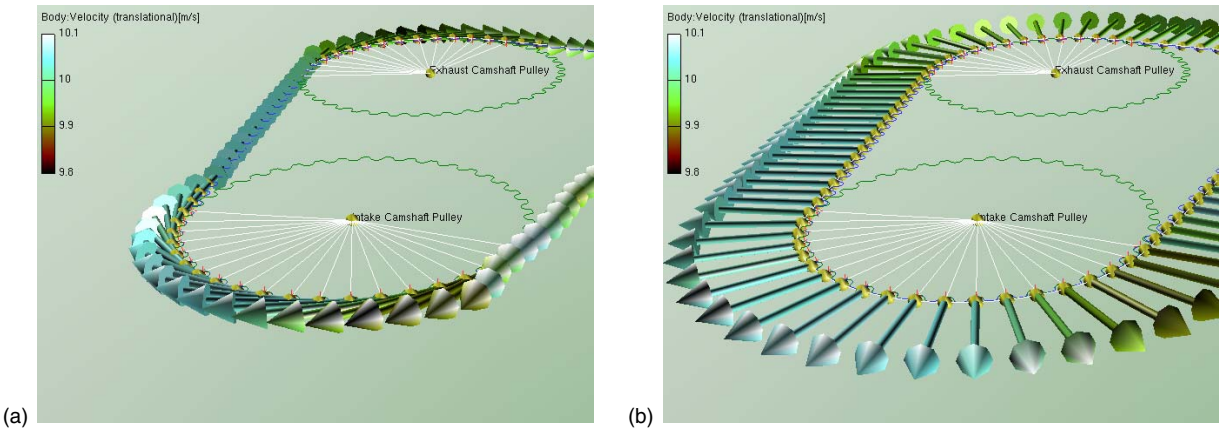


Figure 5: Tangential velocities of belt drive slices. (a) Arrows occlude each other when drawn in their real direction. (b) Occlusion reduced by displaying tangential velocity normal to the real direction.

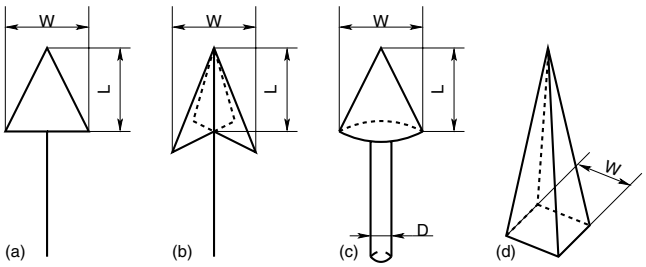


Figure 6: Models of (a) flat head, (b) cross head, (c) cone head and (d) pyramid-shaped arrows. Customizable parameters (width and length of arrow head, diameter of shaft) are also indicated.

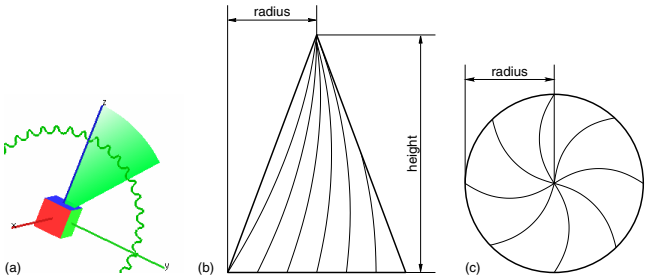


Figure 7: Glyphs for angular attributes: (a) sector glyph on one body visualizing angular velocity, (b), (c) sketch of a spiral mapped on the side of a cone. (b) side view, (c) top view.

a priori. It must be mentioned that such a rendering may be open for misinterpretation to novice users. However, experienced mechanical engineers are used to such representations because it has been a common method of displaying chain forces in figures. An example is shown in [Niemann and Winter 1986].

The arrow glyph can visualize the assigned attribute by changing its length or by full 3D scaling. Although the latter introduces the "visualization lie" [Tufte 1986] effect because of the nonlinear change in surface, in some scenarios it still turns out to be preferred by users because it makes extremely large attributes more noticeable.

It is also possible to apply optional color coding to the arrows to reflect their magnitude. Users may be interested in the actual value a glyph represents, thus annotations can be attached to the arrow glyphs as shown in Figure 12.

4.4 Sectors

Visualizing angular velocity and acceleration is somewhat more challenging. In computer graphics orientation and angular velocity are generally described by the axis of rotation (a vector) and the angle of rotation (a scalar) around that axis. To the contrary, the simulator we interface to represents rotation with Euler angles and the engineers expect to see these three angles in visualization, too.

Coming up with a glyph that visualizes 3 rotational degrees of freedom in an intuitive and unambiguous way is not easy. Therefore we have opted to visualize only one component of the angular velocity in the body's local coordinate system with one glyph. Additional glyphs of the same type can be instantiated to visualize further components.

One glyph that has proven to be intuitive is a sector. The sector's center is at the center of gravity of the body. The plane is rotated normal to the axis of rotation, thereby directly indicating it. The magnitude is visualized by the angle of the sector. The direction of the rotation (clockwise vs. counter-clockwise) is made clear by changing the transparency along the arc. This immediately defines the direction of the movement since it gives the impression of an arm leaving traces during sweeping over the arc. Figure 7 (a) shows a single body (the center of gravity of a sprocket) and a sector glyph that indicates its angular velocity.

The radius and the angle mapped to the maximum of the attribute can be specified by the user. As with arrows, the user is given the option to use color coding as a further means of stressing magnitude.

4.5 Spirals

Although the sectors mentioned above are suitable they have a major drawback. They are 2D objects, and when viewed under very small angles they are difficult to interpret. In the extreme case they appear as thin lines on the screen completely unable to convey anything other than perhaps the color.

To overcome this problem we have developed a glyph that is

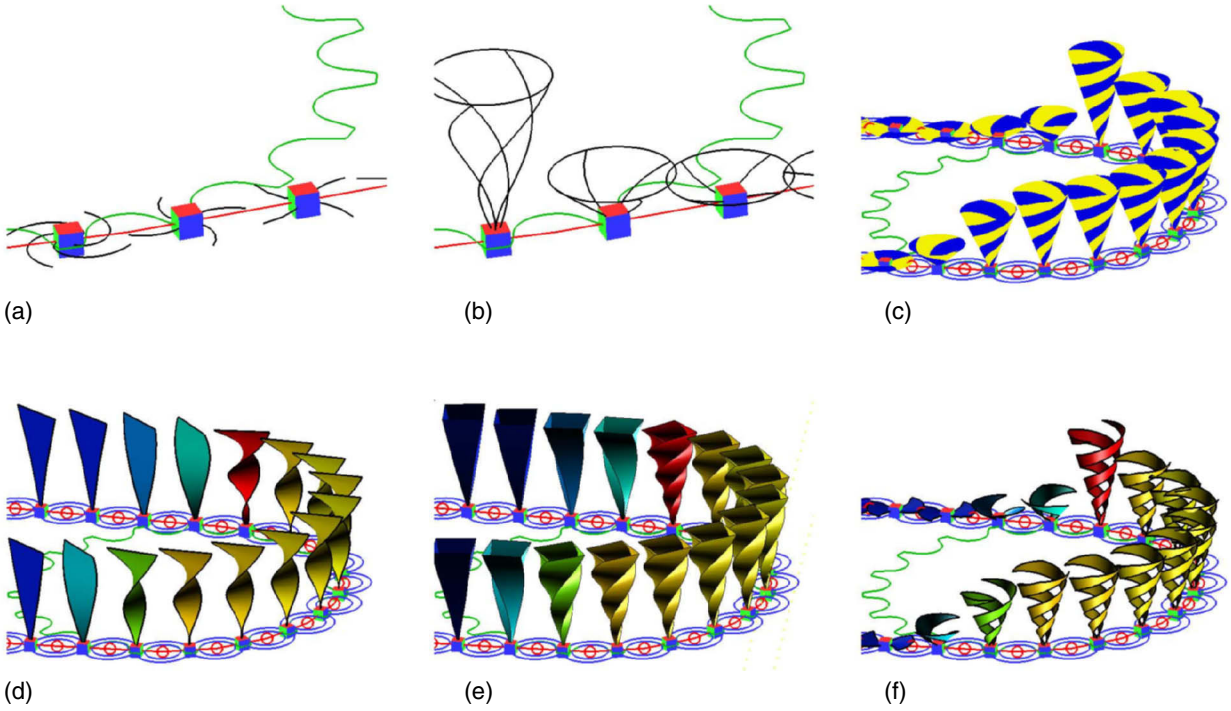


Figure 8: Rendering styles for spirals: (a) wireframe, (b) wireframe with circle, (c) filled, (d) helix, (e) pyramid, (f) ribbon.

capable of visualizing angular attributes reliably no matter where from it is viewed. The basic idea is simple: take a star-shaped glyph with a few arms and transform it into a spiral by twisting its center. The larger the attribute the more the spiral is twisted. This object is still 2D, therefore it is prone to similar visibility problems as the sectors. Now map these lines onto the surface of a cone and make the cone's height also a function of the attribute being visualized. This results in a 3D glyph shown in Figure 7 (b) and (c) which can visualize angular quantities reliably regardless of the viewing angle. When viewed from the top or bottom the twisted spiral arms are easily interpreted. When seen from the side the spiral arms are not as clearly visible as from the top. However, in this case the height of the cone provides additional clues to the viewer.

The cone is rotated so that its axis coincides with the axis of rotation. The direction of the rotation is made clear by twisting the spiral clockwise or counter clockwise, respectively.

We again offer many rendering options to facilitate simultaneous visualization of various attributes and allow the user to choose whichever style they prefer. The user can change (1) the maximum twist angle (2) radius of the disk at the base of the cone (3) number of spiral arms (4) scaling factor of the cone's height. There are 6 rendering styles as shown in Figure 8: wireframe (with or without the circle at the base of the cone), filled, helix, pyramid and ribbon. The cones can be made the same height as seen in (d) and (e) in Figure 8. Color coding and annotations are again optional.

A valuable feature is the optional animation (rotation) of the spirals. The speed and direction of the rotation reflects the attribute visualized. This has proven to be very popular with users because it indicates rotation in the most natural way.

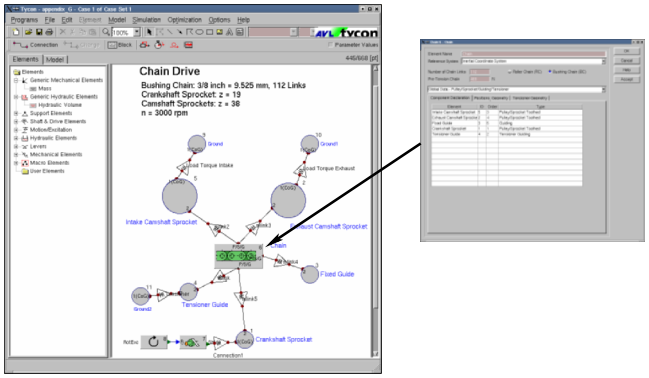


Figure 9: Block model of a chain drive in Tycon with the property dialog of one element.

5 The Application

AVL offers a full range of software for simulation in the automotive industry. The AVL Workspace is a common user-friendly framework for simulation tools including, amongst others, *Tycon* for valve train and timing drive dynamics that is based on rigid body dynamics. It also includes a common visualization tool called *Impress* which can process output from all simulation modules. The rigid body visualization capabilities of Impress are based on the principles described in this paper. We will now guide the reader through the typical workflow of an engineer working with AVL Workspace. We will put more emphasis on detailing the process of evaluating the simulation results.

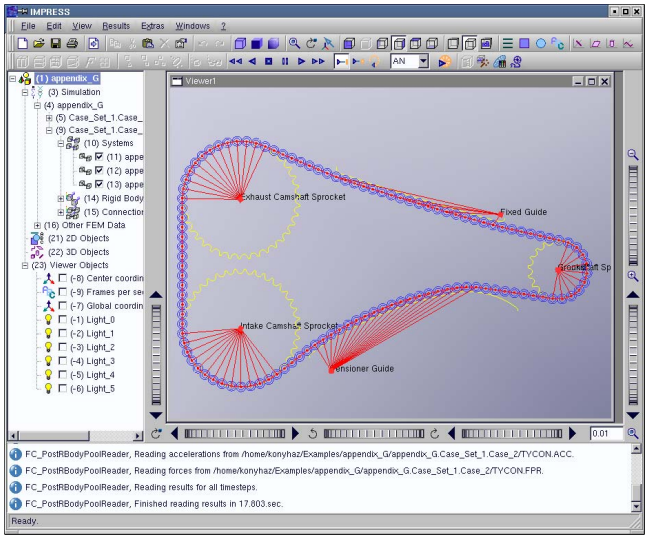


Figure 10: Main window of Impress.

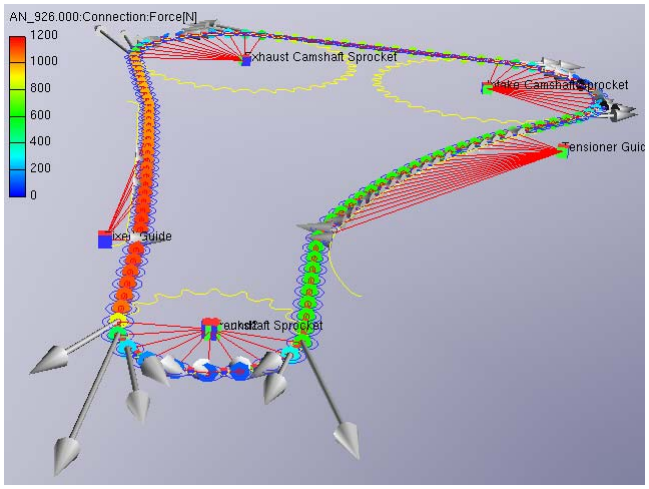


Figure 11: Forces between chain links (colored disks) and contact forces between chain links and sprockets (arrows) This figure also demonstrates the capability of creating several visualization objects simultaneously.

5.1 Model Generation

Tycon enables the user to build a 2D block model of the timing drive. The Tycon user interface is shown in Figure 9. There is a tree view of elements that can be inserted into the workspace. Properties such as name, mass, initial position and link points can be specified in popup dialogs for each element. There are specialized tools to assist the definition of contours of profiled elements, e.g sprockets. Connections are created by clicking the appropriate link points on the bodies, properties such as stiffness are again defined in popups. Models of standard configurations can be stored as templates for further use, thus the generation of modified variants is easy and fast.

5.2 Simulation

Once the model and the boundary conditions (engine RPM, etc) are defined the simulation can be executed. For complex models and long simulation runs (several engine cycles) this can be a fairly time-consuming process. Simulation jobs running for over 10 hours are not uncommon. The resulting files can be as large as 1 GB.

5.3 Evaluation of Results

When the simulation has finished the user can launch the visualization tool Impress and load the results. The Impress application shown in Figure 10 has a tree view of objects on the left, can have any number of independent 3D viewing windows plus the standard menu and icon bars.

5.3.1 Managing Rigid Body Systems in Impress

Groups of bodies, e.g bodies that belong to one chain are created automatically. These groups are referred to as *systems* and they are displayed in the tree. The user can show or hide the components of the systems shown in Figure 3 as well as customize their colors and sizes. It is possible to hide or drag groups that are temporarily out of interest in order to reduce occlusion. Dragged objects can be returned to their original position by a single click.

The view can be panned, rotated and zoomed as desired. The model can be animated using VCR-like controls. The user can perform all tasks during animation, too. They can change the viewpoint, adjust properties of objects and create new visualization objects.

5.3.2 Visualizing Attributes

In order to view attributes of bodies or connections in a system the users selects the system and clicks the icons ”Rigid body results” or ”Connection results”, respectively. A *visualization object* consisting of a group of glyphs referring to the bodies is created. A dialog pops up where the user can select the attribute and glyph tpye shown as well as customize glyph parameters. Any number of these visualization objects can be created for each system, and they can be hidden or dragged just like the systems themselves.

The engineers typically start exploration by examining if the simulation was correctly set up. They run the animation looking for obvious signs of incorrect initial conditions. Mismatching chain and sprocket contours, sprockets defined at the wrongs position or in the wrong orientation are very easy to spot. With toothed belt drives, loose tensioners may allow the belt to skip one tooth of the pulley, which is also very noticeable in the animation. These errors are very hard to discover without interactive visualization.

Then they proceed by examining forces along the chain drive. looking for areas of extremely high forces. These areas are quickly found by creating colored disks and running the animation. In Figure 11 the span of the chain on the left entering the sprocket is easily identified as being under heavy tension. Engineers are interested in the contact position and contact force of the chain links and the sprockets. This information is valuable in order to understand which areas will be subjected to extreme wear. In Figure 11 the location and direction of these forces is immediately seen which was not possible at all with the diagrams used before.

The vibrational motion of chain links in long spans can also be of interest to the engineer. This can be visualized by showing the translational and angular velocities of chain links as seen in Figure 12. In this case showing only the normal component of the velocity is highly useful.

Rotational acceleration and torque are very well visualized using the spirals introduced in Section 4.5. Figure 13 shows a close-up of rotational acceleration of chain links. It is clearly seen in the image that chain links undergo heavy acceleration as they hit the yellow

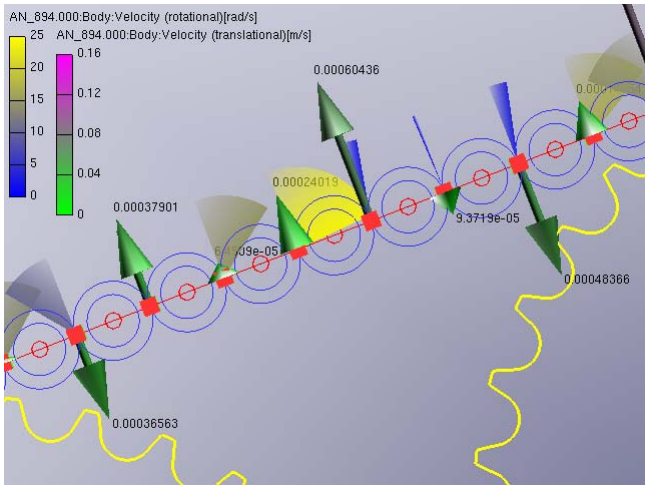


Figure 12: Rotational velocity of chain links visualized using the sector glyph. Annotated arrows show the normal component of the translational velocity. Vibrational behavior is clearly seen.

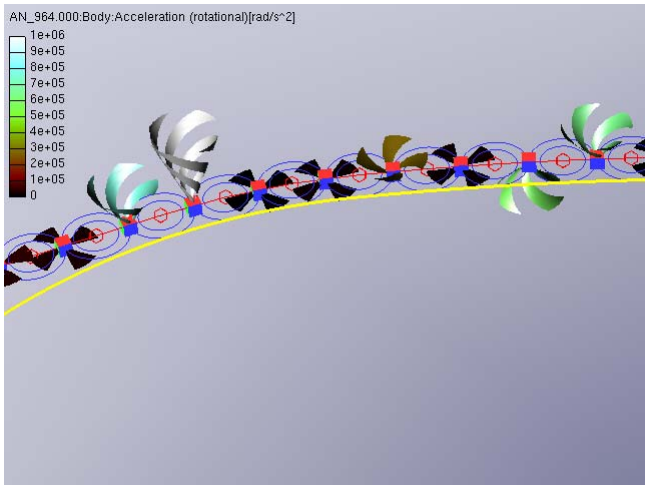


Figure 13: Rotational acceleration depicted by spiral glyphs. Opposing directions of rotation are distinguished by opposite twist directions and different orientation of the cone, too.

tensioner guide. This can be an indication to the engineer to change the profile of the tensioner to allow smoother contact. Naturally, this can reduce noise levels and the extend the chain's life span.

We must mention that not all types of color scales used are perceptually uniform [Levkowitz 1996]. The one seen in Figure 11 for instance has a large section of green colors which are hard to distinguish. Actually, this is not a disadvantage because users are generally more interested in the extremely small or large magnitudes.

We must not forget that interactive 3D visualization cannot entirely replace the traditional 2D diagrams. However, they can coexist as means of partly different evaluation tasks. Interactive 3D visualization helps in exploration, locating areas and attributes of interest, whereas engineers will always refer to diagrams or actual numbers for more accurate information. Body names can be displayed optionally to assist looking up the corresponding diagrams.

All settings are saved in a project file to allow the user to continue exactly where they left off. There is a movie director module similar to the track view found in 3D Studio Max [3ds n. d.] to facilitate creation of AVI or MPEG movies for presentations. The user can define events that will be executed at given frames. Events include show/hide objects, zoom in/out, rotate view, etc. An event can be assigned to a range of frames. This enables smooth zooms and rotations, for example.

6 Evaluation

For many years experienced engineers have been using the 2D diagrams quite effectively. However, they still face difficulties understanding position and orientation of bodies as well as directions in 3D space. It is especially difficult to judge and compare relative positions and distances. Characteristics of periodic behavior such as frequency of vibration are also hard to perceive without animation. The visualization system we have described remedies these shortcomings.

6.1 User Feedback

Users have stressed the following advantages to their previous workflow:

- Contours that are in contact and actual positions of contact points are clearly identified. This is especially important if the model is somehow incorrect, for example the contour of a sprocket is not continuous. Discovering a very loose toothed belt that skips the teeth of a pulley as seen in Figure 14 is also made very easy. Errors in modelling and initial conditions became a lot easier to detect. Finding these types of problems used to be a slow process because the only indication had been irrational simulation results.

- Directions of forces are made clear. This is important when the engineer expects a force to act in one direction, but simulation proves that it acts in some other direction. In fact showing the direction and magnitude of vector attributes at the same time is something they could not do before.

- Motion of tensioners can be seen clearly. This has also been very difficult to understand by studying diagrams.

- Animation shows all chain links at a given time step creating an overall impression of how the chain moves. The graphs are created per chain link and the overall motion is not seen in them.

- It is easier to spot periodic behavior at a given point in space.

- Distribution of forces between chain links is seen. Areas of extreme tension can be found very quickly.

Less experienced engineers and professionals working in neighboring fields (manufacturing, namely) can also profit a lot from the more accessible presentation of the simulation results. Furthermore, advantages of 3D visualization for the marketing department are obvious. Convincing movies for presentations can be generated with little effort.

It is clear that interactive 3D visualization enhanced the existing application significantly. All user groups ranging from inexperienced, young engineers through experienced professionals to marketing department benefit from our solution.

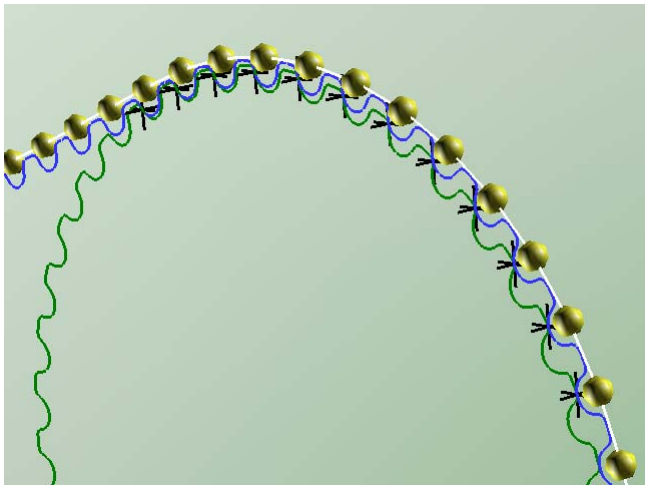


Figure 14: A toothed belt skipping over the teeth of a sprocket due to a loose tensioner. Contact points are depicted by black crosses. This phenomenon is very difficult to spot without 3D visualization.

6.2 Performance

Our implementation of rigid body visualization integrated into Impress runs on Windows 2000, Linux and various Unix systems. Naturally, performance depends heavily on the capabilities of the graphics hardware. Our tests were performed on a 1 GHz Pentium III with HP Visualize *fx* graphics accelerator and running Red Hat Linux 7.2. Rendering and navigation – implemented in a separate thread – runs at over 40 frames per second for typical scenarios. Animation of the chain model consisting of 134 chain links shown in Figure 11 updates at around 15 frames per second. Response to user interaction is generally immediate; creating new visualization objects requires about a second, though.

7 Conclusion and Future Work

Up to now little effort has been invested into visualization of rigid body dynamics in the automotive industry. Engineers were using 2D diagrams to evaluate simulation results. We have proven that interactive 3D visualization improves exploration and analysis significantly. The animated 3D view of the chain drive makes it easier the discover phenomena that were obscured before. Perception of position and direction has been enhanced remarkably. Understanding correlations of various attributes is also easier. These are valuable and welcome achievements even for experienced engineers. Our solution can be useful in training novice engineers and communicating simulation results to professionals not working directly with simulation. Finally, the capability of generating movies showing actual motion and attributes is an important feature when creating presentation.

Future work can extend in multiple directions. Users have already asked for the capability of positioning glyphs at given points in space. However, the simulator computes attributes that are valid at the actual position of the chain link they belong to. Therefore the value these glyphs visualize should be interpolated from the attributes of the chain links in the vicinity of the glyph's location.

There is also interest in integrating the visualization module in Tycon itself to facilitate instant model verification in 3D.

Aesthetic value can be enhanced by attaching surfaces to the bodies that resemble the actual object. This is not very straightforward though, because in Tycon no such model of the bodies exists.

Currently we visualize raw simulation results only. Visualizing derived attributes and states such as vibration would probably open new possibilities in the application.

Finally, we are also planning to introduce similar glyph-based visualization for flexible body simulation.

Acknowledgements

We would like to thank AVL List GmbH (`www.avl.com`) for their support and help. Parts of this work were carried out in the scope of applied and basic research at the VRVis (`www.VRVis.at`) which is funded by an Austrian governmental research program called K plus. We also thank Robert S. Laramee of VRVis and Wolfgang Hellinger, Mark A. Mitterdorfer, Martin Sopouch and Christian Vock of AVL for their valuable contributions and feedback. All simulation data sets displayed in this paper appear courtesy of AVL.

References

3D Studio Max, http://www.discreet.com.

AVL List GmbH, http://www.avl.com.

BARAFF, D. 1989. Analytical Methods For Dynamic Simulation Of Non-penetrating Rigid Bodies. *Computer Graphics 23*, 3, 223–232.

DE LEEUW, W., AND VAN WIJK, J. 1993. A Probe For Local Flow Field Visualization. In *Proceedings Visualization '93*, IEEE Computer Society Press, Los Alamitos, CA, G. Nielson and R. Bergeron, Eds., 39–45.

HOFFMAN, D., AND DOWLING, D. 1999. Modeling Fully Coupled Rigid Engine Dynamics And Vibrations. In *SAE Noise & Vibrations Conference & Exposition*.

LEVKOWITZ, H. 1996. Perceptual Steps Along Color Scales. *International Journal of Imaging Systems and Technology 7*, 97–101.

MIRTICH, B., AND CANNY, J. F. 1995. Impulse-based Simulation Of Rigid Bodies. In *Symposium on Interactive 3D Graphics*, 181–188, 217.

MIRTICH, B., AND CANNY, J. F. 1996. Hybrid Simulation: Combining Constraints And Impulses. In *Proceedings of First Workshop on Simulation and Interaction in Virtual Environments*.

NIEMANN, G., AND WINTER, H. 1986. *Maschinenelemente*. Springer-Verlag.

PARK, J., AND FUSSELL, D. S. 1997. Forward Dynamics Based Realistic Animation Of Rigid Bodies. *Computers and Graphics 21*, 4 (–), 483–496.

POST, F., VAN WALSUM, T., POST, F., AND SILVER, D. 1995. Iconic Techniques For Feature Visualization. In *Proceedings of the 6th IEEE Visualization Conference*, IEEE Computer Society Press, 288–295.

SAUER, J., AND SCHÖMER, E. 1998. A Constraint-based Approach To Rigid Body Dynamics For Virtual Reality Applications. In *Proceedings of the ACM symposium on Virtual reality software and technology 1998*, ACM Press, 153–162.

TUFTE, E. R. 1986. *The Visual Display Of Quantitative Information*. Graphics Press.

VAN WALSUM, T., POST, F., SILVER, D., AND POST, F. 1996. Feature Extraction And Iconic Visualization. *IEEE Transactions on Visualization and Computer Graphics 2(2)* (June), 111–119.

Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data

Helmut Doleisch, Martin Gasser, Helwig Hauser

VRVis Research Center, Vienna, Austria
http://www.VRVis.at/vis/
Doleisch@VRVis.at, Martin.Gasser@VRVis.at, Hauser@VRVis.at

Abstract

Visualization of high-dimensional, large data sets, resulting from computational simulation, is one of the most challenging fields in scientific viualization. When visualization aims at supporting the analysis of such data sets, feature-based approches are very useful to reduce the amount of data which is shown at each instance of time and guide the user to the most interesting areas of the data. When using feature-based visualization, one of the most difficult questions is how to extract or specify the features. This is mostly done (semi-)automatic up to now. Especially when interactive analysis of the data is the main goal of the visualization, tools supporting interactive specification of features are needed.

In this paper we present a framework for flexible and interactive specification of high-dimensional and/or complex features in simulation data. The framework makes use of multiple, linked views from information as well as scientific visualization and is based on a simple and compact feature definition language (FDL). It allows the definition of one or several features, which can be complex and/or hierarchically described by brushing multiple dimensions (using non-binary and composite brushes). The result of the specification is linked to all views, thereby a focus+context style of visualization in 3D is realized. To demonstrate the usage of the specification, as well as the linked tools, applications from flow simulation in the automotive industry are presented.

1. Introduction

Visualizing high-dimensional data resulting from computational simulation is a demanding procedure, posing several complex problems which include, for example very large size of data sets and increased dimensionality of the results. In this paper, we present a formal framework that supports interactive and flexible analysis of complex data using a descriptive and intuitive language for defining features and multiple linked views with information visualization (InfoViz) and scientific visualization (SciViz). In the following, we shortly discuss a few key aspects, which are important for the new approach presented in this paper.

Feature-based visualization – visualization which focuses on essential parts of the data instead of showing all the data in the same detail at the same time, is called feature-based visualization. This kind of visualization gains increasing importance due to bigger and bigger data sets which result from computational simulation, so that not all of the data can be shown simultaneously. For feature-based visualization, proper feature extraction methods are essential.

Up to now, feature extraction mostly is done (semi-)automatically[14] with little interactive user intervention, often as a preprocessing step to the visualization. But for interactive analysis, in many cases, the question of what actually is (or is not) considered to be a feature refers back to the user: depending on what parts of the data the user (at an instance of time) is most interested in, features are specified accordingly. Therefore, flexible feature extraction requires efficient means of user interaction to actually specify the features.

Separating focus and context in InfoViz – when dealing with large and high-dimensional data sets in InfoViz, simultaneous display of all the data items usually is impossible. Therefore, *focus-plus-context* (F+C) techniques are often employed to show some of the data in detail, and (at the same time) the rest of the data, at a lower resolution, as a context for orientation[3, 6]. Thereby the user's attention is di-

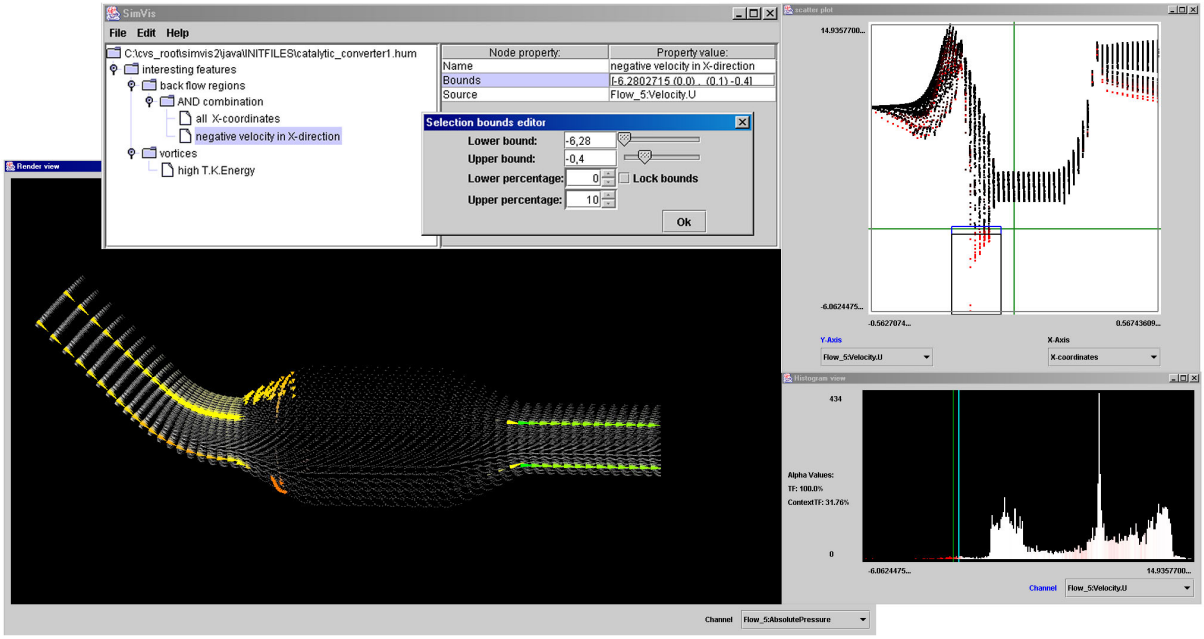


Figure 1: Flexible Feature Specification: *simulation data of a catalytic converter is shown, two features have been specified based on our feature definition language, using the different views for interaction and visualization. (see also colorplate)*

rected towards the data in focus (e.g., through visual enlargement), whereas the rest of the data is provided as context in reduced style (translucently, for example). This is especially useful when interacting with the data, or when navigating through the visualization.

To discriminate data in focus from context information, a so-called *degree of interest* (DOI) function can be used[6], assigning a 1D DOI-value out of the unit interval to each of the n-dimensional data items (1 represents “in focus”, 0 is used for context information).

Defining the DOI function – in literature, implicit techniques for DOI-specification are described (e.g., focus specification through dynamic querying[15]) as well as explicit techniques, such as interactive object selection[9] or brushing[1, 17]. When brushing, the user actively marks a subset of the data set in a view as being of special interest, i.e., in focus, using a brush-like interface element.

In addition to standard brushing, several useful extensions to brushing have been proposed. Multiple brushes and composite brushes[12], and the use of non-binary DOI functions for smooth brushing[4] extend the available toolset for interactive DOI specification. Also, more complex brushes like those designed for hierarchical data[5], or such using wavelets[18] or relative information between different data channels[8] have been proposed recently.

Complex and high-dimensional feature definition – when analyzing simulation data, one very often encountered problem is the limited flexibility of current brushing and interaction techniques. Brushing is usually restricted to simple combinations of individual brushes, as well as missing support of high-dimensional brushes due to the tight coupling of GUI interactions and the representation of the brush data itself. For fast and flexible analysis of the usually large and high-dimensional simulation data, complex and also high-dimensional brushes are necessary. In this paper, we present a formal framework, that is very closely coupled to the data, allowing to define and handle such brushes interactively.

Linking multiple views – the combination of InfoViz and SciViz methods[7, 4], especially for the interactive visualization and analysis of simulation data, improves the understanding of the data in terms of their high-dimensional character as well as the data relation to the spatial layout. Linking several views[2] to interactively update all changes of the data analysis process in all views simultaneously is a crucial property for making optimal use of multiple (different) visualization views.

In previous work[4] we showed how a scatterplot (or a histogram) can be used to smoothly specify features in multi-dimensional data from simulation, and how this focus+context discrimination can be used for selective visualization in 3D. In this paper, we now present a formal framework (our feature definition language, see section 2)

for specifying features in simulation data together with advanced interaction techniques (see section 3), allowing for fast and flexible exploration and analysis of complex and high-dimensional data (application examples in section 4). Finally, a short overview about implementation details is given, as well as conclusions and some future work topics are presented.

2. Using a Feature Definition Language

When dealing with results from computational simulation, usually very large and high-dimensional data sets are investigated. Previous work already showed, that interactive specification of features with tight reference to the actual data attributes is very valuable for visualization of such data sets[7, 4]. For a fast and flexible analysis of these results, powerful and intuitive tools are needed – the here described approach provides flexibility in terms (a) of multiple options to differently view the data, and (b) a wide range of user interactions to construct and adapt feature specifications. Whereas previous work mainly focussed on viewing (a) so far, we mostly improve on interaction (b) in this paper.

To generalize the specification of features (enabling feature descriptions which are portable between data sets, for example) and to also formally represent the state of an analysis session, e.g., to allow for loading/saving of interactive visualization sessions, we present a compact language for feature specification, i.e., a *feature definition language*, here called FDL for short.

```
<FSpecData>:     Reference to data,
                 <FeatureSets>          // mult. feat. sets possible
<FeatureSets>:   <FeatureSet>+          // only one active at a time
<FeatureSet>:    <Feature>+             // implicit OR of features
<Feature>:       <FeatureChar>+         // implicit AND of f.-chars.
<FeatureChar>:   <SimpleFChar>          // RxR-map (one channel->DOI)
                 || <ComplexFChar>      // induces recursion
<SimpleFChar>:   Reference to channel,  // direct data access
                 mapping2DOI-info       // info for DOI calc.
<ComplexFChar>: AND <FeatureChar>+      // AND-comb. of subsequ.chars.
                 || OR <FeatureChar>+   // OR-comb.  --"--
                 || NOT <FeatureChar>   // NOT of subsequent char.
```

Figure 2: Feature definition language: sketch of its structure.

A sketch of the FDL-structure is presented in Fig. 2. Here the different key components of this language are shown, namely the feature specification itself (root), feature sets (level 1), features (level 2) and feature characteristics (level 3). In the following subsections, these four different hierarchical layers of the FDL are discussed in more detail.

2.1. Feature Specification

A description of a feature specification usually is closely coupled to a data set (the one that is to be analyzed). Alternatively, it could also be portable to similar data sets, when data semantics coincide. In the regular case, a feature specification therefore has a reference to the source data set, as well as to one or multiple feature sets (see below).

Our FDL is realized as an XML[13] language application, which makes it easy to handle and the resulting FDL-files readable. This also allows to save feature specifications as files, and load them again at any later point in time to resume an analysis session. Additionally, XML-files can be edited using a text-editor, which allows to re-adjust feature specifications also on a file level.

The explicit representation of feature specifications in the form of FDL-files makes using feature specifications on other data sets possible. Of course, care has to be taken that only data channels are referred to, which are available in all these data sets. With portable feature specifications it is possible to generate general feature definition masks, which can be applied very easily (and interactively adapted, if necessary).

2.2. Feature Sets

A feature set subsumes an arbitrary number of features which all are to be shown simultaneously (like an implicit logical OR-combination). Within each single view, always only one feature-set is used for F+C discrimination, all the other feature-sets are inactive at that time. Multiple feature sets can be used to interactively switch foci during an analysis session or to intermediately collect features in a "repository" feature set, not used at a certain point in time. Multiple views can be used for simultaneously showing different feature sets (one per view).

2.3. Features

Features are specified by one or multiple feature characteristics. The DOI function related to each feature is built up by an (implicit) AND-combination of all DOI functions of all associated feature characteristics. Multiple features are used to support named feature identification and intuitive handling of interesting parts of the data by the user. Each feature can be moved or copied from one feature set to any other.

In Fig. 1 two distinct features have been specified, one denoting areas of backflow, and another one, showing vortices. The latter one consists only of one simple feature characteristic (see below), brushing high values of turbulent kinetic energy, whereas the first feature consists of a logical combination of two separate feature characteristics.

2.4. Feature Characteristics

Feature chararateristics can be either simple or complex. Whereas simple feature characteristics store direct brushing information with respect to one data attribute (or channel) to derive a DOI function, complex feature characteristics imply a recursion.

Simple feature characteristics store a reference to the data channel which it is based on, as well as infomation about

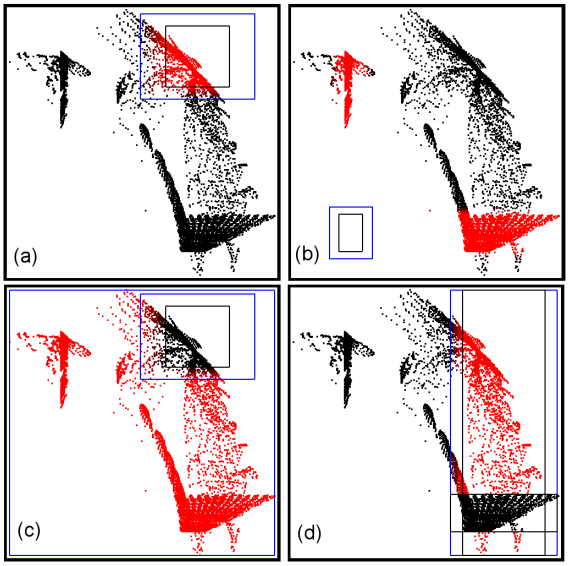


Figure 3: four examples of 2D brush types which users found useful during interactive analysis (catalytic converter example, pressure [x] vs. velocity [y]): *(a) "high velocity and high pressure" (logical AND), (b) "low velocity or low pressure" (log. OR), (c) "all but high vel. and high pressure" (NOT-AND), and (d) "high pressure but not low velocity" (SUB = AND-NOT). (see colorplate for shades of red)*

how the data of this channel is mapped to a DOI function (being the output of this characteristic). Especially the possibility for the user to directly interact with the data attributes by specifying feature charcateristics and modifying them interactively is very intuitive and straight-forward. In Fig. 1 a simple feature characteristic named "negative velocity in X-direction" is shown in the selection bounds editor. Simple feature characteristics support discrete and smooth brushing (via specifying percentages of the total brushing range, where the DOI-values decrease gradually).

Complex feature descriptions on the other hand provide logical operations (AND, OR, NOT) for the user to combine subsequent feature characteristics in an arbitrary, hierarchical layout. For combining smooth brushes, which can be interpreted as fuzzy sets, fuzzy logical combinations are used, usually implemented in form of T-norms and T-conorms[11]. We integrated several different norms for the above mentioned operations. By default, we use the minimum norm (T_M) in our implementation: this means, when doing an AND-operation of several values, the minimum value is taken, and for the OR-operation the maximum respectively.

In Fig. 3, four examples of 2D brush types, which users found useful during interactive analysis sessions, are shown. The data displayed in the scatterplot views comes from the catalytic converter application shown in Fig. 1 (which is also explained in more detail in section 4), pressure (x-axis) vs. velocity (y-axis) values are plotted. Interactive operations "NOT-AND" and "SUB" are mapped to "NOT"-"AND" and "AND"-"NOT" combinations in FDL, respectively.

3. Interaction

One main aspect of analyzing results from simulation is that investigation is often done interactively, driven by the expert working with the visualization system. Therefore, interaction is one of the key aspects that has to be considered when designing a system which should support fast and flexible usage (as described previously). Especially the task of searching for unknown, interesting features in a data set, and extracting them, implies a very flexible and intuitive interface, allowing new interaction methods. In the following subsections, we categorize the main types of interaction which our system supports. Note that these interactions are designed to meet users' most often requested requirements for such an analysis tool.

3.1. Interactive Feature Specification through Brushing

The first type of interaction that has to be considered when designing an interactive analysis tool for exploring simulation data, is *brushing*. In our system, interactive brushing of data visualization is possible in all views except for the 3D SciViz view, which is used for 3D F+C visualization of the feature specification results (see section 4). Brushing is used to define feature characteristics in the FDL interactively. As many types of applications also request non-binary brushing, we allow smooth-brushing[4] in all the interaction views. One example of using a 2D smooth brush, employing a logical AND operation of two simple feature characteristics is shown in Fig. 3 (a). Here, a region of relatively high velocity and high pressure values is brushed in a scatterplot view, defining (a part of) a feature. As can be seen from this figure, a smooth brush defines two regions. A core part of the brush is defined, where data of maximal interest is selected (mapped to DOI values of 1). It is padded by a border, where DOI values decrease gradually with increasing distance from the core part.

3.2. Interactive Feature Localization

Another very often used type of interaction is the so-called feature localization. It is usually provided in the context of simulation data, that has some spatial context. When analyzing this kind of data, the first interest is often, *where* features of specific characteristics are located in the spatial context of the data. Interactively defining and modifying features in different views, coupled with linking, the specification immediately results in a 3D rendering which provides fast localization of the features in the spatial context of the whole data set. For an example see Fig. 4 (a)-(c), where the backflow regions are interactively localized to be in the entrance of the catalytic converter chamber.

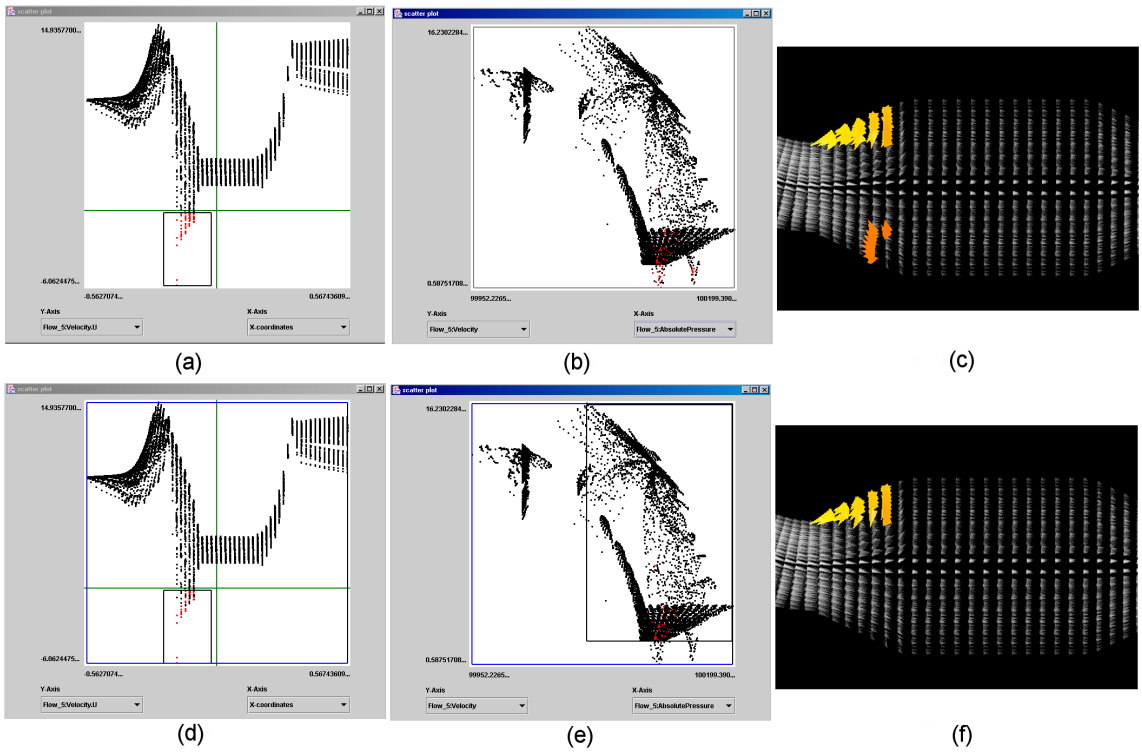


Figure 4: Interactive feature specification and refinement: *(a)-(c): first step: defining backflow region in a catalytic converter (see also Fig. 1) in a scatterplot view (a) by selecting negative x-flow values, direct linking to a second scatterplot view (b) and the 3D view (c). (d)-(f): second step: AND-refinement with a new selection in the second scatterplot view (e), back linking of the interaction via feedback visualization (color of points according to newly calculated DOI values) to the first scatterplot view (d). Now only the backflow region is selected, that exhibits general velocity above a specified threshold (f).*

3.3. Interactive FDL Refinements

After having defined multiple features via brushing and localized them, often interactive refinement of these features is the next step. Refining the feature specification can be either done by interactive data probing (see below) or by imposing further restrictions on the feature specifications, e.g., by adding additional feature characteristics to the actual state of a feature. One example of such an interactive FDL refinement is shown in Fig. 4 (d)-(f). As a first step (first row), all parts of the data, that exhibit backflow, have been selected, defining a feature that spans over two distinct regions in the spatial domain. In the refinement step (second row) a logical AND-combination of the first feature specification (a) with a new selection in a second scatterplot view of the same data (but showing two other data attributes) is performed (e). Thereby only those back-flow regions of the data are put into focus, which exhibit a general velocity above a specified threshold (f).

3.4. Interaction with Tree Viewer

Interaction with a tree viewer (see Fig. 1, left upper window) as a GUI for FDL is another very useful way to adapt or extend feature sets and features, as well as their characteristics. The tree viewer provides standard GUI elements, such as textfields for manual input of numbers or range sliders, for example. Naming of the different nodes of the FDL, as well as editing all the feature characteristics, and also the management of the tree structure (through copy, delete, or move of the different nodes and subtrees) are the most often used interaction methods in this viewer. It strongly depends on the nature of users of whether mouse-interactions or keyboard-input are preferred when specifying features. Sometimes, in the case of well-known thresholds, for example, the keyboard-input to the tree viewer is faster and more accurate then mouse-interaction to an InfoViz view.

3.5. Interactive Data Probing

Another form of interactively exploring features is using a data-probing approach. Thereby, after having specified a feature (via brushing, for example) the one or other feature characteristic can be changed interactively (e.g., by using a range slider). In all linked views (showing the same data and showing different data attributes) immediate feedback of DOI changes can give new insights into different data as-

pects. Especially for exploratively investigating value ranges and better understanding of associated patterns in the data sets, this interaction metaphor is very useful.

3.6. Interactive Management of Views

One key aspect of a system which provides multiple, different views of one data set, is the interactive management and linking of these views. Our system supports an arbitrary number of InfoViz views (currently scatterplots and histograms), as well as SciViz views. Views can be opened and closed at any point in time without distracting the feature specification. In the InfoViz views, the mapping which assigns data channels to the axes can be changed interactively. In the 3D SciViz view the mapping of a data attribute to rendering properties (color and/or opacity) via transfer functions can be interactively modified, too. Additionally, the different axes of all available views can be linked (and unlinked) interactively, allowing rapid updates in multiple views.

4. Visualization and Results from Applications

After having discussed our feature specification framework as well as the important role of interaction for analysis of simulation data, now the visualization part and typical applications are presented.

Below general aspects of visualization during analysis are presented. Then, two different application examples are described in detail. For high quality versions of the images presented here, as well as for additional examples and movies which illustrate the interactive behaviour of working sessions with our framework, please refer to *http://www.VRVis.at/vis/research/fdl-vis/* .

4.1. Visualization for Analysis

When visualization is used to support analysis of large, high-dimensional data sets, the use of multiple views, as well as of flexible views (with respect to data dimensionality) is very important. Our system supports an arbitrary number of each type of InfoViz views, as well as SciViz views. When interactively working with data, two types of views in a multiple views setup can be distinguished: *Actively linked views* are the views, which are primarily used for interaction purposes, i.e., for specifying the features, whereas *passively linked views* are primarily used for F+C visualization of the data, providing interactive updates.

3D SciViz views – The 3D SciViz views of our system are used as passively linked views for providing a F+C visualization and interactive feature localization. The F+C discrimination is mainly accomplished by using different transfer functions for focus and context parts (and interpolating inbetween, for smooth F+C discrimination). The transfer functions in use do not only specify color and opacity, but also the size of the glyphs, that are used to represent single data items (see Fig. 1, lower left window for a 3D SciViz view, showing a smooth F+C visualization).

Two main tasks of this F+C visualization can be identified. The support for feature localization and the visualization of data values through color mapping. Feature localization, as already described in section 3, plays a major role in interactive analysis based on features. By using a F+C visualization, the user attention is automatically drawn to the more prominently represented foci, i.e., the features. Value visualization is another very useful task of visualization in this view, and it is accomplished by coloring glyphs according to the associated data channel.

Of course, interactive user manipulation of rendering parameters (opacity, size of glyphs, or zoom and rotate) are necessary, very useful, and support the analysis task, too.

InfoViz views – Apart from supporting interaction, the InfoViz views (scatterplots & histograms in our system) are very valuable for visualization purposes, too. They visualize the data distribution (1D or 2D) and also give visual feedback of F+C discrimination. Points in the scatterplot views, for example, are colored according to the DOI value of the associated data item. Fully saturated red points are shown for data in focus, whereas the saturation and lightness of points decreases with decreasing DOI values, respectively (see Fig. 3 for examples).

In the InfoViz views it is especially useful that the mapped data attributes can be changed interactively. Mapping spatial axis information to one of the scatterplot axes, for example, is very intuitive in our applications (see below). Additionally, using several scatterplots, comparable to a (reduced) scatterplot matrix, often adds information about the data and internal relations of different data attributes.

4.2. Results from Air-Flow Analysis

We now want to give a step-by-step demonstration of how a typical analysis session takes place, especially to show the importance of interaction when analyzing simulation data.

(1) In a first step, a data set is loaded: in our example, results from air-flow simulation around a car (just on one central slice, from front to back of the car) are shown. To also cope with 2D-slices of 3D-data, we adapted our 3D-rendering view accordingly. It should be noted, that the general flow direction in this application is in X-direction, past the car from front to back. Before a tree viewer is opened automatically, an empty feature set is generated for preparation of an analysis session. A SciViz view is then opened interactively, to show the general spatial layout of the data (see Fig. 5 for the initial view setup). In this figure the unstructured grid of the data set is shown, overall velocity information is mapped to color (green denotes low, red relatively high velocity values).

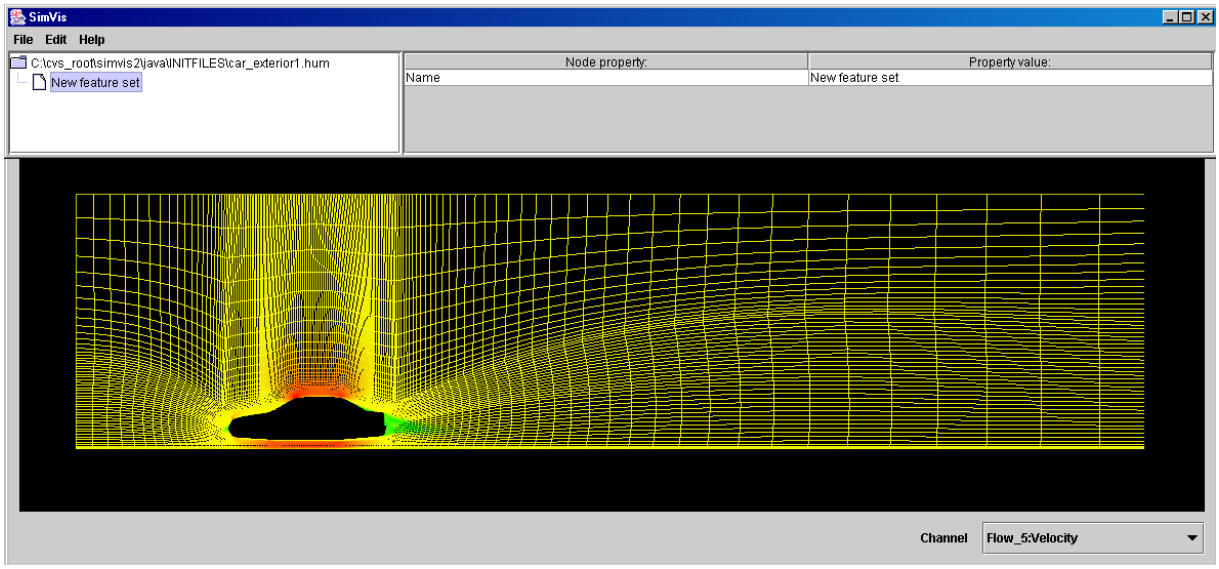


Figure 5: Air-Flow around a moving car: *After loading the data set, an empty feature set is created, and the spatial layout of the data is shown, overall velocity information is mapped to color (green denotes low, red high velocity).*

(2) As a first start into feature specification (focussing on non-horizontal, slow flow at this step of the analysis) a scatterplot view is opened, showing V-velocity (vertical component of overall velocity values), mapped to both axes. In this scatter plot an OR-brush is used to select relatively large positive V-flow, as well as relatively large negative one, too. Then the x-axis of the scatterplot view is changed to show overall velocity and an AND-refinement is done to limit the feature specification to slow flow (see Fig. 6, upper right view).

To furthermore visualize the feature specification up to this step, a second scatterplot view is opened, showing feature and context distribution with respect to the spatial X-coordinates and viscosity (mapped to y-axis of the view, see Fig. 6, lower right). In an interaction panel of the tree viewer, the restriction of V-velocity components is further adapted, to meet the user's needs (see Fig. 6 for a screen capture after this step).

(3) A further AND-refinement, restricting the feature specification to "high viscosity" values is added by using the second scatterplot view. As a result of this step, only features behind the car are part of the new focus (see Fig. 7).

(4) Yet another AND-refinement, further restricting the feature specification to high values of turbulent kinetic energy (a value also computed by the simulation), is performed in the tree viewer (see Fig. 8). This clips away parts of the previously selected features, leaving only the parts that exhibit stronger rotational behavior.

(5) To get a better idea of the vortical structures induced, interactive probing on one part of the feature specification (positive V-velocity) is performed. When limiting the focus to negative V-flow only, the downfacing parts of the upper as well as of the counterrotating, lower vortex become visible (see Fig. 9).

4.3. Results from Catalytic Converter Analysis

A second example presented here is an application, where the data comes from a simulation of a catalytic converter from automotive industry. The results of another analysis session are shown. The data is given on an unstructered grid in 3 spatial dimensions, and has 15 different data attributes for each of the approximately 12000 cells of the grid.

The data set and a corresponding feature specification is shown in several views in Fig. 1. The data set consists of basically three spatially distinct parts, the flow inlet on the left hand side, the chamber of the catalytic converter (middle), and the flow outlet on the right-hand side (see Fig. 1, left lower window for a 3D SciViz view). The other views shown in Fig. 1 include: the tree view for handling the FDL (including a pop-up window for changing the brush properties on the x-component of the velocity), a scatterplot view (right upper window) plotting x-velocity vs. x-coordinates for each data point, and a histogram, showing the distribution of x-velocity values over the data range.

Two distinct features have been specified using the InfoViz views and the FDL tree viewer. The first feature defines all backflow regions in the data set (with negative x-component of the velocity, as general flow is in x-direction). Two such regions are identified at the entrance of the chamber, a weaker one at the bottom of the catalytic converter,

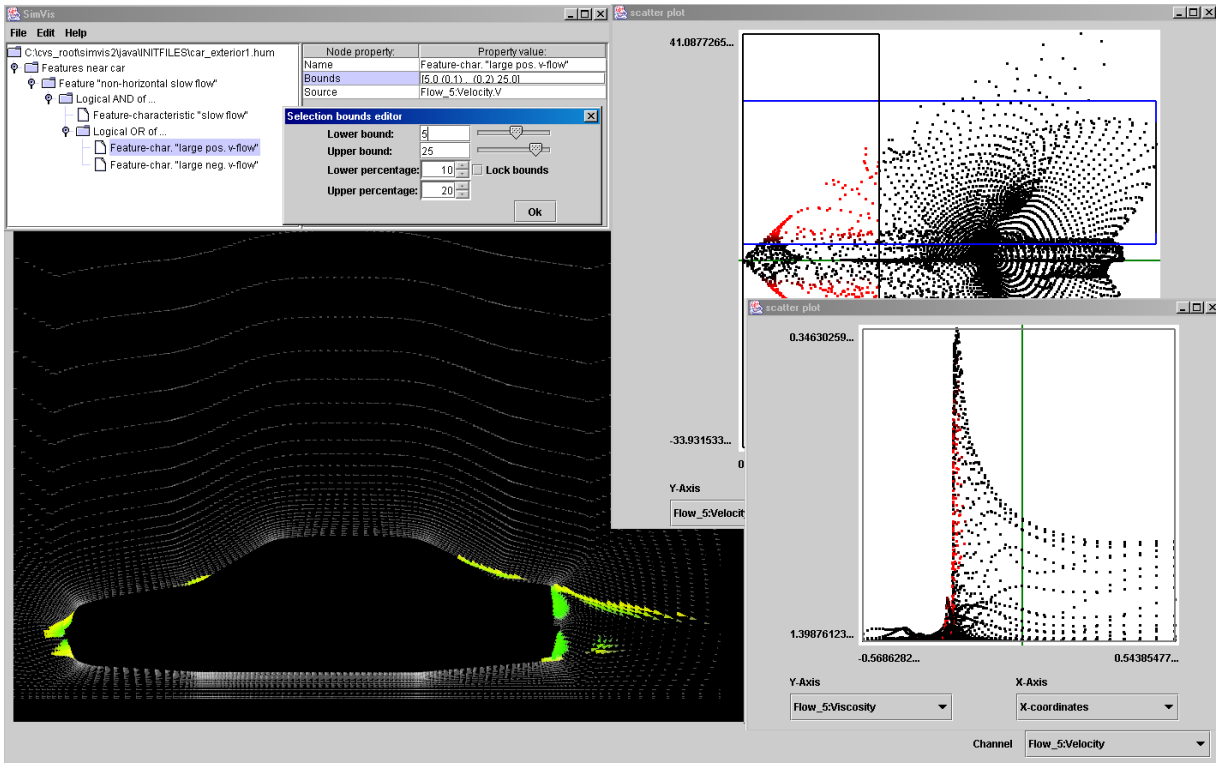


Figure 6: First step of analysis (non-horizontal slow flow): *a tree viewer showing the current feature specification in the upper left (interaction panel for adjusting a simple feature characteristic shown), a scatterplot view used for feature specification in the upper right (velocity vs. V-Velocity component), the SciViz view for f+c visualization in the lower left, a second scatter plot for visualization of f+c distribution (X-coordinates vs. viscosity).*

and a stronger one at the top. The second feature description defines all regions, with high turbulent kinetic energy, these are the regions, where vortices are appearing usually in the flow. As can be seen, two vortex cores are easily separated from the rest of the data at the inlet and outlet of the catalytic converter in this case.

Both, the vortices and the backflow regions have been brushed smoothly, to show some information about the gradient of the values in the 3D rendering view. Note, that the coloring in the 3D view is mapped from another data channel, namely data values of absolute pressure. This allows to visualize an additional data dimension for all the data, that was assigned to be in focus beforehand. In the here applied color mapping, green denotes relative low values of absolute pressure, and red corresponds to relative high values.

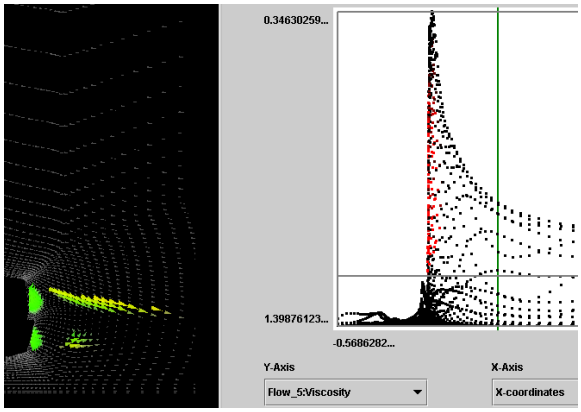


Figure 7: Step 2 of analysis: *AND-refinement, restricting feat. spec. to high viscosity values in the second scatterplot view. Only features behind the car are part of focus now.*

5. Implementation

The presented prototype system includes the described simulation data analysis tools and runs interactively on a standard PC (P3, 733MHz, 756MB of memory, GeForce2) for the data sets shown (in the range of 20.000 to 60.000 cells, 15 to 50 data attributes associated to each cell). The cells of the data are organized in unstructured grids. For the rendering of these grids a visibility algorithm was implemented,

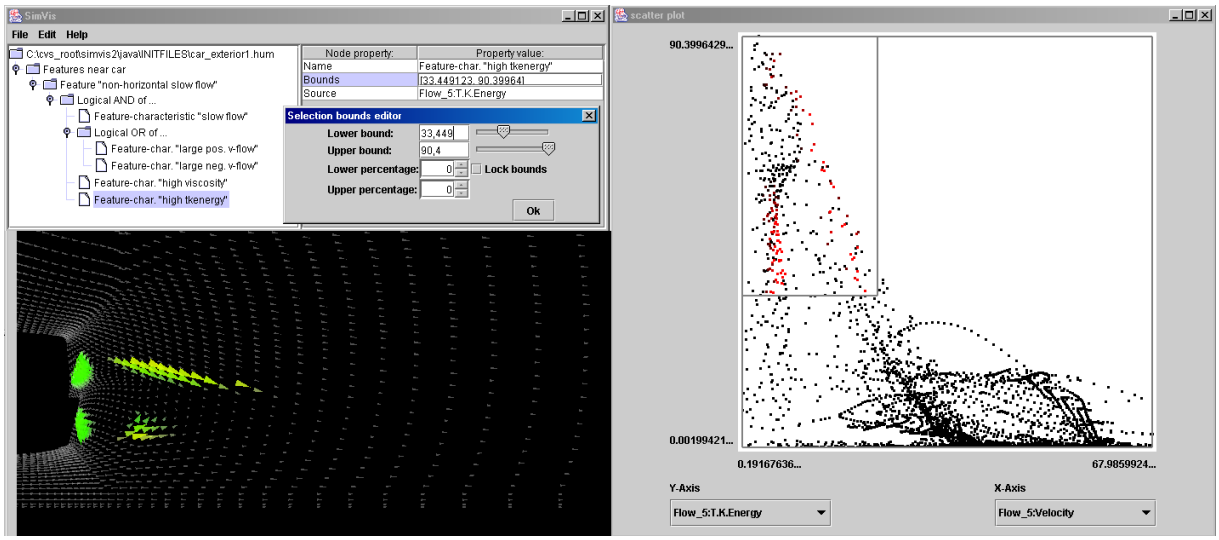


Figure 8: Step 3 of analysis: *another AND refinement, further restricting to high values of turb. kinetic energy, performed in the tree viewer. Only parts with strong rotational component are in focus. (see also colorplate)*

based on the XMPVO algorithm[16] presented by Silva et al. With newer, more powerful PC-setups we already managed to visualize data sets consisting of over a million data cells, but sorting for 3D rendering can not be performed interactively anymore.

For the implementation of our prototype, a hybrid approach was taken; UI Interaction and handling of the FDL is realized in Java, whereas mesh access and the rendering of the visualization views is implemented in native code (we used MS Visual C++). Native methods are called via the JNI API, and the gl4java package was used to make the GL rendering contexts available to the Java GUI toolkit. The mesh access has been realized by using our own data mesh format. Data coming from different data sources can be easily converted to this format via linked readers.

When designing the presented FDL, several considerations were taken, including for example: ease of implementation (close to the visualization sytstem and the data), allow for manual input by the user (preferably ASCII-based, with semantics), verification should be possible (to check for invalid definitions), and many more. To meet all these design considerations, is was decided to use the XML language[13] for storage of the FDL and as interface to other applications. For writing and reading feature specifications to and from FDL-files, the Apache Crimson parser (delivered with the SUN Java SDK) is used, but any other validating XML Parser could also be used. We use a *DTD* (Document Type Definition) for the verification of the FDL trees. The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

6. Conclusions and Future Work

We presented a framework for flexible and interactive, high-dimensional feature specification for data coming from computational simulation. For analyzing simulation data, a feature-based F+C visualization is a good approach, to cope with the data sets' large and high-dimensional nature and to guide the user and support interactive analysis. For F+C visualization interactive focus specification is very useful, if real-time updates of multiple linked views are available. Actual features in simulation data often only are captured with a complex type of specification (hierarchical specification, multiple data channels involved). This is why we believe, that using a simple language to define features hierarchically, namely our feature definition language, helps to extract and manage features during an interactive analysis session. In combination with using multiple InfoViz views (for data examination and feature specification) and SciViz views (for F+C visualization of the interactively extracted features) it is a very useful approach.

Future work will include extensions of the here presented FDL as well as of the analyzing tools. A parallel coordinates view[10] which has been developed earlier[8] can already be used passively to visualize the high-dimensional data, and will be integrated fully in the very near future, as well as new (hardware- and software-based) volume rendering techniques will be included, too. FDL extensions will mainly deal with including the views setting and couple it more tightly with the feature specification tree, as well as time-dependent issues. Currently only steady simulation data can be visualized and the logical next step will be, to enhance the FDL as well as all the corresponding visualization and inter-

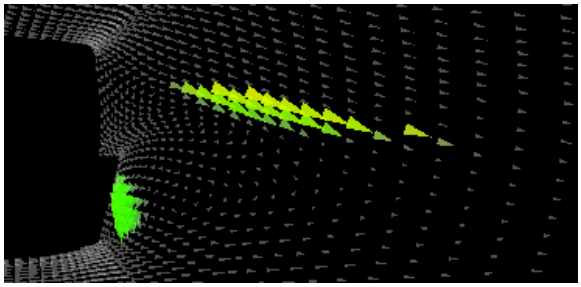


Figure 9: Step 5 of analysis: *interactive probing of V-velocity reveals different behavior of vortical structures, only downfacing parts are shown here.*

action views to cope with time-dependent data sets. Feature specification for time-dependent data sets will be one of the key-issues of future research.

Acknowledgements

This work has been carried out as part of the basic research on visualization at the VRVis Research Center in Vienna, Austria (http://www.VRVis.at/vis/), which partly is funded by an Austrian research program called Kplus. All data presented in this paper are courtesy of AVL List GmbH, Graz, Austria.

The authors would like to thank Robert Kosara, for his help with preparing this paper. Special gratitude goes also to our collegue Markus Hadwiger, who helped with parts of the implementation of the underlying mesh-library system, and the collegues from the Software Competence Center in Hagenberg, Austria, who helped with their knowledge about fuzzy sets and fuzzy combinations.

References

1. R. Becker and W. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.
2. Andreas Buja, John A. McDonald, John Michalak, and Werner Stuetzle. Interactive data visualization using focusing and linking. In *Proc. of IEEE Visualization '91*, pages 156–163.
3. S. Card, J. MacKinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers, 1998.
4. Helmut Doleisch and Helwig Hauser. Smooth brushing for focus+context visualization of simulation data in 3D. In *Proc. of WSCG 2002*, Plzen, Czech Republic.
5. Ying-Huey Fua, M. O. Ward, and E. A. Rundensteiner. Structure-based brushes: A mechanism for navigating hierarchically organized data and information spaces. *IEEE Trans. on Visualization and Computer Graphics*, 6(2):150–159, 2000.
6. George W. Furnas. Generalized fisheye views. In *Proc. of the ACM CHI '86 Conf. on Human Factors in Computing Systems*, pages 16–23, 1986.
7. D. L. Gresh, B. E. Rogowitz, R. L. Winslow, D. F. Scollan, and C. K. Yung. WEAVE: A system for visually linking 3-D and statistical visualizations, applied to cardiac simulation and measurement data. In *Proc. of IEEE Visualization 2000*, pages 489–492, 2000.
8. H. Hauser, F. Ledermann, and H. Doleisch. Angular brushing of extended parallel coordinates. In *Proc. of IEEE Symp. on Information Visualization*, pages 127–130, 2002.
9. H. Hauser, L. Mroz, G. I. Bischi, and E. Gröller. Two-level volume rendering. In *IEEE Transactions on Visualization and Computer Graphics*, volume 7(3), pages 242–252. IEEE Computer Society, 2001.
10. A. Inselberg and B. Dimsdale. Parallel coordinates: a tool for visualizing multidimensional geometry. In *Proc. of IEEE Visualization '90*, pages 361–378.
11. E. P. Klement, R. Mesiar, and E. Pap. *Triangular Norms*, volume 8 of *Trends in Logic*. Kluwer Academic Publishers, Dordrecht, 2000.
12. A. Martin and M. O. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proc. of IEEE Visualization '95*, pages 271–278.
13. Webpage of the World Wide Web Consortium on XML. See URL `http://www.w3.org/XML/`.
14. F.H. Post, H. Hauser, B. Vrolijk, R.S. Laramee, and H. Doleisch. Feature extraction and visualization of flow fields. In *Eurographics State of the Art Reports*, pages 69–100, 2002.
15. Ben Shneiderman. Dynamic queries for visual information seeking. Technical Report UMCP-CSD CS-TR-3022, Department of Computer Science, University of Maryland, College Park, Maryland 20742, U.S.A., January 1994.
16. C. Silva, J. Mitchell, and P. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *Proc. of IEEE Symp. on VolVis '98*, pages 87–94, 1998.
17. M. O. Ward. XmdvTool: Integrating multiple methods for visualizing multivariate data. In *Proc. of IEEE Visualization '94*, pages 326–336.
18. Pak Chung Wong and R. Daniel Bergeron. Multiresolution multidimensional wavelet brushing. In Roni Yagel and Gregory M. Nielson, editors, *Proc. of the Conf. on Visualization*, pages 141–148, Los Alamitos, October 27–November 1 1996. IEEE.

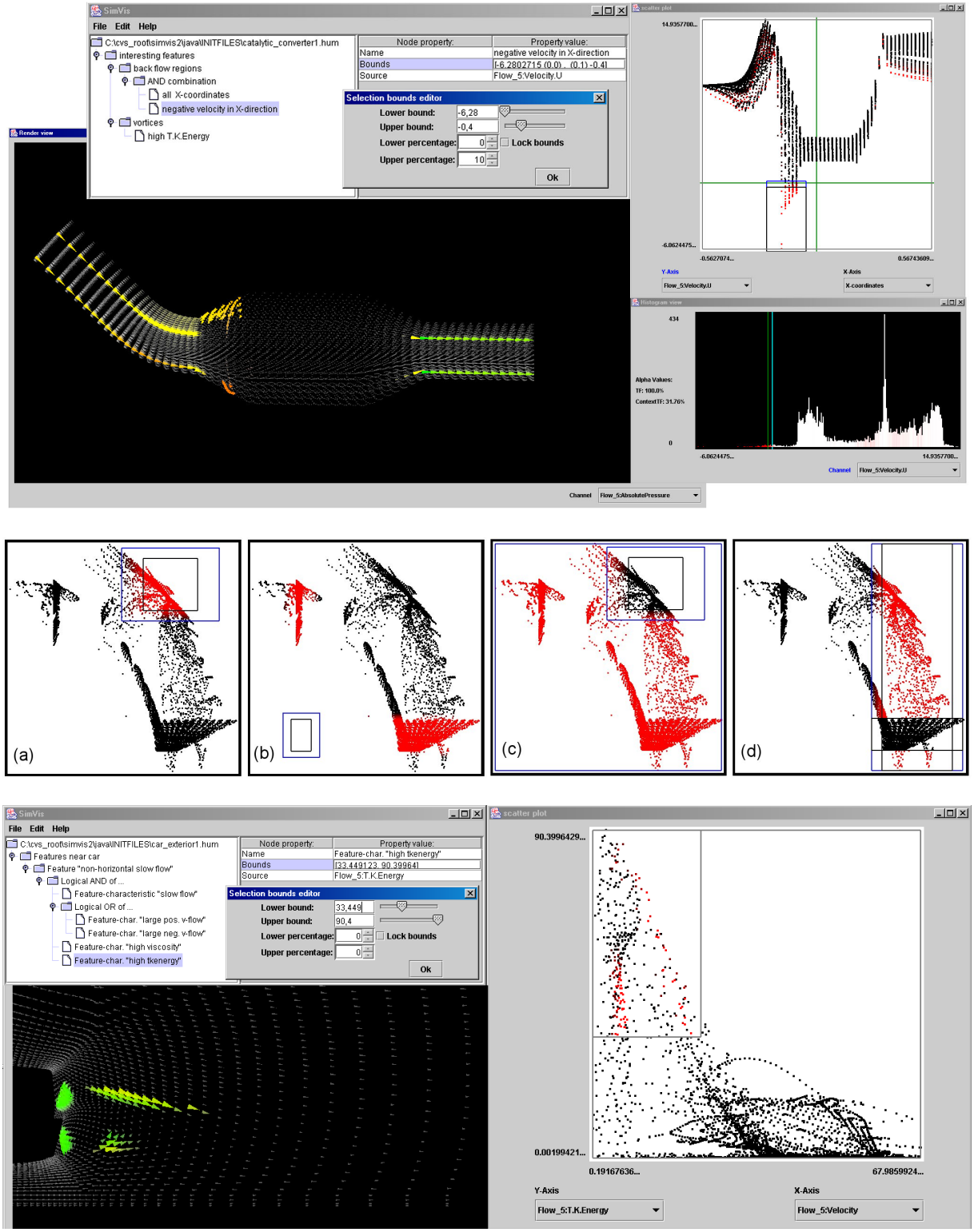


Figure 10: Two examples for feature-based flow visualization using our framework for interactive feature specification and four illustrations of different combination modes for smooth brushes (middle row) *(for verbose captions see figures 1, 3, and 8).*

Useful Properties of Semantic Depth of Field for Better F+C Visualization

Robert Kosara Silvia Miksch
Vienna University of Technology, Vienna, Austria
http://www.asgaard.tuwien.ac.at/
{rkosara,silvia}@asgaard.tuwien.ac.at

Helwig Hauser
VRVis Research Center, Vienna, Austria
http://www.VRVis.at/vis/
Hauser@VRVis.at

Johann Schrammel Verena Giller Manfred Tscheligi
Center for Usability Research and Engineering (CURE), Vienna, Austria
http://www.cure.at/
{schrammel, giller, tscheligi}@cure.at

Abstract

This paper presents the results of a thorough user study that was performed to assess some features and the general usefulness of Semantic Depth of Field (SDOF). Based on these results, concrete hints are given on how SDOF can be used for visualization. SDOF was found to be a very effective means for guiding the viewer's attention and for giving him or her a quick overview of a data set. It can also very quickly be perceived, and therefore provides an efficient visual channel.

Semantic Depth of Field is a focus+context (F+C) technique that uses blur to point the user to the most relevant objects. It was inspired by the depth of field (DOF) effect in photography, which serves a very similar purpose.

Categories and Subject Descriptors (according to ACM CCS): I.3.36 [Computer Graphics]: Methodology and Techniques; H.5.2 [Information Interfaces and Presentation]: User Interfaces

1. Introduction

Like few other areas in computer science, visualization involves the user as the most important part. No matter how good a visualization technique is in terms of its computational cost, its clever design, or the pretty pictures it produces – if it does not convey information to the user efficiently and effortlessly, it is useless. Visualization therefore lacks elegant, formal proofs for its methods, and instead requires the "dirty work" of user studies and psychological tests to asses which methods and techniques are useful, and which are not. Such studies have been neglected in the past, but the awareness of the need of proper evaluation of methods is slowly growing[3, 5, 10].

This paper reports the results of a study that was performed to evaluate a new method called Semantic Depth of Field. We also present conclusions we drew about how and where this method can be used.

1.1. Semantic Depth of Field (SDOF)

Semantic Depth of Field (SDOF)[6, 7] is a focus+context (F+C) technique that uses selective blur to make less important objects less prominent, and thus point out the more relevant parts of the display to the user (e.g., certain chess figures in Figure 1). It is based on the depth of field (DOF) effect known from photography and cinematography[8], which depicts objects sharply or blurred depending on their distance from the lens. This is used to guide the viewer's attention, and is quite effective and intuitive. SDOF extends this effect to decide for every object whether to display it sharply or blurred, not based on geometry, but on the object's current relevance.

We measure blur as the diameter of a circle over which the information from one pixel is spread when it is blurred. Thus, a blur diameter of 1 means a perfectly sharp image, with larger values creating more and more blurred depictions.

Because blur is generally known to be slow in computer graphics, we developed a fast implementation that uses texture mapping on commodity graphics hardware[6] to make interactive applications possible on state-of-the-art PCs.

1.2. Study Goals

The overall goal of the study was to find out if SDOF is an effective means of guiding the user's attention, and if it supports the user in applications.

Effectiveness was assessed in two ways: a) by testing the ability to preattentively (i.e., very quickly and without serial search, see Section 3) detect and locate objects, as well as estimate the percentage of sharp objects; and b) by comparing search times for different cues, i.e., sharpness versus color and orientation (Section 4), and also checking for the interplay between SDOF and those other cues. We also tested the thresholds necessary to tell different blur levels apart, and also the blur levels necessary for an object to appear sharp or blurred (Section 5). Applications [7] were also tested, but are not presented here because of space constraints.

The following section presents some high-level results we obtained from the study; the sections after it go into the details – they first present the hypothesis to be tested, then the test method, and finally the results. Technical details of the study (sample, etc.) are given in the appendix.

2. Results – How to Use SDOF

The following points are the key findings of our study:

- SDOF can be used to quickly and effectively guide the user's attention.
- SDOF makes it possible to discriminate between a small number (about two to four) of object groups.
- Interaction is very important, because people do not like looking directly at blurred objects (if they do so, the application is badly designed).
- SDOF enables the user to get a quick overview of data by letting him or her ask questions quickly and efficiently.
- Blur levels have to be chosen carefully. For normal viewing conditions, we found a blur of 7 pixels too small, and a value of 11 pixels sufficient.
- Things that don't need to be blurred shouldn't be.

3. Preattentivity

Preattentive processes take place within about 200 ms after a stimulus is presented[1, 4, 9], and are performed in parallel, without the need for serial search. Such processes involve a limited set of features (e.g., orientation, closure, color, proximity, etc.) for which certain tasks (e.g., detection, location, count estimation, recognition of groups, etc.) can be performed with ease. Using preattentive features for visualization makes the information easier to see in order to get an overview. Especially methods for pointing out information have to make the relevant objects immediately stand out.

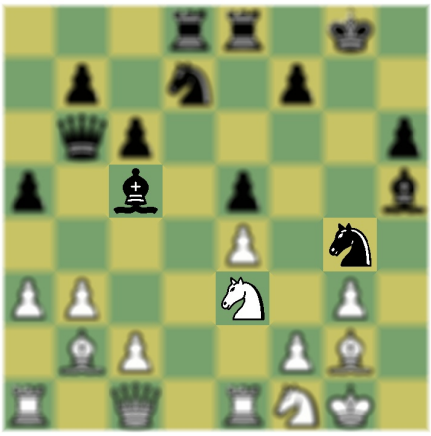


Figure 1: *Chess board application, with the chessmen threatening the knight on e3 in focus (from Kosara et al.[6])*

Experience suggests that sharp objects can be preattentively recognized among blurred ones: depth of field is a very effective means in photography and also cinematography, where the eye can be guided from one object to the other with focus changes. And blur is also present in the human eye, which also only has a limited depth of field (like a camera lens), but we hardly notice that – we simply ignore blurred areas.

3.1. Test Procedure

We tested two preattentive abilities: being able to detect and locate a sharp object, and being able to estimate the percentage of targets among distractors.

The images for target detection and location showed ellipses whose main axes were horizontal, and which were scattered over the image (Figure 2b). The reason for choosing ellipses was that we needed objects that would not change their shape drastically when blurred to rule out shape perception effects. Ellipses seemed perfect for this, because they don't change, and they can also be rotated (which was needed in the interplay trial, Section 4). Participants were shown images with 3, 32, or 63 distractors, with or without a target (50% with, 50% without a target) and one of the seven combinations of three different blur levels (7, 11, and 15 pixels) – resulting in 42 different combinations. For each combination, participants were shown five images (randomly picked from 30 generated ones), resulting in 210 images per participant.

The test procedure consisted of four steps (Figure 2a): First, an empty screen was shown for 300 ms, followed by the image, which was shown for 200 ms. After that, the answer screen was presented, which gave the participant the

choice between clicking on one of the four quadrants or buttons for “no target” and “target not locatable”. After the answer was given, a screen with the German word for “Next” was shown, which required a key-press to continue with the next iteration. This was done to provide the participants with a means to control the speed of the test. Additionally, a screen encouraging the subject to take a short break was shown after every 30 images. For percentage estimation, the sequence was identical, except that the answer screen contained only three buttons for the estimated number of targets: “few” (up to 19 targets), “intermediate” (20 to 45) and “many” (more than 45 targets). The images shown in this trial only used one blur level per image, and always contained 64 objects, with 5% to 95% of targets (in steps of 10%), and the rest distractors (Figure 2c).

3.2. Results

Finding sharp targets among blurred distractors is indeed performed preattentively. Figure 3a shows the accuracies for correct location of targets, which were very high ($> 90\%$) or high ($> 60\%$) depending on the blur level. When the lowest blur level (7 pixels) was present, the accuracy dropped significantly – this is most likely due to the fact that participants were not able to differentiate between sharp and slightly blurred objects. There is also a significant difference in accuracy between the cases with three distractors and those with 32 or 64 (Figure 3b), which was to be expected. Accuracies were almost identical for cases with and without targets, only for the case with only the smallest blur level present, it was much higher in the no-target case. This is most likely due to the participants not being able to distinguish between the slightly blurred distractors and the sharp target, and thus not finding it – so stronger blur than 7 pixels is needed (this is also quite apparent from Figure 3c).

Estimation of the percentage of sharp objects can also be done preattentively. The accuracy for all blur levels is significantly better than chance. When analyzed by number of targets, the accuracy is lowest close to the borders of the intervals (“few”, “many”, “intermediate”), and slightly higher on the low and high end than in the middle – which is not surprising, because for these numbers, the participants can make the decision more easily. The dependence on blur levels is weaker than for target location, and does not differ significantly between the lowest one and the stronger two. For the smallest blur level, more objects were perceived as sharp (Figure 3d), which led to more errors.

The results clearly show that SDOF is an effective method that can draw the user’s attention to objects quickly. Getting a first idea of data (e.g., in a scatter plot) seems also possible. The smallest blur level (7 pixels) clearly was too small for these viewing conditions, because it seriously impeded the subjects’ performance. Proper parameterization of the method for the user’s viewing conditions is therefore necessary.

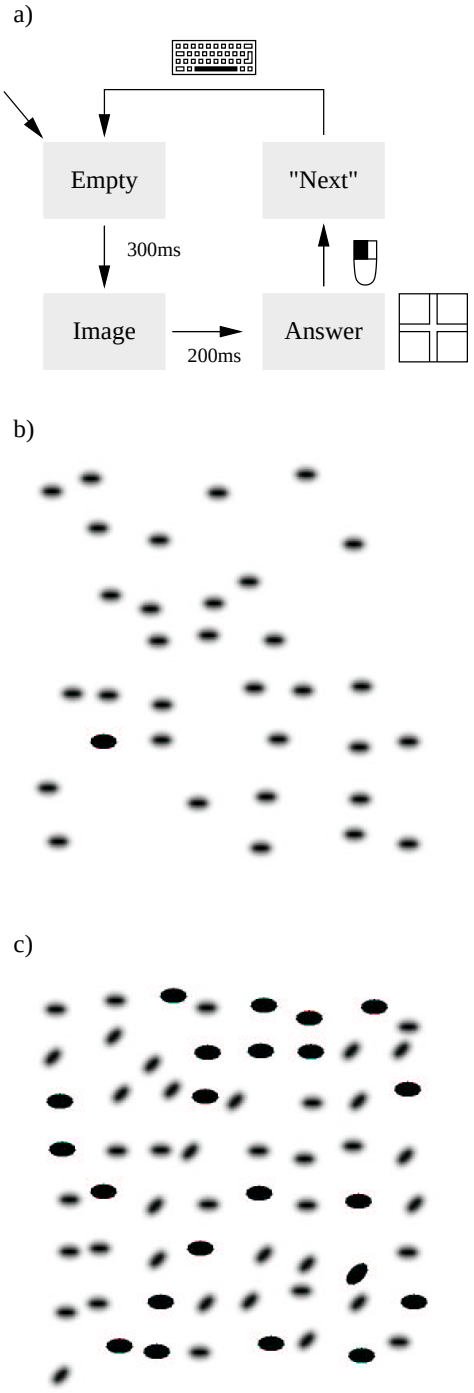


Figure 2: *Test sequence and sample images.* ***a)*** *The sequence of screens for testing preattentive location of objects (Section 3) and interplay (Section 4);* ***b)*** *Sample image for target detection and location with 32 distractors of the highest blur level, and a target;* ***c)*** *Example image for interplay: Find the rotated, sharp object.*

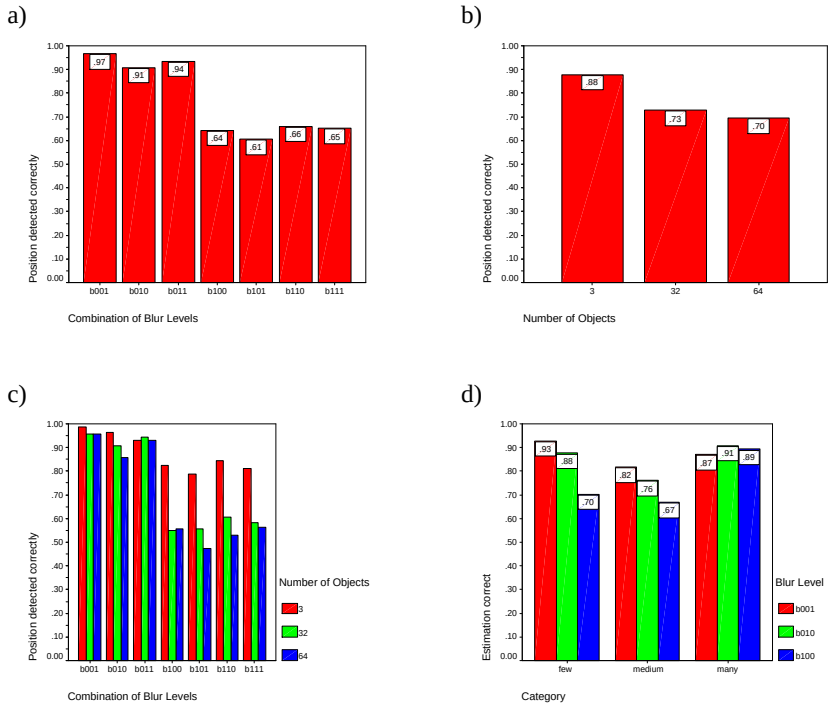


Figure 3: *Results for preattentiveness:* ***a)*** *Ratio of correctly located targets depending on blur levels used (encoding of blur levels see below);* ***b)*** *Ratio of correct answers by number of objects;* ***c)*** *Correct answers by blur level and number of objects;* ***d)*** *Ratio of correct estimations by blur level and number of targets.* ***Encoding of blur levels:*** *for each of the three blur levels, a 1 indicates that it is present, and a 0 that it is not. So for "b011", the lowest blur level was not present, the higher ones were.*

4. Interplay

SDOF will very likely not be used without any other visual cues, which is why we were interested in its interaction with other features; for this test we selected color and orientation.

4.1. Test Procedure

For this part, images similar to the ones used for the preattentiveness test were used, with the additional properties color (red, black) and orientation (main axis horizontal or at 45°).

The user interaction was similar to the first blocks, only this time subjects could look at the image as long as they wanted to find the answer – they were, of course, encouraged to answer as quickly as possible.

We tested *simple*, *disjunctive*, and *conjunctive* searches. Simple searches are based on the presence of one feature in the target, with the distractors not being different from one another. In a disjunctive search, the subjects looked for one feature in the targets, but the distractors could also differ in another one (e.g., if the red object is the target, all distractors were black but could be sharp or blurred). Conjunctive searches required the participant to look for a combination of two features in the targets (e.g., the red sharp object), while the distractors could have any other combination of the two.

4.2. Results

In terms of search time, SDOF is not significantly worse than color – this is perhaps the most interesting and surprising finding of this study. There is no significant difference between a simple search for colored or for sharp objects (Figure 4a). The conjunctive searches for color and blur, orientation and blur, and color and orientation differ significantly from each other, with color and orientation being the slowest – each of these two features combined with SDOF is faster. Also, the conjunctive search for color and blur is not significantly slower than the simple and disjunctive searches, which is quite contrary to what we expected, because conjunctive searches usually are slower[9].

Search times were longer when no target was present (Figure 4b), which is not surprising, because it takes longer for subjects to make sure they have not overlooked a target[2]. The total number of errors in this block was only 10 (i.e., less than 0.7%) for the whole test (90 images per participant, 1440 in total). This shows that subjects took the tests seriously, and did not sacrifice accuracy for speed.

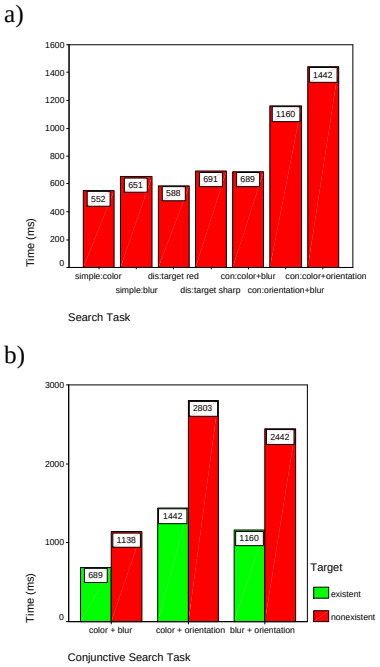


Figure 4: *Results for interplay of features.* ***a)*** *time needed for search when target present ("simple": only look for one feature, with no other feature present; "dis": disjunctive search for one feature with two distractor features; "con": conjunctive search for combination of features;* ***b)*** *search times for conjunctive search by search task and existence of target.*

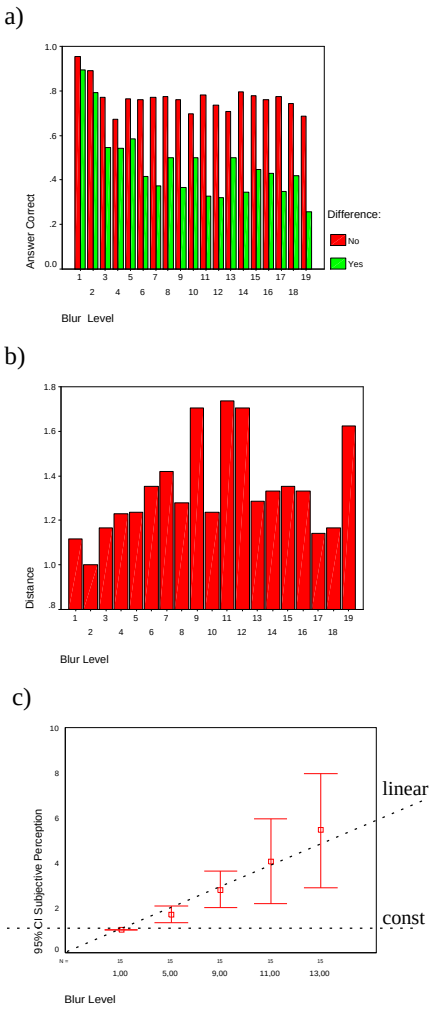


Figure 5: *Results for blur perception.* ***a)*** *Correct answers for identical ("no") and different ("yes") objects, by blur level;* ***b)*** *Distance needed to detect difference, by blur level;* ***c)*** *Numerical answer to perception of absolute blur value, by displayed blur value (error bars for 95% of values).*

5. Blur Perception

One aspect of SDOF we were planning was to use it as a fully-fledged, separate visualization dimension that could be used in addition to the existing such as space, color, etc. In order to do this, we needed to assess the minimal difference in blur that can be perceived, and the rate at which "steps" in blur are perceived. Our original hypothesis was that there would be an exponential relationship between the blur level and the perceived blur (similar to the way luminance is perceived, for example).

5.1. Test Procedure

This test consisted of several parts. In the first, we tested the ability to tell whether or not two objects had the same blur level. For this, we showed the subjects two objects next to each other, with equal or different blur. Subjects had to decide whether the blur was equal or different – if they decided it was equal, the blur of one of the objects was increased (starting with sharp objects), and the objects were shown again. Another part started with strong blur and decreased the blur level. In a third part, participants had to decide, which of the two was sharper, or if they were equal.

We also tested for the absolute thresholds of blur perception, by showing just one object, which was sharp in the beginning and got increasingly blurred until the participant judged it as blurred. This test was also performed starting with a strongly blurred object that got increasingly sharper (until it was perceived as sharp).

In the final part, participants had to tell the perceived relation in blur in terms of a ratio of two numbers. They were free to use any numbers they wanted (i.e., not restricted to "1:x"), which were later normalized. These numbers were given orally, and recorded by the test supervisor.

5.2. Results

SDOF cannot be used as a full visualization dimension. Participants were able to tell the difference between objects of different blur levels (Figure 5a) with a good accuracy (which even stayed quite constant even for strong blur). But they were not able to correctly identify objects of the same blurriness, and did not better than chance for blur over 7 pixels.

The differences in blur needed to tell the blur levels apart (Figure 5b) do not show a clear trend. The distances are quite small (less than 1.8), and overall appear quite constant, which suggests a good differentiation between blur levels – in accordance with the above results. In terms of absolute values, a blur diameter of 3.27 (on average) was already judged a sharp object, when the participant was presented a very blurred object that got sharper; but a blur level of only 1.46 was already judged as blurred when starting out with a sharp object.

When judging the ratio of blur of two objects, subjects reported very small numbers compared to the real ratio (Figure 5c). Their answers also differed very much, so that no clear trend could be made out. This is quite contrary to the above results about being able to differentiate between blur levels. So while subjects were able to see a difference, they were not able to quantify it – another peculiar similarity to color perception.

The quantitative results of this part of the study form a consistent image with the participants' comments, that they disliked having to look at blurred objects and to compare them. It therefore appears to be necessary to make sure that no important parts of the display are blurred, and that the user can switch to a different view, or back to a completely sharp image at any time.

6. Conclusions and Future Plans

This study has shown that SDOF is, indeed, an effective and efficient method for guiding the user's attention. We were surprised to find the similarities with color (even though they were not significant), which we had not expected. We now also have some data for parameterization of SDOF for the use in standard desktop environments, and can perhaps extend this to other viewing conditions as well.

This study has revealed a lot of interesting information about SDOF, but it was only a first step. We want to continue investigating SDOF properties and parameters in new studies which we want to design based on this one.

Acknowledgments

We would like to thank Peter Fröhlich and Birgit Rabl for their work to make this study possible.

This work is part of the Asgaard Project, which is supported by *Fonds zur Förderung der wissenschaftlichen Forschung* (Austrian Science Fund), grant P12797-INF. Parts of this work have been carried out in the scope of the basic research on visualization at the VRVis Research Center (`http://www.VRVis.at/vis/`) in Vienna, Austria, which is funded by an Austrian governmental research program called Kplus. Furthermore, this work is partly supported by the City of Vienna.

Appendix: The Gory Details

The test setup was designed in workshop sessions between the computer scientists at the University of Technology/VRVis and the usability experts at the Center for Usability Research and Engineering (CURE); the test software was developed by Robert Kosara. All tests were performed in August 2001 in the CURE usability lab in Vienna, Austria.

For significance testing, we used chi-square tests and ANOVAs with Scheffé tests for post-hoc analyzes. All results that are described as significant in this paper were tested for with a probability for error of $p < 0.001$. The base level for the whole study was $p < 0.05$.

To rule out large differences in perception between test participants, and to allow for a rather small sample size due to financial and time constraints, we selected a rather narrow group of participants who all fulfilled the following requirements: male, aged 18–25, very good vision (no contact lenses or glasses), student at university, basic computer knowledge.

The sample size was 16 individuals, which we recruited from different universities in Vienna. Each participant was paid a small amount of money for taking part. Each test session took about two hours.

References

1. Lyn Bartram. Perceptual and interpretative properties of motion for information visualization. In *ACM Workshop on New Paradigms in Information Visulization and Manipulation*, 1997.

2. E. Bruce Goldstein. *Sensation and Perception*. Brooks/Cole Publishing Company, 5th edition, 1998.

3. Christopher G. Healey, Kellogg S. Booth, and James T. Enns. Visualizing real-time multivariate data using preattentive processing. *ACM Transactions on Modeling and Computer Simulation*, 5(3):190–221, July 1995.

4. Christopher G. Healey and James T. Enns. Large datasets at a glance: Combining textures and colors in scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):145–167, April 1999.

5. Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, January-March 2000.

6. Robert Kosara, Silvia Miksch, and Helwig Hauser. Semantic depth of field. In *IEEE Symposium on Information Visualization 2001 (InfoVis 2001)*, pages 97–104. IEEE Computer Society Press, 2001.

7. Robert Kosara, Silvia Miksch, and Helwig Hauser. Focus and context taken literally. *IEEE Computer Graphics & Applications, Special Issue on Information Visualization*, 22(1):22–29, January/February 2002.

8. Hsien-Che Lee. Review of image-blur models in a photographic system using principles of optics. *Optical Engineering*, 29(5):405–421, May 1990.

9. Anne Treisman. Preattentive processing in vision. *Computer Vision, Graphics, and Image Processing*, 31:156–177, 1985.

10. Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000.

SMOOTH BRUSHING FOR FOCUS+CONTEXT VISUALIZATION OF SIMULATION DATA IN 3D

Helmut Doleisch and Helwig Hauser

VRVis Research Center in Vienna, Austria,
mailto:{Doleisch, Hauser}@VRVis.at,
http://www.VRVis.at/vis/

ABSTRACT

We present the usage of a non-discrete degree of interest (DOI) function, obtained by brushing multi-valued 3D simulation data in information visualization views, to define opacity, color, and geometrical transfer functions for 3D rendering in a scientific visualization view via linking. To reflect the smooth nature of features in flow simulation data, smooth brushing was chosen. Different available views and interaction methods of a prototype system are discussed, and examples from 3D flow simulation are shown.

Keywords: F+C Visualization, Linking & Brushing, Information Visualization, Scientific Visualization, 3D Visualization, Transfer Function Modulation, Flow Simulation

1 INTRODUCTION

In this paper we present a new solution for feature-based visualization, which is especially useful for analysis or exploration of simulation data. In our case, the data comes from flow simulation in automotive applications. We are dealing with multi-dimensional and multi-modal data-sets from flow simulation, which are layed out on unstructured grids in two or three spatial dimensions.

In general, simulation data often exhibits a rather smooth distribution of data values along spatial dimensions. This requires special treatment when dealing with feature specification (see later about smooth brushing). As a special feature of the solution described here, a 3D visualization of the data is inherently integrated into our approach.

Dealing with occlusion in SciViz – when rendering truly three-dimensional information in scientific visualization (SciViz), such as, for example, medical data acquired from computer tomography, it is very important to decide *how* to render the data. But even more important, the question of *what* parts of the data should be displayed needs to be addressed [6]. Because of occlusion, not all of the data can be shown concurrently.

In volume rendering, usually a so-called *transfer function* is employed, which assigns an opacity value to each of the data items. A compositing procedure is used, featuring semi-transparency for image synthesis. The design of a transfer function is a difficult challenge, that strongly depends on the specific goals of the visualization process. At the IEEE Visualization conference in 2000, the most recognized approaches have been

discussed in an interesting panel [8].

Separating focus & context in InfoViz – apart from SciViz as discussed above, information visualization (InfoViz) also deals with data-sets of tremendous size and increased dimensionality. Since the simultaneous display of all of the data items usually is impossible, *focus-plus-context* (F+C) techniques are often employed to show some parts of the data in detail, and at the same time the rest of the data as a context for orientation. This is especially useful when interacting with the data, or navigating through the display [2].

To discriminate data *in focus* from context information, a so-called *degree of interest* (DOI) function can be used [4]. It assigns to each of the n-dimensional data items a 1D DOI-value out of the unit interval (1 represents "in focus", 0 is used for context information).

Using a DOI function for opacity modulation – in this paper we demonstrate how the idea of F+C visualization and the use of a transfer function for volume rendering can be combined (through *linking*, a well-known concept from InfoViz, see below). To do so, we use a DOI function that is defined by interactive means on the n-dimensional domain of the simulation data. It is then used as a transfer function for opacity modulation. Thereby, a 3D F+C visualization of flow features is realized: parts of the data which are *in focus*, i.e., the flow features, are displayed rather opaque, whereas the rest of the data (the context) is shown translucent.

This means, that concentrating on features during visualization also allows to improve on the occlusion problem which is inherent to 3D rendering.

Smooth Brushing – for specifying the DOI function on the n-dimensional domain of simulation data, we use an interactive brushing tool which is based on separate InfoViz views. To cope with the rather smooth distribution of simulation data, we use so-called *smooth brushing* which results in a DOI function that continuously maps to the $[0, 1]$ range. The hereby derived DOI function also can be interpreted as a fuzzy set describing a "degree of being *in focus*" [15]. For scientific visualization, the DOI function, as specified in an InfoViz view, corresponds to modulated opacity values in 3D rendering (through linking).

2 RELATED WORK

As the work presented in this paper relates to several different areas of visualization research, this section consists of several parts. A short discussion of the most important approaches is given here, for more detailed information the reader is pointed to the indicated references.

Visualizing n-dimensional data from flow simulation – in general, simulation of flow data yields multiple values per data item such as, for example, pressure, temperature, and velocity. Visualization of all these different values is useful for understanding the results of the simulation, and for enabling the analysis process. Standard 2D solutions (color plots, graphs, etc.), as well as surface-based solutions embedded in 3D (iso-surfaces, for example) are widely available in commercial software products.

On top of standard solutions, Kirby et al. propose simultaneous visualization of multiple values (of 2D flow data) by using a layering concept related to the painting process of artists [9]. In another approach for 2D CFD data, called Linked Derived Spaces [7], Henze uses multiple 2D views (featuring geometrical connectivity), which are linked, and allow discrete brushing – opposed to *smooth* brushing as featured in our approach – in all of the views.

Another related topic is feature-based flow visualization, e.g., detection and visualization of vortices or vortex cores – recent work has been presented by Roth and Peikert [11] as well as Sadarjoen et al. [12].

Linking & Brushing for connecting SciViz & InfoViz – *linking & brushing* is a useful and well-appreciated concept, known from InfoViz. *Brushing* is a process in which the user can interactively highlight, select, or delete a subset of elements with regard to visualization by using some appropiate brushing tool. Often, brushing is associated with *linking*, a process in which brushing some elements in one view directly affects the visual appearance of the data in other (linked) views. Already in 1987, Becker and Cleveland applied linking & brushing to high-dimensional scatter-plots [1].

The principal idea of linking SciViz & InfoViz views has already been demonstrated in a system called WEAVE, by Gresh et al. [5]. In this approach, which deals with data from a heart simulation, (discrete) brushing is performed in InfoViz views – a 3D view is linked via through coloring. However, this system does not feature volumetric rendering based on semitransparency, only surface- and point-based methods are used. Also in the linked derived spaces [7] (see above) it is possible to show the two spatial dimensions of the grid in one of the scatter-plots for 2D scientific visualization.

DOI functions for discriminating Focus & Context – DOI functions (as described above) have been introduced by Furnas in 1986 [4]. Multiple ways of how to define a DOI function have been presented since then, using either explicit or implicit specification. Whereas many F+C solutions build on an explicit specification of the focus, e.g., by pointing at data item of specific interest, others use an implicit and data-driven specification of what is *in focus*. Examples are querying methods as, e.g., used in the XmdvTool [14].

Martin et al. [10] extended the usage of brushes to define a DOI function by several new concepts, including non-discrete brush-boundaries, simultaneous display of multiple (up to four) brushes, and creating composite brushes via logical operators.

3D rendering / dealing with occlusion / F+C solutions – when rendering 3D data with a volumetric approach, the problem of occlusion needs to be solved. In standard 3D rendering, the concept of using an opacity transfer function for this purpose has been well accepted in the field of scientific visualization [8]. A recent approach of volumetric flow rendering was Raycasting Vector Fields presented by Frühauf [3], which directly renders all of the available data elements, and therefore lacks easy spatial perception.

A similar problem of occlusion, due to a lot of data being shown, is apparent in information visualization – the input often comprising very large and high-dimensional data sets. Here the concept of *focus+context visualization* was established [2]. The basic idea of this concept is to enable users to have the object of primary interest presented in detail, while still preserving an overview or context available at the same time.

There are few attempts to utilize F+C solutions also in scientific visualization, one is two-level volume rendering [6], where different rendering techniques can be applied to different objects in a 3D data set, depending on whether they are in focus or not.

3 FOCUS+CONTEXT VISUALIZATION OF SIMULATION DATA BASED ON SMOOTH FEATURES

Inspired by the work of Gresh et al. (WEAVE [5]), and two-level volume rendering [6], we found F+C visualization to be also very useful in scientific visualization. Both approaches use a different visual appearance for data *in focus* vs. the display of context information. In WEAVE, InfoViz views are used to specify data in focus, whereas in two-level volume rendering objects appear *in focus* through explicit selection. While WEAVE uses linked color coding for 3D F+C visualization, two-level volume rendering uses different techniques of (more or less) translucent 3D rendering for F+C display.

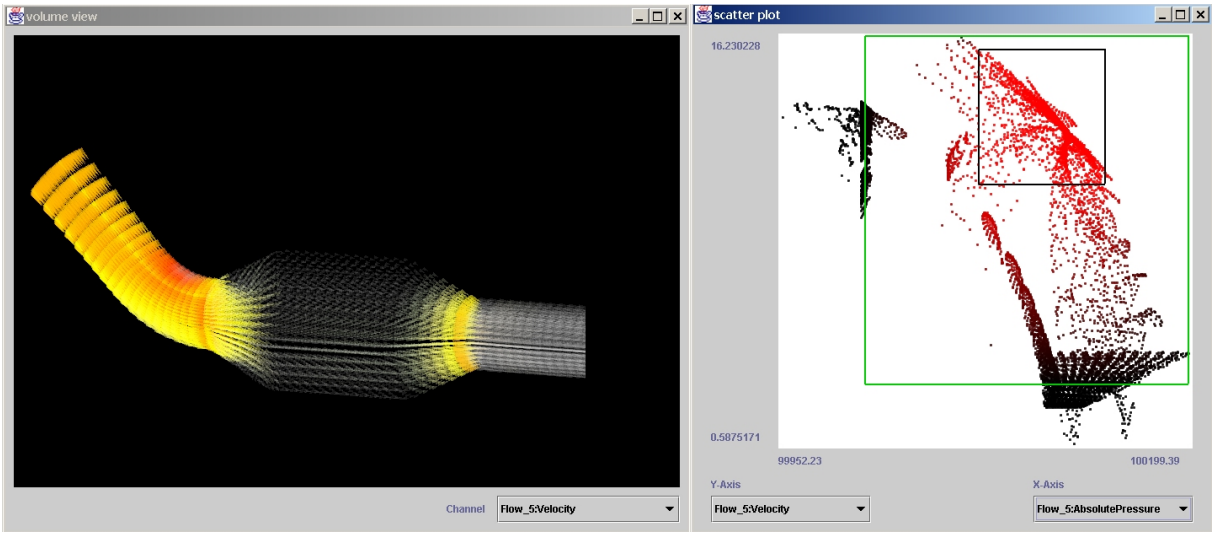


Figure 1: *Smooth Brushing and Linking:* simulation data of a catalytic converter is shown. Right side:a scatter-plot, where a non-binary DOI function was defined by smooth brushing a cluster of high velocity/high pressure data; Left side: a 3D SciViz view, employing opacity/color modulation for the 3D arrows.

We think that a F+C approach in scientific visualization fits together very well with the challenge of solving the occlusion problem in 3D rendering. F+C solutions in InfoViz and the notion of features in feature-based visualization correspond very well, as in both fields a level of interest is specified for special parts of the data. When dealing with 3D rendering of the data, occlusion is an inherent problem to tackle. It can be solved by modulating the opacity of data items in 3D rendering according to a degree of interest function.

Both from InfoViz and WEAVE, we know that the concept of linking & brushing is very useful for discriminating focus and context in visualization. In our approach we use InfoViz views on high-dimensional data from flow simulation to specify the desired DOI function interactively according to different attributes of the data items. This DOI function is linked to the 3D view, showing the features of interest in a F+C style.

As a speciality of data from flow simulation, data values are distributed rather smoothly along spatial dimensions. Different to data from medical visualization, for example, almost no sharp boundaries of (flow) features are given. To cope with this special type of data, we use a non-discrete brushing technique, which we call *smooth brushing*, to specify a continuous DOI function for F+C visualization.

For evaluation of our ideas, we implemented a software prototype that supports different views from SciViz & InfoViz together with smooth brushing in InfoViz views, as well as linking of the 3D view to InfoViz views through the continuous DOI function. In 3D rendering, DOI values influence the opacity of display elements, and/or their color coding, as well as their geometrical properties, e.g., their size.

Fig.1 gives an example of the solution described here, showing an application featuring data which comes from a 3D simulation of flow in a catalytic converter. The left view shows a scientific visualization with small 3D arrows representing data items. Velocity is mapped to color, with red corresponding to high velocity. The right view is a scatterplot where velocity values are plotted against values of static pressure. A non-binary DOI function was defined by smooth brushing a cluster of data items exhibiting high velocity and rather high pressure. This DOI func-

tion was used to modulate the opacity as well as the color of the 3D arrows in the rendering view – display elements not in focus are drawn in a gray-scale fashion. One can see that mainly the inlet of the catalytic converter (upper left part in the 3D view) exhibits values of high pressure and high velocity. Through the smooth brush parts of the outlet are also of partial interest, since values there are not much different to the core focus of this visualization, and thus are also partially influenced by the F+C mapping.

4 VIEWS AND INTERACTION

To test our ideas we implemented a prototype including the two most prominent InfoViz views (scatter-plot and histogram view) and a 3D rendering view with different representation modes.

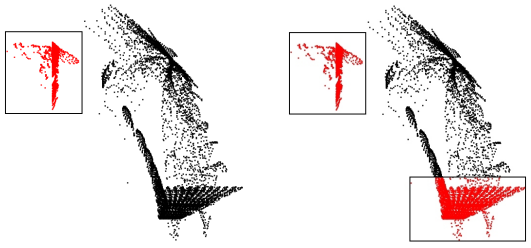
In the scatter-plot view the two attribute-to-axis mappings are interactively configurable, any attribute dimension available in the data set can be used on any of the two axes. Additional feedback is provided by showing zero-axes and numerical output of the boundaries of the data ranges for the two corresponding axes.

An example scatter-plot view is shown on the right side of Fig.1. It visualizes the data distribution of the data items according to two attributes. Clustered regions, but also outliers and trends are easily identified by using this view. Two-dimensional brushing of the data items in the scatter-plot is accomplished by interactively defining a rectangle (with the mouse) and thus creating a binary selection (see Fig.2(a)). To increase flexibility in brushing the data, multiple brushing regions can be defined (see Fig.2(b)), allowing to put distinct data clusters or subsets into focus simultaneously. Future work will include logical combinations of different brushing regions.

As already explained, smooth brushing was the concept of coping with the specialities of data from flow simulation. The boundary region of non-zero DOI values is visualized by a second rectangle enclosing the initial (binary) brushing region. This second rectangle can also be changed interactively, providing the possibility to separately define the region of interpolation of DOI values between 0 (context) and 1 (full focus) for each of the four directions. The DOI value is also encoded in the color of the points representing the data items in the scatter-plot.

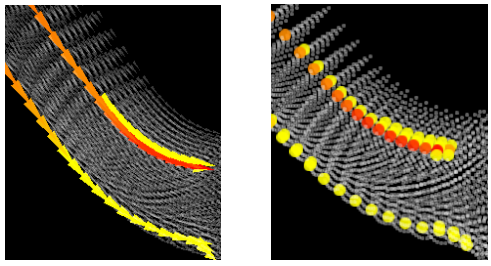
The histogram view (see Fig.5, for example) visualizes the number of occurences per (range of) data item(s) for one attribute, in contrast to the scatter-plot where only the data distribution and not the count is visible. In our approach it is linked to one of the axes of the scatter-plot and thus provides, at the same time, additionally data count information to the distribution information shown. Brushing mechanisms are similar to the scatter-plot, but obviously only 1D.

The histogram view also provides the user with a feedback visualization (color and opacity values) of the two transfer functions



(a) (b)

Figure 2: binary brushing of a cluster (a) and multiple brushing regions (b) in the scatter-plot



(a) (b)

Figure 3: 3D arrows (a) vs. 3D blobs (b), showing vortex core lines for the inlet of the catalytic converter application

employed in the 3D rendering view.

In the 3D rendering view (Fig.1 left side) currently two different rendering modes are available. *3D arrows* (Fig.3(a)) vs. *3D blobs* (Fig.3(b)) for every cell of the grid. The arrows are tetrahedron-like (length and orientation encode magnitude and direction of the flow at the current grid position), the blobs' geometry is a sphere. Opacity and color of the primitives are triggered by the transfer function as explained earlier.

Although volume rendering is currently being developed, the 3D arrows and blobs have the important advantage of giving a one-to-one representation of single elements in the data set, often asked for by special tasks of interest. For both modes the transfer function maps one attribute, in Fig.3 this is magnitude of the turbulence kinetic energy. 3D arrows provide the possibility of conveying additional information, such as direction and magnitude of the flow – that is only reasonable, if detailed information is desired about small regions of the data. When showing an overview, the advantages of the (then too) small arrows get lost, and rendering blobs is the better choice (keeping complexity lower, too).

All the available views are linked via the (non-discrete) DOI function defined in one of the InfoViz views by brushing the data. This DOI function is especially used to modulate the appearance of the 3D rendering for the data elements. Optionally opacity, color, and/or size of geometric objects can be modulated, depending on the F+C discrimination defined by the DOI value.

In the current setup of the prototype, also the attribute to axis mapping of the histogram axis and one of the scatter-plot axes are linked, as mentioned above.

Another feature of the prototype is that the brushing information (DOI function) persists until the next brushing interaction in any of the InfoViz views is performed. This allows to compare different attributes in the 3D rendering view for the same features. Also, analyzing and comparing data distributions in the scatter-plot by changing the attribute to axis mapping with the same DOI feedback visualization per data item is enabled through this concept. The next section discusses an example of this feature.

5 RESULTS AND IMPLEMENTATION

In Fig.4(a) the application of joining two flows (from left and upper pipes) into one (going to the right side) in a so-called T-junction is shown. Fig.4(b) shows the corresponding scatter-plot to Fig.4(a). A low velocity/low pressure area has been brushed smoothly, to focus on the recirculation area of the mixing flows. In the lower two images the currently active attribute to be visualized by the transfer functions in the 3D view and the currently active attribute to axis mapping for the y-axis in the corresponding scatter-plot view was changed to be the turbulence kinetic energy instead of the velocity. Note the persistence of the DOI function as described above.

Fig.5 presents a different example, coming from a simulation of the ventilation system in a car. This time, only a 2D slice of cells from front to back of the car was considered during simulation. The inlet of the ventilation system is the red region on the left side, where maximum velocity can be observed. The data has been brushed in a histogram view on velocity values, shown on the right side. Relatively high velocity values have been brushed smoothly, to specify the DOI function. Again the transfer function of the features in the 3D view is a colored one, red defining highest velocity values and green being associated with relatively small values (which are not in focus and thus not visible here). The context transfer function is a gray-scale one, and the size of the arrows used for the representation of the data elements is also smaller, nearly not visible. Linear interpolation of the alpha and color values, as well as of the size of the arrows is applied according to the DOI values.

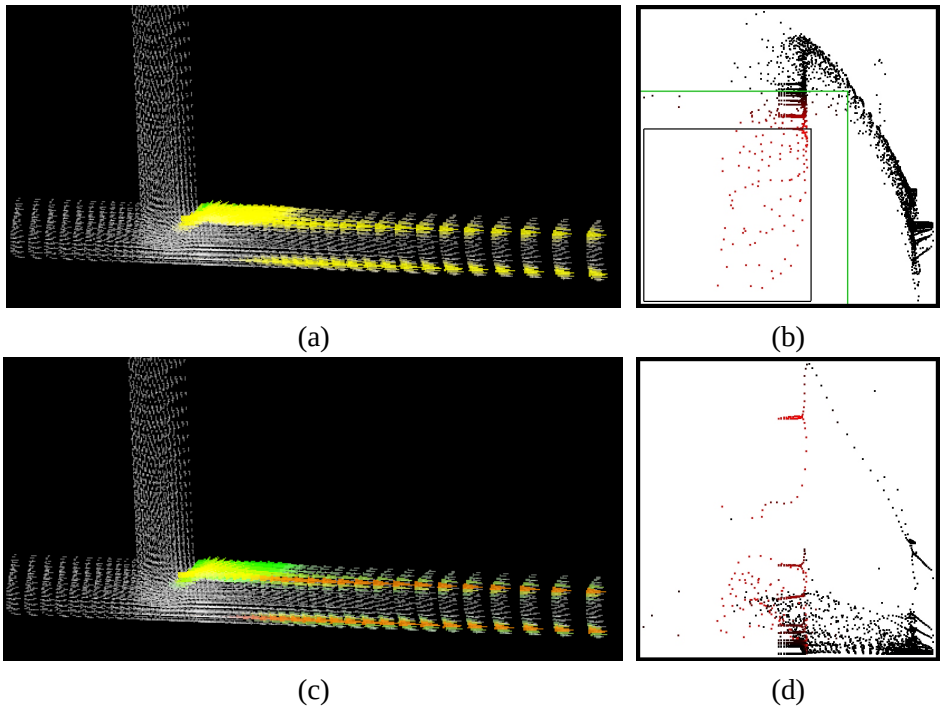


Figure 4: Smooth brushing of the recirculation area in a T-Junction application (a) and (b), changing of the attribute to axis mapping and attribute to transfer function mapping to turbolence kinetic energy on the previously brushed information (c) and (d)

Further results and video-sequences of working with the system are available at www.VRVis.at/vis/research/smooth-brush/.

The presented system has been implemented in a self-developed environment for visualization, called OFVis (Open Framework for Visualization). It is a combination of Java (for graphical UI) and C++ (for data access via libraries of our primary partner company AVL List GmbH), using Open GL for the rendering parts (using the *gl4Java* package). This allows for flexible data access, and possible extensions by using only data access routines that are different for other data sources.

The prototype system presented runs interactively on a standard PC platform (P3, 733MHz, 756MB, GeForce2) for the data sets shown (in the range of 20.000 to 60.000 cells, 15 to 40 data attributes associated to each cell). The cells of the data are organized in unstructured grids. For the rendering of these grids a visibility algorithm was implemented, based on the XMPVO algorithm [13] presented by Silva et al.

Acknowledgements – this work has been done at the VRVis Research Center, Vienna, (www.VRVis.at/vis/), which partly is funded by an Austrian research program called Kplus. All data-sets presented in this paper are courtesy of AVL List GmbH, Graz, Austria. The authors also thank Cl. Silva for his support with 3D rendering. Also, we thank M. Hadwiger, L. Mroz, and R. Kosara for helping with the design of the software prototype and valuable comments on this project.

REFERENCES

[1] R. Becker and W. Cleveland. Brushing scatterplots. *Technometrics*, 29(2):127–142, 1987.

[2] S. Card, J. MacKinlay, and B. Shneiderman. *Readings in Information Visu-*

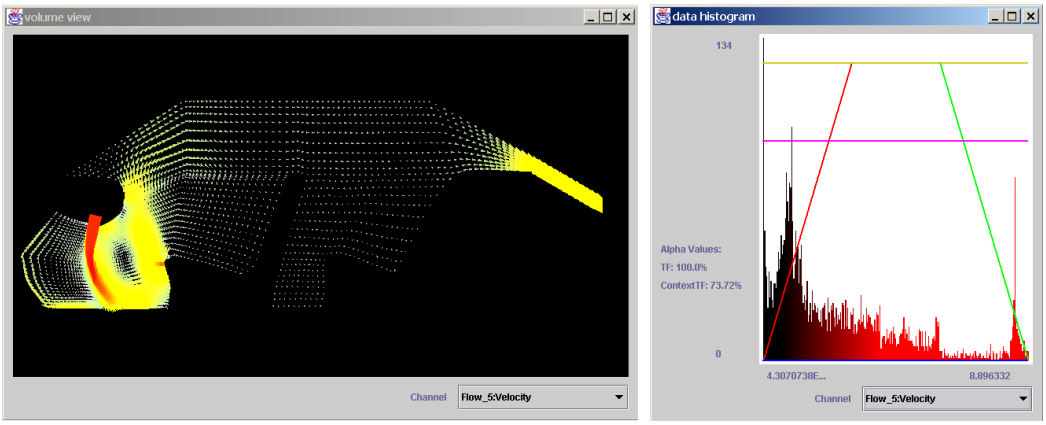


Figure 5: Smooth brushing in the histogram view (right) and linking to the 3D view (left): a 2D slice of the simulation of a car ventilation system, high velocity in focus.

alization: Using Vision to Think. Morgan Kaufmann Publishers, 1998.

[3] T. Frühauf. Raycasting vector fields. In *Proc. of Vis '96*, pages 115–120, 1996.

[4] G. Furnas. Generalized fisheye views. In *Proc. of ACM CHI'86 Conference on Human Factors in Computing Systems*, pages 16–23, 1986.

[5] D. Gresh, B. Rogowitz, R. Winslow, D. Scollan, and C. Yung. WEAVE: A system for visually linking 3-D and statistical visualizations, applied to cardiac simulation and measurement data. In *Proc. of Vis 2000*, pages 489–492, 2000.

[6] H. Hauser, L. Mroz, G. Bischi, and M. Gröller. Two-level volume rendering. *IEEE TVCG*, 7(3):242–252, 2001.

[7] C. Henze. Feature detection in linked derived spaces. In *Proc. Vis '98*, pages 87–94, 1998.

[8] H.P. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Sobierajski-Avila, K. Martin, R. Machiraju, and J. Lee. Visualization viewpoints: The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–23, 2001.

[9] R. Kirby, H. Marmanis, and D. Laidlaw. Visualizing multivalued data from 2D incompressible flows using concepts from painting. In *Proc. of Vis '99*, pages 333–340, 1999.

[10] A. Martin and M. Ward. High dimensional brushing for interactive exploration of multivariate data. In *Proc. of Vis '95*, pages 271–278, 1995.

[11] M. Roth and R. Peikert. A higher-order method for finding vortex core lines. In *Proc. of Vis '98*, pages 143–150, 1998.

[12] A. Sadarjoen, Fr. Post, B. Ma, D. Banks, and H.G. Pagendarm. Selective visualization of vortices in hydrodynamic flows. In *Proc. of Vis '98*, pages 419–422, 1998.

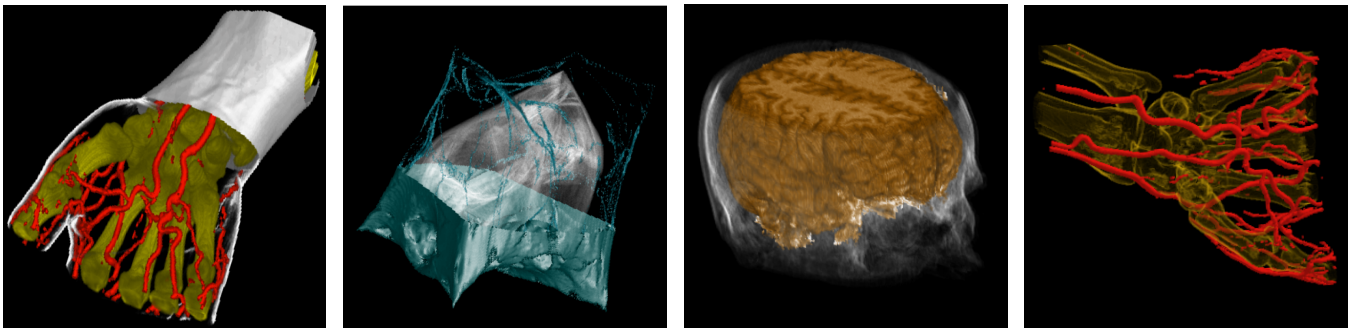
[13] C. Silva, J. Mitchell, and P. Williams. An exact interactive time visibility ordering algorithm for polyhedral cell complexes. In *Proc. of IEEE Symp. on VolVis '98*, pages 87–94, 1998.

[14] M. Ward. XmdvTool: Integrating multiple methods for visualizing multivariate data. In *Proc. of Vis '94*, pages 326–336, 1994.

[15] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

RTVR – a flexible Java library for interactive volume rendering

Lukas Mroz and Helwig Hauser*
VRVis Research Center; Vienna, Austria;
http://www.VRVis.at/vis/



Abstract

This paper presents several distinguishing design features of RTVR – a Java-based library for real-time volume rendering. We describe, how the careful design of data structures, which in our case are based on voxel enumeration, and an intelligent use of look-up tables enable interactive volume rendering even on low-end PC hardware. By assigning voxels to distinct objects within the volume and by using an individual setup and combination of look-up tables for each object, object-aware rendering is performed: different transfer functions, shading models, and also compositing modes can be mixed within a single scene to depict each object in the most appropriate way, while still providing rendering results in real-time. While providing frame rates similar to volume visualization using 3D consumer hardware, the approach utilized by RTVR offers much more flexibility and extensibility due to its pure software nature. Furthermore, due to the memory-efficiency of the data representation and the implementation in Java, RTVR can be used to provide volume viewing facilities over low-bandwidth networks, with almost full control over rendering and visualization mapping parameters (clipping, shading, compositing, transfer function) for the user. This paper also addresses specific problems which arise by the use of Java for interactive Visualization.

Key words: interactive volume visualization, Internet-based visualization, Java

1 Introduction

Volume visualization has proven to be a valuable tool for exploration, analysis, and presentation of data from numerous fields of application, such as medicine, geo sciences, or mathematics, for example. Within the visualization process, interactivity is not only crucial for efficient exploration and analysis of data; the communication of visualization results to a viewer also benefits from the ability to manipulate the visualization output while viewing, especially if complex 3D interrelations have to be understood. Data exploration and interactive presentation with low demands on computational and/or networking resources has been one of the driving factors for the development of the RTVR library.

*Mroz@VRVis.at, Hauser@VRVis.at

Volumetric data-sets contain a variety of structures with different characteristics. With respect to the structure of the data and the goal of the visualization, different visualization techniques are appropriate for different objects to best convey the nature of the data. The most common approaches are surface rendering [12, 13], opacity-weighted blending (direct volume rendering, DVR) [10], or maximum intensity projection (MIP) [23]. The second motivation for the development of RTVR is to provide the user with means for individually selecting the best-suited combination of visualization parameters – transfer function, shading model, and compositing technique – for each object (i.e. part) of a volume, while still providing interactive frame rates.

RTVR integrates and extends several previously published techniques for interactive rendering [4, 8, 16, 15] and efficient data transmission [14] into a flexible framework which can be utilized to provide volume visualization on PC-hardware.

A common approach to the acceleration of software-based volume rendering is to employ techniques for efficiently avoiding the processing (projection) of irrelevant parts of the volume, i.e., empty (transparent) space, or interior parts of entirely opaque objects. Numerous techniques and data structures for this purpose have been published, utilizing, for example, octrees [11], distance volumes [3], or run-length encoding [11]. Based on the observation, that for some compositing techniques, like MIP, parts of the volume which are relevant for the visualization result are strongly intermixed with irrelevant parts [15], a different approach for empty space leaping has been chosen for RTVR. By pre-filtering the volume data, voxels which may be of relevance for the visual representation of objects are identified and stored into a derived data structure – an enumeration of possibly relevant voxels – which is well-suited for fast rendering [15, 16]. Although a similar strategy has been already introduced for shell rendering [22], our approach exhibits several advantages:

- The derived data is always traversed in a sequential (memory-aligned) order during rendering, thus increasing the efficiency of the processor cache.
- Due to the projection technique we use (a fast shear-warp approach), no strictly spatially ordered traversal of the data is required, allowing reordering of data for efficient skipping of entire blocks of irrelevant voxels.

- Extracted voxels can be reordered in a way which allows efficient encoding, compression, and transmission of the data by exploiting spatial coherence. Rendering can be performed without restoring the original arrangement [14].

The above property makes RTVR well-suited for providing volume visualization over low-bandwidth networks, either for interactive presentation of data which has been generated off-line, or within a split client/server approach for on-line visualization. The challenge of interactively presenting images of volumetric data over networks, which also can be manipulated interactively on standard desktop-hardware, has been addressed by several approaches. The simplest way to display objects contained within volumetric data-sets is to extract a polygonal surface representation of the object and to render a sufficiently simplified version of the model at the client (via a VRML browser, for example). Although current consumer 3D hardware is already quite powerful, it is still not possible to render highly detailed models from real-life data-sets at interactive frame rates. To overcome this problem, Engel et al. [7] place the data-set on a server and use progressive transmission and progressive refinement to allow interactive surface extraction and viewing. They also presented an approach for providing direct volume rendering (DVR) at low-end clients [6]. First a small, subsampled version of the data-set is transmitted to the client. During interactions which influence the rendered image, the local copy of the data is rendered using texture-mapping capabilities of consumer 3D hardware. After finishing the interaction, a high-quality rendering of the full-resolution data-set is computed on a server and transmitted to the client. Although these approaches work well for a limited number of users who share the same server, they can not be applied if an interactive visualization is published to a large group of viewers, for example over the Internet.

An approach which is better suited for "public" distribution of visualization results has been presented by Höhne et al. [20]. A multi-dimensional array of images is rendered and stored in an extended Quicktime-VR format. The viewer can browse through different views of the data, imitating an interactive rotation, dissection, or segmentation, for example. While this approach provides high-quality images [21] on low-end hardware, the user interaction is restricted by the "hidden" browsing mechanism (inbetween precomputed views). Furthermore, the size of even small-scale movies already becomes a limiting factor for viewing over low-bandwidth networks. The necessity to transmit an entire volume over the network is also the limiting factor to the approach of Hendin et al. [9]. They introduced a VRML-based viewer, which performs volume rendering by the means of texture-mapping, displaying a set of axis-aligned, textured polygons.

The approach implemented by the RTVR library is located inbetween the methods discussed above. The amount of data which actually is transmitted to the client for visualization is very low (about the size of several images), especially in comparison to the Quicktime-VR approach. The viewer is not restricted to precomputed views and has full control over visualization parameters. The only restriction for rendering is that just those parts of the volume which have been pre-selected (extracted) for presentation and transmission can be rendered. As usually just limited intervals of transfer function parameters produce meaningful results for a given visualization task, this restriction has proven not to be problematic.

Volume rendering by the use of special purpose hardware like the VolumePro board [17], or using 3D consumer hardware [18], achieves highly interactive frame-rates, allowing fast and efficient tuning of visualization parameters for entire volumes. Unfortunately, neither the VolumePro board, nor recent approaches based on textured polygons, allow to work with segmented data and to specify parameters on a per-object basis.

When used in a distributed client-server scenario, the software-only rendering approach of RTVR provides much more flexibility in terms of rendering parameters than volume previewing using texture mapping hardware, still at comparable or even lower costs in terms of bandwidth requirements.

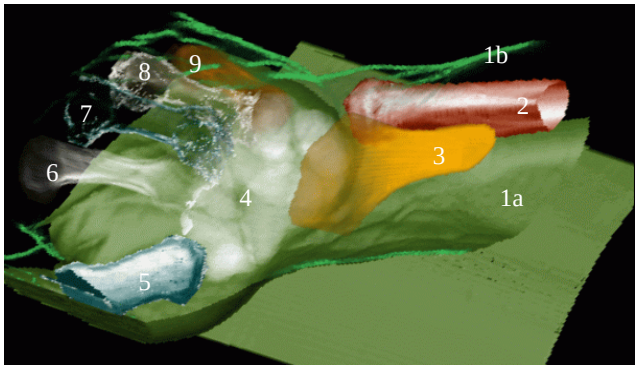


Figure 1: RTVR allows the assignment of shading and compositing methods on a per-object basis. The combination of Phong shading (objects 1a, 2, 5, 8) with non-photorealistic methods (1b, 5, 7, 8), of surface rendering (1, 2, 5, 7), DVR (9), MIP (3, 8), and summation (x-ray) rendering (4 and 6) is done in real-time.

In contrast to volume visualization toolkits like VolVis [1] or VTK [19], which cover a very broad range of data representations and applications, RTVR is focused on fast visualization of isotropically spaced rectilinear volumes, with flexible handling of rendering parameters. The restriction to isotropical spacing is required to ensure comparable rendering quality regardless of the viewing direction. Data defined on other types of grids has to be resampled for rendering (this can be done on the fly, during the extraction of relevant voxels).

In the following we describe some of the distinguishing design issues of RTVR which are responsible for its excellent efficiency with respect to real-time volume rendering as well as its flexibility in terms of rendering parameters. Section 2 gives an overview over the basic concepts behind RTVR, as well as an overview over rendering features and visualization techniques which are realized on this basis. Section 3 presents RTVR's internal data structure, performance-relevant issues, and the rendering algorithms used. Timings for typical application scenarios are given in section 4, followed by the presentation of sample applications, which are based on the RTVR library (section 5). Interactive visualizations with RTVR corresponding to the images of this paper can be found at `http://www.VRVis.at/vis/research/rtvr/`

2 Concepts and Capabilities

From a visualization-user's point of view, a volumetric data-set rarely resembles a monolithic block of data. Usually the data is assumed to be composed of a collection of spatial structures, i.e., objects, which have to be rendered or omitted from rendering to achieve the desired visualization goal. An obvious consequence of this observation, is to treat those structures within the volume individually during rendering, allowing separate adjustment of their visualization and rendering parameters (see Fig. 1 for an example). Providing more degrees of freedom for the individual parameter adjustment allows better fitting of the visualization method to the inherent properties of the visualized objects. Consequently, within RTVR each voxel of the input volume is assigned to an object having a common set of rendering parameters, consisting of opacity and color transfer functions, a shading model, and a compositing method (for example MIP or DVR) for combining the voxels of an object during rendering.

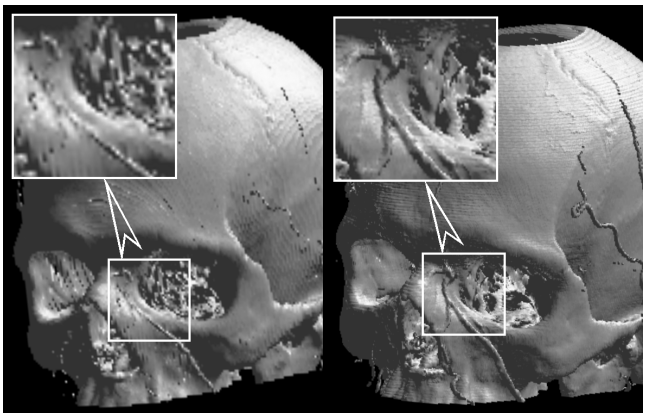


Figure 2: Rendering quality: 1024^2 image of a 256^3 and a 512^3 volume

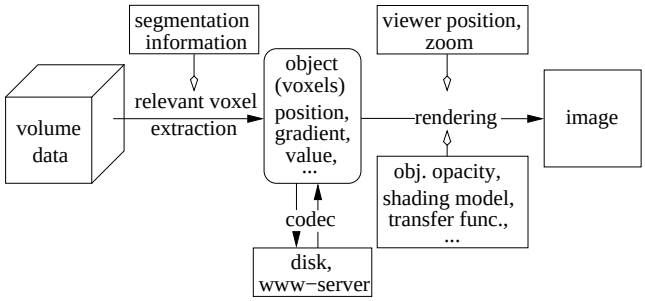


Figure 3: Volume data flow within RTVR: first, voxels with actually contribute to the visualization are extracted, then this representation of volumetric objects is used for fast and flexible rendering.

Depending on the chosen set of rendering parameters, voxels of an object may or may not contribute to a rendering of the object. Especially if MIP is used for compositing, relevant and non-relevant voxels are strongly intermixed, making the usual approaches for skipping non-relevant data (octrees or distance volumes) inefficient. On the other hand, for most scenarios only a small percentage of an object's voxels contribute to an image. Considering this facts, overhead for skipping irrelevant voxels during rendering can be entirely avoided at moderate memory cost if potentially relevant voxels are extracted from the volume and stored within a derived enumeration data structure. For rendering, just this data structure, which contains a high percentage of relevant voxels, has to be considered.

Consequently, the basic rendering primitive of RTVR is a voxel, i.e., a single data-sample from the volume. As no spatial neighborhood information is available within the derived data, no interpolation can be performed inbetween samples without accessing the original volume. To avoid this expensive operation, the extracted voxels are projected individually. A method well-suited for fast rendering of such "sparse" voxel data is shear-warp projection, with nearest-neighbor interpolation within the base-plane. For objects with sharp opacity transitions, this rendering method allows zoom factors up to two with sufficient image quality (i.e. an image size of 512^2 for 256^3 data sets, or 1024^2 for 512^3 volumes, see Fig. 2). For fuzzy objects even higher zoom factors are acceptable.

The visualization procedure using the above techniques is a two-step process (Fig. 3). During a segmentation and data extraction step, voxels which actually are relevant for the user-defined visualization goal are identified and extracted. The segmentation information which is required to distinguish objects within the data may

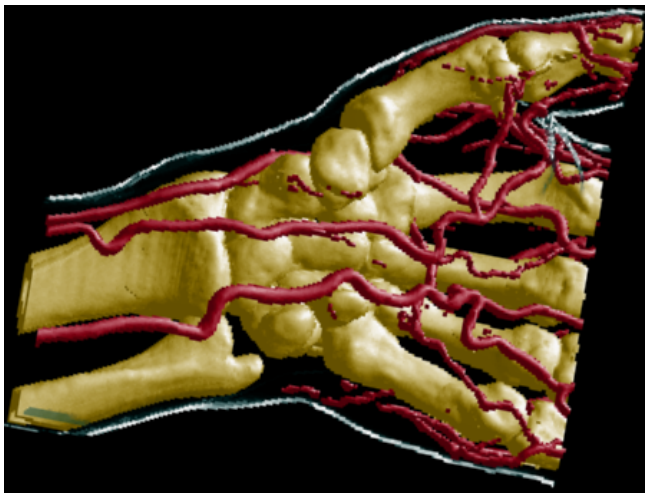


Figure 4: Combination of different shading models: Phong lighting for bones and vessels, contour rendering for the skin.

be obtained together with the volume data itself from an external data source, or interactively computed using threshold-based segmentation. The actual selection of voxels can be balanced between a radical elimination strategy, which just selects voxels relevant for a specific parameter setting (for example, just the surface voxels of an entirely opaque object), and the selection of all voxels of an object, on the other extreme. Although the latter approach at the first glance seems to be useless as an acceleration of rendering, it allows to freely adjust the transfer function during rendering. For the chosen shear-warp based rendering technique, the order of projecting voxels which share the same distance (z) to the base-plane is not relevant. Thus it is possible to shuffle such voxels without any restriction. By arranging voxels with the same z into two sub-groups which contain voxels relevant, respectively irrelevant to the current transfer function, efficient skipping of irrelevant parts within the extracted data is achieved. Using specific voxel sorting schemes for different types of transfer functions and compositing modes allows to skip voxels irrelevant for the current parameter settings without resorting [4, 15]. If, for example, voxel opacity is chosen to correspond to gradient magnitude, and voxels are ordered according to this attribute, entire blocks of voxels can be skipped as soon as the first entirely transparent voxel within a group is encountered.

The voxel extraction step usually leads to a significant data reduction, as only a small portion of the original volume actually belongs to objects of interest. Especially for surface-like structures, the extracted voxel data can be efficiently compressed exploiting coherence among voxel positions and attribute values. The resulting compact representation of the volume can be used to store visualization results for later interactive viewing, or for transmission and on-line viewing over low-bandwidth networks [14].

For performing the actual visualization and rendering, the extracted voxels are assigned to distinct objects and inserted into a scene graph. Using a fast, look-up table based approach (see section 3.4 for details), not only opacity and color, but also the shading model can be defined individually for each object. This allows to combine rendering using standard shading models like Phong shading with objects that are rendered by the use of non-photorealistic shading [4, 5]. Such combination is, for example, useful to provide context and shape information almost without occlusion (see Fig. 4).

Most volume rendering packages only allow to render a whole data-set using either the usual opacity-blended compositing (DVR) [10], and surface rendering [12, 16], or maximum intensity

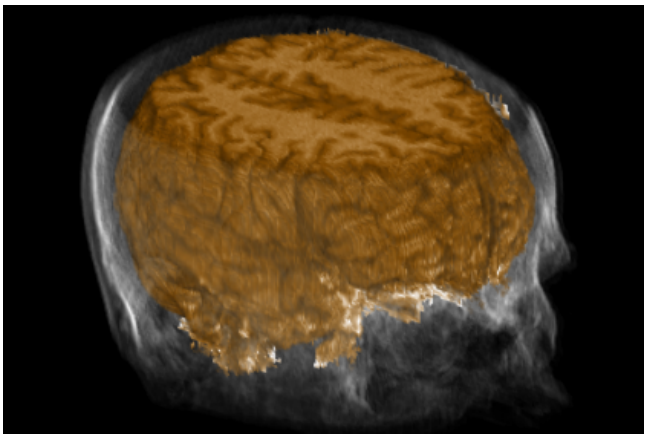


Figure 5: Combination of different compositing modes: DVR for the brain, summation for the surrounding bone and tissue.

projection (MIP) [15]. RTVR allows to separately define the compositing mode within each object (DVR, MIP, or summation), as well as an inter-object compositing mode (two-level volume rendering [8]). This allows to choose the most appropriate compositing technique for each object, depending on the structure of the data and the goal of the visualization (see Fig. 5, color plate d).

Among other "standard" techniques, RTVR supports the clipping of volumes or sets of individual objects at planes and more complex structures. Clipped parts of objects can be omitted from rendering – which is the most common approach – or rendered using a different set of rendering parameters (see color plate e). By using for example Phong shading for non-clipped voxels and a contour-only rendering for clipped parts, insight into an object can be given, while still providing a sketch of the most significant features of the clipped part as a context (see Fig. 1, skin).

Another feature of RTVR is the support for visualization of time series of volumetric data and of multi-dimensional parameter-series of volumes from simulation. The large memory demands of such data are compensated by the fact, that data extracted from a volume and used by RTVR for rendering is usually much smaller than the original volume. Only extracted data of the currently displayed volume has to be kept in memory for rendering, remaining parts of the volume and data which belongs to other time (parameter) steps can be kept on disk.

3 Intrinsics and Implementation

For each voxel identified during the extraction as potentially relevant for rendering, the coordinates and a set of attributes, like data value, gradient direction, and magnitude are stored in the derived data structure. For rendering, a subset of the attributes is selected as an information source for visualization mapping, and transformed into a compact representation which is well-suited for fast rendering.

The attribute values are used to index look-up tables to obtain and modulate color and opacity values in a way which is defined by the selected rendering mode. The look-up tables allow to implement different transfer functions and shading models in a very effective way.

3.1 The `RenderList` as Data Representation

The voxel extraction is performed by scanning the volume slice by slice, producing for each slice of each object a so-called

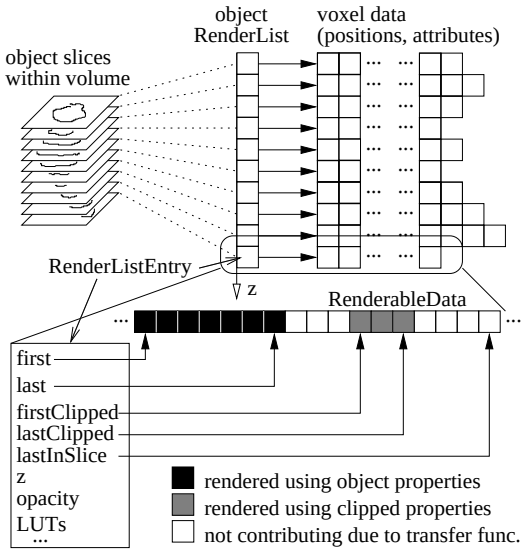


Figure 6: Volumetric object representation: voxels which are relevant for rendering an object are extracted slice by slice from the volume and stored into `RenderList`s.

`RenderListEntry`, containing the object's relevant voxels within the slice (Fig. 6). The `RenderListEntry`s of each object are grouped into an array – a `RenderList`. Thereby, the original (implicit) spatial arrangement of data values within the 3D array is sacrificed for an object-aware enumeration scheme of arbitrarily arranged voxels. All the attributes of a voxel are stored in separate arrays, the `RenderListEntry` itself just stores additional information which is required for rendering:

- an object-level opacity value for clipped and regular voxels
- look-up tables for mapping of clipped and regular voxels
- specification of rendering and compositing modes for clipped and regular voxels
- a reference to an array which contains a renderable representation of voxel data (derived from voxel attributes). Within this array, voxels between `first` and `firstClipped` belong to the regular part of an object, voxels between `firstClipped` and `lastInSlice` belong to the clipped part. Only voxels between `first` and `last`, respective `firstClipped` and `lastClipped` have to be rendered, voxels between `last` and `firstClipped`, and `lastClipped` and `lastInSlice` are not relevant for the current transfer function and rendering mode.

The "blocking" of voxels into non-contributing, clipped, etc., as shown in figure 6, is achieved by simply reordering the voxels within `RenderListEntry`s during clipping and optimization operations. The optimization, i.e., identification and reordering of currently non-relevant voxels within `RenderListEntry`s is performed by a background thread, which is activated whenever the application is idle and no rendering is performed. No effort has to be spent on skipping those voxels during following rendering passes. The background optimization is especially useful for accelerating the rendering of "fuzzy" objects, where no exact information about the relevance of voxels is available at the time of extraction.

For fast rendering, position and attribute information for each voxel is fitted into a single 32 bit integer. The x and y coordinates of the voxel are stored using 8 bit each, the z coordinate is identical for

all voxels within a `RenderListEntry` as they are all extracted from the same slice of the volume and thus it is stored just once. The common coordinate stored at the `RenderListEntry` for all voxels is referred to as z for reasons of simplicity. In fact, three copies of the data and thus three `RenderLists` are required for the shear-warp algorithm – each one grouped and sorted by one of the three coordinates. Using just 8 bits per coordinate limits the maximum extent of an object to 256^3 voxels. Larger volumes and objects are internally split into 256^3 pieces and the missing high bits of the coordinates are encoded into an offset, which is also stored once at the `RenderListEntry`. The remaining 16 bits are typically split into a 12 bit and a 4 bit field which store the data attributes used for rendering as previously described. This "renderable" voxel representation is attached as an additional array to each `RenderListEntry`, and is actually the only per-voxel information accessed during rendering.

Although the limitation to two voxel attributes with an overall of 16 bit for rendering is clearly a limitation with respect to flexibility and accuracy, the compact representation is perfectly suited for very fast rendering. In combination with the ability to re-order voxels within a `RenderListEntry` the rendering process turns into a "streaming" of sequential chunks of voxels – an optimal scenario for caching and prefetching as implemented by recent processors. The problem of the low bit resolution of data attributes for rendering can be addressed by applying intelligent remapping when copying voxel attribute data into it's renderable form: instead of clipping low bits of an attribute, a logarithmic remapping can be performed, or a certain sub-range of attribute values can be mapped to the range of values available for rendering. For scenarios which require more than two attributes for evaluating a voxel's contribution, special rendering modes can be defined which use more than 32 bit of information per voxel, at the cost of slower rendering of the affected object.

Java Peculiarities – due to the specific way of memory management as employed by current Java virtual machines (VM), a special data handling and caching functionality is used by RTVR to support the visualization of huge data-sets (series of dozens to hundreds of volumes), which are produced, for example, by numerical simulation applications [2]. The maximum amount of memory which is available to a VM has to be fixed at initialization time. As the garbage collection and object allocation mechanism sweeps through the entire address space of the VM, allocating more memory to the VM than physically available would lead to excessive paging and strong performance degradation. Instead of allocating sufficient memory to fit even the largest data-sets, RTVR uses a separate memory and disk cache for space-demanding parts of it's data structures, i.e., the original volume data, the extracted voxel attributes, and the renderable voxel data. Data, which is currently not used for rendering, is placed into the memory cache and thereby potentially written to disk by a background thread. Requests for recently used data can usually be satisfied out of the memory cache, whereas reading less recently accessed data may require to fetch it from disk. The cache feature is used when large data-sets are visualized locally, and is disabled when RTVR is used within a web-browser.

3.2 The RTVR Scene Graph

After extraction, the `RenderLists` and attribute data of volumetric objects are encapsulated into `VolumeObjects` and added to a common scene graph for rendering (Fig. 7). A common task of all types of nodes within the scene graph is to deliver up-to-date `RenderLists` which represent the content of their subgraphs. In the following a short overview over the most important types of nodes is given:

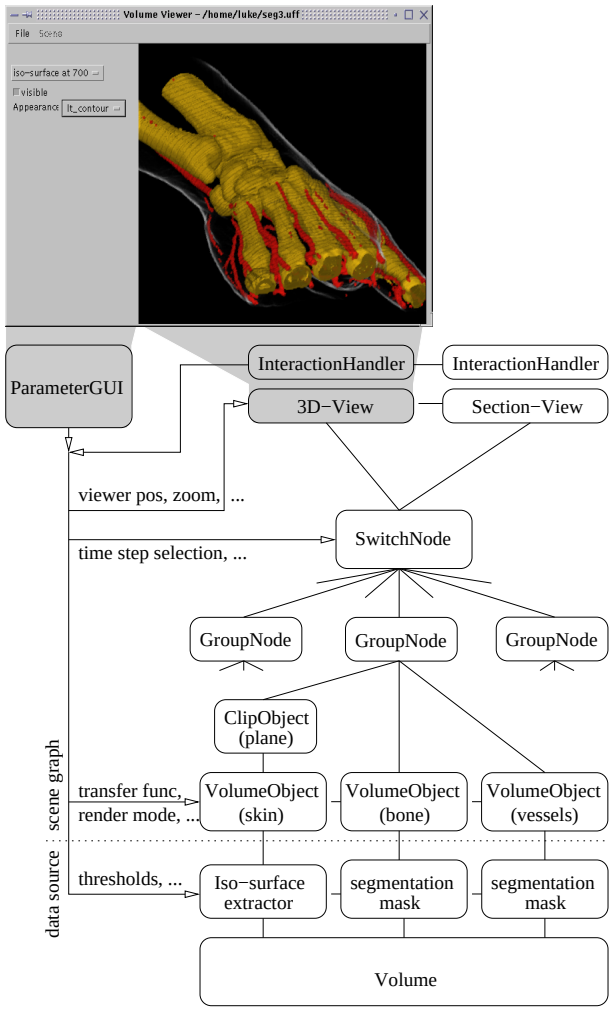


Figure 7: RTVR scene graph and user interaction handling

- `VolumeObject`: holds the `RenderList` of a single object as well as information on all parameters which affect the appearance and visualization mappings for this object.
- `GroupNode`: the shear-warp renderer performs a back-to-front rendering of `RenderListEntrys`. In addition to providing a simple way of handling multiple objects, the main purpose of the `GroupNode` is to merge and sort the `RenderLists` of its sub-graphs into a single list which is sorted by the current main viewing axis (z).
- Depending on the value of a selection parameter, the `SwitchNode` provides the `RenderList` of one of it's children. `SwitchNodes` allow to browse through multi-dimensional arrays of volumes, like time series, or parameter-dependent simulation results.
- `ClipNodes` filter and reorder the voxels of it's child nodes to implement clipping.

Each node is responsible for tracking changes of parameters which affect it's content and for performing appropriate actions according to changes. The actual update of renderable data to reflect parameter changes is carried out as late as possible, i.e., when a request for the affected voxel data is issued for rendering (lazy evaluation). Keeping just the currently visible data up-to-date improves

the responsiveness of the visualization during interactive parameter changes significantly.

3.3 User Interaction

The philosophy of data manipulation within RTVR is object-oriented. One of the objects within the currently displayed scene is selected to be the "active" object, for example by pointing into the rendered scene. The most important properties of the active object can be changed by pressing one of the mouse keys and dragging over the image. Transfer function contrast (color and alpha), opacity, and color can be changed directly within the 3D view. Furthermore, camera position, light source position, and zoom factor can be set within the view. The mapping of mouse actions to parameter changes is performed by an `InteractionHandler` component, which can be adapted to meet the needs of specific applications (for example to implement stream line integration from the position of a mouse click for a flow visualization application).

As a supplement to parameter manipulation within the rendered view, all parameters of the active object can be adjusted using standard user-interface components which are automatically generated by RTVR. This parameter panel allows an explicit selection of the active object and adjustment of its parameters, and can (but does not have to) be used within any application which utilizes RTVR for visualization.

Java Peculiarities – for Java-based graphical user interfaces, basically two APIs are available: the AWT, available in its present form since Java version 1.1, and the more sophisticated SWING 1.1, which is part of the Java runtime since version 1.2. The front end (GUI and rendering output) of RTVR is provided using both, either AWT or SWING. As most web browsers currently provide a 1.1 virtual machine only, an AWT implementation is provided for compatibility reasons, despite of all its inconveniences and deficiencies. The rendering performance of the SWING implementation benefits, for example, from a faster image handling (`BufferedImage`) introduced in Java 1.2.

3.4 Rendering

To achieve interactive rendering rates even on standard desktop hardware, a fast shear-warp based parallel projection is used. Rendering to the base-plane is performed using back-to-front compositing of voxels by the use of nearest-neighbor interpolation. The warp step of the algorithm, which especially for large image sizes may be more time-consuming than the voxel projection itself, can also be carried out by texture mapping using OpenGL. In comparison to a previously published version of this fast algorithm [16], RTVR includes an extended version, which provides more flexibility for mapping voxel attributes to color and opacity. Three look-up tables (LUTs, typically 1x4 bit, 2x12 bit) are available at each `RenderListEntry` for implementing shading and transfer function mapping. A set of combination patterns for the voxel attributes and look-up tables is provided by RTVR (See Fig. 8) and selected by choosing an appropriate rendering mode for an object. This scheme of combining LUTs allows efficient processing while still enabling various ways of selectively applying visualization techniques to objects within the data. The `RenderListEntry` can also be extended to provide user-defined rendering functionality for it's voxels, which finally allows to implement any desired operation on voxel attributes and look-up tables.

Shading operations are performed using an approach which is based on look-up tables, with a 12-bit representation of the gradient vector as an index. Two shading models are provided by RTVR: a Phong shading table (color plate a), a non-photorealistic shading table which enhances the contour of an object [5] (Fig. 4), and a combination of both (color plate b). The shading tables have to be

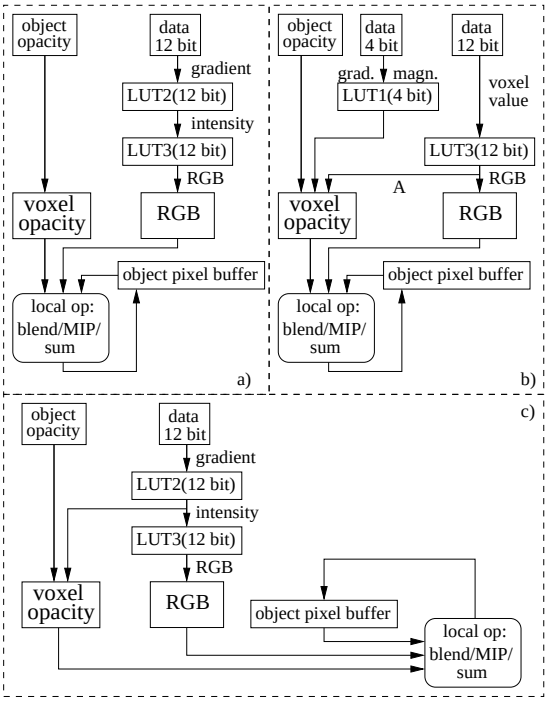


Figure 8: Sample LUT combinations during voxel rendering within RTVR: a) shaded surface b) non-shaded DVR c) contour rendering. An overall of 4 combination schemes for voxel opacity and 9 combination schemes for voxel color are available.

re-computed after every change of viewer or light source position, which is not time-critical due to their small size (4096 entries). For rendering, the shading table is placed into LUT2 (Fig. 8), and indexed by the 12 bit data-channel which contains the gradient vector. The output of the look-up is an intensity value, which is then used to access the color transfer function in LUT3. Splitting shading into two stages allows to reuse the same shading table for objects with different color transfer functions.

The opacity of a pixel is influenced by several sources. An all-object opacity value is always included into the computation and can be used to tune the overall opacity of entire objects. The individual opacity of each voxel is modulated by various combinations of data channel and look-up operations. In the following, a few sample color and opacity calculation setups will be discussed, which implement different volume rendering approaches.

- *display of (iso-)surfaces* (Fig. 8a): the surface voxels of an object have to be shaded and blended using the object opacity. A Phong shading table is put into LUT2, the resulting intensity value is used to access a color transfer function in LUT3. The transfer function is a ramp of object color values starting with the color of the highlight (white) and evolving towards maximum saturation and minimum lightness (object color, ambient light, see color plate a). By just rendering a thin layer of voxels which form the surface, the object opacity can be used to influence the transparency of the surface in the same way as an alpha-value influences the appearance of a polygonal surface model.

- *non-shaded DVR* (Fig. 8b): LUT3 contains the color and opacity transfer function and and is indexed by the 12 bit data value (Fig. 5, brain). Optionally, voxel opacity can be modulated by the content of LUT1 accessed, for example, by gradient magnitude encoded in the 4 bit data field.

- *Bright object outlines* (Fig. 8c): LUT2 is loaded with a shading table which maps the angle between viewing direction and gradient direction to intensity. LUT3 contains a color transfer function which is used to tune contrast and color for the specific object. If the result of the LUT2 look-up is also used as voxel opacity, the object becomes almost entirely transparent – except for the contours which remain opaque (color plate c, skin).

The implementation of object-aware compositing requires the use of two separate pixel buffers, one for compositing within an object and one for compositing of the global image (Fig. 8). Scenes which require only a single compositing mode (within and inbetween objects) do not require two pixel buffers and are rendered more efficiently with the usual single-buffer approach.

If pure MIP is used, voxels can be sorted and grouped into `RenderListEntry`s by value instead of the z coordinate [15]. In this case, projecting sorted voxels from lowest valued to highest valued ones eliminates the need for maximum search. However, if MIP is combined with other compositing techniques within the scene, back-to-front rendering and thus sorting by the z coordinate is required also for objects composited by MIP, as they may interleave with other objects rendered with different techniques.

4 Performance

High responsiveness of a visualization system to user actions is a crucial factor for the effectivity of data exploration and analysis. The rendering times for the surface rendering [16], MIP [15] and two-level rendering approach [8] used by RTVR have been published in previous work. Thus, instead of broadly surveying the behavior of each method, a comparison of the measured times for rendering the same data-set with RTVR using various methods is given in table 1. The measurements have been carried out on a PII/400MHz PC using the virtual machine of JDK1.3 from Sun and the AWT front-end of RTVR. The size of the rendered images is 512^2. The first row shows timings for the data-set shown in figure 4. Skin, bones, and vessels are represented by their surface voxels. The rendering is carried out using MIP, DVR, a gray-scale DVR view, and a combination of DVR for the vessels and MIP for bones and skin. The last column contains the times for rendering the same scenes using texture mapping and OpenGL for the warp step of the projection. In contrast to software warp, which for large images (1024^2) usually is already the dominant factor for rendering time, GL based warp is insensitive to the size of the output image, and takes 10-20ms on most current consumer graphics cards. The second row displays timings for the head data shown in color plate a, with vessels and skin displayed as additional surfaces. The data-set in row 3 is similar to the one depicted in color plate f. The basin is represented by its surface voxels, the chaotic attractor has a highly complex internal structure, and is thus considered a volumetric object.

The pure rendering time reflects the rendering performance for most interactions. These include interactive changes of the viewing parameters (viewer position and zoom), changes to content of look-up tables (moving light source, changing transfer function), and changes to the parameters and rendering modes of objects. Clipping operations require scanning and reordering of object voxels. During simple clipping of all objects at an axis aligned plane, the response time increases by approximately 40% compared to when changing viewer position. Time required for clipping at more complex objects depends on the complexity of the test which has to be performed for each voxel. Clipping of a complex scene at an oblique plane, for example, can be done with 1–2 frames per second. During browsing through large (time or parameter) series of volumes, voxel data may have to be fetched from disk cache, thus increasing the response time by the time required to read the data. Depending on the size of the scene, this may range from few milliseconds, to more than one second. The time for extraction of new objects from a volume depends on the complexity of the segmentation criteria and on the amount of voxels selected (gradient computation). The extraction of an iso-surface from a 256^3 volume for example requires approximately 1.5 seconds, including gradient computation.

The choice of the virtual machine used to execute the application has severe impact on the performance. Among the tested runtime environments, fastest execution and rendering has been observed for the VMs (1.1.6++, 1.2, 1.3) from Sun on Windows and (1.1.8, 1.2, 1.3) from IBM on Windows and Linux. Virtual machines provided by web-browsers are in general slower, probably due to additionally performed security checks. Worst results are obtained by the VM which is used by Netscape browsers (Version $\leq$ 4.7.4) on Linux – more than ten times slower than the timings in table 1.

5 Sample Applications of RTVR

The RTVR library has been successfully used to provide volume visualization functionality within two projects. The first application is a volume viewer which can be used for fast volume data exploration and analysis in the field of medical data. Visualization results created within the viewer (extracted objects and visualization parameters) can be stored using a compact representation (typically just a few hundred kilobytes) for later interactive viewing or for publication on the Internet. An applet version of the viewer which provides the same functionality, except for the extraction and creation of new objects, can be used to view previously stored data within web pages (interactive versions of this paper's images can be found at `http://www.VRVis.at/vis/research/rtvr/`).

A second application which makes use of the capabilities of RTVR is a visualization and analysis system for 3D dynamical systems (discrete non-invertible maps) [2]. The application is used to analyze and visualize structures and events within the phase space of the systems. For this application, objects of interest are attractors (often complex and chaotic), their basins of attraction (i.e., the set of all system states which are attracted by them) and surfaces which separate regions with different properties (color plate f). Events (bifurcations) can be caused by contacts between structures as some parameter of the dynamical system is changed. The process of visualization is split into two parts. A volumetric representation of the structures within phase space is computed offline (256^3 volumes) and stored in a space-efficient form. For the analysis of bifurcations, sequences of up to hundreds of volumes are computed for different values of the bifurcation parameter. For investigation the data produced by the simulation is loaded into the viewer (the disk-cache is extremely useful for large sequences of volumes) which provides application specific functionality, like the shooting of trajectories by pointing with the mouse at the start position. To ease the detection of contacts between objects, distance information can be mapped to voxel color, as shown in color plate f. The feature of mixing MIP with other compositing methods has proven to be especially useful for visualizing chaotic attractors. Their complex internal structure is well captured using MIP while producing little occlusion. At the same time, the attractor's basin of attraction can be rendered as a shaded surface.

6 Conclusions

Using an efficient data representation and a fast rendering method volumetric data can be displayed at an average desktop PC at frame rates which are comparable with those which are achieved for volume rendering by consumer 3D hardware, while providing significantly more flexibility, like object-wise transfer functions, shading

data-set	*size*	*scene voxels*	*MIP*	*DVR*	*DVR/mono*	*two-level*	*two-level, GL-warp*
hand & vessels	256^2x124	380k	80ms	135ms	80ms	180ms	160ms
head & vessels	256^2x158	640k	100ms	185ms	110ms	250ms	220ms
attractor & basin	256^3	1M	130ms	250ms	140ms	320ms	280ms

Table 1: Timings for various data-sets and rendering modes

models and compositing methods (MIP, DVR, ...). Taking into account peculiarities of Java, all those capabilities can be made available to users with standard desktop hardware using different operating systems. Using a compact volume representation, the RTVR library can be also exploited to provide highly interactive and flexible presentations of visualization results over networks, like the Internet.

7 Acknowledgements

This work has been carried out as part of the basic research on visualization at the VRVis Research Center in Vienna, Austria (`http://www.VRVis.at/vis/`), which partly is funded by te Austrian Kplus research program. The medical data-sets depicted in this paper are property of Tiani Medgraph, Vienna, Austria. The authors want to thank Meister E. Gröller, Csébfalvi Balázs, and the team of Tiani Medgraph.

References

[1] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang. VolVis: A diversified volume visualization system. In *Proceedings IEEE Visualization '94*, pages 31–39, 1994.

[2] G.-I. Bischi, L. Mroz, and H. Hauser. Studying basin bifurcations in nonlinear triopoly games by using 3D visualization. Accepted for publication in the Journal of Nonlinear Analysis.

[3] D. Cohen and Z. Sheffer. Proximity clouds – an acceleration technique for 3D grid traversal. *The Visual Computer*, 10(11):27–38, 1994.

[4] B. Csebfalvi, L. Mroz, H. Hauser, A. König, and E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. Accepted for publication in Proceedings of EUROGRAPHICS 2001, 2001.

[5] D. Ebert and P. Rheingans. Volume illustration: non-photographic rendering of volume models. In *Proceedings IEEE Visualization 2000*, pages 195–202, 2000.

[6] K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings IEEE Visualization 2000*, pages 449–452, 2000.

[7] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *Proceedings IEEE Visualization '99*, pages 139–146, 1999.

[8] H. Hauser, L. Mroz, G.-I. Bischi, and E. Gröller. Two-level volume rendering - fusing MIP and DVR. In *Proceedings IEEE Visualization 2000*, pages 211–218, 2000.

[9] O. Hendin, N. John, and O. Shochet. Medical volume rendering over the www using VRML and Java. In *Proceedings of MMVR*, 1998.

[10] J. T. Kajiya. Ray tracing volume densities. In *Proceedings of ACM SIGGRAPH '84*, pages 165–174, 1984.

[11] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorisation of the viewing transform. In *Proceedings of ACM SIGGRAPH '94*, pages 451–459, 1994.

[12] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.

[13] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH'87*, pages 163–189, 1987.

[14] L. Mroz and H. Hauser. Space-efficient boundary representation of volumetric objects. In *Data Visualization 2001, Proceedings of the Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, pages 193–202, 2001.

[15] L. Mroz, A. König, and E. Gröller. Real-time maximum intensity projection. In *Data Visualization '99, Proceedings of the Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, pages 135–144, 1999.

[16] L. Mroz, R. Wegenkittl, and E. Gröller. Mastering interactive surface rendering for java-based diagnostic applications. In *Proceedings IEEE Visualization 2000*, pages 437–440, 2000.

[17] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volume pro real-time ray-casting system. In *Proceedings of ACM SIGGRAPH'99*, pages 251–260, 1999.

[18] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of SIGGRAPH/EUROGRAPHICS Graphics Hardware Workshop 2000*, 2000.

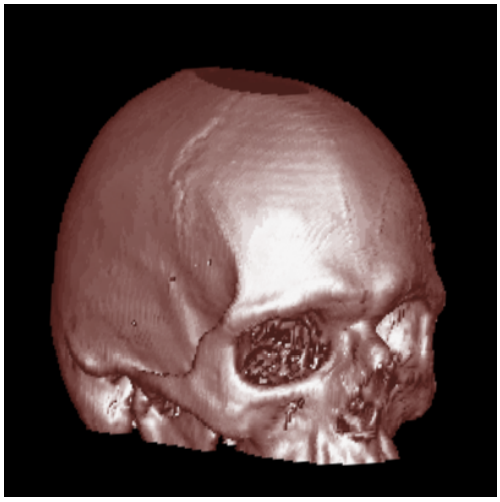
[19] W. Schroeder, K. Martin, and W. Lorensen. The visualization toolkit. In *An Object-Oriented Approach to 3D Graphics*. Prentice Hall, 1996.

[20] R. Schubert, B. Pflesser, A. Pommert, K. Preiesmeyer, M. Riemer, Th. Schiemann, U. Tiede, P. Steiner, and H. Höhne. Interactive volume visualization using intelligent movies. In *Proceedings Medicine Meets Virtual Reality*, 1999.

[21] U. Tiede, T. Schiemann, and K.H. Hohme. High quality rendering of attributed volume data. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *Proceedings IEEE Visualization '98*, pages 255–262. IEEE Computer Society Press, 1998. Isosurface and Volume Rendering.

[22] J. K. Udupa and D. Odhner. Shell rendering. *IEEE Computer Graphics & Applications*, 13(6):58–67, Novemner 1993.

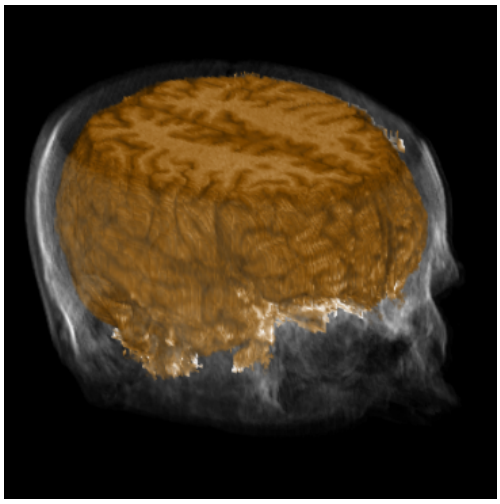
[23] K. J. Zuiderveld, A. H. J. Koning, and M. A. Viergever. Techniques for speeding up high-quality perspective maximum intensity projection. *Pattern Recognition Letters*, 15:507–517, 1994.



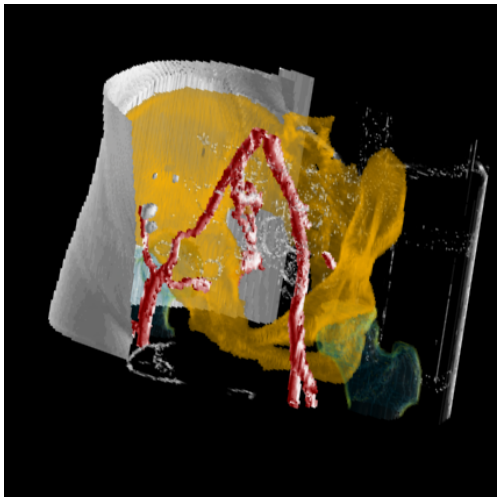
a) surface representation of a skull

b) shaded and contour enhanced vertebrae

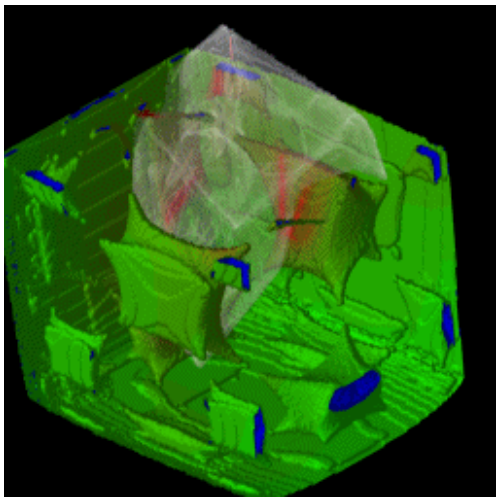
c) surface rendering for for bones and vessels, contours only for skin



d) DVR for brain, summation for bones and outer tissue



e) surface rendering for vessels, MIP for hip-bone, contour enhanced surface for thighbones, contours for clipped part of skin



f) contact bifurcation - one out of a sequence of 40 volumes from a numerical simulation

Space-Efficient Boundary Representation of Volumetric Objects

Lukas Mroz and Helwig Hauser

VRVis Research Center, Vienna, Austria
http://www.vrvis.at/vis/
{mroz|hauser}@vrvis.at

Abstract. In this paper we present a compression technique for efficiently representing boundary objects from volumetric data-sets. Exploiting spatial coherency within object contours, we are able to reduce the size of the volumetric boundary down to the size of just a few images. Allowing for direct volume rendering of the down-scaled data in addition to compression ratios up to 250:1, interactive volume visualization becomes possible, even over the Internet and on low-end hardware.

1 Introduction

One major challenge of visualization in general is to deal with a whole lot of data. Especially in volume visualization, common data-sets range between several hundreds of Kilobytes, at the minimum, up to Gigabytes of uncompressed size. In medical visualization, for example, volumetric data-sets of size $256^3 \times 16$ Bit, i.e., 32 MBytes in total, are quite usual. If standard compression like `gzip` [7], for example, is applied, data-sets usually shrink to about 30–60 percent of the original size – still MBytes.

Processing huge data-sets itself poses high-performance requirements on the visualization software, but also storage and transmission of volumetric data-sets easily get into bandwidth problems, especially if multiple data-sets are to be treated. From medical applications, for example, we know that archiving 3D data-sets, which accompany diagnosis data, significantly stresses storage devices currently available in common clinical setups.

Even more critical, concerning the size of volumetric data-sets, and compared to storage problems, is visualization over the Internet. Web applications like remote diagnosis, for example, suffer from low transmission rates, even over local networks. In general, client-server solutions in the field of visualization usually are classified by the point, at which the visualization pipeline [8] is cut into a server-part and a client-part. Doing most of the visualization job at the client, for example, usually is referred to being a fat-client solution [10]. Thin clients, on the other hand, just display results of the visualization process, namely images, which entirely have been computed at the server beforehand. The trade-off between thin- and fat-client solutions is driven by the fact, that cutting the visualization pipeline at an earlier stage (fat-client solution) allows for more flexibility at the client's side (without any need to reload data). However, this advantage is gained at the expense of large-sized (volumetric) data to be downloaded,

whenever necessary (initial download, changes to parameters of the preprocess). Respectively, thin clients deal with smaller data – just result images, for example – but need to download new data, whenever any of the parameters, even just viewing parameters, are changed.

The applicability of the more flexible fat-client solution to volume visualization strongly depends on the effectivity of the compression techniques used for transmission of the data-set. Lossless compression techniques – general purpose [7] as well as volumetric data specific [6] – usually achieve rather low compression ratios (around 2), which is not sufficient to significantly widen the bandwidth bottleneck. Using lossy compression [17, 2, 12] ratios in the range of 5 to 50 can be achieved while maintaining acceptable quality of the visualization results. On the other hand, medical applications, for example, prohibit changes to the accuracy of the data, as induced by lossy compression methods. Hierarchical methods, like wavelet compression [12] combine advantages of lossy and lossless compression. By transmitting and considering just a small fraction of the coefficients (around 5%) images of acceptable quality can be generated, data values of the original volume can be reconstructed if all coefficients are considered. A useful property of wavelet compression and many lossy compression techniques is the ability to render compressed data directly, without prior expansion and decompression.

Polygonal representations of structures within the volume (e.g. of iso-surfaces) can be used to realize solutions which are compromises between a pure thin and fat client approach. The volume is maintained at the server, just the polygonal model is transmitted and rendered at the client. Changes of viewing parameters require local rendering only, just changes affecting the shape of the model require a recomputation at the server and transmission of surface data over the network. To reduce the bandwidth required to transmit the model and to improve the interactivity of rendering at low-end clients, progressive refinement as well as focus and context techniques can be used [5], trading quality of representation (in less relevant regions of the volume) for speed. A combination of server-side and client-side approaches for direct volume rendering has been presented by Engel et al. [4]. They transmit a sub-sampled volume to the client and use it for local rendering during interactions. The original volume at the server is used to create and transmit a high-quality image whenever the interaction is finished/paused.

Pure thin-client solutions on the other hand, allow to perform visualization using low-end clients making at the same time shared use of special purpose hardware at the server (multiple CPUs, VolumePro board [18] for example).

One approach to determine the effectiveness of compression techniques for volumetric data-sets and their suitability for Internet-based visualization is to compare the size of compressed volumes versus the size of images of the same data. This comparison is useful as it directly corresponds to the trade-off between thin and fat-client solutions. If sizes of compressed volume data-sets range in the same magnitude as sizes of images thereof, and given the client to provide sufficient computational performance to carry out most of the visualization steps itself, then fat-client solutions become feasible even via the Internet. In our case, we achieve compression rates such that, given a 256^3 data-set as well as 512^2 images (24 Bits per pixel) in compressed GIF-format, about 2–5 images already are bigger in size than the compressed volume data-set.

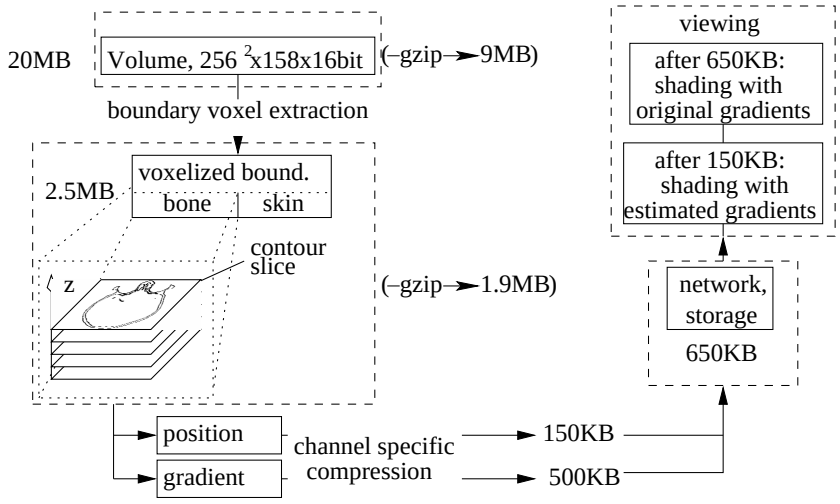


Fig. 1. Boundary extraction, compression and visualization pipeline

2 The Basic Idea

The effectivity of our approach is based on the observation that for the vast majority of applications, especially in medical visualization, volumetric data is rendered by displaying either iso-surfaces [14] or surface-like structures defined by areas of high gradient magnitude [13]. In both cases, the result of the visualization is determined by contributions of just a small fraction of all volume samples. By just coding those voxels of an object, which actually contribute to its visual appearance, the size of the data-set is greatly reduced. Thereby, a small-scale boundary representation of volumetric objects is generated (Fig. 1, Sect. 3). Compression of the boundary representation, which exploits spatial coherence among neighboring voxels, produces an extremely compact object representation (Sect. 4) which is well-suited for network transmission (Sect. 5). The information contained within this representation of objects allows interactive rendering at a client without any dependency on hardware-support, and with more flexibility regarding visualization parameters than polygonal surface representations (Sect. 6).[1]

The first step to obtain an efficient representation of bounded objects within a volumetric data-set is the identification and extraction of voxels which contribute to the object's visual representation, i.e., the boundary of the object. In our case, boundary voxels are data samples located within the object and have at least one neighboring voxel outside the object. The extraction process generates a separate boundary representation for each object within the volume. Usually 5–10% of all voxels belong to the boundary representation.

Typically, gradient information is required to evaluate a shading equation at each voxel during rendering. It is more efficient to precompute voxel gradients during bound-

[1] A demonstration applet is available from
`http://bandviz.cg.tuwien.ac.at/basinviz/compression/`

ary extraction than to store all data values required for gradient computation at boundary voxels at rendering time.

Within our representation of an object, voxels are grouped into slices sharing the same z coordinate (See Fig. 1). Within a slice, the boundary voxels form contours of the object – a set of connected sequences of voxels. Exploiting spatial coherence of the contour, the positions of voxels within the slice are efficiently encoded into a compressed data stream. Voxel gradients are compressed in the same order as the corresponding positions, using a special compression scheme. Additional streams of voxel attributes (= data channels), like data value, gradient magnitude, etc., can be optionally encoded in a similar way. The output of the compression step is a boundary representation of volumetric objects, typically compressed by a factor of 10–100 compared to the original volume.

By transmitting the data channels in a smart order, for example, position data first, gradients last, a preview of the objects with full spatial accuracy can be displayed (appendix, Fig. 3) after transmitting just a few Kilobytes of data (using estimated gradients for shading).

The decompressed boundary representation can be rendered directly [15, 9], without prior reconstruction of a full-sized volume. Compared with a polygonal representation of the boundary surfaces, our approach preserves the full accuracy of the data-set at much lower memory cost, allows interactive rendering on low-end hardware and provides more flexibility with respect to rendering parameters. Transparency, non-photorealistic shading and the fusion with truly volumetric objects (which can be compressed using the same method) are easily possible without performance degradation.

3 Extraction of Boundary Voxels

In our approach we either use the iso-surface metaphor to specify boundary voxels, or use a predefined and explicit segmentation mask for this purpose. In the first case, voxels with a data value $\geq$ iso-value and at least one 26-connected neighbor with a value smaller than the iso-value are considered to be part of the boundary. This definition results in 6-connected sets of boundary voxels, a property useful for exploiting coherence during compression of the contours. Boundaries of objects defined using a segmentation mask, can be extracted in a similar way and also result in 6-connected sets of voxels. As voxel identification accounts only for a small part of extraction time (gradient computation is most expensive), a simple sweep method is used for this purpose. The extraction of object boundaries is performed during an interactive volume visualization session. The resulting object representation can be rendered immediately, the compressed boundary can be stored for later viewing and presentation of visualization results.

Although best compression efficiency is achieved for surface-like voxel sets, truly volumetric objects can be extracted and compressed in the same way. This is especially useful for the visualization of spatially complex structures, like vessels in medical angiography data-sets or complex chaotic attractors in the field of dynamical systems [1].

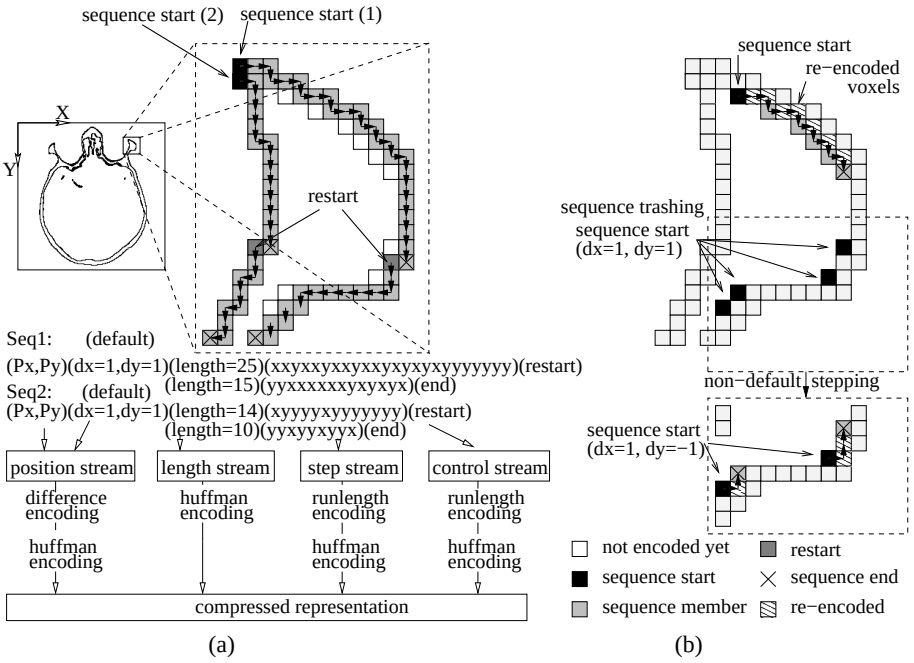


Fig. 2. Encoding of voxel positions: slice scanned from top left to bottom right a) long sequences and sequence continuations b) re-encoding of voxels and non-default stepping direction to reduce number of sequence starts and thus position specifications

For the extracted voxels, attributes (data channels) like voxel position, data value, gradient direction and magnitude, and application specific attributes are stored. When only the display of shaded surfaces is desired, storing voxel position and gradient direction is sufficient.

4 Data Compression

Individual objects within the volume are compressed separately. Voxels of each object are grouped into slices of voxels with the same z coordinate which are processed sequentially (See Fig. 1). To ensure effectivity, different data channels have to be compressed using specialized compression methods.

4.1 Position Data Channel

Boundary voxels within a single z-slice form object contours which consist of face-connected voxels (See Fig. 2). Exploiting spatial coherence and connectivity, voxels can be grouped into "*sequences*" which spatially follow the object contour. The approach is similar to "chain coding" used in binary image (text) compression [11] for contour encoding. In contrast contours of 2D objects, the boundary of a volumetric object may be thicker than a single layer of voxels, and thus requires a modified approach.

During compression, the slice is scanned for non-encoded voxels. Whenever one is found, a new sequence is started and the position of the voxel (P_x, P_y) is stored. The sequence is continued, by selecting and appending one of the neighbor voxels at it's end. As the contour voxels are face-connected, potential candidates for continuation are located at $(P_x + dx, P_y)$ or $(P_x, P_y + dy)$ with dx, dy being respectively -1 or 1. Encoding the selection of one of the four neighbors as a successor would require 2 Bits. If the choice is restricted to two neighbors by using constant values of dx and dy for a whole sequence, each voxel continuing a sequence can be specified by a single Bit, which defines whether a step by dx or dy is used. Although this restriction reduces the flexibility and thus the average length of sequences, the cost per voxel within a sequence is cut by half, outweighting the disadvantage of shorter sequences.

In cases where a direct neighbor of the trailing voxel of a sequence is present, but can not be reached using the current (fixed) dx and dy values, a *sequence restart* can be performed, continuing the sequence at this neighbor with a new value for dx or dy. To realize this, each sequence is followed by a command code which specifies whether the sequence ends, or restarts with a different stepping direction. The presence of a restart code implicitly defines the position of the start voxel of the new sequence. As the previous sequence had to be terminated, no successors of it's last voxel are present in it's dx and dy direction. One of the remaining two neighbors is the second but last voxel of the interrupted sequence, so the other one necessarily is the starting voxel of the new sequence. The new values of dx and dy are derived from dx and dy of the old sequence. Depending on whether the last step of the sequence was dx or dy either dx or dy is inverted. Although being more restrictive than with an explicit specification of dx and dy, this strategy still allows encoding of cyclic structures with a single position specification and restart commands within a *chain of sequences*.

For each combination of dx and dy values, one of the possible stepping directions is preferred, whenever both ways can be taken. The preference is chosen in a way, that a clockwise processing of closed objects will stay as close to the outer border as possible. For example, for $dx = 1$ and $dy = 1$ like in the first sequence of Fig. 2a, steps by dx are preferred.

After the creation of long sequences, usually groups of short sequences or even non-connected voxels remain. Starting a new sequence for each of these voxels is expensive. Usually most of these voxels can be encoded at a lower cost by joining them into sequences re-using voxels already encoded earlier in the process (Fig. 2b). In general, a sequence has to be continued, reusing already encoded voxels, if this allows to reach non-encoded voxels at a cost which is lower than a "sequence end" and the start of a new sequence.

As the scan for non-encoded voxels within a slice is performed in ascending x and y direction, using $dx = 1$ and $dy = 1$ as a default stepping direction for newly started sequences is usually a good solution – voxels with smaller x and y coordinates compared to the current one are already encoded in this case. In some cases however, keeping $dx = 1$ and $dy = 1$ as default directions tends to generate a lot of short sequences "sequence trashing" (Fig. 2b). Instead it is better to first search "backwards" (using $dx = -1$, $dy = 1$) and to start the new sequence using $dx = 1$ and $dy = -1$ at the last voxel found (for reasons of simplicity no backward scan was performed for the

sequences of Fig. 2a). At each sequence start 1 Bit is used to store, whether the default stepping direction $(1, 1)$, or the direction of the backward scan $(1, -1)$ is used.

For further compression, the sequence data is separated into four streams. The *position stream* stores starting positions and stepping directions. Positions are stored using Huffman encoded differences between successive coordinate values (Typically 12 Bits per starting code). The *length stream* stores information about sequence lengths (Huffman encoded, 5 Bits per sequence). The *step stream* stores the information for building up sequences (1 Bit per voxel). As dx and dy steps tend to cluster due to the presence of a preferred stepping direction, this information is run-length encoded, using again Huffman encoding for the run-lengths. The *control stream* is used for the sequence control information (end/restart, 1 Bit per sequence). As many restarts at the beginning of encoding a slice are followed by short sequences collecting isolated voxels towards the end of encoding, which leads to clustering of restart and end commands, run-length encoding combined with Huffman encoding is also used here. Combining all those streams, an average of 2 Bits is required to encode the position of a single voxel.

Within all other data channels, voxels are encoded in the same order as their position data. This ordering allows to exploit spatial coherence within voxel sequences also for attribute encoding. For subsequent occurrences of re-encoded voxels, no attribute information is stored.

4.2 Gradient Direction Channel

As a first step, gradient vectors are normalized, transformed to polar coordinates and quantized to 2x6 Bits. This gradient representation is also used by our rendering algorithm for interactive shading. By exploiting spatial coherence within the encoded stream of voxels the gradient information is reduced to 3–8 Bit per voxel, depending on the smoothness of the boundary. Both polar coordinates are encoded into separate streams, storing differences between coordinates of successive voxels. As most of the difference data consists of sequences of values in the range of $[-1, 1]$ which are occasionally interrupted by larger values or clusters thereof, the encoder switches between two different coding schemes. The first scheme is used to encode sequences of differences in the range of $[-1, 1]$ using 1 Bit for 0 (most common), 2 and 3 Bits for -1 and 1, and a 3 Bit code to switch to the other encoding scheme. Larger differences are encoded using Huffman coding with an extra symbol to switch to the encoding scheme for small values. A switch to the code for small values is only performed to encode sufficiently long sequences of small values (cost of switching).

The use of prediction techniques for estimating gradients and the encoding of the prediction error instead of encoding gradient differences seems to promise good results at the first glance. Nevertheless, tests performed using linear regression [16] with a diameter of 3 and 5 for gradient estimation, indicate that compression rates obtained using this technique are worse than our current approach.

4.3 Other Data Channels

Additional data channels, like gradient magnitude, data value, etc., are compressed in the same order as the positions of the voxels to exploit spatial coherency also. Huffman

encoding of differences of successive values and additional `zlib` compression (for further reduction of uniform areas) is used.

5 Data Transmission and Decompression

The compressed data-set consists of two parts: a header, which contains control-information about the objects and their position within the data, information about additional data channels and how to use them for rendering. The body contains voxel positions and other data channels for all objects. The data within the body is arranged in a way which allows to obtain a view on the data as early as possible during loading. Objects and data channels which are more significant for the preset visualization mappings are stored and transferred earlier than less significant data. Data channels are subdivided into blocks of a few Kilobytes each. As soon as an entire block has arrived, it can be decompressed and displayed while the following data is arriving. This allows voxel data to be rapidly updated, without having to wait for the arrival of the entire channel. Finally, as gradient information usually accounts for most of the data to be transmitted (See table 1), for boundary objects a locally computed gradient approximation (linear regression [16] with a filter size of 5 while interpreting the data as a binary object) can be displayed before the original gradient data arrives (appendix, Fig. 3). For inherently binary objects, like basins of attraction within the phase space of a dynamical system [1] the locally computed gradients can entirely replace the transmission of gradients, significantly decreasing the amount of transmitted data.

6 Rendering

In our test application, tendering of the data is performed by a Java applet at the client. A fast shear-warp-based method previously described by the Authors [15, 9] is used and extended to provide more flexible influence of data channels on the results of rendering. The 12 Bit gradient representation is used to directly index a look-up table containing shading information for interactive lighting (appendix, Fig. 4a). Using a shading table filled according to a non-photorealistic shading equation [3] and using the result to modulate voxel opacity, interactive non-photorealistic rendering can be realized. Using gradient-based shading and an additional gradient magnitude channel, the classical gradient-modulated transfer functions of Levoy [13] can be realized. Using one additional data channel (containing distance information) to modulate either color or opacity (appendix, Fig. 4b) the visualization of contacts between objects can be enhanced.

7 Results

Table 1 presents the compression rates obtained by applying our technique to a collection of data-sets from different application fields. The head and hand data-sets are CT scans containing objects typical for medical applications. Bone and skin surfaces extracted from the data are usually made up from 1–4% of all voxels. Using our compression scheme the boundary data is compressed by a factor of 20–90 compared to

data-set	*volume size*	*obj. voxels*	*Bit/pos*	*Bit/grad*	*Bit/voxel*	*file(w/o grad.)*	*ratio to gzipped vol.*
head-bone	$256^2 * 158$	378k	2.0	7.0	9.0	430k(95k)	1:22(1:97)
head-skin	$256^2 * 158$	231k	2.1	5.8	7.9	229k(60k)	1:40(1:154)
hand-bone	$256^2 * 232$	191k	2.5	7.8	10.3	246k(60k)	1:45(1:186)
hand-skin	$256^2 * 232$	170k	2.0	4.0	6.0	126k(41k)	1:89(1:273)
engine	$256^2 * 110$	298k	1.7	5.1	6.8	253k(64k)	1:13(1:51)
teapot	256^3	152k	1.7	3.4	5.1	80k(28k)	1:4(1:11)
attractor	256^3	769k	1.8	4.9*	6.7	639k(170k)	–**
basin	256^3	292k	2.2	0.6*	2.8	104k(80k)	–**

Table 1. Compression survey. * Scalar value channel instead of gradients. ** The attractor and basin data-sets have been extracted from a volume with a vector of several scalar values at each voxel directly within the simulation application. No volumetric representation was available.

the original volume when compressed with `gzip`. If gradient information is not stored but approximated at the client the compression factor increases to 100–270. The cost of compressing voxel positions within such data-sets is relatively independent of the surface shape 2–2.5 Bit/voxel. The cost for storing gradients depends on the smoothness and curvature of the surface and varies between 4 and 8 Bit/voxel. For objects with artificial, "well-behaved" surfaces like the CT scan of an engine block or the voxelized teapot, better compression is achieved for both voxel position and gradient data. The attractor and basin of attraction data, obtained from a simulation of a dynamical system, is also effectively compressed – especially as the basin boundary is derived from a binary classification of space and no gradient information has to be stored – it can be reconstructed from the surface shape at the client. Compression for each of the examples mentioned above takes approximately one second on a PIII/733 PC. Decompression timings for locally stored data are similar on the same PC. An applet which implements the techniques described in this paper and all compressed data-sets discussed and depicted here are available at `http://bandviz.cg.tuwien.ac.at/basinviz/compression/`.

8 Conclusions

Many applications of volume visualization require the display of objects boundaries. Using our compact volume representation, volume visualization becomes feasible even over the Internet, while still providing full spatial accuracy. Representing just the boundary voxels of objects reduces the amount of data to be transmitted or stored dramatically. By exploiting known properties of the boundary voxels (like spatial coherence and inter-voxel connectivity) the data is further compressed. The resulting data representation is smaller by a factor of 20-250 than the volume compressed with `gzip`. The location of voxels within the volume is compressed very efficiently to about 2 Bit/voxel. The compression rates for gradient data are lower, in the range of 3-8 Bit/voxel, as gradient data is derivative information compared to the original data, containing less spatial coherence. Using a proper gradient reconstruction scheme, gradients can be estimated from voxel positions only, allowing to display objects just after the well-compressed position data has arrived, instead of waiting for the original gradient information. The display of

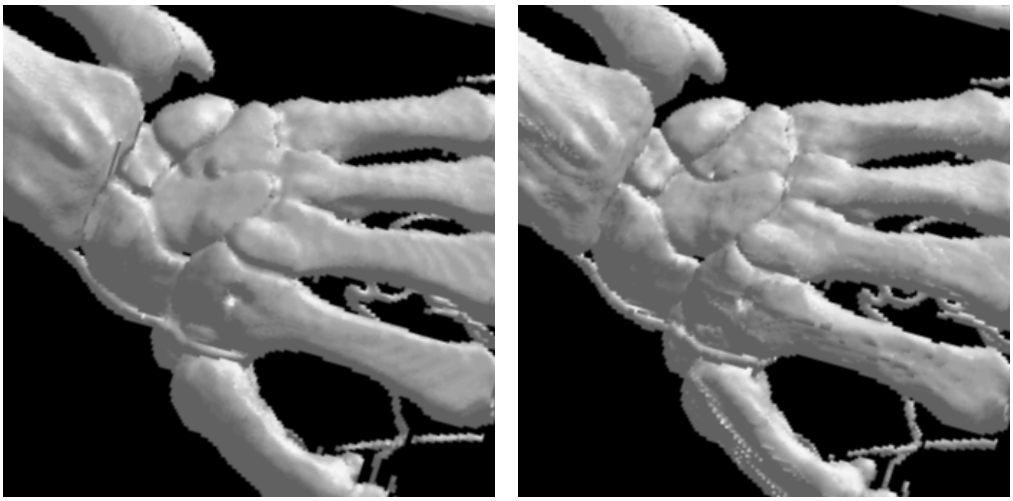
the boundary data can be performed in pure software (Java) at interactive frame rates without the need for any hardware support .

Acknowledgements

The work presented in this paper has been financed within the Kplus program of the Austrian government, as well as the BandViz project (`http://bandviz.cg.tuwien.ac.at/`) which is supported by the FWF under project number P 12811. The medical data-sets are courtesy of Tiani MedGraph.

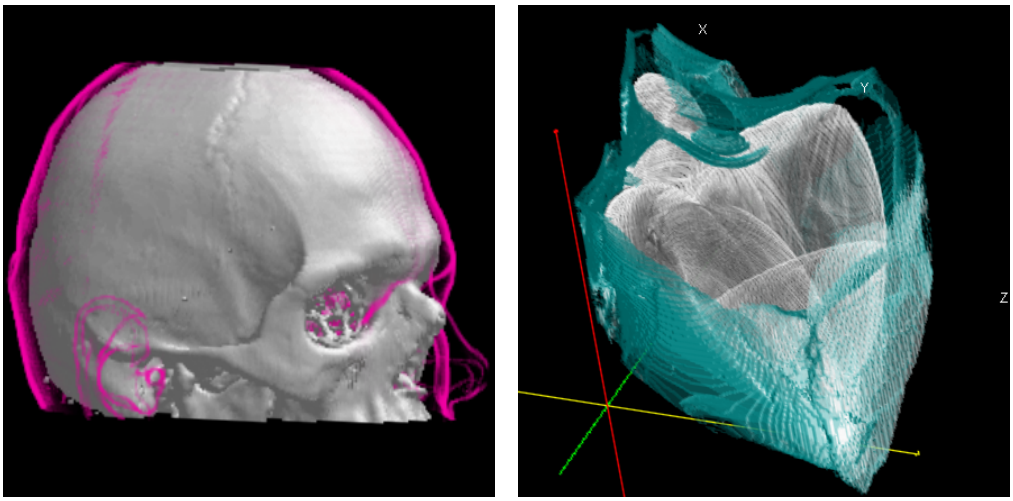
References

1. G.-I. Bischi, L. Mroz, and H. Hauser. Studying basin bifurcations in nonlinear triopoly games by using 3D visualization. *accepted for publication in Journal of Nonlinear Analysis*.
2. T. Chiueh, C. Yang, T. He, H. Pfister, and A. Kaufman. Integrated volume compression and visualization. In *Proceedings IEEE Visualization '97*, pages 329–336, 1997.
3. D. Ebert and P. Rheingans. Volume illustration: non-photographic rendering of volume models. In *Proceedings IEEE Visualization 2000*, pages 195–202, 2000.
4. K. Engel, P. Hastreiter, B. Tomandl, K. Eberhardt, and T. Ertl. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings IEEE Visualization 2000*, pages 449–452, 2000.
5. K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *Proceedings IEEE Visualization '99*, pages 139–146, 1999.
6. J. Fowler and R. Yagel. Lossless compression of volume data. In *Proceedings IEEE Volume Visualization Symposium '94*, pages 43–50, 1994.
7. J. Gailly and M. Adler. gzip. URL: `http://www.gzip.org`.
8. R. Haber and D. McNabb. *Visualization idioms: A conceptual model for scientific visualization systems, visualization in scientific computing*, pages 74–93. 1996.
9. H. Hauser, L. Mroz, G.-I. Bischi, and E. Gröller. Two-level volume rendering - fusing MIP and DVR. In *Proceedings IEEE Visualization 2000*, pages 211–218, 2000.
10. M. Jern. Information drill-down using web tools. In *Proceedings of the 8th EUROGRAPHICS Workshop on Visualization in Scientific Computing*, pages 1–12, 1997.
11. R. Estes JR and V. Algazi. Efficient error free chain coding of binary documents. In *Proceedings of the Data Compression Conference*, pages 122–132, 1995.
12. C. Kurmann L. Lippert, M. Gross. Compression domain volume rendering for distributed environments. In *Proceedings of Eurographics '97*, pages C95–C107, 1997.
13. M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.
14. W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH '87*, pages 163–189, 1987.
15. L. Mroz, R. Wegenkittl, and E. Gröller. Mastering interactive surface rendering for java-based diagnostic applications. In *Proceedings IEEE Visualization 2000*, pages 437–440.
16. L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4D linear regression. In *Proceedings of Eurographics 2000*, pages C–351–C–357.
17. P. Ning and L. Hesselink. Fast volume rendering of compressed data. In *Proceedings IEEE Visualization '93*, pages 11–18, 1993.
18. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro real-time ray-casting system. In *Proceedings of ACM SIGGRAPH '99*, pages 251–260, 1999.



(a) (b)

Fig. 3. Estimated gradients (a) are used for shading until the original gradient data has arrived (b)



(a) (b)

Fig. 4. a) By adjusting the visualization mappings at the client the skin surface has been rendered using a non-photorealistic technique over the conventionally shaded skull. b) A data channel containing distance information has been used to modulate opacity of the basin surface to emphasize areas of almost-contact between the surface and the attractor contained within.

Interactive High-Quality Maximum Intensity Projection

Lukas Mroz, Helwig Hauser †, Eduard Gröller ‡

Abstract

Maximum Intensity Projection *(MIP) is a volume rendering technique which is used to visualize high-intensity structures within volumetric data. At each pixel the highest data value, which is encountered along a corresponding viewing ray is depicted. MIP is, for example, commonly used to extract vascular structures from medical data sets (angiography). Due to lack of depth information in MIP images, animation or interactive variation of viewing parameters is frequently used for investigation. Up to now no MIP algorithms exist which are of both interactive speed and high quality. In this paper we present a high-quality MIP algorithm (trilinear interpolation within cells), which is up to 50 times faster than brute-force MIP and at least 20 times faster than comparable optimized techniques. This speed-up is accomplished by using an alternative storage scheme for volume cells (sorted by value) and by removing cells which do not contribute to any MIP projection (regardless of the viewing direction) in a preprocessing step. Also, a fast maximum estimation within cells is used to further speed up the algorithm.*

1. Introduction

The ability to depict blood vessels is of great importance for many medical imaging applications (angiography). CT and MRI scanners can be used to obtain volumetric data sets, which allow the extraction of vascular structures. Especially data originating from MRI, which is most frequently used for this purpose, exhibits some properties which make the application of other volume visualization techniques like ray casting[4] or iso-surface extraction[6] difficult. MRI data sets, for example, contain a significant amount of noise. Inhomogeneities in the sampled data make it difficult to extract surfaces of objects by specifying a single iso-value. In addition, vascular structures and especially thin vessels cover a wide range of data values, which makes the extraction by conventional techniques also difficult.

Maximum intensity projection (MIP) exploits the fact, that within angiography data sets the data values of vascular structures are higher than the values of the surrounding tissue. By depicting the maximum data value seen through each pixel, the structure of the vessels contained in the data is captured. A straight-forward method for calculating MIP is to perform ray casting and search for the maximum sample value along the ray instead of the usual opacity and color composition. In contrast to direct volume rendering, no early ray termination is possible, making standard MIP even more expensive.

Usually, MIP contains no shading information, depth and occlusion information is lost. Structures with higher data value lying behind a lower valued object appear to be in front of it. The most common way to ease the interpretation of such images is to animate or interactively change the viewpoint while viewing. This can be achieved using one of the interactive MIP techniques[1, 3, 7] which - up to now - were not able to generate high-quality images. For performance reasons these techniques perform no resampling of the original data values during maximum evaluation, thus introducing aliasing and producing images of moderate quality (Figure 1). For exact depiction of even small features there is need for algorithms which produce high-quality MIP in real-time. In contrast to interactive MIP techniques which perform just nearest neighbor interpolation, more accurate evaluation of ray maxima is required for the generation of high-quality MIP. At the cost of significantly longer computation times, this allows to create much more detailed images and accurate animations.

Depending on the quality requirements of the resulting image and the desired performance, different strategies for finding the maximum value along a ray can be used:

† former name: Helwig Löffelmann

‡ Institute of Computer Graphics, Vienna University of Technology, Karlsplatz 13/186/2, A-1040 Vienna, Austria. email:{mroz, hauser, groeller}@cg.tuwien.ac.at

(a) (b)

Figure 1: *MIP with a) shear-warp projection with nearest neighbor interpolation b) ray casting and trilinear interpolation.*

- **Analytical solution:** Usually data values within a cell are reconstructed using trilinear interpolation of the data values at the cell vertices. For each data cell, which is intersected by the ray, the maximum value encountered by the ray is calculated analytically. For trilinear interpolation within cells this means that the maximum of a cubic polynomial must be determined. This is the most accurate but also computationally most expensive method[9].
- **Sampling and interpolation:** As usually done for ray casting, data values are sampled along the ray using trilinear interpolation (See Figure 6a). The cost of this approach depends on the resampling step size along the ray. Depending on how many interpolations, which do not affect the result, can be avoided[9, 10], acceleration can be gained up to a certain limit.
- **Nearest neighbor interpolation:** Values of the data samples closest to the ray are taken for maximum estimation. In combination with discrete ray traversal this is the fastest method. As no interpolation is performed, the voxel structure is visible in the resulting image as aliasing[1] (See Figures 1a, 6b).

Recent algorithms for MIP employ a set of approaches for speeding up the rendering:

- **Optimization of ray traversal and interpolation:** Sakas et al.[9] evaluate cell maxima only if the maximum value of the examined cell is larger than the ray-maximum calculated so far. For additional speedup they use integer arithmetics for ray traversal and a cache-coherent volume storage scheme. Zuiderveld et al.[10] apply a similar technique to avoid trilinear interpolations. In addition, cells containing only background noise (usually rather low values) are not interpolated. For further speedup a low-resolution image containing lower-bound estimations of the maximum of pixel clusters is generated before the main rendering step. Cells with values below this bound can be skipped when the final image is generated. Finally a distance-volume is used to skip empty spaces. A careful definition of "empty space" is required to avoid that small, low valued structures are missed.
- **Use of graphics hardware:** Heidrich et al.[3] use conventional polygon rendering hardware to simulate MIP. Several iso-surfaces for different threshold values are extracted from the data set. Before rendering, the geometry is transformed in a way, that the depth of a polygon corresponds to the data value of its iso-surface. MIP is approximated by displaying the z-buffer as a range of gray values. Recent versions of SGI OpenGL include blending modes for maximum evaluation and the SGI Volumizer API includes a MIP option. However the advantages of hardware-acceleration for high-quality MIP are only available for upper range graphics workstations. Recently the VolumePro board[8] became available as a purely hardware-based solution for volume rendering and also MIP. For generating MIP of a quality which is comparable to our approach, 4 times super-sampling has to be employed, reducing the frame rates of the VolumePro board to approximately 4 fps for a 128^3 volume.
- **Splatting and shear warp factorization:** Several approaches[1, 2] exploit the advantages of shear-warp rendering[5] to speed up MIP. Cai et al.[1] use an intermediate "worksheet" for compositing interpolated intensity contributions of voxels for projection of a single slice of the volume. The worksheet is then combined with the shear image to obtain the maxima. Several splatting modes with different speed / quality tradeoffs are available. Run-length encoding and sorting of the encoded segments by value are used to achieve further speedup.
- **Elimination of irrelevant voxels and alternative storage schemes:** In a previous paper we showed, that many voxels of a volumetric data set do not contribute to any MIP[7]. Significant speedup is gained by identifying them before rendering and excluding them from image generation. To avoid overhead for skipping them during rendering, voxels which might contribute to a MIP image are stored in a list and sorted according to their data value. By processing this list from low to high intensity voxels and

projecting them using a computationally inexpensive projection (template-based or shear-warp parallel projection), real-time MIP is achieved (See Figure 6c). One purpose of this paper is to extend this idea to high-quality MIP using trilinear interpolation.

Two major limitations to the performance of state-of-the-art software based high-quality MIP can be identified. First, although regions of the volume which do not contain meaningful data can be identified in advance and skipped during rendering using distance volumes or similar structures[10], the overhead associated with evaluating the additional information and stepping over these cells significantly limits the possible speedup of volume traversal. Secondly, despite of space leaping approaches, the number of interpolations which are actually performed is still far from optimum. As the volume is traversed in a spatially ordered manner along viewing rays, local maxima are usually encountered and evaluated before the global ray maximum is reached. Moreover, lots of unnecessary evaluations are performed on the rising slopes of data value which precede a maximum.

In this paper we present a new algorithm for the generation of high-quality MIP (parallel projection) which is approximately one to two orders of magnitude faster than other software-based approaches with comparable quality. In Section 2 we present a preprocessing scheme, which can (but not necessarily has to) be applied to identify and exclude non-contributing cells from the volume. To maximize the amount of cells, which do not contribute to any MIP, 12 sets of cells are generated, corresponding to 12 clusters of viewing directions. Usually, more than two thirds of all cells can be eliminated from the data completely, no longer causing any overhead to identify them later and skip over them. This is achieved by resorting the cells according to their maximum value (Section 3). As the spatial order of processing cells is not relevant for maximum evaluation, cells containing high data values are evaluated first. This reduces the number of evaluations required for MIP significantly. To avoid even more of the relatively expensive trilinear interpolations, we use a fast method for estimating the maximum value along a ray-cell intersection (Section 4). Using these techniques, we are able to greatly reduce the number of trilinear interpolations required per image pixel, achieving interactive high-quality MIP.

2. Preprocessing of Volume Data

The methods described in the following sections are based on an interpretation of the data set as a regular grid of cells, each one defined by eight data samples of the volume located at the cell's vertices. Within cells trilinear interpolation is assumed. For image generation continuous rays are shot through the pixels of the image, allowing arbitrary image sizes as well as oversampling.

2.1. Motivation

Although time used for volume traversal is more significant as a portion of overall rendering time for computationally inexpensive maximum evaluation and projection methods, identifying and skipping not interesting regions also lowers the number of candidates for a costly maximum evaluation. For trilinear interpolation within cells the cell maximum is always located at one of the vertices. Besides skipping pre-identified empty regions of the volume using distance volumes, another simple way to increase efficiency is to perform maximum evaluations only within cells with a cell maximum C_{max} greater than the current ray maximum. This technique avoids the evaluation of cells reached by rays after processing a (local) maximum. If in addition a good lower-bound estimation of the ray maximum can be obtained before rendering, also cells encountered before the maximum can be skipped, if their maximum is lower than the lower-bound estimation.

Although these methods reduce the number of required evaluations significantly, the lower-bound estimation has to be performed for each new viewing direction and much time is spent during rendering on identifying and then skipping cells and empty regions.

To increase time savings during rendering, cells which do not contribute to any MIP should be identified and removed during a preprocessing step.

2.2. Cell Removal

A cell C does not contribute to MIP from any viewing direction (and therefore should not to be considered for rendering), if all rays passing through it collect a higher value by passing through other cells D either before or after C is processed. Our first approach is to investigate direct neighbors D_i of a cell C only. C does not contribute to any ray through it if $\forall i | D_{imin} \geq C_{max}$. As we assume continuous rays to be traced through the volume, only face-connected neighbors of C have to be considered (A ray entering C through an edge or vertex at least "touches" some face-connected neighbors).

Applying such a unified relevance detection for cell elimination, which considers the whole domain of viewing directions, is very ineffective and leads to low cell removal rates. Significantly better results can be achieved if several distinct clusters of similar viewing directions are distinguished. For rendering, the set of cells corresponding to the cluster which contains the current viewing-direction is selected and used for MIP. To minimize the number of neighbors which have to be checked for elimination, a decomposition of all viewing directions into 24 clusters is performed first. Each cluster of viewing directions corresponds to a quarter-face of the directional cube as depicted in Figure 2b. Considering just viewing directions out of one cluster a cell's relevance to a MIP depends on just three neighbors (Figure 2c) instead of

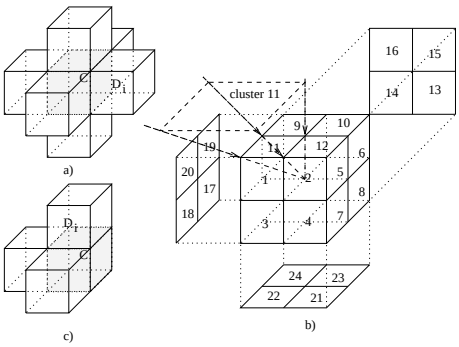


Figure 2: *a) relevance of cell C depends on values of 6 face-connected neighbors D_i. b) decomposition of the viewing-domain into 24 (12) clusters of viewing directions. c) for any viewing direction within cluster 11 relevance of C depends on the values of at most 3 direct neighbors*

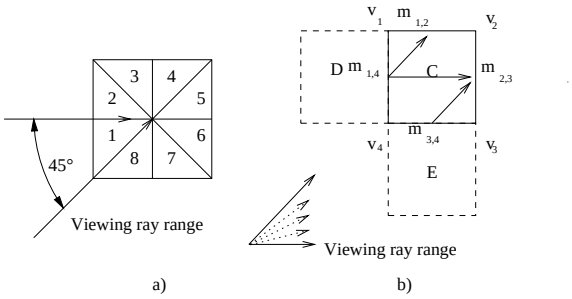


Figure 3: *a) Viewing domain decomposition in 2D. b) face-maximum propagation for cluster 1*

six as in the case of viewpoint-independent preprocessing (Figure 2a).

As the direction of ray traversal is not relevant for MIP and a MIP generated from a certain viewing direction produces exactly the same image as a MIP from the opposite direction, sets of possibly contributing cells have only to be calculated for 12 of the 24 clusters of viewing directions. Furthermore, as can be seen in Figures 2b and c, three clusters of viewing directions around each corner of the directional cube result in the same set of direct neighbors on which a cell's relevance depends (for example clusters 1, 11 and 20). Unfortunately, the clusters differ in the way in which the estimations for rays entering through cell faces combine to the estimations for exiting faces. As our removal algorithm uses face estimations for determining non-contributing cells, the clusters can not be combined without sacrificing removal efficiency.

To achieve effective cell elimination, not only the influence of direct neighbors, but also the influence of more distant cells on rays has to be investigated. This can be done by applying a simple two-pass scheme for each cluster of viewing directions. For reasons of simplicity, the procedure will be explained in 2D. The extension to 3D is straight-forward.

For cell removal in 2D, the viewing domain is decomposited into 8 clusters, each one covering a range of viewing directions spanning 45 degrees (Figure 3a). Rays within cluster 1 can enter cell C only through edge $\overline{v_1v_4}$ or $\overline{v_4v_3}$. Considering just the values at the cell's vertices, cell C has no influence on the maximum of rays through $\overline{v_1v_4}$, if $min(v_1, v_4) \geq max(v_2, v_3)$. In this case, the maximum contribution of the cell to any ray entering through $\overline{v_1v_4}$ and leaving through $\overline{v_1v_2}$ or $\overline{v_2v_3}$ is located on the edge $\overline{v_1v_4}$. As this edge is shared with cell D which is traversed by the rays earlier, C does not contribute to the rays. Similarly, the cell has no influence on rays through $\overline{v_4v_3}$ if $min(v_3, v_4) \geq max(v_1, v_2)$ due to the contribution of cell E. The condition for $\overline{v_4v_3}$ has to consider the influence of v_1, as a large data value at v_1 may influence the data gradient within the cell in a way that rays through $\overline{v_4v_3}$ obtain a larger value within the cell than at $\overline{v_4v_3}$ or $\overline{v_2v_3}$.

Preprocessing the volume in a spatially consecutive order, the influence of cells on ray maxima can be propagated with an advancing front approach. In the example in Figure 3 cells D and E are processed before cell C is reached. For both of them, lower-bound estimations of the maximum for rays which leave the cells have been calculated (for this cluster, rays leave cells either through face $\overline{v_2v_3}$ or through face $\overline{v_1v_2}$). This means, that at the time C is processed, lower-bound estimations $m_{1,4}$ for rays entering through v_1v_4 and $m_{3,4}$ for rays entering through v_3v_4 are available, which already include the influence of more distant (preceding) cells and the influence of the edges themselves ($min(v_1, v_4)$ and $min(v_3, v_4)$). Thus, the revised test for the irrelevance of cell C is $m_{1,4} \geq max(v_2, v_3)$ and $m_{3,4} \geq max(v_1, v_2)$. If both conditions are true, C is removed and never ever considered for MIP anymore. After classifying the cell, the lower-bound estimations for the maxima of rays leaving the cell have to be computed. As for the investigated cluster of viewing directions only rays which have entered the cell through $\overline{v_1v_4}$ can leave through $\overline{v_1v_2}$, the estimation for rays through $\overline{v_1v_2}$ is $m_{1,2} = max(min(v_1, v_2), m_{1,4})$. As rays entering through both, $\overline{v_1v_4}$ and $\overline{v_3v_4}$ can leave through $\overline{v_2v_3}$, the estimation for $v_2v_3 = max(min(v_2, v_3), min(m_{1,4}, m_{3,4}))$.

While the first sweep allows to identify and remove low-valued cells located behind higher-valued parts of the volume, a second sweep in the opposite direction is required to propagate the influence of high-valued cells to cells reached earlier by the rays of this cluster. The second sweep is identical with the first sweep with the exception of the inverted orientation of the rays and thus an inverted volume scan order. Cells which have been removed during the first sweep do not influence ray values during the second sweep. Considering also cells removed during the first sweep would result in mutual elimination of cells and lead to holes in the volume.

The two-pass method presented above can be extended to

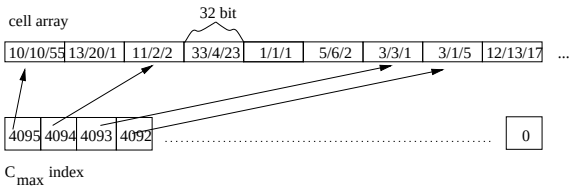


Figure 4: *Cell array storage scheme: all cells are sorted according to C_{max} in descending order. Their position in space is stored in an array (4 bytes per cell). An additional index points to the first cell in the array for each possible value of C_{max}. All cells with the same C_{max} are sub-sorted according to C_{min} in descending order.*

3D in a straight-forward way. For a decomposition of the viewing domain into 24 (12) clusters, rays may enter and leave cells through three faces for each cluster. Lower-bound estimations have to be propagated for faces instead of edges.

2.3. Noise Compensation

To even more increase the efficiency of removal and compensate for noise which is usually present in MRI data sets, the removal process can be modified to remove also cells which violate the exact criteria by a factor which does not exceed a user specified threshold (removal tolerance). After the preprocessing with a 1% tolerance, on average only about 30% of all cells remain as possibly contributing for a single view-set. For detailed results please refer to Section 5 and Table 3.

3. Cell Storage

In the following, we will use $(x,y,z) = (min(x_i), min(y_i), min(z_i))$, (x_i,y_i,z_i) being the vertex coordinates, as reference coordinates of a cell.

As the order in which cells are processed is not relevant for MIP, we are able to abandon the usual storage scheme for volumes, which is a three dimensional array. We use, instead, a storage scheme which is well suited for omitting irrelevant cells during rendering[7]. Cells are sorted according to descending maximum values within a 1D cell array. For every cell its reference coordinates are stored using 4 Bytes – packing 3 indices (x,y,z) allows to store volumes up to a size of 2048x2048x1024 cells. Assuming data values to be represented by 2 bytes, this alternative storage scheme requires twice the amount of memory (without any cell removal). Using this alternative storage scheme it is trivial to start MIP with high-valued cells. Speed up factors of 10-20 compared to other optimized high-quality MIP approaches are gained by just this part of our approach in combination with the usual optimizations like evaluating only cells with $C_{max} >$ray max and noise skipping (See Table 1, row 3).

Although sorting all cells within a set by C_{min} would most effectively reduce the number of interpolations required, sorting them by C_{max} has several advantages. First, C_{max} is implicitly encoded into the position of a cell in the array by using an additional array of indices which contains, for each possible value of C_{max}, the index of the first cell in the cell array with the particular value of C_{max} (see Figure 4). The implicit encoding of C_{max} allows to access it in an extremely efficient way for testing a cell's relevance during projection. Within a group of cells with the same C_{max} sub-sorting is done according to descending C_{min}. Due to the small range of different values, sorting by C_{max} and C_{min} can be done using fast histogram-based sorting (complexity $O(N)$). During projection, cells with high C_{max} and C_{min} are processed first.

For display of medical images usually a remapping of data values to gray levels (windowing) is applied to enhance the contrast for a specific range of data values which is of utmost interest to the viewer. A window is defined by a *center c* and a *width* value *w*, and defines a mapping of data-values below $c - w/2$ to black, values above $c + w/2$ to white, and values between $c - w/2$ and $c + w/2$ to a uniform ramp of gray levels. If cells are sorted by C_{max} and rendering is started with highest values of C_{max}, rendering can be stopped after reaching the first cell with C_{max} mapped to black. For realistic window definitions as used by medical doctors this can significantly speed up the rendering (up to several times faster).

Besides the fast computation of MIP due to re-sorting, another big advantage is gained: progressive refinement is achieved automatically, as cells which are most relevant to MIP are projected first. Projection can be stopped any time - the result will always be optimal for the given time-constraints. Also, computation of cheap previews is simple. Since interaction is crucial for using MIP, this advantage is also very important for practical use.

In the following a rough comparison between the traditional way of volume storage and our cells array is given: Storing the position instead of data values doubles the amount of memory required. As roughly 30% of all cells remain after the preprocessing step (cell removal), the amount of memory required is about 0.6 * original volume size (per cluster of viewing directions). For 12 clusters this results in an approximately sevenfold increase in memory size compared with the original data set. Considering data sets from medical applications and regular hardware resources, the storage requirements are acceptable. If, nevertheless, the memory resources are a limiting factor, the data set can also be transformed into a single cell array without prior view-dependent preprocessing. This, of course, slightly increases rendering time (See Table 5) , but still gains results significantly faster than conventional approaches of comparable image quality, while requiring just twice as much memory as the original data set.

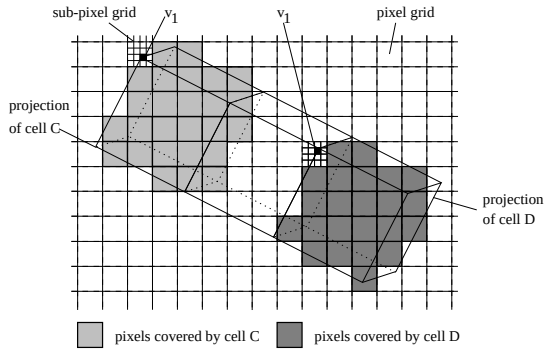


Figure 5: *Cell projection templates. As the projections of cell C and cell D (position of v_1) have different sub-pixel offsets, the shape of pixel-sets affected by the cells differs. Thus C and D require different templates for projection.*

4. Rendering

To achieve maximum rendering performance, precalculated templates are used to determine all pixels of the image which are covered by a cell's projection (Section 4.1). A fast parallel projection is used to calculate the position of a cell's projection in the image (Section 4.2). Finally, the order of traversal of the cell array and a fast heuristic estimation of the upper-bound for the maximum value along a cell-ray intersection reduce the number of more costly and accurate trilinear maximum evaluations required (Section 4.3).

4.1. Template Calculation

The main purpose of the template is to allow fast identification of the pixels affected by the projection of a cell. At each of these pixels, the cell's influence on the maximum of the ray through this pixel and thus to the pixel value has to be evaluated. A sufficiently accurate calculation of this contribution requires several steps of trilinear interpolation along the ray within the cell. To save time during cell projection, it is quite useful to pre-calculate entry and exit coordinates of the ray for each pixel of the template. Interpolation weights (u, v, $w \in [0,1]$ for trilinear interpolation) of the entry and exit points are stored for this purpose.

As only parallel projection is used, the shape and size of the projected image of all cells is identical in a continuous image space. Due to arbitrary scaling and rotation of the volume for viewing and the discrete nature of a pixelized image, images of cells differ by an individual sub-pixel displacement with respect to the pixels of the image (see Figure 5), which also leads to differing sets of pixels covered by the cell's image. To account for this shift with sufficient accuracy, the projection has to be performed with sub-pixel accuracy, allowing to place a cell's image in between image-pixel positions. The placement on a 4x4 grid within a pixel produces satisfying results.

The placement of cell images on sub-pixel positions leads to slightly different shapes of the templates for different x/y displacements and also requires the calculation of individual ray entry/exit positions for each of the templates. The resulting (4x4) array of templates can be directly accessed during rendering using the sub-pixel displacements of a cell's projection.

To optimize the rendering performance each element (=pixel) of the template stores a set of values:

- the offset of this element's pixel from the pixel containing the projection of the cell's origin (cell vertex v_1). As the image of v_1 can be located anywhere within the cell's image depending on the viewing direction, the pixel offsets may also become negative.
- the interpolation weights (u,v,w) for ray entry and exit coordinates at this element.
- the number of interpolation steps required along the ray / cell intersection
- A (du,dv,dw) vector defining a step along the ray.

Before rendering, the template is optimized to speed up access. To avoid the necessity of skipping non-covered pixels within a template, just a list of covered template entries is stored instead of a 2D array. Thus, each template is just an array of image offset / ray information elements for locations covered by a cell's projection.

4.2. Projection

As the parallel projection of a point can be performed by independently projecting it's x, y, z coordinates,

$$P\begin{pmatrix} x \\ y \\ z \end{pmatrix} = P\begin{pmatrix} x \\ 0 \\ 0 \end{pmatrix} + P\begin{pmatrix} 0 \\ y \\ 0 \end{pmatrix} + P\begin{pmatrix} 0 \\ 0 \\ z \end{pmatrix}$$

The projected positions x_{img}, y_{img} can be precalculated for the projection of all possible x, y, and z cell coordinates within the volume. This results in six arrays - one for each x_{img}, y_{img} position of each x, y, and z coordinate.

For performance reasons we use integer positions. As the summation of 3 integer values to obtain the x_{img} or y_{img} position of a cell's projection introduces an error of up to 1.5 pixels in x_{img} and y_{img}, the arrays have to contain sub-pixel coordinates to keep the error below one pixel.

In combination with precalculated templates, the projection of a cell becomes simple and efficient. `Cmax` is obtained in a way described in Section 4.3, `img_width` is the width of the image in pixels. `.pixel` is the x or y coordinate of an image pixel, `.subpix` is a sub-pixel offset.

```
(xp,yp)=projection(cell.v1);
imgpos=xp.pixel+yp.pixel*img_width;
template=subtemplate[xp.subpix, yp.subpix];
for all elements in template
{
```

```
  if(image[imgpos+
     template_element.imgoffset]<Cmax)
    evaluate cell contribution at this pixel
}
```

4.3. Evaluation of the Maximum

Compared to the other stages of MIP generation, the evaluation of maximum values within cells (trilinear interpolation) is by far the most expensive part. The reduction of the number and effort of evaluations required to generate an image is crucial for performance. Performance can be improved on one hand by using a less expensive but usually also less accurate evaluation method to approximate the maximum. On the other hand, more evaluations can be omitted if a good guess for the ray maximum can be found early. The sorted cell array allows to access and render most promising cells first. If the rendering is started with the projection of cells which have the highest cell maximum and minimum, the probability of having to evaluate successive cells projected on the same pixels is significantly reduced. As can be seen in Table 4, only about 2-4% of the cells of the original data set require the use of trilinear interpolation to evaluate their possible contribution to a MIP. The evaluation of the remaining cells is stopped either after checking C_{max} or after performing the slightly more expensive maximum estimation described below.

Values of pixels covered by the current cell which are lower than the cell maximum potentially have to be replaced by a higher value. An analytical solution for the maximum along the ray through the pixel is extremely expensive, and a few trilinear interpolation steps along the ray are also quite costly. A cheap estimation of an upper bound for the ray maximum which is more restrictive than the cell maximum C_{max} can greatly reduce the number of more costly exact evaluations of the maximum. We facilitate the following observations to present such a heuristic:

- If the maximum is located on the entry or exit point of the ray: applying two bilinear interpolations on the entry and exit faces of the cell gains the exact value for the maximum as $max(entry, exit)$.
- If for this ray the maximum is located within the cell, there must be some positive deviation from a linear evolution between the entry and exit point of the ray. If a cheap approximative guess for this nonlinearity within the cell can be found, the upper-bound of the ray can be estimated as $min(C_{max}, max(entry, exit) + deviation)$.

A fast approximative estimation of this deviation is $deviation = Max(0, Max((v_i + v_j)/2) - c)$ with v_i, v_j being data values at the vertices located at the ends of the four space diagonals of the cell and c being the trilinearly interpolated value at the center of the cell (cheap, as special case). Although this estimation is not a strict upper bound in all cases (99%), no visible impact on images of real-world data sets has been found. On average, this estimation for the ray maximum within a cell is 30% lower compared with C_{max} as an estimation. When using this estimation about 60% less of significantly more expensive trilinear evaluations have to be performed (Table 4). As only 25-30% of the trilinear evaluations lead to a change of a pixel value, a more tight upper bound estimation could gain even more performance. If the estimated upper bound for the ray is above the value of the examined pixel, several steps of trilinear interpolation within the cell are performed utilizing information stored in the templates to obtain the ray maximum.

method	*interpol.*	*cells*	*pix.set*
brute force	100%	100%	24
$+C_{max} > max$	26%	29%	24
+ignore noise	7.7%	7.8%	13
cell array	1.3%	2.5%	1.65

Table 2: *Comparison of interpolation efficiency: brute force (row 1) interpolates at every step along ray (col. 1), within every cell (col. 2). Each pixel changes 24 times (col. 3). Row 2: interpolation only if $C_{max} > max$. Row 3: also ignore low-valued cells. Row 4 shows the results for our approach.*

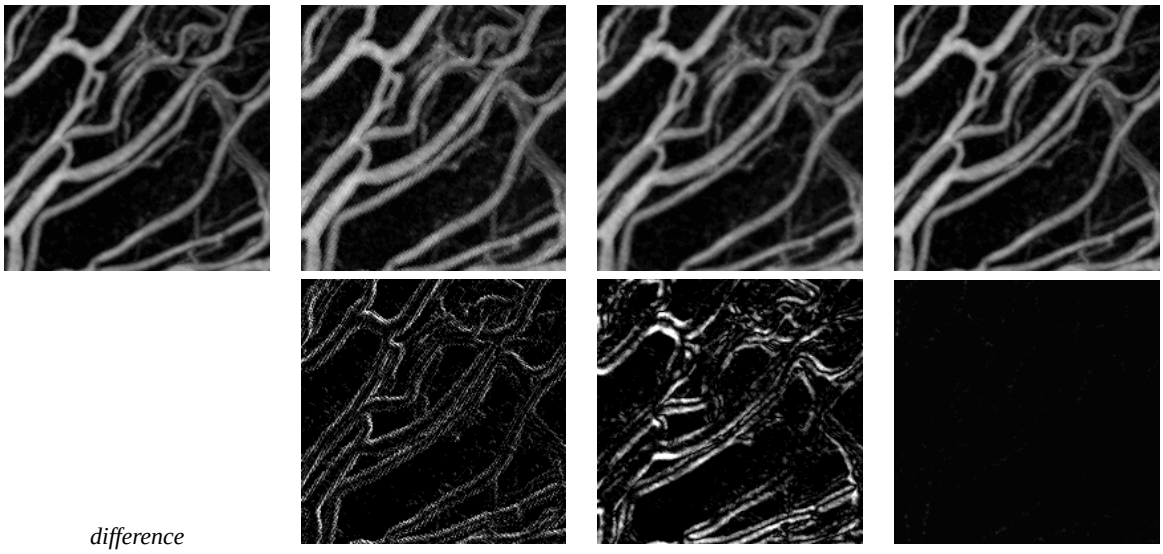
data set	*size*	*toler.*	*sweep 1*	*sweep 2*	*total*
hand	256^2x100	1%	81%	3%	84%
hand	256^2x100	2%	93%	2%	95%
kidney	256^2x69	1%	56%	5.5%	61%

Table 3: *Cell elimination results*

Projecting high valued cells first and using an additional estimation of the ray/cell maximum is very efficient, as the value of each non-background pixel of the image is set only 1.3 to 4 times compared to 10-25 times for MIP using conventional ray-casting with optimizations.

For the following pseudo-code summary of the projection of the cell array, `gray[]` stores the mapping from data-values to gray levels defined by the windowing function.

```
cells=get_cell_set(viewmatrix);
calculate_template(viewmatrix);
calculate_projection_arrays(viewmatrix);
for Cmax=highest to 0
  if(gray[Cmax]>0)
    for cell=cells.first_cell_with_Cmax
           to cells.last_cell_with_Cmax
      project(cell);
```



difference
a) trilinear interpolation *b) n. neighbor ray casting* *c) n. neighbor shear-warp* *d) cell array*

Figure 6: *Quality comparison of MIP produced using a) ray casting with trilinear interpolation, b) ray casting with nearest neighbor interpolation, c) shear-warp projection with nearest neighbor interpolation, d) sorted cells and trilinear interpolation. The difference images depict amplified difference with respect to ray casting with trilinear interpolation.*

levels of our approach	*preprocessing time*	*memory factor*	*speed-up compared to* *brute force*	*+$C_{max} < max$*	*+ignore noise*
cell array	6s	2.0	1.3-1.4		
+$C_{max} < max$	6s	2.0	6.1-7.4	3.9-5.3	3.1-4
+ignore noise	6s	2.0	20-40	14-25	11-20
+cell elimination	93.6s	3.6-12	28-43	18-27	24-22

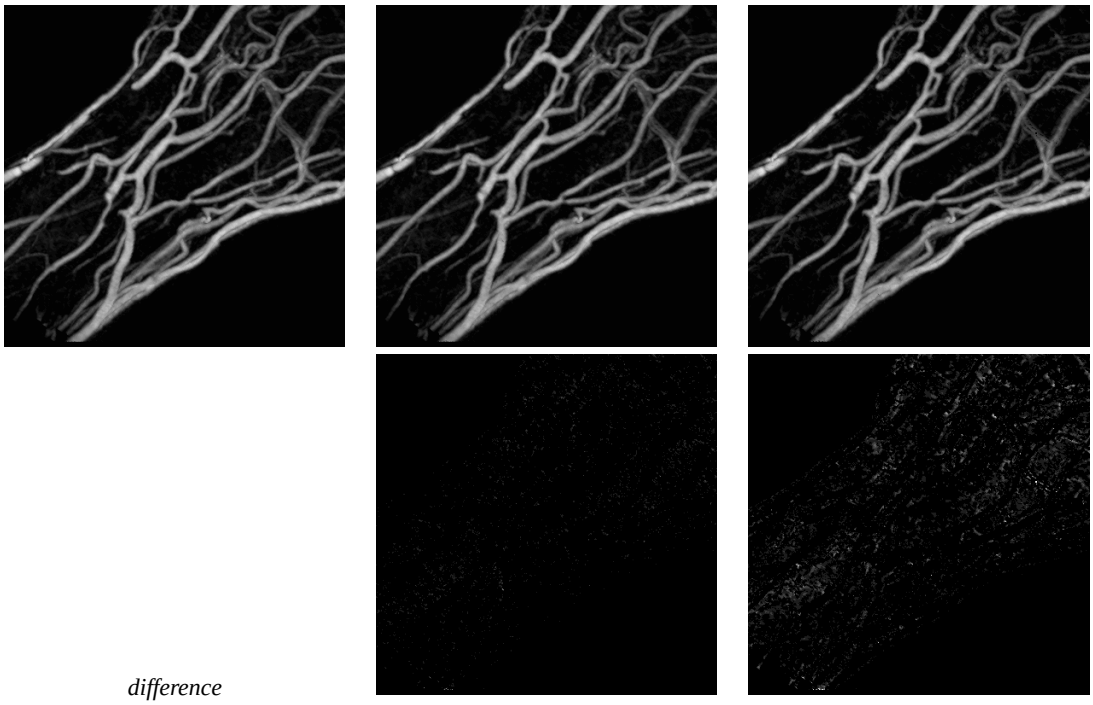
Table 1: *Comparison of ray-casting based MIP (trilinear interpolation, columns 4 and 5 with optimizations) with our approach (cell array and optimizations).*

data	*img. size*	*cell acc.*	*appr.*	*eval.*	*pixel set*
hand	465^2	2.5%	1518066	26%	1.65
hand	512^2	4.1%	2439476	27%	1.5
kidney	465^2	3.2%	1077898	29%	1.46

Table 4: *Efficiency of maximum value evaluation: Column "cell acc." gives the percentage of cells involved in the evaluation process. "Appr." is the total number of approximation operations, "eval" is the percentage of approximated ray intersections which required more exact evaluation. "Pixel set" is the number of writes to a non-background pixel.*

data set	*size*	*img size*	*no prepr.*	*prepr.*
hand	256^2x100	256^2	295ms	215ms
		512^2	615ms	450ms
kidney	256^2x69	256^2	180ms	150ms
		512^2	350ms	315ms
head	256^2x124	256^2	180ms	170ms
		512^2	380ms	365ms

Table 5: *Rendering times for different data sets and image sizes. all data sets have been preprocessed with a 1% tolerance threshold. The timings depend on the used window definition. Some of the resulting images can be seen in Figures 6d and 8a, b.*



difference tolerance 0% *tolerance 1%* *tolerance 2%*

Figure 7: *Results for sorted cell array MIP with different tolerance values for preprocessing (hand data set). As tolerance is increased, deviations appear mainly within areas containing low-valued noise.*

5. Results

The MIP algorithm presented in this paper has been implemented as a Java applet. The applet and further high-resolution results are available at (*http://www.cg.tuwien.ac.at/research/vis/vismed/CMIP/*). We expect a C implementation to exhibit a 30-40% better performance.

Table 1 depicts the speed-up achieved by our method compared to brute-force ray casting with trilinear interpolation (integer arithmetic) and to optimized ray-casting (no interpolation in cells with $C_{max} < max$, skip background noise windowed to black). The second row (sorted cells, evaluation only if $C_{max} \geq max$) clearly shows the advantage of projecting high-valued cells first (factor 3-7). Even more is gained by the ability to skip low-valued cells without any overhead (row 3). This results in a speed-up factor of 11-40. Finally, by eliminating cells during the preprocessing step, factors of up to 43 compared to brute force and 24 compared to optimized ray-casting can be achieved.

Rendering times as compared in Table 1 may depend on the optimization of the implementation. To obtain a less implementation dependent measure for the efficiency of the approach, we compare the number of interpolations performed, cell and image-access statistics of our algorithm to ray-casting in Table 2. Compared to even optimized ray-casting, the cell array requires five times less interpolations and changes ray-maxima and thus pixel values significantly less frequently. Due to the difference in the number of interpolations performed, we can conclude that a speed-up factor of approximately five is related to the avoidance of cell evaluations. The remaining portion of the speed-up is based on efficient volume traversal (strictly linear access to the cell array) and cell skipping.

Table 3 shows cell removal statistics for the preprocessing of the hand data set depicted in Figures 6 and 7 and the kidney data set of Figure 8a. As the second sweep eliminates just few additional cells, it can be optionally omitted to shorten the preprocessing. Increasing the tolerance for preprocessing mainly affects low-gradient areas which usually do not contain vessel data (Figure 7).

Table 4 presents statistical information on the number of cells involved in the image creation, the number of upper bound estimations for ray-cell intersections and the percentage of intersections which require a more accurate trilinear interpolation to evaluate the maximum within a cell. Rows 1 and 2 represent the hand data set for two different image sizes, Row 3 shows data for the kidney data set.

Finally, Table 5 presents rendering times for MIP using the cell array approach. The rendering times have been measured on a PIII/750 PC.

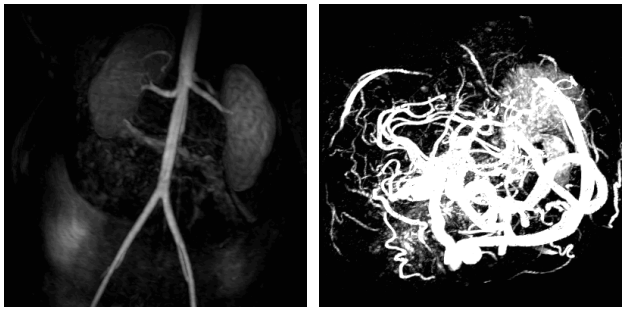


Figure 8: *a) kidney data set b) head data set*

6. Future Work

The performance of MIP methods based on sorted cell arrays can be still improved by further development in two areas: a more selective preprocessing technique, which is able to remove more cells, can reduce memory requirements and unnecessary processing of invisible cells. As the test results discussed in Section 4.3 and summarized in Table 4 show, more rigid upper bound approximation techniques could eliminate even more exact maximum evaluations. Further research should be done on these topics.

7. Conclusion

In this paper we have presented a new high-quality method for the generation of MIP images. During a preprocessing step cells which do not contribute to any MIP (regardless of the viewing direction) are identified and removed. To minimize dependencies and maximize the number of irrelevant cells, the preprocessing is performed for 12 distinct clusters of viewing directions. The positions of the remaining cells are sorted according to the cell maximum. For rendering, the cluster containing the current viewing direction is selected and projected. The cells are processed starting with cells with high values using precalculated templates to determine pixels affected by the projection. As pixels of the image are initially set to high values by the projection of high-valued cells, many unnecessary maximum evaluations which are performed by other MIP approaches can be avoided. To further decrease the number of costly maximum evaluations, a cheap and effective estimation of the upper bound for the maximum of a ray through a cell is performed first.

The method provides interactive frame-rates on high-end PCs and is well-suited for fast generation of animation loops especially to depict small details within the data which may be missed or deformed using other fast, but less accurate methods. Compared to ray-casting based high-quality MIP approaches our method avoids trilinear interpolations in a significantly more efficient way and achieves at least 20 times better performance, while providing better quality than other (shear-warp or hardware based) approaches.

8. Acknowledgements

The work presented in this publication has been funded by the FWF as part of project P-12811/INF (BandViz project) and by the $V^{is}M^{ed}$ project (`http://www.vismed.at`) which is supported by *Tiani Medgraph*, Vienna, `http://www.tiani.com`, and the FFF, Austria. The hand and kidney data sets are courtesy of Tiani Medgraph GesmbH, Vienna. The head data set is available from the United Medical and Dental Schools (UMDS) Image Processing Group in London `http://www-ipg.umds.ac.uk/archive/heads.html`

References

1. W. Cai and G. Sakas. Maximum intensity projection using splatting in sheared object space. In *Proceedings EUROGRAPHICS '98*, pages C113–C124, 1998.
2. B. Csebfalvi, A. König, and E. Gröller. Fast maximum intensity projection using binary shear-warp factorization. In *Proceedings of the 7-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99, WSCG'99*, pages 47–54, 1999.
3. W. Heidrich, M. McCool, and J. Stevens. Interactive maximum projection volume rendering. In *Proceedings Visualization '95*, pages 11–18, 1995.
4. J. T. Kajiya. Ray tracing volume densities. In *Proceedings of ACM SIGGRAPH'84*, pages 165–174, 1984.
5. P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorisation of the viewing transform. In *Proceedings of ACM SIGGRAPH '94*, pages 451–459, 1984.
6. W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM SIGGRAPH'87*, pages 163–189, 1987.
7. L. Mroz, A. König, and E. Gröller. Real time maximum intensity projection. In *Data Visualization '99, Proceedings of the Joint EUROGRAPHICS - IEEE TCCG Symposium on Visualization.*, pages 135–144, 1999.
8. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volume pro real-time ray-casting system. In *Proceedings of ACM SIGGRAPH'99*, pages 251–260, 1999.
9. G. Sakas, M. Grimm, and A. Savopoulos. Optimized maximum intensity projection. In *Proceedings of 5th EUROGRAPHICS Workshop on Rendering Techniques*, pages 55–63, Dublin, Ireland, 1995.
10. K. J. Zuiderveld, A. H. J. Koning, and M. A. Viergever. Techniques for speeding up high-quality perspective maximum intensity projection. *Pattern Recognition Letters*, 15:507–517, 1994.