Master's Thesis

Quantitative Measurement and Visualization Of Four-Dimensional Cardiovascular Blood Flow

Bernhard Kainz, bakk. techn.

Institut for Computer Graphics and Vision
Graz University of Technology

Supervision: Univ.-Prof. Dipl.-Ing. Dr. techn. Dieter Schmalstieg
Dipl.-Ing. Michael Kalkusch

Graz, July 2007

I would like to dedicate this thesis to my grandma who died during the creation of this work. She was always proud of me and supported me as good as she could.

I hereby certify that the work presented in this master thesis is my own and that work performed by others is appropriately cited.

All images used in this work are made by myself and therefore belong to me unless otherwise noted. Please ask me before reusing my images!

Ich versichere hiermit, diese Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Alle in dieser Arbeit gezeigten Grafiken sind, solange nicht anders angegeben, von mir erstellt worden, und daher mein Eigentum. Bitte fragen Sie mich, bevor Sie Bilder wiederverwenden!

Abstract

This thesis describes a workflow to obtain high level, hardware accelerated scientific visualizations from the measurement of four dimensional cardiac blood flow data through phase contrast magnetic resonance imaging (PC-MRI). Extending an existing medical data viewer, called iMEDgine, which is based on Coin3D, an open source scene graph library, enabled the implementation of point based and sparse velocity field visualizations with an additional presentation of the underlying anatomical image data. The workflow includes a preprocessing of the raw data with a Toolbox provided by Siemens Medical Solutions. To evaluate the results of the presented visualization algorithms several real human heart datasets and an additional artificial flow phantom setup were measured with different parameters. The results of these sequences showed that a high-level visualization may provide a deeper insight in the cardiac cycle for diagnostic and research purposes.

Keywords: phase contrast magnetic resonance imaging; PC-MRI; cardiac blood flow; artificial flow phantom; scientific visualization; flow visualization; scene-graph; Cash-flow; iMEDgine; GPGPU

Kurzfassung

In dieser Arbeit werden Abläufe definiert, die die Nachverarbeitung und Darstellung von vierdimensionalen Kernspinresonanz-Phasenkontrast Aufnahmen des menschlichen kardiovaskulären Blutflusses ermöglichen. Hierzu werden verschiedene Möglichkeiten der hardwarebeschleunigten Strömungsvisualisierung untersucht und als Erweiterung zu einem existierenden medizinischen Bildbetrachtungsprogrammes implementiert. Als Basis dient hierzu ein szenengraph-basierter Ansatz der vollständig auf Grafikhardware ausgeführte Visualisierungsalgorithmen in dessen Knoten kapselt. Um die Ergebnisse von punkt- und trajektorienbasierten Algorithmen zu evaluieren wurden mehrere Datensätze von gesunden Probanden mit verschiedenen Parametern aufgenommen und ein Flussphantom gemessen, das eine künstliche Stenose eines Blutgefäßes simuliert. Die Ergebnisse zeigten, dass eine interaktiv nutzbare, frei kombinierbare und effiziente Visualisierung mit verschiedenen Methoden zu neuen diagnostischen Verfahren und einem besseren Verständnis des menschlichen kardiovaskulären Systems führen kann.

Schlagwörter: Phasenkontrast Magnetresonanzverfahren; PC-MRI; Strömungsvisualisierung; kardiovaskulärer Blutfluss; Szenengraphen; iMEDgine; GPGPU; Flussphantom

Contents

Chapter 1

Introduction

Assessment of blood flow properties is crucial in the understanding and diagnosis of many cardiovascular diseases. The Magnetic Resonance (MR) through-plane phase contrast method provides a lot of useful information from flow through cross sections or velocities in preferred directions. However, its usefulness in situations involving complex fluid dynamics - as for example in the cardiac chambers - is limited, because the main directions of flow are neither known nor constant in time. Conceptually the easiest way to acquire three-dimensional blood flow data is to measure both through-plane and in-plane velocity components via phase contrast sequences. Velocity vectors are determined on each imaging plane: In the case of the combined through-plane and in-plane measurement for each pixel.

This measurement method is already available [Reiter2006; ClincalMRI2006], but the vast amount of data and its complex composition requires the development of tools and methods for a clear representation aimed at physicians who may evaluate possible vascular diseases or understand the human cardiovascular system in a better way. High-performance real time GPU accelerated visualization techniques help to investigate the resulting four-dimensional datasets in this work. These methods are also applicable to other kinds of flow data.

The problem we are dealing with is that radiologists and physicians can form mental images of a certain structure from human anatomical images slices, but flow information given in additional images is not perceptible in a similar way. The main problem with flow sensitive sequences is that three dimensional flow data requires three additional images for each measured slice image and that they have to be investigated in addition to basic morphological data. For physicians this is not feasible. Flow is also changing

over time, so the resulting datasets are four-dimensional, which implies that high-performance interactive visualizations are required for a presentation of this data in combination with a conventional presentation of medical data.

The desired solution is a workflow which integrates these measurement sequences with a preprocessing software into an interactive visualization framework. Therefore, we extended a medical image viewer with scene graph based and hardware accelerated point based visualization algorithms to combine the measured velocity fields with the morphological background.

Chapter 2 gives an overview of possible visualization techniques for medical data and flow data. This approach introduces *scientific visualization* used for *n-dimensional flow data*. Due to the high computational costs for rendering flow and anatomical datasets, hardware acceleration techniques are required. The corresponding graphics hardware architecture is also presented in chapter 2.

It is crucial to present interesting parts of blood flow in an instructive way, so that a possible diagnostic use is given. The special MRI sequence which was developed to continuously measure the movement of blood in human vessels is described in chapter 3 after a short introduction into the basics of magnetic resonance imaging.

To apply these algorithms chapter 4 defines the workflow with all preprocessing steps for the raw data which are required until an advanced data visualization can be performed. This workflow is one of the main attainments of this work and is outlined in figure 1.1.

This work concentrates on *point-based* and *sparse flow representations* as defined by [Weiskopf2007]. We implemented different types of direct particle based visualizations and particle trajectory based line visualizations. Additionally we provide several cutting plane visualizations and improvements of basic line approaches. All these visualization algorithms can be combined arbitrarily and are specified in detail in chapter 5.

To reduce the development work for GUI and further overhead, a medical image viewer called iMEDgine was extended, which is based on scene graphs as provided by the Coin3D [SystemsInMotion2007] library. On the one hand data flow abilities were added as provided by a scene graph extension library called Cash-flow [Kalkusch2005] and on the other hand high-performance and parallel executed shader algorithms for complex flow visualizations were developed. Chapter 5 describes the details for these combinations and outlines their dependencies.

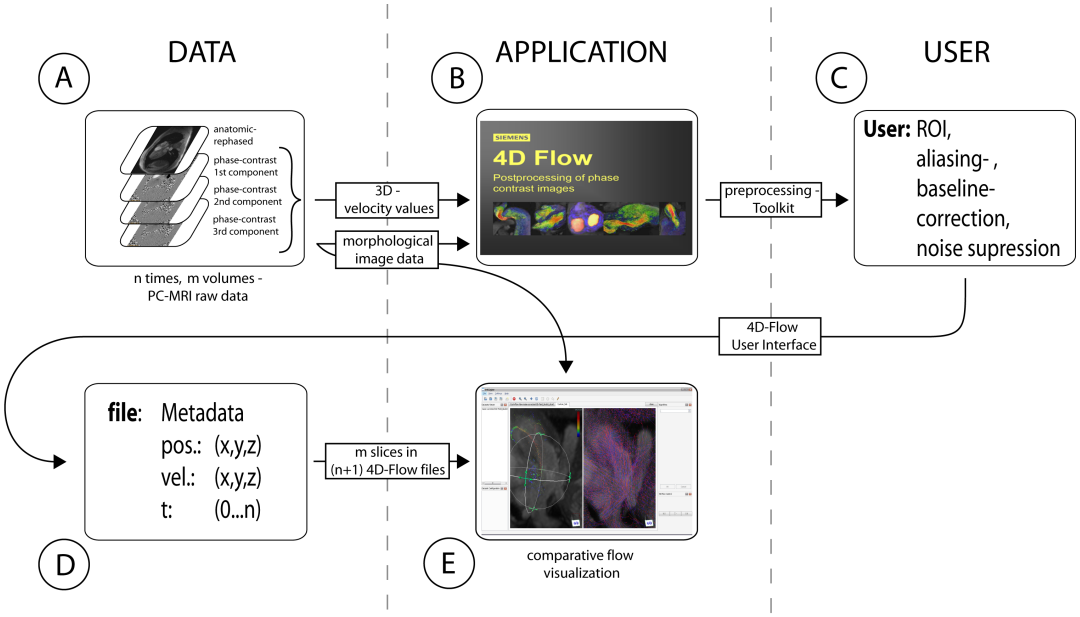


Figure 1.1: This workflow diagram describes the traversal of the measured raw data. Data acquisition is marked with (A) and described in chapter 3. Following the first arrow rightwards leads to the data preprocessing (B) which is described in chapter 4. From the preprocessing-Toolkit module on, the user has to perform some data improvement (C). The processed data is stored in files (D). These files are used in combination with the anatomic raw images from (A) by our flow visualization framework (E). This framework is further discussed in chapter 4 and chapter 5. (A) and (D) belong to the data layer, (B) and (D) to the application layer and (C) needs user interaction. The arrows additionally indicate the course of the data flow.

For experiments we have accomplished several collateral PC-MRI measurements at the radiology department of the Landesklinikum Graz and the Diagnostikzentrum Graz. Besides the acquisition of datasets from real human hearts, a flow phantom was built to measure flow through an artificial narrowing. The results of this setup compared to real human blood flow are presented in chapter 6. Performance analyses of all implemented visualization techniques are compared on two reference PC-systems in section 6.4.

Finally a forecast for future possibilities is made in chapter 7. Among others, further developments may be seen in the fast and accurate calculation of derived flow quantities and better control mechanisms for an arbitrary interactive visualization. Quantities derived from velocity values could be partial pressure differences or correlated tissue

stress visualizations. These applications may provide crucial indicators for a possible course of a vessel surgery. Currently, these quantities are determinate by perilous catheter based measurements. They may get obsolete through further investigations of 4D PC-MRI cardiac flow measurements and according visualization techniques.

Chapter 2

Visualization

Visualization is a broad term referring to various fields and disciplines. It is always related to a human perception, so the interpretation of visualization can differ from person to person.

In the focus of this work, visualization generally means the representation of complex data with visual means. In this context the goal of visualization is to give different observers the same perception of the same coherences of the same data. This chapter introduces different kinds of presentation techniques and accordingly splits them up to the underlying data and the information which has to be transported.

Such high-level definitions can be subdivided into numerous sections, which can be organized in a tree structure. This chapter will follow one path of this tree up to possibilities suited for complex datasets as described in chapter 3. The junctions through this tree will define technical visualization and its divisions into scientific visualization and information visualization which are separated in section 2.1. Passing some insights in ongoing graphics hardware developments will yield a high level description of the programing abilities of such a graphical processing unit (GPU) with a special view on vector processing and flow visualization opportunities.

The focus of this thesis is the processing of image data with additional cardiac flow patterns, recorded from a MR imaging device. Consequently, cardiac flow data visualization belongs to the field of scientific visualization which will be illuminated in section 2.1.1.

2.1 Technical Visualization

Visualization can be defined as a technique for exploration and presentation of data as well. According to [Wijk2005] the creation of images, diagrams or animations to communicate a certain message, is just as important as the exploration of new and unknown datasets.

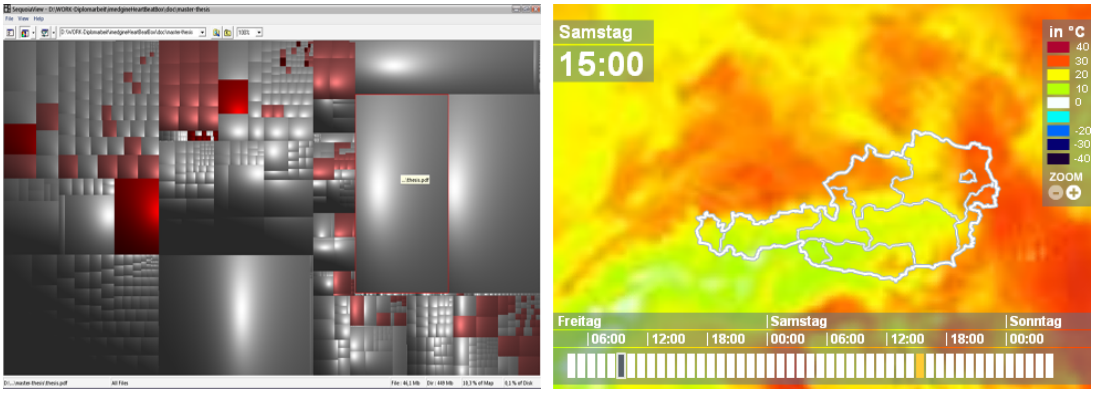
Two main fields of technical visualizations can be divided in the following: *information visualization* and *scientific visualization*. The difference between them is the kind of data they operate on. Information visualization attempts to efficiently map data variables onto visual dimensions in order to create graphic representations [Gee2005]. Abstract data is data that has no inherent mapping to space and can cover nearly everything from databases to statistics or biochemical processes. The main objective is in most cases the arrangement of abstract data intuitively for the user sight.

Scientific visualization should be treated in opposition to information visualization since it deals with physically-based data mainly with spatial character. This means all but everything which can be measured with a sensor or sensor-related devices. This makes it relatively easy to visualize physical data in an intuitive way by mapping the space coordinates in the dataset to screen coordinates as discussed by [Voigt2002]. Figure 2.1 illustrates the differences between information- and scientific visualization by two examples. They show on the left hand side the files used for this thesis but ordered by their size, using the tool from [Eindhoven2002], which refers to information visualization. The right hand side example shows a weather heat-map for the temperature distribution of a day in June which refers to scientific visualization due to the spatial correlation of its base data.

In the next section the field of scientific visualization will be investigated more closely. The aspects of presentation of potentially huge quantities of data and representation approaches to aid the exploration, presentation and hypotheses generation from them will be a matter of particular interest.

2.1.1 Scientific Visualization

The main parts of this area of research are **volume visualization** and **flow visualization** [Hauser2002]. To provide an overview over this topic some major visualization approaches are discussed in the following.



(a) (b)

Figure 2.1: Figure (a) shows the *"information visualization"* approach for the organization of the files used to generate this thesis. They are ordered here by their size with the tool SequoiaView from [Eindhoven2002]. Figure (b) visualizes the (measured or predicted) temperature differences by means of a heat map and can be associated with scientific visualization. This images were taken from *http://www.wetter.at/wetter/oesterreich/*. These examples illustrate how different these two areas of visualizations are with respect to the underlying data and the resulting presentation.

The context in which visualization is used should be taken into account first. In figure 2.2[1] we show visualizations organized depending on the dimensionality of the underlying data from the inside outward. Besides several other fields of usage we could identify the area where this work is located in. This place is marked with a red cross at the border between three-dimensional and n-dimensional data.

Another good choice for an ordering concept of scientific visualization was presented by [Tory2004], who tried to squeeze this field of research into a taxonomy shown as in table 2.3. This taxonomy is based on the data structures by which the underlying data can be described and on the number of independent variables. These variables can for example refer to the dimensionality of the space in which the data is embedded. The data structures are *scalar, vector, tensor* or *multivariate* ones. In this ordering concept we can identify the area of interest for this thesis more precise since the basic data structure for the measured flow data are multidimensional, respectively three-dimensional vectors. The number of independent variables is four.

[1] sub-images taken from *http://www.research.ibm.com/dx/imageGallery/*, *http://www.wetter.at/wetter/oesterreich/* *http://www.btci.org/bioimaging/2005/preview05.html*, *http://openqvis.sourceforge.net/gallery.html* and *http://aspadvice.com/forums/thread/28774.aspx*.

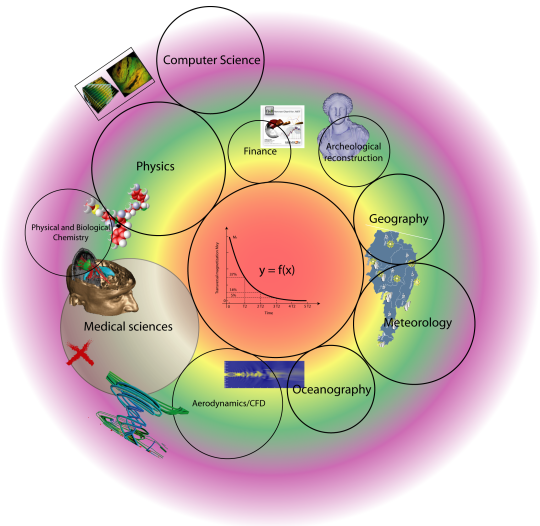


Figure 2.2: We could identify some fields of technical visualization sorted from 1D to nD data and from red to purple. We have centered the largest and oldest part of visualization which is suitable for all sciences but mostly restricted to 1D. Then we added other fields of research outwards, depending on the data-dimensionality these sciences are used to. The content of this thesis is marked with a red cross which is located approximately at the lower left outer green-to-purple border of "Medical sciences".

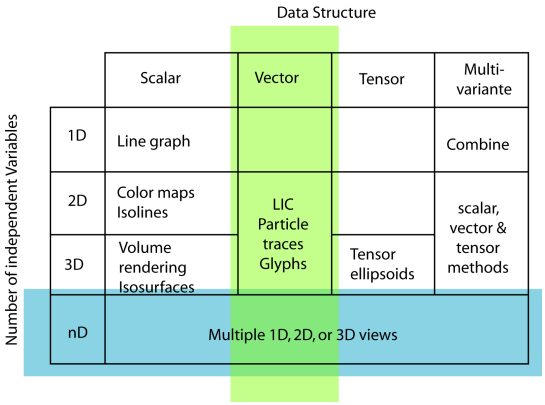


Figure 2.3: This table provides and ordering concept for scientific visualization by means of their underlying kind of data structure and the number of independent variables. The actions considered in this thesis are marked with the green and the turquoise bar since the basic data structure for the measured flow data are multidimensional, respectively three-dimensional vectors and the number of independent variables is four. This table was taken from [Tory2004]

Next a common concept known as *visualization pipeline* is introduced. Referring to illustration 2.4, the data has to be acquired from an arbitrary device on the top of this pipeline. This data and several other physical aspects of the modality will influence necessary preprocessing steps depending on the desired and provided information of the measurement task. The acquired solutions must deal with such processing steps since the raw data is not implicitly suitable. The detailed description of the preprocessing is given in chapter 4. Further down this pipeline a rendering process brings images to a screen. Certainly, no (medical) measurement method is capable of telling the graphics hardware directly where to render polygons or pixels, so the data parts which are used for displaying desired information are in most cases completely different from the raw ones. Figure 2.4 provides an overview of these described steps. Visualizing during or short after data acquisition will enable to optimize a measurement based on the given visualization results. This is even one of the great goals for workflows like the one presented here. Consequently this would then provide a user-steerable data acquisition and visualization process.

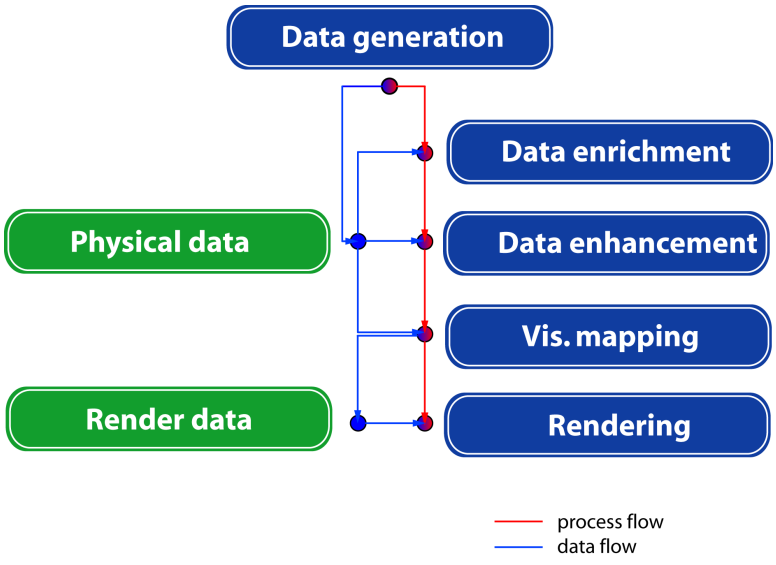


Figure 2.4: The traditional visualization pipeline containing steps as presented in a lot of publications like [Upson1989; Kalkusch2005]. Depending on the data properties and its recoring modalities, several preprocessing steps will be performed in the most cases before a visualization rendering can take place.

Applying these general considerations will require a deeper insight into the spe-

cial needs of medical image data and physiological flow data. The upcoming sections identify dedicated visualization techniques for both.

2.1.1.1 Representation Techniques for Volumetric Morphological Data

To be able to perform medical volumetric data visualization one first has to consider the underlying two dimensional images which build-up an image stack for further volumetric approaches together. The following considerations refer predominantly to morphological data which means images presenting an accurate representation for the shape of different tissues.

Displaying even two dimensional medical image data is in general not as trivial as decoding and presenting an image in a common format. Unlike a picture taken with an ordinary picturing device providing red, green and blue components $R[0...255]$, $G[0...255]$, $B[0...255]$, structural projection or scanning techniques deliver only gray values ($R = G = B$) but with a greater range. In a commonly known medical imaging format called **D**igital **I**maging and **Co**mmunications in **M**edicine (DICOM), [DICOM2007], the intensity values are encoded with 12 bits, which allows to encode more information in one pixel. The problem hereby is the limitation of all display devices to 256 values and the constraints on the human visual system. [Barlow1956; Lu1999; Barten1992] have identified these constraints by estimating the required signal stimulus energy required for an observer to maintain a perception and found that the perception is dependent on a non-linear function. Due to these limitations a windowing mechanism has been developed among others by [NEMA2003] to provide a mapping from the measured 12-bit data to a 256 gray value gradient. Such a mapping is called *windowing* and briefly outlined in figure 2.5.

This constraint of pixel value mapping has to be kept in mind when working on medical data, especially when using them with some special volumetric rendering techniques for image stacks as they are appearing next. Later used techniques, which are described in more detail, use transfer functions which implicate the windowing in their parameters. The remaining rendering techniques are itemized in the end of this section along with the definition of volume rendering transfer functions.

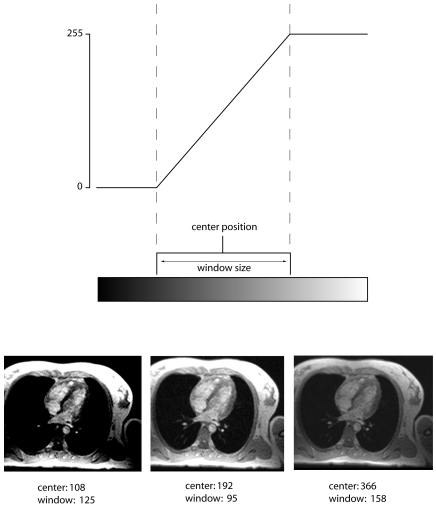


Figure 2.5: The mapping of 12 bit medical data can be performed by choosing a transfer window centered at a certain position. The range of values covered by this window can then be linear converted to 256 gray values.

2D texture planes define probably the simplest but also the most common technique to represent medical data. Even today a radiologist will not use sophisticated volume rendering techniques for diagnostic purposes. He will prefer to scroll through the image stack and has consequently to build up a 3D-imagination of the structures mentally. For a wide range of medical applications, like these analyzed in the presented workflow, another image based approach is used which is more related to a 3D visualization. The measured images of a slice are rendered on top of each other with different transparency values and a slidable main focused/brightest image. The distance of the rendered image planes are in this case defined by the selected slice distance during the measurement itself. For spatial good resolved volumes, arbitrary additional planes can be interpolated perpendicular or in any direction to the actual measured slices. This rendering technique is as powerful as simple because it does not produce any interpolation related artifacts and only a few but easily comprehensible occlusions, in contrast to research done on multi-planar-reconstruction [Shekhar2003; Rothman2004] and sampling techniques for arbitrary cutting planes. Due to figure 2.6 we have found, that physicians favor real measured slices over interpolated ones. That means that we had to keep in mind this approach for further visualizations of morphological data, even if it is not a classical volume rendering technique.

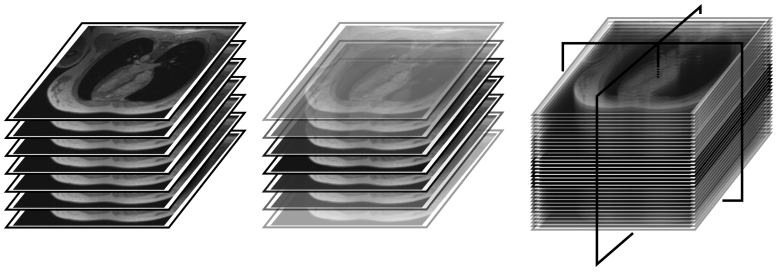


Figure 2.6: The left image illustrates a spare measured volume which can be rendered with different transparency values on top of each other (in the middle). If the dataset is dense enough, which means that there are enough slices available in one direction, additional arbitrary reconstructed slices can be displayed as suggested on the right. We have found, that physicians favor real measured slices over interpolated ones so the approach in the middle would be the mean of the choice for an accurate presentation to them with this technique.

Transfer Functions The concept of transfer functions as described by [Pfister2001; Engel2006] defines the way of mapping arbitrary data from a volumetric element (voxel) to a RGB-color-space. For the following, concerning direct volume rendering techniques, a transfer function is the crucial part for the final result and representation of the underlying data.

In general, transfer functions define the opacity of a certain data point and further a color which can help to distinguish features in the volume. This implies that such a function can implicitly enable the above described medical data windowing and an additional segmentation in one step. The actual difficulty is to generate a function which fits exactly to the desired structures and features and suppresses the unwanted details. Several simple one-dimensional functions are distinguished by [Hauser2003] in the following way:

- taking the maximum occurring data value is called *Maximum Intensity Projection* (*MIP*),
- accumulation of the occurring values on a ray is called *Compositing* which results a semi-transparency of the volume,

- averaging of all volume elements on a ray produces an image similar to an *X-Ray*-image,
- taking the first occurring value on a ray is called *first hit* and renders the first pixels above a certain threshold. In the optimal case this technique would result in the same as extracting a surface from the volume but the estimation of the threshold can be time-consuming as well.

An one-dimensional transfer function may be applicable only to adjust the opacity of the volume. Multidimensional functions can also be used to color certain parts of the volume, despite the increasing complexity the design of the function may get accomplished by trial and error. [He1996] identified a parameter optimization problem and proposed user driven algorithms to optimize transfer functions. [Kniss2001] denoted furthermore that the complexity of defining a transfer function is based in the enormous number of degrees of freedom in which the user may get lost.

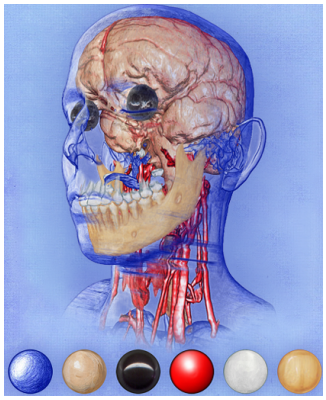
The automatic adjustment of adequate parameters is still a topic of research. The currently best ways to define a multidimensional transfer function for arbitrary datasets are mostly interactively driven as proposed by [Kniss2001], who defined manipulation widgets for 3D transfer functions. They defined the axis of the 3D function space with the data value, the gradient magnitude and the second derivative. To underline their results they demonstrated their work for multivariate data in a case study [Kniss2002a].

Recently new and more powerful kinds of transfer function designs are developed. [Bruckner2007] presented a technique for illustrative volume visualization to enhance important features without rendering uninteresting ones. They introduced the concept of style transfer functions by using the data values and eye-space normals, so thickness controlled contours are possible by approximating the normal curvature along the viewing direction.

In later chapters we concentrate on on one-dimensional transfer functions since the focus of this work lies in the visualization of flow patterns, where direct volume rendering approaches with opacity mapping serve for a spatial localization of these patterns in the volume. In our opinion additionally mis-colored rendered morphological data would lead to a confusion with the painted flow visualizations. Nevertheless, these techniques will have to be kept in mind for a meaningful combination with flow visualizations in future work.

(a) One-dimensional transfer function for mapping of data values to opacity from section 4.4.1.1.

(b) Manipulating a three-dimensional transfer function with control widgets as presented by [Kniss2001]



(c) Segmented Volume data rendered with style-transfer functions as introduced by [Bruckner2007]

Figure 2.7: Comparison of different direct volume rendering approaches using transfer functions.

3D texture slicing Another technique is the well known rendering of 2D textures intersecting a 3D volume texture. The main difference is that independent of the number of measured slices, a constant number of planes are cut through a 3D texture which contains the actual volumetric data. These slices are always directed towards the viewer and perpendicular to the viewing direction. Rotation around this volume will then only affect the mapping of these slices - with the constraints of aliasing artifacts - whereas the 3D volume texture is placed fix. Consequently, different viewing directions will result in different data mapped onto the 2D texture planes. Hence, a constant transparency value of a slice or a special function related to them will then provide the impression of a semi-transparent volume as shown with *transfer function* in paragraph 2.1.1.1. For example a constant transparency value

$$\alpha = \frac{1}{num_{slices}}$$

for each slice - will result in the same as the projection of the averages from the occurring values found on a ray starting from the eye point as described in the next paragraph. Other transparency functions will allow arbitrary combinations of the evenly

spaced sampling points of the volumetric texture. Figure 2.8 from [Engel2006, page 49] illustrates these principles.

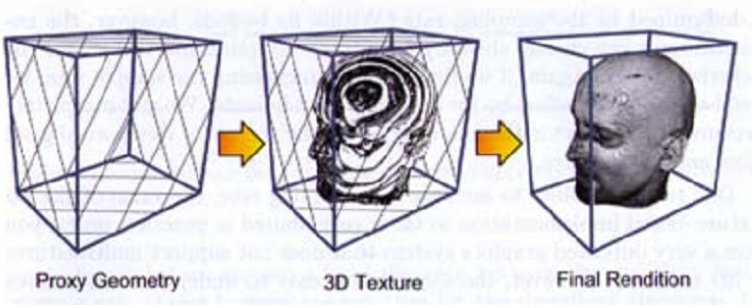


Figure 2.8: Generating semi-transparent slices always perpendicular to the viewing direction leads to a evenly spaced sampling of a fixed 3D-texture volume. This will provide the impression of a semi-transparent volume depending on the transparency distribution or more complex mapping functions. This illustration is taken from [Engel2006, page 49]

Ray casting is based on the idea of sampling a the three dimensional scene along perspective arranged lines in viewing direction. This concept was first introduced by [Roth1982] and results in a two dimensional projection of the observed volume parts. With this algorithm a more fruitful usage of transfer functions as described in paragraph 2.1.1.1 is possible but sampling the volume is time consuming and the problem of possible aliasing artifacts is evident. Nelson L. Max has identified in his work [Max1995] the vital coherences of absorption, emission and the combination of them with additional scattering considerations for direct volume rendering. Ray casting is in this section an absorption only process.

Many work was done on this topic, so only the main idea can be presented in figure 2.9. Early ray termination [Levoy1988], octree decomposition [Levoy1990; Levoy1990a], or color cube techniques, adaptive sampling and empty space skipping [Scharsach2005; Rottger2003] would be vital for the performance and an interactively usable application.

The usage of even multidimensional transfer functions as presented by [Kniss2002] adapted to the desired highlighted tissues will already give a very good impression of the actual appearance of the scanned area. As an example given for a simple transfer function, the popular maximum intensity projection (MIP) approach which produces

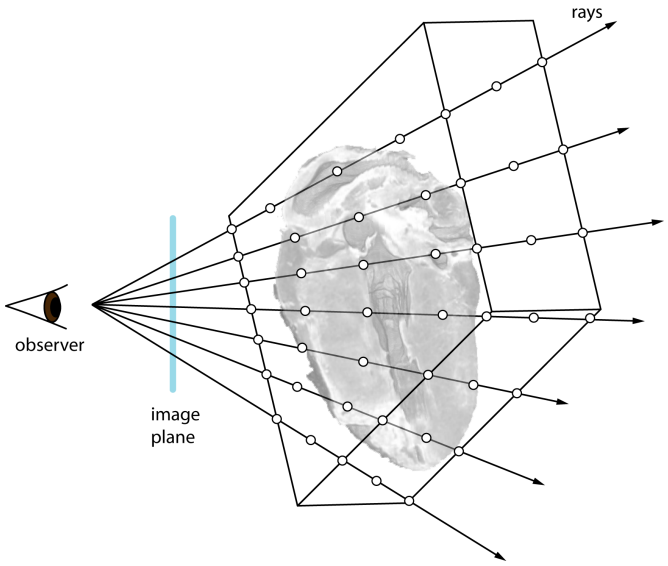


Figure 2.9: Perspectively arranged rays sampled through a volumetric dataset and evaluated by a transfer function are commonly known as *Ray casting*. Several smart sampling techniques and ray termination assumptions have to be made to guarantee interactive handling abilities. The ray-casted image of the heart is taken from [Kerwin2007]; a related illustration can be found in [Engel2006].

images in the style of X-Ray angiography images can be denoted here referring to [Heidrich1995]. This function simply projects the value with the highest intensity value found on the given ray. Actually this algorithm is commonly used with angiography methods since for these applications the vessels and the contained blood will produce the highest signal. Projecting this maximum intensities will result in a almost clear sight on the arteries or veins, depending on the time between the injection and the scan. Additionally a two pass rendering was presented by [Hauser2001] which makes it possible to combine a MIP with another volume rendering transfer function to show additional interesting features. Nevertheless, figure 2.10 taken from [ClincalMRI2006] points out, that the pure MIP rendering method can proof its value only with an either interactive or permanent rotation of the volume.

The remaining direct volume rendering approaches were not described in this section due to the following reasons:

- *2D texture slicing*, even performed with graphics hardware implementations as

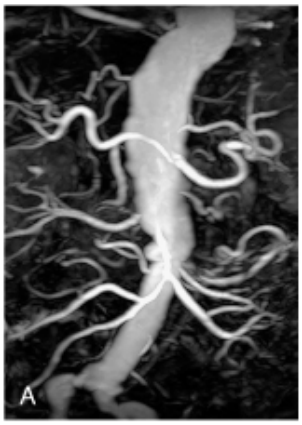


Figure 2.10: Frontal MIP of an gadolinium contrast-enhanced abdominal MRA scan taken as an example for a basic but effective transfer function. Image taken from [ClincalMRI2006].

presented in [Engel2006], is based on axis parallel 2D texture planes but was neglected in this section due to its complex texture switching mechanism.

- *Splatting* as technique for high detailed visualizations and refinements was introduced by [Laur1991] as an idea to send the information from to object to the screen. It is not used in this work since we had not the necessary capability of parallel graphics processing for an interactive usage of this technique.

- *Shear Warp* as described in [Lacroute1994; Engel2002] simplifies the projection of voxels by first shearing, then projecting and finally warping the viewed points to a 2D plane. This algorithm is mostly performed on special hardware cards which we could not obtain.

- The *Fourier reconstruction method* stores the volume in a 3D-Fourier space and performed the projections by a 2D inverse Fourier transform of a cutting plane through the Fourier volume. This technique was presented by [Dunne1990] and [Malzbender1993; Moeller1996] but is restricted to gray values and was therefore not used.

Extraction of surface meshes Beside several global representations of a given volume, certain patterns of a dataset can be extracted and presented for example as mesh

relatively easy.

The term "iso", from the greek word $\iota\sigma o\zeta$ which means "similar", is used in this context to underline what the visualiztion is actually doing: It tries to produce a triangle mesh by computing an iso-surface with a certain iso-value representing the same property of the volume. A very popular method to extract a certain property in a given volume is the so called marching cubes algorithm [Lorensen1987].

The underlying voxel grid is divided into small cubes where always four corners are built up of four pixels in adjacent slices. The goal is to determine how this cube is intersected by the desired object's surface and thus the adjacent slices are intersected. Choosing an iso-value will result rarely in one of the cubes corners directly; in most cases the positions of the desired value will not be found without an interpolation along the intersected edges of the cube. In the majority of cases a surface will intersect three or more edges of these cubes consisting of vertices with values greater and smaller than the iso-value. A connection of these intersections will lead to one or at most three triangles of the desired iso-surface and subsequent processing of all cubes in the volumetric dataset ("marching") consequently unfolds the whole surface. Then the algorithm tries to find the edges which are intersected. This reduces the intersection problem for eight vertices to $2^8 = 256$ cases which can be further generalized by rotations and symmetries to 15 cases. Finding these intersected edges is rather easy. Every cube's corner with a value greater or equal the iso-value will be assigned to one and the others to zero. In the first case the vertex is behind or on the surface and in the latter it is in front of it.

The creation of a lookup table for these cases enables an efficient build-up; the table can be accessed via an adopted notation referring to each case by an index created from a binary eight bit interpretation of the corner's state. The exact surface intersection on the edges can then be linear interpolated with the real image values where the vertices are placed.

Finally the vertex normals for correct shading have to be investigated. These components can be estimated along the tree coordinate axis by the gradient vectors at the cube vertices and linearly interpolated depending on the light model.

Two ambiguous cases and certain incompatible triangulated cube-types in the above described approach have been resolved by introducing new types of intersecting families [Nielson1991; Montani1994; Chernyaev1995, and others]. Based on this idea many

improvements have been developed. For Example using tetrahedrons instead of cubes overcomes topological inconsistencies [Treece1999].

Obviously especially in this case the quality of the constructed surface depends on the number of measured slices of the considered volume. Using one of the former described direct volume rendering approaches would be more advisable for a dataset containing only a few image planes.

2.1.1.2 Flow Visualization

The data acquired for this work can be defined in the following way:

- time dependent, which means that the discreet field values can change over time,
- organized on a regular grid which implies that the field values are evenly spaced inside a three-dimensional coordinate system,
- discretized by three dimensional velocity vectors for each position in space.
- Additionally the morphological information is provided by separate images in the dataset which can be treated with the techniques described in section 2.1.1.1.

The complete definition of the available datasets is done in section 3 and will not be further discussed here. However, for the following considerations of flow sensitive visualization algorithms, noise-less and consistent flow vector fields are assumed.

Color mapping According to the definitions given below a first information transporting issue has to be discussed: the use of color. Color may be mapped in an arbitrary way to an arbitrary visualization technique. It will always be necessary to define a certain kind of color-gradient as shown in figure 2.11 to allow a mapping between values of the desired color for the minimum parameter and the maximum parameter. Color has the benefit that it can reduce visual clutter and that the human visual system is rather designed to identify homogeneous colored regions quickly [Barten1992] and a mental clustering is possible. The most popular parameter to map is for flow fields the absolute magnitude of the underlying velocity. Nevertheless other parameters may be used, like the absolute value of variation, vorticity/rotor, attributes of tensors and many more or the combination of them by giving the observer additional meta-knowledge.

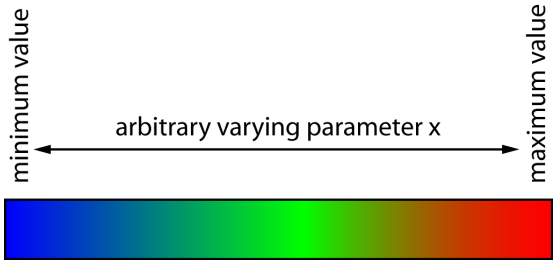


Figure 2.11: An exemplary color gradient for arbitrary parameter value mapping. A gradient like this can even be used to suppress certain flow properties if for example an appropriate transparency-value definition can be performed.

The main visualization techniques for any kind of flow data can be divided into four main classes. A related taxonomy can be found in [Weiskopf2007] which we summarized in figure 2.12. Furthermore some of these techniques are only valid or behave differently for steady or unsteady flow and can thus be subdivided again. Steady flow refers in this context to unchangeable vector fields and unsteady flow to changeable fields over time.

- **point-based direct flow visualization**

  implies that some fixed geometry is rendered for each velocity vector. A large viewport showing enough of these representatives enables the observer to interpret the whole vector field. The problem of visual clutter and occlusion is evident for volumetric datasets.

- **sparse particle tracing techniques**

  relies on calculations done based on the movement of massless particles injected into the field and envelop everything from concrete particle effects to complex trajectory integrations.

- **dense particle tracing techniques**

  mostly rely on a specialized convolution of a texture with an even volumetric flow field. Again, strategies to cope with visual clutter and occlusions have to be investigated.

- **feature based visualization approaches**

try to extract desired patterns like vortices from the raw vector field to provide an abstract representation of the contained important information. These are the most complex methods since almost a kind of artificial intelligence has to be developed to find out the important parts of the data. Similar algorithms can be found in the field of computer vision.

Figure 2.12: A visualization taxonomy and therefore some possibilities for datasets as described in chapter 3. The combination of morphological and physiological MRI data requires both; a tissue and a flow visualization technique. Ray casted and iso-value triangulated images of the heart on the left from [Kerwin2007]; Images of spot noise and 3D-texture advected Tornado dataset from [Weiskopf2007]; stream balls-image from [Brill1994]; critical points from [Frits1993]; field topologies on the left from [Tricoche2004]. The rest of the shown images are based on results from this work and prior achieved ones from [Reiter2006].

Visualizing glyphs, arrows and color maps are based on rendering of simple geometrical structures at each point of the grid. Therefore these techniques do not need further explanations. The next step will be to make a definition of particle movements in steady or unsteady flows. Related trajectories can be calculated by different approaches. This thesis concentrates on discrete particle injection methods and finite differences approximations. Texture based advection techniques which are based on Lagrangian approaches were not implemented.

$$\frac{\partial x}{\partial t} = v(x(t), \tau), \quad x(t_0) = x_0, \quad x : \mathbb{R} \rightarrow \mathbb{R}^n \tag{2.1}$$

denotes the continuous movement of a massless particle under the influence of a even varying vector field, [Kipfer2004; Krueger2005; Weiskopf2007]. v describes a sampled vector field whose sampled values depend on the current position of an particle $x(t)$. In the case of a time varying vector field, which equals unsteady flow, τ serves as parameterization of the time. Together with the initial condition $x(t_0)$ this differential equation is complete. Keeping in mind that the measurement method from chapter 3 produces a vector field defined on a regular spaced lattice and that general programming approaches cannot solve differential equations in complete form, numerical integration methods have to be used.

The simplest form to solve the initial value problem 2.1 numerically is the standard explicit Euler-approach [Euler1768]. Choosing a discretization step size $\Delta t = h > 0$ for

$$t_{k+1} = t_k + h, \tag{2.2}$$

leads to an approximation for the exact solution of

$$x_{k+1} = x_k + hv(x_k, t_k, \tau). \tag{2.3}$$

The accuracy depends on the selected step size Δt which is indirect proportional to the computational costs for the same length of a certain trajectory.

To reduce the integration error or the computation effort of the former described numerical solution, several correction mechanisms can be added. One example is the well known Runge-Kutta method [Runge1895; Kutta1901] of a certain order [Albrecht1996;

Arnold1998]. The order defines the number of pre-integration steps before they are averaged added to the previous step x_k. Equation 2.3 can be amplified to

$$x_{k+1} = x_k + h \sum_{j=1}^{n} b_j c_j, \tag{2.4}$$

with procedure depending coefficients b_j and the intermediate steps c_j. Each c_j is calculated per step with a basic Euler integration step. Defining a concrete n, for example $n = 4$, will lead to the popular Runge-Kutta method of fourth oder or denoted by *RK4*. An approximated integration with four intermediate steps will then be defined by

$$\begin{aligned} x_{k+1} &= x_k + \frac{h}{6}(c_1 + 2c_2 + 2c_3 + c_4), \ where \\ c_1 &= v(x_k, t_k, \tau), \\ c_2 &= v(x_k + \frac{h}{2}c_1, t_k + \frac{h}{2}, \tau), \\ c_3 &= v(x_k + \frac{h}{2}c_2, t_k + \frac{h}{2}, \tau) \ and \\ c_4 &= v(x_k + hc_3, t_k + h, \tau). \end{aligned} \tag{2.5}$$

Compared to the former described explicit Euler method the benefits of this technique are shown in figure 2.13. Consequently, it can be denoted, that the Runge-Kutta method will produce more accurate results with less sampled supporting points of the desired trajectory. This technique will be favored later on.

Storing the supporting points as a result of the calculation of a certain amount of particles will produce a $n \times m$-array with all sampled trajectory points depending on the step sizes Δt and the time parametrization τ.

This array can be utilized for the further presented visualization techniques except of a concrete particle system effect itself. Visualizing particles in a flow field directly will only require the calculation of the *next* position of the particle, so previous trajectory points need not to be stored. Line based visualizations, or similar approaches must not discard these "older" positions.

Based on simple particle calculations a vast number of trajectory based flow visualization techniques have been investigated. Referring to [Weiskopf2007] the following

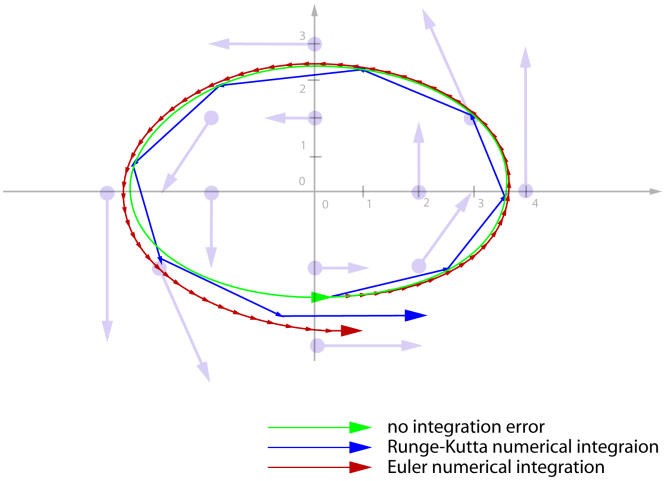


Figure 2.13: The benefits of the Runge-Kutta method compared to the explicit Euler method are summarized in this image. Obviously, the Runge-Kutta method will produce more accurate results with less sampled supporting points of the desired trajectory. Vice versa a numerical integration with the Euler method will always need much more steps to obtain a related result. This image is similar to one presented in [GrabnerKalkusch2006] which is originally based on [Hauser2005-2006].

enumeration briefly describes methods that generate *sparse representations* of a velocity field.

- **Stream lines** are based on the abstract idea of a completely relaxed line with diameter zero representing the trajectory of a particle in a steady flow. Relaxed means that the velocity component perpendicular to the tangents in the supporting points are zero and a steady flow can be defined by a constant value for τ.

- **Path lines** are basically composed in the same way as **stream lines** whereas path lines are built-up piecewise with varying value for τ. This states that path lines are the same as streamlines for steady flows and a constant τ. Path lines can be observed in unsteady flows in nature whereas stream lines only occur in non-natural steady flows.

- **Streak lines, time lines** are in general the same as a direct particle visualization but with different seed-point strategies. These two techniques are based on the classical (physical) flow illustration methods which mostly utilized some kind of

dye in a flowing fluid or air. The first type of line uses evenly spaced points for a persistent injection of particles and the second approach defines a continuous line of seed points with evenly spaced injection times. Hence streak lines can be compared with the commonly known technique of smoke in a wind tunnel and time lines with a permanent dunking of a dye plate in a streaming fluid. Following [Kalkusch2005] they are defined for a varying τ only.

- **Stream ribbons, stream tubes** and **stream balls** can be generated by additional considerations of the rotational effect of a flow field along a trajectory. Stream ribbons will behave similar to completely elastic strips brought into an unsteady flow. Stream tubes are the same as stream lines but with a constant diameter or an elliptic diameter to get information of the rotation component as well and balls are geometries placed at supporting point positions with variable defined elongation depending on the flow's direction and strength at these positions.

For the sake of completeness, another non-particle trace based approach for spare flow field representations can be mentioned. With the principles described in section 2.1.1.1, iso-surfaces can be used to show flow characteristics as well. Surfaces may be extracted, for example from positions with the same velocity magnitude or related parameters as shown in figure 2.14(e).

Even illumination of structures like massless lines or particles can be very important for depth perception as compared in figure 2.14(b) . Stream ribbons, stream tubes and stream balls consist of connected geometry parts, which allows a classical Gouraud shading or Phong shading due to well defined normal vectors of the surfaces. In the case of massless particles and lines illumination is not as trivial because of arbitrary direction of the normal vector within a tangential perpendicular plane according to the line. These inconveniences have been resolved by [Zockler1996] with a reformulation of the Phong lightning model. The light intensity at a point on a surface is given by

$$I = k_a + k_d(\vec{L} \cdot \vec{N}) + k_s(\vec{V} \cdot \vec{R})^n \tag{2.6}$$

with the ambient, diffuse and specular coefficients k_a, k_d, k_s and n. The unit length vectors $\vec{N}$ for the normal of a surface, $\vec{L}$ and $\vec{V}$ pointing toward the light source and the eye-position and $\vec{R}$ referring to the reflection of $\vec{L}$ and $\vec{N}$. These are all defined except

of $\vec{N}$. $\vec{N}$ can be approximated by choosing it in a plane perpendicular to the tangent $\vec{T}$ so that $(\vec{L} \cdot \vec{N})$ and $(\vec{V} \cdot \vec{R})$ are maximized. Following [Banks1994; Stalling1997; Peeters2006] $\vec{T}$ is given so that $(\vec{L} \cdot \vec{N})$ and $(\vec{V} \cdot \vec{R})$ can be approximated by

$$\begin{aligned} \vec{L} \cdot \vec{N} &= \sqrt{1 - (\vec{L} \cdot \vec{T})^2} \\ \vec{V} \cdot \vec{R} &= (\vec{L} \cdot \vec{N})\sqrt{1 - (\vec{V} \cdot \vec{T})^2} - (\vec{L} \cdot \vec{T})(\vec{V} \cdot \vec{T}). \end{aligned} \tag{2.7}$$

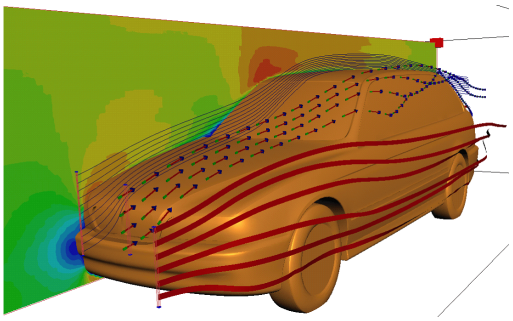
With these considerations an illumination of even diameter-less lines or particles can be implemented directly. Figure 2.14 gives an overview of the above described techniques on certain examples from different publications.

Accordingly a view onto dense vector field representations will be taken next. Since these techniques are not as exploited as particle traces in this work, only the main representatives are presented but none of them have been implemented.

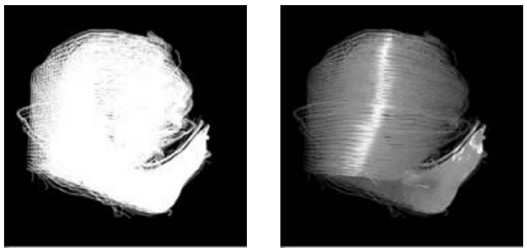
The class of dense flow visualization attempts to provide a complete, dense representation of the flow field. This may be achieved by influencing an arbitrary texture with an underlying velocity field, which is commonly known as convolution. For example, facilitating a filter kernel to smooth and convolute a given vector field with a white noise texture leads to the first and popular method of **L**ine **I**ntegral **C**onvolution (LIC).

LIC, as defined by [Cabral1993], is basically capable of 2D vector fields, steady flows and computational expensive. Figure 2.15(a) shows an example for a LIC based visualization with additional velocity magnitude coloring. Due to these deficiencies several extensions to LIC have been developed. Pseudo LIC (PLIC) [Vivek1999] figure 2.15(b), for example uses template textures mapped onto a spares streamline representation of a field which reduces the calculation work. Subsequent updating of the LIC-texture for unsteady flow has been resolved reasonably efficient by Accelerated Unsteady Flow LIC (AUFLIC) by [Liu2002] which is an extension on UFLIC [Shen1998].

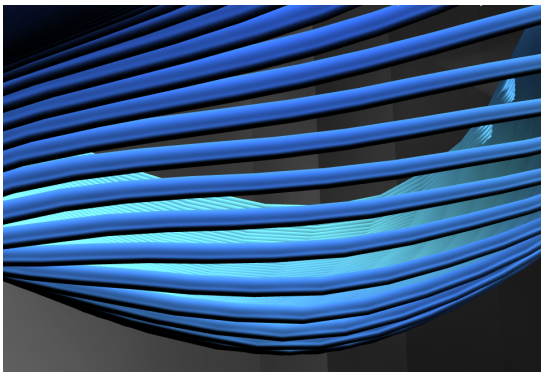
However, extending LIC to 3D may seem to be straightforward from an algorithmic point of view but the crux of this idea is the problem of occlusion in a three dimensional space. As always for volumetric data several approaches try to deal with these problems. Firstly an interactive approach with cutting planes can be chosen to either show a 2D-LIC texture on the cutting plane or only the volumetric rendered 3D-LIC texture behind the cutting plane [Rezk-Salama1999]. Secondly more complex algorithms can be



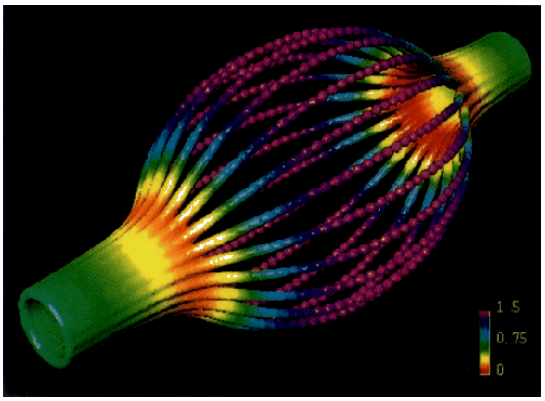
(a) Comparison of color plane, stream lines, arrows and stream ribbons in a wind tunnel simulation from [Schulz1999].



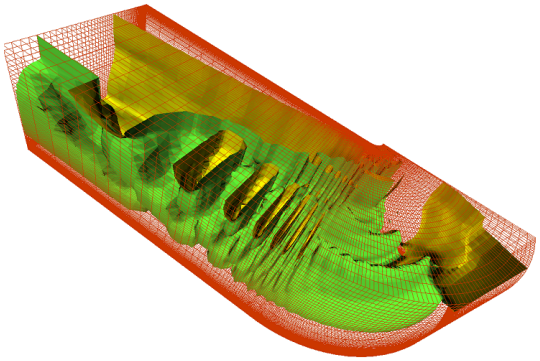
(b) Comparison of stream lines without and with illumination from [Peeters2006]. Compare to illumination techniques from [Banks1994].



(c) Stream tubes from section 5.3.3.5



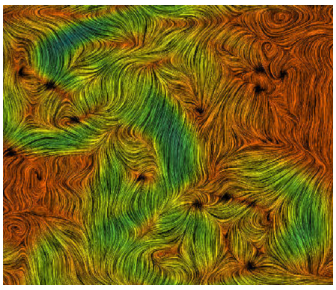
(d) Stream balls from [Brill1994].



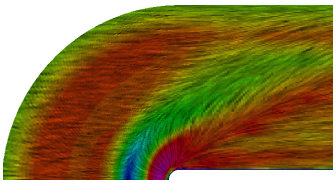
(e) Stream surfaces from [Pascucci2004].

Figure 2.14: A comparison between examples of different sparse flow vector field representations.

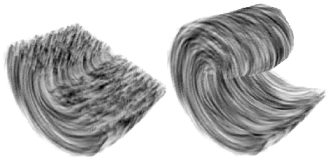
applied to bring out or suppress several important or fewer interesting areas of the 3D texture. Volume rendering techniques as describes in section 2.1.1.1 can here be applied as well, for example setting a transfer function. Figure 2.15(c) shows an applicatory example of such an approach from [Interrante1998].



(a) A LIC texture from [Cabral1993] of a fluid dynamics field multiplied by a color image.



(b) A PLIC evaluated dataset colored by the underlying velocity magnitude from [Vivek1999].



(c) Flow simulation with interactive settable transfer function and clipping plane [Interrante1998].

Figure 2.15: A comparison between examples of different dense flow vector field representations.

The algorithms introduced so far can be directly implemented on graphics hardware. Depending on the complexity such an approach will provide a certain speedup in contrast to implementations on a CPU. How graphics hardware can be programmed to execute algorithms which are actually intended to be used on a CPU is described in the next section.

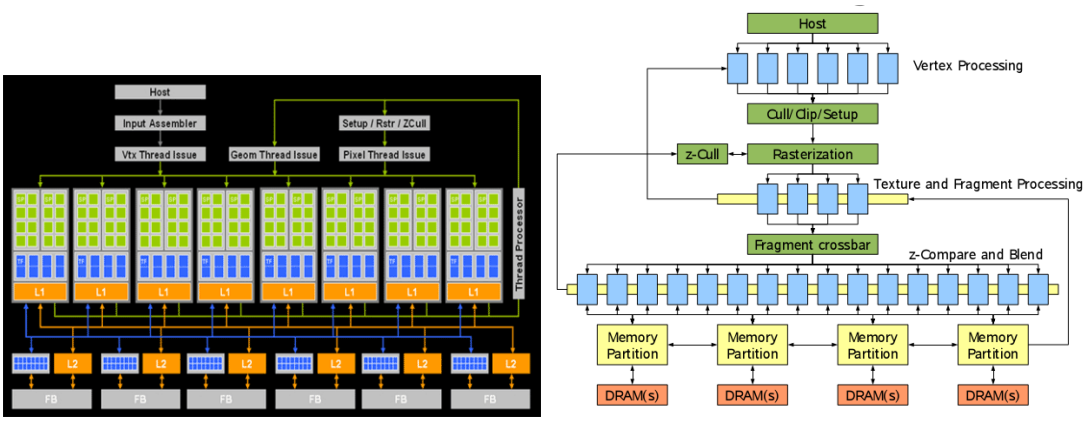
2.2 Advanced Graphics Processing

Today's personal computers are, even without 3D-acceleration cards, high-end graphics work stations compared to specialized hardware from 1982. The next sections will illuminate the newest available **G**raphics **P**rocessing **U**nit (GPU) and the principals of performing general purpose computations using this hardware unit. The reader is referred to [GbR2007][1] for a deeper insight into hardware issues. [GLSL2004; Cg2007; Pharr2005; FernGPU2004] will provide hints and definitions for programming on a GPU.

[1] and to http://en.wikipedia.org/wiki/Graphics_card

2.2.1 The Latest Available Graphics Hardware

After more than 15 years of usage, the basic vertex processing pipeline has been superseded by a new architecture. The classic programmable pipeline has several disadvantages. An unbalanced processor load for example will be an effect of much more pixel shader instructions than vertex shader instructions. Building one Arithmetic-Logic Unit (ALU) per vertex and fragment processing will result in an overloaded and an unchallenged unit. Furthermore it was nearly impossible to reprocess already calculated results in the pipeline. These drawbacks have been resolved by the G8x processor used on GeForce 8x cards. In this architecture at most 128 stream processors (GeForce 8800 GTX[1]) are able to process both vertex and fragment geometry instructions. Theoretically, employing a "Thread-Processor" for this should enable each ALU to operate at full capacity.



(a) Block diagram of the GeForce 8x architecture. (b) Block diagram of the GeForece 6x architecture.

Figure 2.16: Comparison of Nvidia's G8x (a) and G6x (b) processor architecture from [Nvidia2006] and [Pharr2005]. Remarkable is that the G8x stream processors are now able to do both, which was still separated on the G6x series.

Due to increasing programmable requirements of graphics hardware, Nvidia decided to design the architecture around the processor as illustrated in figure 2.16(a). Consequently, it is now possible to overcome the two main drawbacks described above - "real loops" and "balanced load". However, although the implementation of this work in chapter 5 has been performed on a GeForce 8800 system, mainly the hardware

[1] released in November 2006,

programming features available since Shader Model 3.0[1] and the increased processing speed have been used. This is because of the desired compatibility to currently available systems in clinical routine. The next section outlines only the basic concepts of shader programming since Shader Model 3.0.

Currently three main shader languages exist:

- **Cg** which stands for "C for graphics" and is developed by Nvidia,
- **GLSL** stands for "**G**raphics **L**ibrary **S**hader **L**anguage" and is provided for OpenGL,
- **HLSL** stands for "**H**igh **L**evel **S**hader **L**anguage" and applies to DirectX[2] based applications.

They are all C-like programming languages and are of the same value. The implementation in chapter 5 concentrates on GLSL. However the following techniques can be implemented with all other languages as well.

In general two types of shader programs can be distinguished. Firstly a so-called *vertex program* executed by the vertex processing unit is at least able to define a new position for each given vertex position processed for each frame with any parameters as input. Secondly a so-called *fragment program* processes all pixel candidates which were produced between the vertices by a rasterization unit. The output of a fragment shader will always be a RGBA color value. The output of such shader does not necessarily need to be rendered to a display device. Instead it can be written into a buffer or texture for further use. Furthermore the implicit float arithmetic of graphic microprocessors allows calculations for general purposes. A benefit - in contrast to CPU processing - will be achieved if graphics operations are performed on multiple input data and vectors data types. Since graphics hardware matches these constraints it is also called a **S**ingle **I**nstruction **M**ultiple **D**ata (SIMD) Processor. A perfect example for such a SIMD operation will be the update mechanisms on millions of particles moving through a flow velocity field as described in section 2.1.1.2.

[1] Shader Model 3.0 (SM 3.0) was developed by Nvidia with the GeForce 6x-series in 2004.
[2] http://www.gamesforwindows.com/en-US/AboutGFW/Pages/DirectX10.aspx

2.2.2 GPGPU - General-Purpose Computation on the GPU

Programming the GPU for general purposes has several constraints. The techniques to handle them are presented in this section and much more in detail in [Thompson2002; Pharr2005; Nvidia2005].

2.2.2.1 Graphics Card/GPU Memory

The access to memory on GPUs is restricted to various kinds of indexed texture memory lookups into one, two or three-dimensional banks of fast accessible texture memory. The amount of memory has to be defined in advance through a texture of a certain size. Each of its "memory" element (texel) gets processed during one iteration. First and foremost this means that the programmer has no opportunity to allocate memory during the shader program's execution or to use pointers. Nevertheless, these concepts are generally not necessary since the complete data is represented by a stream and interpreted by a stream-processor. Consequently, using memory on graphics hardware in both, vertex and fragment shader will mostly look like the following algorithm:

1. define a texture of a certain size with the main application,
2. pass the texture-ID to the desired shader program,
3. define in the shader program what to do with each texture's element,
4. tell the hardware when to use the shader in each rendered frame.

Using memory is always accompanied by two basic concepts:

- **Gathering** describes an operation which takes values from different memory addresses to obtain one result.
- **Scattering** is the opposite an operation which stores several results in different memory addresses at once.

On a GPU only the first concept is available because of the fixed defined output address of the process data stream. However, it is possible to store a result in multiple render targets at this predefined position. This implies that graphics hardware is able to render calculated pixel values not only on screen but also in a dedicated buffer in the

texture memory. Multi-target rendering has therefore available on common graphics hardware for a long time.

2.2.2.2 Execution concepts

Due to the stream architecture of GPUs some well known concepts which are common with CPUs have to be refined. Table 2.1 provides an overview of these refinements and proposes the relative complements of these two architectures. To evaluate if it is fruitful to implement a certain algorithm on the GPU the

$$arithmetic\ intensity = \frac{operations}{words\ transferred}. \tag{2.8}$$

of this algorithm has to be considered. If the *arithmetic intensity* is high enough, the algorithm may be parallelized and therefore executed on a stream processor. Also the pipelined data-flow and a sequential graphics memory access have to be considered. Consequently, the algorithm should use the data in a sequential order as well to guarantee an efficient traversal. For example such algorithms are applicable for solving complex linear systems of equations, physics simulations, shortest-path and finite-differences calculations.

CPU	GPU
Array	Texture
Loop	Fragment processing
Feedback	Render-to-Texture
Memory address	Texture coordinate
Memory allocation	Texture generation
Branch	Branch or Conditional write
Invocation	Frame update

Table 2.1: Analogies between basic CPU concepts and their possible implementation on a GPU. This analogy can akin be found in [Pharr2005, page 500-503] and a related table was presented in [GrabnerKalkusch2006].

The inconveniences with a fixed stream have already been mentioned. An iterative calculation or the ability to update the desired results will only be possible if the actual result can get passed back. The work around on previous graphics cards was to render into a buffer with the first pass and use the buffer or texture as input for a second render

pass. Since passing back the results in the same render pass is currently only available on late-breaking Nvidia G8x architectures, the programmer will have to by-pass this in a way called *multi-pass rendering* .

- *multi-pass rendering* uses the output buffer or texture which was generated in on render pass in the next render pass as input.
- *single-pass rendering*, in contrast, generates the final result in one render pass.

Additional to multi-pass rendering a technique called *double buffering* has to be used due to most hardware's disability to read and write to the same texture. This technique switches two identical textures in each render pass. Depending on the frame rate of the application these textures are subsequently exchanged and updated. Subsequently, only one iteration step can be performed each frame but this for millions of parallel iterations at once. Depending on the settling time of the system or an abort criterion, such calculation updates of parallel systems can be performed much faster on nearly all currently available graphics hardware cards than on a CPU with this method.

Despite these shortcomings there are also several comforts when programming graphics hardware. For example a bilinear or trilinear interpolation, not only for a texture, can be performed with hardware implemented units. This interpolation tends to be really fast. Furthermore all in hardware implemented clipping mechanisms can be used for many parts of standard algorithms which can imply another speedup.

Concluding, it must be noted that shader programs can not replace conventional application programs due to the lack of high level languages but for problems with a high arithmetic intensity as defined by equation 2.8. For sequential data access, GPU stream processors can increase the performance remarkable. Since complex visualizations are commonly linked to highly parallel costly calculations a programmable graphics card is rather predestined for both: Calculation of the desired simulations and a subsequent direct rendering of them.

Chapter 3

Magnetic Resonance Imaging for Flow Sensitive Data

This chapter is directed towards physicists and readers interested in medical imaging. This chapter is not exhaustive a basic understanding of the MRI is assumed.

Since the late 1970's, Nuclear Magnetic Resonance Imaging (NMRI) is one of the most important diagnostic imaging technologies besides Ultrasound-Imaging (US), invasive X-Ray and Computed Tomography Imaging (CT) . Bloch and Purcell could potentially not anticipate in 1946 how fruitful their experiments on NMR would be for future developments in medicine, [Bloch1946; Purcell1946]. They showed independently how nuclei with an odd atomic number absorb and pass back high energy radio frequency impulses within a certain spectrum. To measure this property of matter they placed samples in a strong homogeneous directional magnetic field. This fact builds the core of a MRI-System to this day. To circumvent a historic time line of the MRI development, the number of earned Nobel-prizes directly linked to NMR related topics should announce the significance of this technique. It were not less then **six** to date. [1]

This chapter will give a brief introduction from the basics of NMR signal recovery to the principles of phase contrast flow measurement. More details on magnetic resonance imaging can be found in the compendiums of MRI techniques from [ClincalMRI2006, Section I, pages 23-248 and Section II, pages 693-740 and pages 843-860], [Scott1962; Abragam1989] and more summarized in [Hornak2006; Jakob2006; Slichter1990; Liang1999; Edelstein1987; Haase1986; MR-TIP; Stollberger2007]. Flow

[1] Stern (1943), Rabi (1944), Bloch & Purcell (1952), Ernst (1991), Wuethrich (2002), Lauterbur & Mansfield (2003) - *http://en.wikipedia.org/wiki/List_of_Nobel_laureates*

measurements have additional been described in detail by [Hinshaw1983; Laub1998; Markl2006; Markl2003; Mostbeck1992].

Section 3.1 to 3.5.1 will describe MRI-Signal formation and localization via gradient-echo methods up to parallel imaging with cardiac phase contrast angiography techniques based on [Stollberger2007]. Finally the sequences we used to acquire the flow sensitive datasets are introduced briefly in section 3.5.2.

3.1 Magnetic Field and Magnetization

The MR signal itself is based on the well known spin based view on quantum mechanical particles. According to this concept all particle in elementary systems own a certain intrinsic angular momentum $\vec{J}$. Since a combination of neutrons and/or protons within a nucleus is only stable with different spin directions the net spin of a nucleus with an even number of elementary particles is zero. Consequently only nuclei with an odd atomic number will provide a net nuclear spin which leads to a magnetic momentum $\vec{\mu}$, related with the *gyromagnetic ratio* γ to $\vec{J}$.

$$\vec{\mu} = \gamma \vec{J} \tag{3.1}$$

The *gyromagnetic ratio* γ depends on the observed nucleus and describes the magnetic dipole moment to its angular momentum or in classical physics the ratio of the charge of the particle considered as circuit current to two times its mass, valid as long as its charge and mass are identically distributed.

$$\gamma = \frac{q}{2m} \frac{[Hz]}{[Tesla]} \tag{3.2}$$

For MR imaging typically hydrogen (H^1) is used. One the one hand γ can be specified relatively easy since the hydrogen nucleus consists of only one proton and on the other hand this is the most frequent nucleus in the human body. Its *gyromagnetic ratio* is $\gamma = 2.675 \times 10^8 [rad/sec/Tesla]$ or $\overline{\gamma} = \frac{\gamma}{2\pi} = 42.58 \frac{[MHz]}{[Tesla]}$

From a quantum mechanical point of view the net spin $\vec{J}$ of a nucleus can occupy only two discrete energy states. The magnitude of $\vec{\mu}$ is therefore given as

$$|\vec{\mu}| = \mu = \gamma \cdot \hbar \sqrt{I(I+1)} \tag{3.3}$$

with a general spin quantum number $I = 0, \frac{1}{2}, 1, \frac{3}{2}, 2, ...$ and the Planck's constant $\hbar = h(6.6 \times 10^{-34} Js)/2\pi$. Considering H^1, I is defined by $I = \pm\frac{1}{2}$.

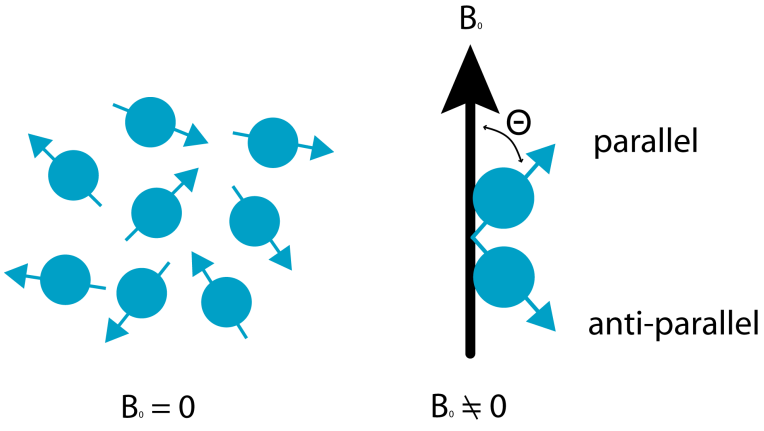
If there is no external magnetic field in equilibrium the orientations of the magnetic dipole moments are uniformly distributed due to of thermo-dynamics and other energy transfer effects. Hence there is no measurable magnetization of a sample containing a large number nuclei as shown in figure 3.1(a) . In an external magnetic field $\vec{B_0}$ the bulk magnetization of the sample accommodates with this field for spin $\frac{1}{2}$ with

$$\cos\Theta = \frac{\mu_z}{\mu} = \frac{m_I}{\sqrt{I(I+1)}} = \pm 54°44' \tag{3.4}$$

$$\mu_z = \gamma \cdot m_I \hbar$$

$$m_I = \pm\frac{1}{2}$$

where m_I is the magnetic quantum number. Figure 3.1(b) illustrates this behavior rudimentarily.



(a) The magnetic moments of the nuclei of a sample without the influence of an outer magnetic field.

(b) The magnetic moments accommodate with an outer magnetic field.

Figure 3.1: Influence of an outer magnetic field on the nuclei with odd atomic number in a sample.

In the classical mechanic approach, after solving the Euclidean movement equation

of the magnetic moments

$$\frac{\partial \vec{\mu}}{\partial t} = \gamma \vec{\mu} \times \vec{B_0} \tag{3.5}$$

the complete magnetization is then given by

$$\begin{aligned} \mu_{xy} &= \mu_{xy}(0) \cdot e^{-i\gamma B_0 t} \\ \mu_z &= \mu_z(0), \end{aligned} \tag{3.6}$$

assuming that $\vec{B_0}$ points along the z-direction. This magnetization is called bulk magnetization.

Consequently, only the magnetization in z-direction is fixed whereas the magnetization in xy-direction relates with the angular frequency γB_0. A common term to say is that the magnetization precesses with the **Larmor-frequency** around the magnetic field-vector B_0. This precession frequency, also known as the natural resonance frequency of a spin system, is given by the Larmor equation and illustrated in figure 3.2:

$$\omega_0 = \gamma B_0 \tag{3.7}$$

$$f_0 = \overline{\gamma} B_0 \tag{3.8}$$

The remaining part to think about is, how much of this directional magnetization can be used for experiments. The following considerations will lead to a possible bulk magnetization

$$\vec{M} = \sum_{n=1}^{N_s} \vec{\mu}_n \tag{3.9}$$

of a given huge number of H^1 nuclei, such as contained for example in a small part of tissue.

Since the magnetic moments can set up both parallel and anti-parallel to B_0, most of them will be nullified.

The energy of a spin state is given by

$$E = -\vec{\mu} \cdot \vec{B_0} = -\mu_z B_0 = -\gamma \hbar m_I B_0 \tag{3.10}$$

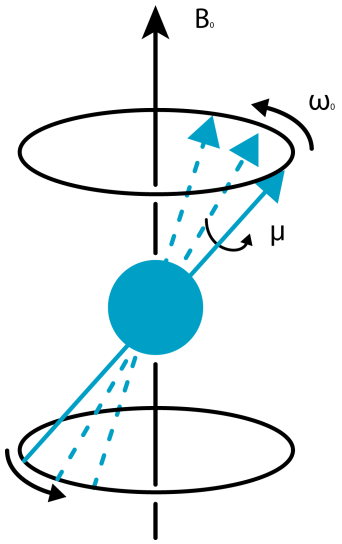


Figure 3.2: Precession of the magnetic moment μ of an isolated spin around a magnetic field B_0 with the Larmor-frequency ω_0.

and especially considering the two states for a hydrogen nucleus by

$$E_{\uparrow}(m_I = -1/2) = \frac{1}{2}\gamma\hbar B_0 \tag{3.11}$$
$$E_{\downarrow}(m_I = 1/2 = -\frac{1}{2}\gamma\hbar B_0.$$

The energy difference between these two states adds up to

$$\Delta E = E_{\downarrow} - E_{\uparrow} = \hbar\gamma B_0 = \hbar\omega_0 \tag{3.12}$$

with the Larmor-frequency ω_0. For the number of spins per unit volume can be assumed with the two and all possible states $N = N_{\uparrow} + N_{\downarrow}$ and therefore the distribution of these states is given by a Bolzman distribution. That means if $N_{\uparrow}$ refers to the spins parallel to $\vec{B_0}$ and $N_{\downarrow}$ define the spins anti-parallel the distribution is given by:

$$\frac{N_{\uparrow}}{N_{\downarrow}} = exp(\frac{\Delta E}{kT} = exp(\frac{\hbar\gamma B_0}{kT}). \tag{3.13}$$

with the Bolzman constant $k = 1.38 \times 10^{-}23J/K$ and T as the absolute temperature of the spin system.

With a net-magnetization of $M_0 = \mu \cdot (N_{\uparrow} - N_{\downarrow})$ and assuming that $\Delta E << kT$ the approximation of 3.12, which is with the approximation ΔE

$$exp(\frac{\Delta E}{kT}) \approx 1 + \frac{\gamma\hbar B_0}{kT} \Rightarrow \quad \Delta E << kTN_\uparrow - N_\downarrow \approx N \cdot \frac{\gamma\hbar B_0}{2kT} \tag{3.14}$$

The total magnetization $M_0 = \mu(N_\uparrow - N_\downarrow)$ results then from

$$M_0 = N \cdot \frac{\gamma^2\hbar^2 B_0}{4kT}. \tag{3.15}$$

Equation 3.13 shows that the energy state of a few more spins is **parallel** - actually the z-component of the magnetization vector - to B_0. Consequently, there is a small magnetization to experiment with left but a decaying transverse (xy) component.

3.2 Excitation of Magnetized Matter

Without excitation magnetization cannot be used to get material-dependent signals from a certain volume unit. The generated magnetization has to be rotated to get measurable signals in a way that the available circuit current and the resulting magnetic field, can be detected by a coil. This rotation of a equilibrium set magnetization can be achieved with the influence of a time-varying circular polarized high-frequency magnetic field[1].

Choosing ω_{hf}, so that it satisfies the on-resonance condition $\omega_0 = \omega_{hf}$, a rotation of the magnetization can be achieved. Such an additional magnetic field is called RF-pulse because ω is situated in the range of radio-frequencies and the duration of the pulse τ_p takes some milliseconds.

Since further considerations would be rather difficult if one remains in a constant laboratory coordinate frame, the upcoming ones will be done in a rotation reference frame (x', y', z'), in which the xy-plane is oscillating with ω_0. This frame provides three main advantages:

- circular signals are represented by their encasing curve
- rotation of a magnetization vector can be adopted as tilting toward y', instead as a circular movement on a spheric curve
- B_1 seems to be a static field along the x' axis

[1] A birdcage resonator produces a pure circular polarized field without any counter-components. All modern MRI-Systems provide this type of resonator.

With these assumptions the flip angle produced by a certain RF-pulse for a given magnetization vector results in

$$\alpha = \omega_1 \tau_p = \gamma B_1 \tau_p \tag{3.16}$$

with ω_1 referring to the spin's precession frequency. Figure 3.3 shows this coherences in the rotating and the stationary coordinate frame.

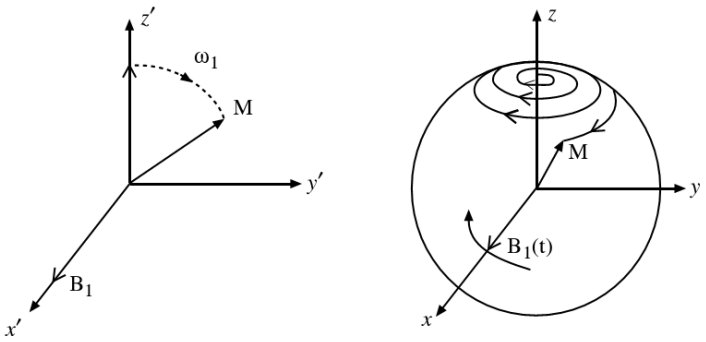


Figure 3.3: The rotation of the magnetization vector in the rotation (left) and the stationary coordinate frame (right). Image taken from [Liang1999]

Accordingly the magnetization vector rotates toward the y'-axis. In the laboratory coordinate system (x, y, z) the movement describes a precession around the z-axis. After τ_p a signal called Free Induction Decay (FID) is detected with in $\vec{B_0}$ placed coils and characterized by an exponential relaxation processes towards the equilibrium as described in section 3.2.1.

3.2.1 Precession and Relaxation

The signal of a spin system which has been excited with a RF-pulse can only be detected for some milli-seconds. The reason is that the spins are influenced by other magnetic fields from their neighborhood as well. In short terms, the excitation energy can be transferred to surrounding spins, which causes a dephasing of the magnetization vector within the $x'y'$-plane referred as *Spin-Spin-Relaxation or transversal relaxation* and additionally the rotated magnetization vector will tip backwards to its stationary position parallel to the direction of the field B_0 which influences the amount of signal for each repetition of the measurement. This is called *longitudinal relaxation or Spin-Grid-Relaxation.* The latter can be phenomenologically interpreted as the

strive for thermo-dynamic equilibrium and the first one as an annoyance (in example an additional excitation of a coincidentally with ω_1 precessing magnetic moment) of surrounding fields. [Abragam1989] provides an insight into the complete quantum mechanical processes and their classical interpretation on these relaxation processes. The upcoming considerations are based on these assumptions, because a detailed derivation of these coherences would go beyond the scope of this thesis.

The description of the relaxational movement of the magnetization vector and hence the amount of detectable magnetization after an excitation pulse can be approximated with a first order process in the rotation coordinate frame

$$\frac{\partial M_{x'y'}}{\partial t} = -\frac{M_{x'y'}}{T_2} \tag{3.17}$$

for transversal relaxation during time T_2 and

$$\frac{\partial M_{z'}}{\partial t} = -\frac{M_{z'} - M^0_{z'}}{T_1} \quad z' = z \tag{3.18}$$

for longitudinal relaxation processes during a longer time T_1. $M^0_{z'}$ describes the available rotatable magnetization in z'-direction before any RF-pulse.

The solution of this system of differential equations leads to

$$M_{xy} = M_{xy}(0_+) \cdot e^{-t/T_2} \cdot e^{-i\omega_0 t} \quad and \tag{3.19}$$

$$M_z = M^0_z(1 - e^{-t/T_1}) + M_z(0_+) \cdot e^{-t/T_2} \tag{3.20}$$

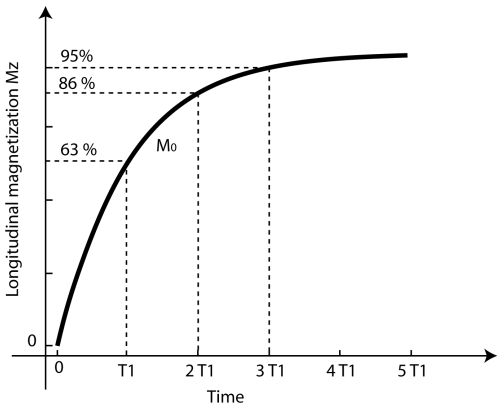
in the laboratory system at a time t after RF-excitation. Figure 3.2.1 illustrates these coherences explicitly.

3.2.2 Nuclear Magnetic Resonance Signal

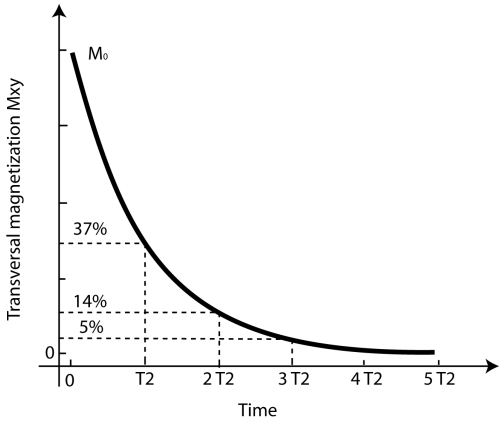
As a result of inevitable inhomogeneities of the magnetic field B_0 the purly material specific reachable time for T_2 is actually not the relevant time for transversal relaxation. Instead the transversal relaxation happens much faster than T_2 during the so called time T_2^* where

$$\frac{1}{T_2^*} = \frac{1}{T_2} + \gamma \Delta B_0. \tag{3.21}$$

This circumstance is not necessarily unfavorable since the dephasing caused by field inhomogeneities can be compensated - in contrast to the irreversible T_2 decay. Two



(a) The time T_1 is defined as the time in which 63% of the primary magnetization M_0 has been restored through the relaxation process in z-direction because the signal comes from induction.



(b) The time T_2 is defined as the time in which 37% of the transversal magnetization is left, which correlates with the time after which 63% of the signal got lost.

Figure 3.4: The coherences between signal intensity, loss of magnetization in transversal direction and the relaxation back to a longitudinal magnetization.

major techniques exists to restore the signal after time T_2^*. A signal produced by the restoration of such a decay of the FID is commonly called an echo.

The dephasing of the stimulated spins can be envisaged as slowing or increasing the rotational speed of the magnetization vectors caused by the fields of neighboring spins[1]. That means that some magnetization vectors will run faster and some slower until all of them populate a uniformly distributed direction so that they compensate each other. One way to rebuild the signal is the very popular Spin-Echo technique, which will not be described in detail here. The interested reader is referred to references given in section 3.

A Spin-Echo technique for example utilizes another RF-pulse, which turns all magnetization vectors around 180 or less degrees. This forces the dephased spins to rephase again, because faster vectors are then behind the aspired magnetization and slower ones in front. In a manner of speaking the vectors gain on each other. The signal will then arise as a Hermitian echo weighted with T_2. Weighted means in this sense that the main contrast mechanism is caused by that certain material specific time.

[1] quantum mechanical: stimulated emission and absorption of energy quantums within the atomic union.

The same effect of rephasing the magnetization vectors can be achieved by applying an additional gradient field after an α-degree RF-pulse in one direction. This additional pulse causes an early dephasing of the magnetization vectors which can be canceled out by applying another gradient field with opposite sign. The effect is a rephasing of the spins and a measurable Hermitian signal but weighted with T_2^*. Figure 3.6 shows the pulse sequence diagram of a fundamental gradient-echo sequence called FLASH (**F**ast **L**ow-**A**ngle **Sh**ot). The time at which the echo reaches its maximum is called **echo time TE**.

In the rotating coordinate frame it can be easily shown that an additional magnetic gradient will force different spin vectors to change their phase according to

$$\phi(x,t) = \gamma \int_0^t -G_x x d\tau = -\gamma G_x x t, \quad 0 \leq t \leq \tau \;\; G_x = const., \tag{3.22}$$

The time until the signal decays is called T_2^{**} and after $\tau > 3T_2^{**}$ the signal is close to zero. The reappearance of the signal as Hermitian echo can be induced by an inverse gradient. For the specialized case of a gradient of the same strength as the dephasing one the spin phases are characterized by

$$\phi(x,t) = -\gamma G_x x \tau + \gamma \int_\tau^t G_x x dt = -\gamma G_x x \tau + \gamma G_x x (t-\tau), \quad \tau \leq t \leq 2\tau, G_x = const. \tag{3.23}$$

After time τ the spin's phases are again zero for that direction and an echo is formed. This echo time TE can be influenced by the strength of the refocusing pulse as well. A stronger gradient will drive the spins to get in phase faster, so TE will be shortened and vice versa.

Typical ranges of the above described relaxation times are for biological tissues:

- $T_{1bio} \approx 300 - 3000[ms]$,
- $T_{2bio} \approx 30 - 2000[ms]$,
- $T^*_{2bio} \approx 10 - 100[ms]$.

3.3 Signal localization

The provoked signal is meaningless without a reasonable localization of the signals' origin. A spatial mapping of a signal is mainly done in two steps:

1. A gradient field modulates B_0, so that only one slice can be excited by a certain RF-pulse with a certain bandwidth.

2. Two other gradients modulate the resulting signal, so that the lines of an excited slice are phase encoded and the contained frequencies can be discriminative split up.

Section 3.3.1 and 3.3.2 describes this mechanism more precisely.

3.3.1 Slice Selection

Modern medical imaging requires the formation of an arbitrary cross section through the body. In MRI such a slice or section[1] can be spatially selective excited by using a linear gradient field augmenting the field B_0 and a 'shaped' RF-pulse of a certain bandwidth. Both the slope of the gradient and the bandwidth of the RF-pulse will influence the thickness of the selected slice and its shape. The lateral profile of a slice is for small flip angles approximately the Fourier-transformed of the RF-pulse itself. The slope of the used gradient affects how many spins will be addressable and how much of the volume will be excited by the RF-pulse. Figure 3.5 shows the relationship between slope and bandwidth and their effects on the segment's profile and dilatation.

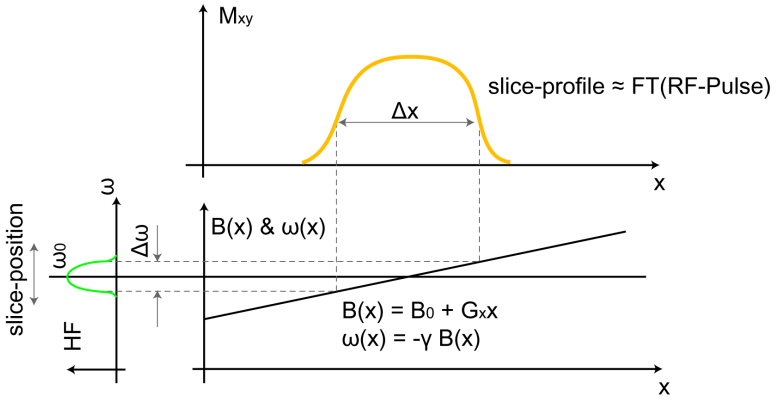


Figure 3.5: The slice-profile depends on the RF-excitation pulse and the slope of the slice selection gradient.

Direct transcription of the former assumption would require using a sinc pulse with a finite frequency bandwidth of $\omega_{hf} = \omega_0$. The Fourier transformation of such a pulse

[1] 'slice' and 'section' refers to the same in the following context.

would result in an ideal square slice profile. In practice the frequency range is higher since on the one hand a perfect sinc pulse is technically not possible and on the other hand a sharper desired slice profile requires a higher energy of the RF-pulse. Applying too much energy to a human body per time unit would lead to a warming of the tissue by more than $1°C$, which can be harmful for living cells. This absorbed energy amount is called *Specific Absorption Rate (SAR)* and restricts the slope and the speed of switching gradients, the energy of the RF-pulses and the sharpness of the slice profile.

The fact that the profiles are not ideal square cause a slice distance of more than zero or a doubled measurement time with interleaved slice measurement. Elsewhere partial volume effects would garble the resulting images depending on the sequence.

Figure 3.7 shows the selection of a segment, placed perpendicular in the xy-plane by a linear gradient G_z in z-direction. However, slice-selection in arbitrary direction and location needs a combination of gradients in three directions

$$G_{ss} = (G_x, G_y, G_z) = \begin{cases} G_x = G_{ss} sin\theta \cos\phi \\ G_y = G_{ss} sin\theta \sin\phi \\ G_z = G_{ss} cos\theta \end{cases} \tag{3.24}$$

and a RF-pulse

$$\omega_{hf} = \omega_0 + \gamma G_{ss} c_0, \quad \Delta\omega = \gamma G_{ss} \Delta c \tag{3.25}$$

with the excitation frequency ω_{hf}, bandwidth $\Delta\omega$, the center of the segment c_0 and its thickness Δc.

A frequency response of all spins situated in the selected segment will be detectable after an excitation done as described in this section. Obviously further dispartment of the signal will be necessary to locate a certain volume element's response within the activated segment.

3.3.2 Voxel Selection

During the precession period the second step of spatial localization is called frequency- and phase encoding. Frequency encoding makes the oscillating frequency of the complex MR-signal dependent on its spatial origin which can be utilized to locate a column of the activated slice. How to impose different Larmor frequencies on different locations has already been described in the previous sections. We need to apply another gradient

in one direction of the responding slice after the selective excitation of a segment. Then the frequencies of the spins will be modulated with

$$\omega(x) = \gamma(B_0 + G_x x) \tag{3.26}$$

for a chosen gradient in x-direction. When neglecting the proportionality relationship to the flip angle and a notional spin density distribution $\rho(x)$ the received signal of a sample with simultaneously activated *frequency encoding gradient* G_x follows from

$$S(t) = \int_{-\infty}^{\infty} \rho(x) e^{-i\gamma(B_0+G_x x)t} dx = \int_{-\infty}^{\infty} \rho(x) e^{-i\gamma G_x x t} dx e^{-i\omega_0 t} \tag{3.27}$$

and after the removal of the carrier signal $e^{-i\omega_0 t}$ (demodulation), a signal

$$S(t) = \int_{-\infty}^{\infty} \rho(x) e^{-i\gamma G_x x t} dx \tag{3.28}$$

results.

Encoding the phase angle of a collection of spins can be performed similarly. Applying a gradient G_y[1] with a certain strength between the RF-pulse excitation and the former described gradient will provoke a preparatory frequency encoding depending on the time T the gradient is applied. In combination with the following frequency encoding gradient[2] each row of the responding segment will accumulate different phase-angles

$$\phi(y) = -\gamma G_y y T, \quad G_y = const. \tag{3.29}$$

which leads to the term phase-encoding gradient for this special type of frequency-encoding gradient. Figure 3.6 shows the resulting pulse sequence diagram to obtain a sampled signal spectrum of one arbitrary line inside the activated slice. Such a sequence has to be repeated to get all voxel signals of a slice.

3.3.3 Image Formation

With the substitutions

$$k_x = -\gamma G_x t, \tag{3.30}$$

$$k_y = -\gamma G_y T \tag{3.31}$$

[1] G_y is not necessarily different from G_x. The notation G_y is only used to distinguish from frequency and phase encoding gradients.

[2] This frequency encoding gradient is also called *read-out gradient and notated as* G_x

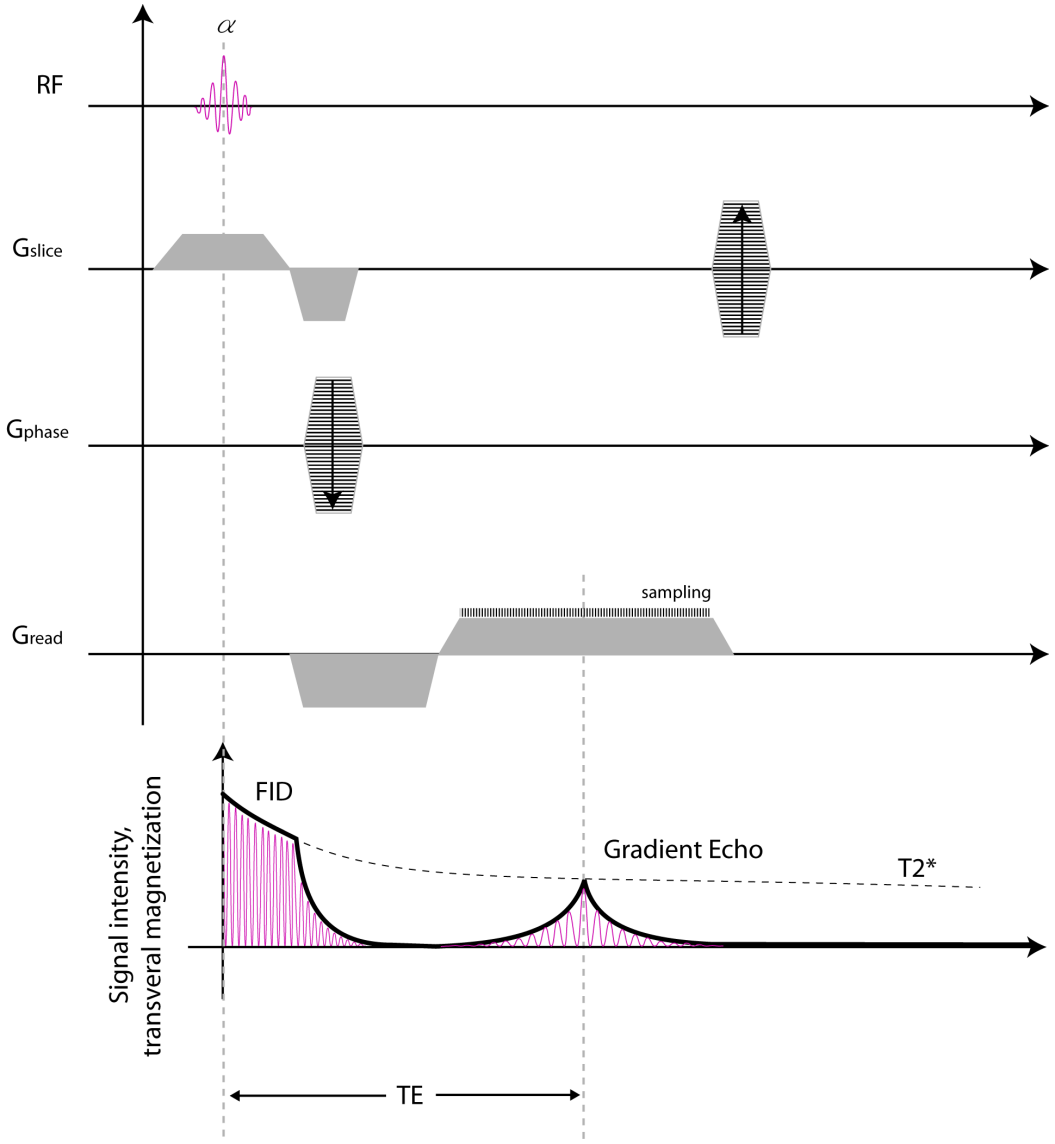


Figure 3.6: This is a complete pulse-sequence diagram of a FLASH gradient echo sequence. After the slice selection gradient and the RF-pulse, an inversed slice-gradient is turned on to rephase the slice. The phase-encoding gradient and the dephasing read-out gradient are turned on simultaneously. Subsequently the read-out dephasing gradient is compensated by the inverse read-out gradient. This provokes an echo which gets sampled with a certain rate. To destroy cohesive magnetization before the sequence is repeated, a gradient spoiling table is applied for the volume.

a commonly known (2D) Fourier relationship can be established

$$S(k_x, k_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \rho(x, y) e^{-i2\pi(k_x x + k_y y)} dx dy \tag{3.32}$$

In case of phase encoding, k_y is a function of the strength of the gradient G_y and in case of frequency encoding, k_x is a continuous function over time which should has been clarified in figure 3.7. k_x, k_y encoded measurement values can now be placed in a two dimensional space called **k-space**. Accordingly to equation 3.32 this two dimensional space is the Fourier-transformed of the desired intensity image of the excited cross section.

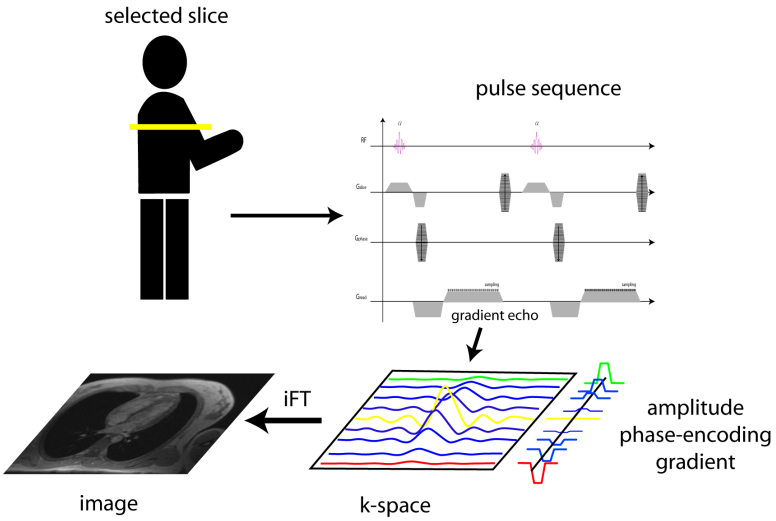


Figure 3.7: A pulse sequence is applied subsequently to obtain each line of k-space. The line position is determinated by the amplitude of the phase-encoding gradient G_y. An inverse Fourier-transformation of the filled k-space produces the desired intensity image.

It has to be mentioned that these coherences are not as smooth as they are described here. Each of the two encoding directions is especially vulnerable to one of two major artifact origins of MRI.

1. In frequency encoding direction the *chemical shift* artifact is recognizable. Small differences in the resonance frequencies due to the chemical bindings of the hydrogen form the basis of the *chemical shift* effect. Some tissues - like fat - seems then to be at a different position because of a shift in their responded frequencies.

This causes a spacial shift in frequency encoding direction which is noticeable as black gaps or overlays at tissue borders in the resulting intensity image.

2. Phase encoding may show ghost images superposed over the desired image. This artifact results from a moving object which causes different spectral maxima at different positions in k-space. Depending on the positions of the corrupted lines in k-space the resulting ghost-images will be blurred or edge based.

3.4 Image Contrast

The intensity of the responded signal - and consequently the brightness of a pixel in the resulting image - of a certain tissue depends on its specific T_1, T_2, or T_2^* decay and the proton density. The time TE when the echo is received and the repetition time TR in which the different phase encoded echoes are measured when the sequence is repeated. Because of the usage of special contrast mechanisms for the data described in section 3.5 the standard contrast mechanisms for morphological data are explained here. Figure 3.4 will give a basic understanding of T_1 and T_2 contrast mechanisms. For more information about the classic image contrast mechanisms, the interested reader is referred to the references. These considerations mainly apply for Spin-Echo sequences and only truncated for the sequences used for this work. Nevertheless, for the understanding of the imaging itself they are essential.

(a) T1-contrast is in a manner of speaking based one the amount of available - already relaxated - spins parallel to B_0. A new RF-pulse will rotate the whole magnetization vector, whereas for some tissue it will be higher than for others. In other words, if at the time of a pulse, spins are still excited, they will be rotated anti-parallel to B_0. Only these which have already relaxated will be turned back perpendicular to the measurement coil and will be noticed afterwards. For example, these coherences explain the secularly known behavior of *water* with T_1 weighted images. Nearly all hydrogen nuclei of H_2O are still excited at a short TR. A new RF-pulse makes them "invisible" or out-of-phase, so water will appear almost black.

(b) The T_2-contrast or T_2^*-contrast is defined by the transversal relaxation. Because of the different relaxation times the loss of signal will vary for different tissues depending on the time TE an echo is enforced. To refer to the showcase *water*, H_2O will produce a bright signal for a longer echo time TE as a result of many spins remaining in phase and "visible".

Figure 3.8: The basic contrast mechanisms for T_1 or T_2 respectively T_2^* weighted images. For T_1 weighting one has to consider mainly the time between the repetitions of the sequence TR and a short echo time. For T_2 weighting the time between excitation and the generated echo TE represents the major factors.

3.5 Cine Cardiac Fast Low-Angle Shot with Parallel-Imaging Technique

With the basics from section 3.1 to 3.4, the sequences used for flow sensitive cardiac imaging can be specified. The data generated as described below will build the starting point for the workflow presented in chapter 4 and the following.

A pulse sequence has to meet several demands for a continuous flow sensitive measurement of the cardiac movement of blood inside a breathing, even moving living organism:

1. The movement of the chest and the myocardium has to be compensated or suppressed. Otherwise moving artifacts would corrupt the data.

2. The duration of one heartbeat can be assumed to be one second. Within this second all images of all slices and all aspired temporal steps are ideally acquired. This implies that such a sequence has to be faster than basic MRI imaging sequences.

3. Additional to morphological data, comprehensible movement of the blood (for example) has to be encoded in all three spacial directions. This multiplies the number of necessary images by four.

4. Since the human ECG is not necessarily constant, a gating mechanism has to be used, which triggers a set of images or parts of them always at the same state of the heart-cycle.

Due to the fact that these requirements cannot all be met within one heart-cycle the virtual sequence utilize the beneficial properties of the k-space, a very fast imaging sequence with parallel imaging and a retrospective gating technique based on the patient's ECG.

A sequence which is fast enough to acquire at least a few lines in k-space of each temporal image is the so called Fast Low-Angle Shot or *FLASH*. FLASH uses the sequence described in image 3.6.

To accomplish a continuous measurement of the heart-cycle, segmented Flash is performed. This means, that only a few lines of the k-space are measured for one image in the cycle during one heartbeat. This implies that not all images of a *cine* series can

be measured at once. The images itself result from a combination of more than one heartbeat with the assumption that the heart-cycle is perfectly periodical and that an image taken at a certain time point of the cycle will always look the same.

To trigger the same point for some lines of a certain image's k-space, the patients ECG is recorded simultaneously for detection of the silhouetted R-wave. The images are then attempted to be equal spaced between the R-waves. Depending on the heart rate and the desired time-steps as many k-space lines as possible are measured for the morphological and the flow-sensitive images. This sounds simple but can be very difficult. Gathering a mV-signal of the human heart in the center of a KW transmitter demands special ECG-electrodes and electrode arrangements so that on the one hand the magneto-hydrodynamic effect[1] of the large vessels can be neglected and on the other hand, dependent on the used wires and their length, as less interferences as possible are absorbed. However, it is enough to have a detectable R-wave in the resulting signal. The signal-conduction is done as shown in figure 3.5[2] and submitted via Bluetooth® to the pulse sequence control unit. Figure 3.9 underlines this measurement process.

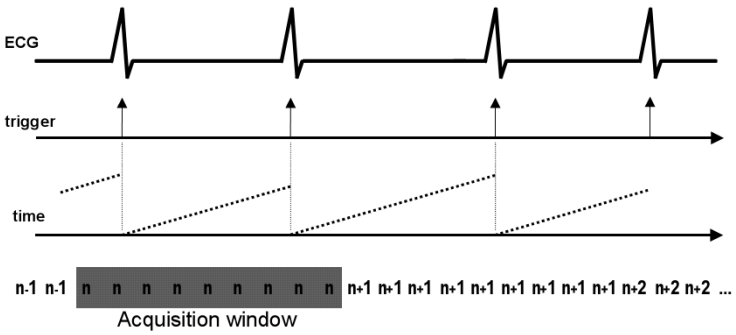
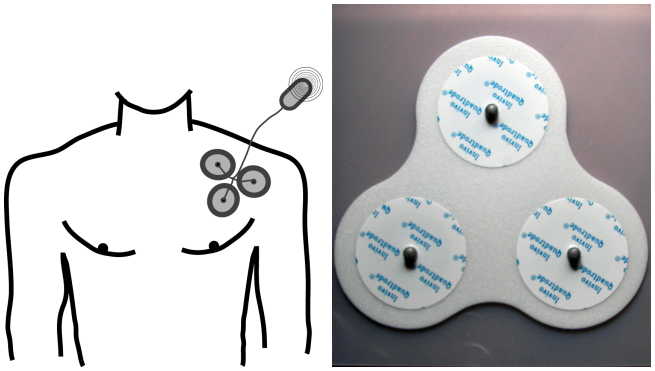


Figure 3.9: Triggered retrospective data acquisition. The phase encoding steps are incremented when the trigger signal is detected. Each line of data is timestamped and retrospectively sorted to reconstruct a series of images covering the entire cardiac cycle. Image taken from [Siemens2003].

As a result of the periodicity of the 2D Fourier k-space an image would be completed if *enough* lines are gathered after some heart-cycles.

[1] The magneto-hydrodynamic effect is is an additional electrical charge generated by ions in blood (loaded particles) moving perpendicular to the magnetic field. see [MR-TIP]

[2] Product (b): Invivo Research Inc., Quadtrode® 3, MRI ECG Electrode Pad, REF 9369N



(a) (b)

Figure 3.10: Figure (a) shows one of the possible interference-less positions for the three ECG gating electrodes and a Bluetooth® transmitter. Figure (b) shows the electrode-triplet itself carried out with fixed distances as electrode pad.

Using a rectangular Field of View (FOV) through larger but fewer phase-encoding steps fastens the acquisition at the same resolution but with a reduced signal to noise ratio. A drawback of this method are back-folding artifacts.

These artifacts would also occur while examining images recorded from the same region with different coils at once. This technique, using multiple coils is called *parallel imaging*.

Parallel imaging uses arrays of detector coils to acquire multiple data points simultaneously rather than one after the other. This technique is combined with certain k-space based reconstruction algorithms like GRAPPA[1] which is frequently used in today's cardiac MRI and contributes much to a reduction of scan time. To avoid a digression to the fundamentals of signal reconstruction from multiple coils, the interested reader is referred to [ClincalMRI2006, Section I-8, pages 231-248]. Figure 3.5 shows a typical body coil array as employed for the data used for this thesis.

The movement of the chest as a result of respiration can be suppressed in a rather easy way: The patient has to stop breathing during the time of acquisition. As a matter of fact, that even cardiac insufficient patients have to be examined, the measurement time is restricted to about 20 seconds. Since not all slices of a volume can be measured in that time, only all temporal encoded images of one slice are measured at once. This implies that the patient has to take a breath and hold it for each desired slice.

[1] GRAPPA = GeneRalized Autocalibrating Partially Parallel Acquisitions

(a) (b)

Figure 3.11: A typical body coil for thorax parallel image acquisition (a). This array-coil fits to the Siemens Magnetom Espree Tim [76 x 18] system, situated at Diagnostikzentrum Graz (b).

An alternative to the above described, are navigator triggered measurements with continuous respiration. For this method the movement of the diaphragm is taken additionally and only images at the same state of breath are taken then. The combination of this method with ECG gating will result in a longer measurement time and more produced images. Nowadays both methods are used whereas the first one does not apply to cine series so the former one is slightly more popular because of its easier application.

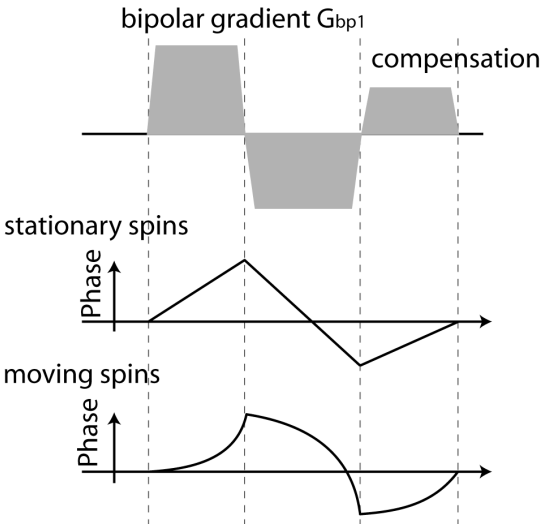
3.5.1 Phase-Contrast Imaging

The last untreated requirement from section 3.5 is the measurement of velocity of moving matter like blood. At the moment *phase-contrast magnetic resonance angiography* (PC-MRA) is one of the most reliable techniques to divide steady from moving spins. Additionally the quantitative velocity value and direction of a voxel can be retrieved by using this method as an add-on to the sequence outlined in section 3.5.

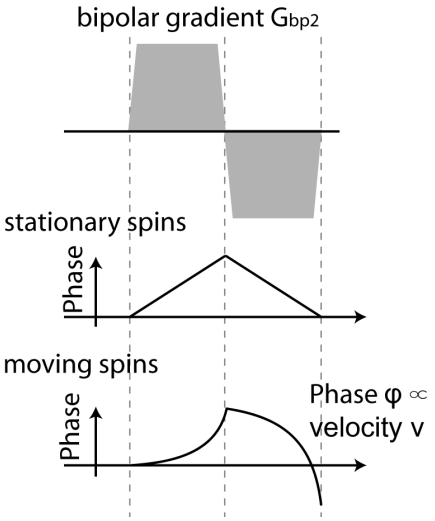
The PC-MRA is a well known angiography method. It relies on the fact that a bipolar gradient results in a zero signal for fixed spins. Moving spins will have a complex phase component as shown in figure 3.13.

Flow encoding contrast requires a supplementary complex reference signal acquired from a flow compensated. The amount of flow sensitivity is controlled by the strength

of the gradient pulse pair which is incorporated into the sequence. Figure 3.5.1 and Figure 3.13, show the remaining phase components of moving spins for the case of a certain bipolar gradient field in one direction.



(a) Measuring a flow compensated data requires to compensate an arbitrary bipolar pulse G_{bp1} so that out-of-phase spins are brought back to the same phase of the stationary spins. The result is a phase image with flow-sensitivity zero.



(b) Applying a self-compensating bipolar gradient in one direction leads to a phase-related flow sensitivity in that direction. Computing the complex difference of this signal with the complex reference signal from (a) produces an intensity value proportional to the quantitative flow value of this voxel in the gradient's direction.

Figure 3.12: Comparing a flow-compensated complex reference signal measurement (a) with a flow-sensitive one (b). Both use a bipolar gradient but in different fashion.

The complex subtraction of two of these data points produces an image with signal intensities depending on local flow velocities. A complex differentiation ΔS performed for one direction is shown in Figure 3.14. The length of the difference ΔS depends on the phase shift Φ_V for each pixel. The amount of phase shifts,

$$\Phi_V = \gamma \cdot A \cdot \tau \cdot v \tag{3.33}$$

depends on the area of each gradient pulse and distance between the pulses, with

γ $[Hz/T]$ ... gyro magnetic ratio

A $[m^2]$ ... area of each gradient pulse

τ $[s]$ ... time between the pulses

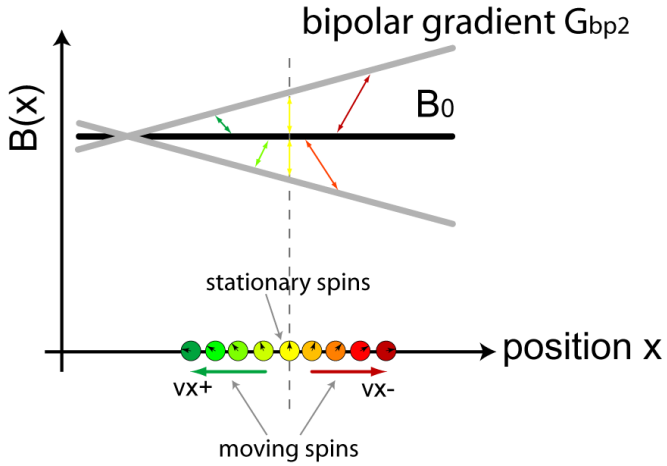


Figure 3.13: Flow sensitive images require a mapping in two directions, so stationary tissue will not appear as minimum gray value in the resulting image. This illustration shows that stationary spins will always see the same strength of the gradient fields but with different sign. For moving spins the gradient field will be too weak or too strong at different positions to compensate the shift, so that a net phase shift remains.

v $[m/s]$ .. velocity of the spins

An image which shows ΔS as signal intensity represents the velocity of spins at each point within the field of view for one direction, whereas the phase shift Φ_V is proportional to the spins velocity $v(x)$.

The phase-interval $-\pi$ to π refers to the gray-value range of g_{min} to g_{max} and even this again the velocity range v_{min} to v_{max}. The mapping velocity- to gray-value is given by

$$v(g) = v_{min}^{blood} + \frac{2v_{max}^{blood}}{g_{max} - g_{min} + 1} \cdot (g - g_{min}). \tag{3.34}$$

A complex phase component is always 2π-periodic. So it is very important to set the expected maximum velocity v_{max} called velocity encoding in advance. The maximum signal for ΔS is given for reverse directions of S_1 and S_2. This velocity is commonly called v_{enc} [1] and dependent on the type of sequence. A reverse direction means a phase difference of π. For velocities according to an amount larger than v_{enc} the difference signal is decreased constantly until it gets zero for a phase difference of 2π. Values occurring during the measurement which are larger than v_{enc} result in pixels suggesting blood or tissue moving towards the opposite direction. Such aliasing

[1] v_{enc} stands for Velocity ENCoding value

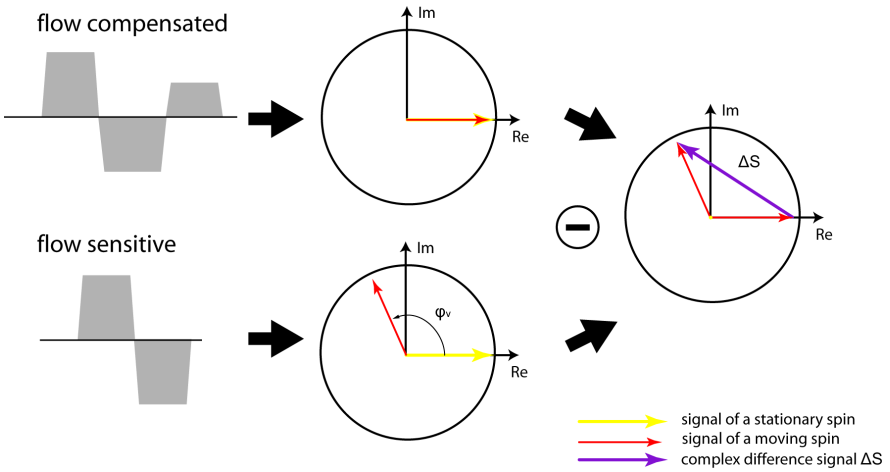


Figure 3.14: Computing the complex difference of a flow compensated and flow sensitive signal produces a complex difference vector for moving signals and zero for stationary spin signals. The image implies that one of the major defiances is the 2π-periodicity of a complex signal. The phase shift has to be mapped precisely between $-\pi$ and π which means that the maximum expected velocity has to be known in advance.

artifacts may occur from time to time because v_{enc} has to be defined empirically. On the other hand a v_{enc} value defined too high will result in a low contrast velocity coding image. That is one of the reasons some further post processing steps are required or the usage of a sequence with pre-measured v_{enc}. Since sequences with automatically estimated v_{enc} were experimental during writing this thesis, further data post processing steps are described in section 4.2 due to empirical defined v_{enc}-values. Figure 3.15 shows examples for $v(x_{ijk}) > v_{enc}$ and $v_{enc} << v_{max}$ and the 2π periodic phase-angle mapping as illustrated in Figure 3.16.

Since the previous method only applies for one spatial direction, two additional orthogonal flow sensitive images, each with its own v_{enc} value, have to be measured. Four images result consequently:

1. flow compensated image $Im1$
2. flow sensitive along x image $Im2$
3. flow sensitive along y image $Im3$
4. flow sensitive along z image $Im4$

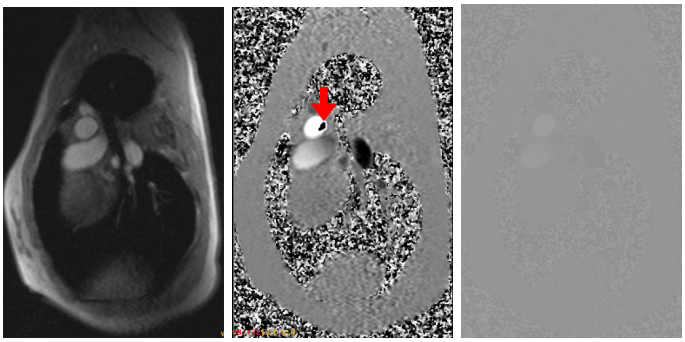


Figure 3.15: Left: Anatomical image of the scanned area. Middle: An example for aliasing artifacts as a result of a too small chosen v_{enc} value, $v_{enc} > v(x_{max})$. The black pixels within the white homogeneous area suggest an opposite flow direction (red arrow). Right: Low contrast if $v_{enc} << v_{max}$. (The images show the *Left Ventricular Outflow Tract* (LVOT), in plane x velocity encoded with sagittal main direction.)

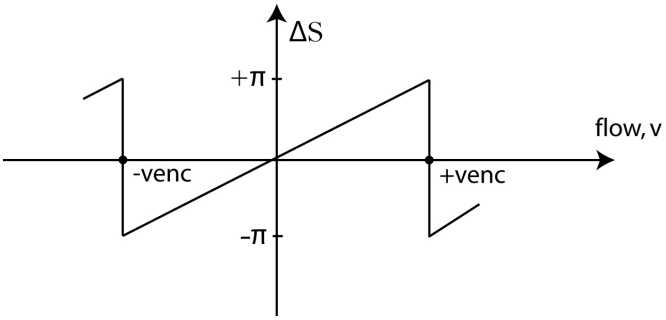


Figure 3.16: v_{enc} has to be set to the expected maximum velocity v_{max} otherwise measured values $v(x_{ijk}) > v_{enc}$ suggest a flow in the opposite direction or for $v_{enc} << v_{max}$ the velocity contrast will get too low.

Then, the spatial discrete velocity field result from

$$\begin{aligned} x_i = &\quad Im1_i - Im2_i, \\ y_j = &\quad Im1_j - Im3_j, \\ z_k = &\quad Im1_k - Im4_k, \end{aligned} \tag{3.35}$$

and

$$v(x_{ijk}) = v(x_i, y_j, z_k) = \tag{3.36}$$
$$(v_x(x_i, y_j, z_k), v_y(x_i, y_j, z_k), v_z(x_i, y_j, z_k)).$$

To acquire the whole dataset t_n (ECG triggered) time steps for all slices of the

volume have to be measured. Hence equation 3.36 amplifies to

$$v(x_{ijk}, s_m, t_n) = v(x_i, y_j, z_k, s_m, t_n) = \tag{3.37}$$
$$(v_x(x_i, y_j, z_k, s_m, t_n), v_y(x_i, y_j, z_k, s_m, t_n), v_z(x_i, y_j, z_k, s_m, t_n)),$$

for a spacial localization x_{ijk}, a slice indicator s_m and a time step t_n.

3.5.2 Data Acquisition

The datasets used for experiments in section 6 were acquired from healthy subjects, figure 3.5 with two 1.5 Tesla MRI systems (Magnetom Sonanta, Siemens, at the radiology department at the Landesklinikum Graz and Magnetom Espree, Siemens at the Diagnostikzentrum Graz).

Measuring a complete exam, as described in the previous sections takes between 10 and 35 minutes with temporarily given instructions to the subject to suspend respiration. The resulting image set is outlined in figure 3.17. The data can be accessed directly via the installed PACS [1] or exported and stored on various data carriers in the well known DICOM [2] file format.

[1] PACS = Picture Archiving and Communications System
[2] DICOM = Digital Imaging and Communications in Medicine

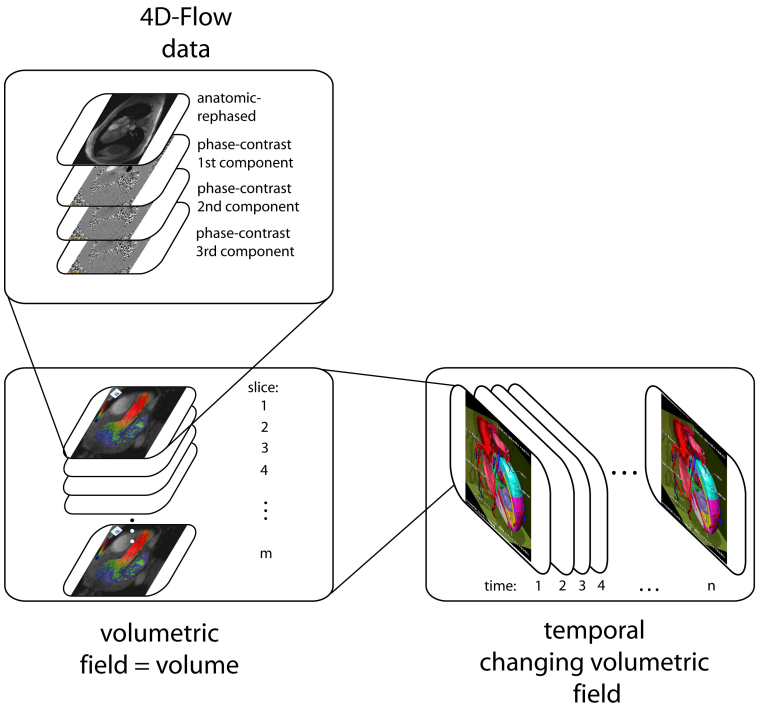


Figure 3.17: An overview of a volumetric time-dependent velocity-encoded dataset. The phase-contrast velocity encoded images contain one spacial direction component each. This part describes one slice for one certain volume. One volume consists of several of these slices, so the number of recorded images for one volume is given by the four images which are required for one slice, times the number of slices itself. Since we handle four dimensional data, there is even one volume per time step. Consequently the final number of raw images results by $\#images = 4 \cdot \#slices \cdot \#timesteps$.

Chapter 4

Work-Flow and System Architecture

This chapter introduces the necessary preprocessing steps before a visualization of PC-MRI datasets can be performed. Section 4.1 gives an overview of the requirements for an accurate visualization and the already available tools. Furthermore the desired software integrations ad combinations are proposed there. Subsequently, section 4.2 outlines the abilities of the Siemens Med 4D-Flow Toolbox [Reiter2006] which produces an intermediate result file for further processing. The parts of this file format which are necessary for this work are described in section 4.3 and appendix A. With these definitions a data reader can be implemented.

The second part of this chapter, starting from section 4.4, deals with utilized tools and libraries. Finally, the programming environment and the medical image viewer *iMEDgine* are presented. iMEDgine was extended with hardware accelerated, visualization algorithms in this thesis.

4.1 Requirements

Visualization of raw PC-MRI data, as they are described in chapter 3, requires preprocessing for aliasing corrections, baseline corrections and noise corrections. Therefore we used a toolkit (4D-Flow Toolbox) which we have developed for Siemens Medical Solutions in a previous project. The correction mechanisms of this toolkit are described in detail in section 4.2. For the resulting intermediate files, which are defined in section 4.3, we could subsequently elaborate the following requirements for a high-level

visualization of corrected four-dimensional velocity fields with additional medical image data.

1. After a measurement sequence, all resulting images are stored in the PACS[1] in the DICOM[2] format. Since only medical staff at facilities with PACS can access them, we decided to export them to DVD's or CD's. However, this implies that we need access to medical DICOM images and to the correlated DICOM-directories which are generated for exported datasets. An suitable interface is therefore the first requirement.

2. As already mentioned, most of the datasets need some preprocessing steps before a visualization can be done. With the 4D-Flow Toolbox it is possible to correct aliasing and baseline effects, to suppress noise and to draw segmentation contours. The resulting velocity field is not stored in the DICOM format but in a couple of its own files. Therefore we need to define a reader to access velocity fields in these 4D-Flow Toolbox intermediate format files.

3. Subsequently an efficient intern administration of the available medical image volumes and the associated velocity fields is crucial. We have to consider that some of the data may be of interest in different parts of the resulting software but that resources are limited on a personal computer. Furthermore most of the data is transformed to GPU-readable 3D-textures, so we have to consider an efficient way to administer data in GPU memory too.

4. The main objectives are interactive and highly efficient visualizations of interesting flow patterns with a concurrent rendering of morphological background information. The rendering of medical image volumes can be costly, so we have to consider fast algorithms for a concurrent calculation and visualization of flow patterns without a high CPU-load. Programs executed on the graphics unit of a PC seem to be ideal, so we need additional support for such kinds of algorithms.

5. To provide an overview of different flow patterns for comparison reasons we have to provide possibilities for a concurrent arrangement of different visualizations. The same data should be presented in different ways of representation.

[1] PACS = Picture Archiving and Communications System
[2] DICOM = Digital Imaging and Communications in Medicine

6. For concurrent views of the same or different datasets configuration mechanisms for each visualization are required during run-time. This can be achieved by using scripts or configuration widgets.

7. To explore the datasets we need interactive usable visualizations and therefore suitable control mechanisms. In particular, efficient global control of the time-variate volume is required.

8. Overall, extensibility and reusability of resulting visualization algorithms, in particular for other kinds of velocity fields should be considered. A general interface for velocity fields is therefore required.

9. To provide a comfortable exploration of the data, exporting of view and camera parameter of each visualization is necessary. Re-importing of this data to the application leads to a reconstruction of an arbitrary previous session.

10. Even if most of the computational work is done on the GPU, low-cost volume rendering techniques for morphological image data should be available. It must not happen that rendering of background information thwarts the whole application.

We decided to implement several extensions to the medical volume viewer iMEDGine. iMEDgine supports free arrangeable perspectives of the same or different datasets with scene graph based view widgets. Consequently, iMEDgine is the main part of the system architecture so this package already specifies most of the further software requirements as they are the Insight Segmentation and Registration Toolkit for medical data, Qt and SoQt as windowing system, Coin3D as scene graph library and SIMVoleon for volume rendering and the boost library for convenience. We added the Cash-Flow library to the iMEDgine framework to provide a scene-graph based data flow library and thus to meet the third requirement from above.

For high-performance visualizations we decided to heavily use general purpose computation shader which are executed on graphics hardware. For this purpose we added the Shallows library to the iMEDgine framework which abstracts the GPU as single stream processor. This makes the subsequent use of shader feasible and encapsulatable in special scene graph nodes.

4.2 Data Preprocessing

The data gathered with a technique as described in chapter 3 cannot be directly processed with algorithms as presented in chapter 2. Several major correction mechanisms have to be applied before a visualization can produce a comprehensible and accurate representation of the measured scenes.

On the one hand the image data itself in DICOM format has to be reorganized to packages of one single measurement series and the velocity data has to be extracted. These basic series-selection mechanisms are not described in detail since all medical image processing libraries [ITK2006; UserLib2004] provide rather easy handling mechanisms for them.

On the other hand it is not guaranteed that every data point has been measured with accurate parameters. Hence the following sections describe a supplementary adjustment of the predefined v_{enc} values and some simple noise, registration and baseline correction mechanisms before the data gets packed into several intermediate files.

Image 4.1 shows a screenshot of the GUI of the required calculation tool from the Siemens Med 4D-Flow Toolbox. Figure 4.2 illustrates the main structure of the 4D-Flow calculation toolbox. According to this illustration further implementations plug-in at the binary intermediate result files.

4.2.1 Venc Adjustment

To adjust the mapping range for the phase differences given by PC-MRI's phase shifts to velocity values, a maximum velocity value to be expected has to be defined before the measurement starts. Since this definition relies on guessing educated (see section 3.5.1), the velocity mapping can be modified with the *4D-Flow Toolbox calculation tool*.

Assuming a constant velocity range of $[-v_{enc}, v_{enc})$ within an interval of $[-2v_{enc}, 2v_{enc})$ the user can adjust a new velocity range by defining new v_{min} and v_{max} values. The centered interval $[-v_{enc}, v_{enc})$ is then shifted with $\Delta v_{adj} = v_{max} - v_{enc} = v_{enc} + v_{min}$. Consequently, the velocity values acquired from gray values are modified as follows:

If $\Delta v_{adj} \geq 0$

$$\begin{aligned} v_{adj} &= v && \text{for } v \geq v_{min} \\ v_{adj} &= v + 2v_{enc} && \text{for } v < v_{min} \end{aligned}$$

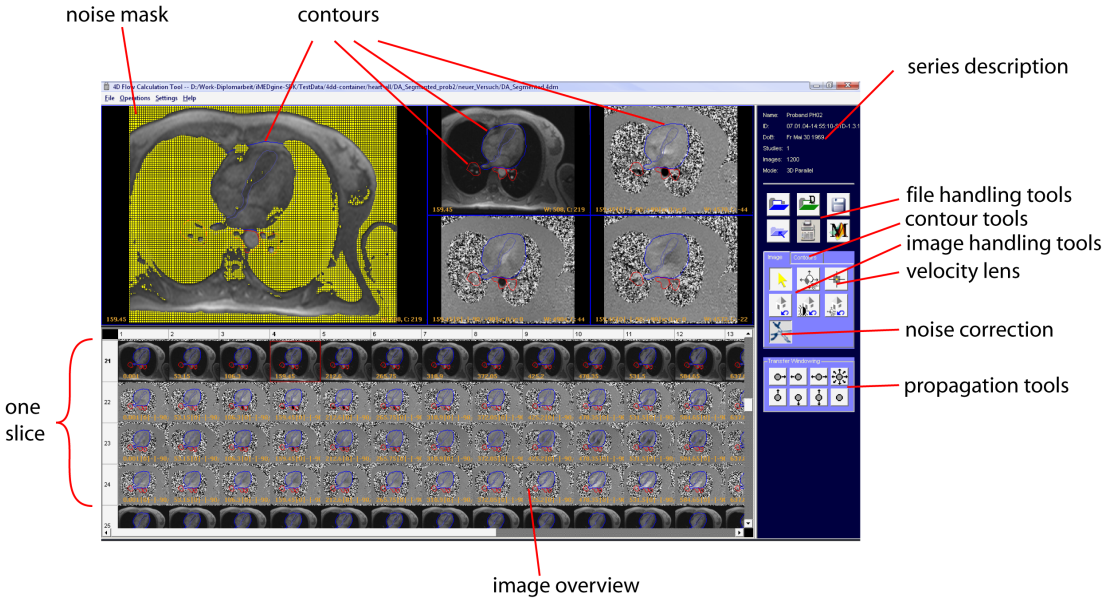


Figure 4.1: A screenshot of the 4D-Flow Toolbox calculation tool. Additional to specialized correction mechanisms several basic contouring and manipulation tools as common for medical datasets are available.

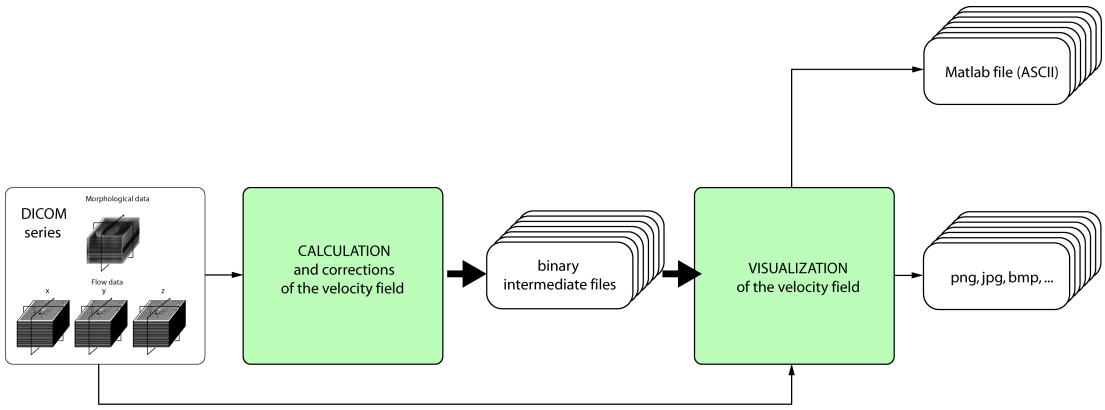


Figure 4.2: This figure refers to the actual 4D-Flow Toolbox [Reiter2006], where a simple visualization tool is already integrated. This tool is able to render hedgehog arrow plots and making screenshots of them. Even a Matlab export is possible. Matlab is a product of **The MathWorks** and can be found at "http://www.mathworks.com/". The visualization environment described in this work uses the intermediate files in the middle and performs visualizations for higher grades.

and if $\Delta v_{adj} < 0$

$$\begin{aligned} v_{adj} &= v && \text{for } v < v_{max} \\ v_{adj} &= v - 2v_{enc} && \text{for } v \geq v_{max} \end{aligned} \tag{4.1}$$

If a v_{enc} adjustment was done, the value v_{adj} is stored in the intermediate file as well.

An automatic v_{enc} correction is possible as well. This assumes a temporal field, since this correction analyzes the temporal course of the velocity values. Figure 4.3 illustrates the typical temporal course of a velocity value with aliasing.

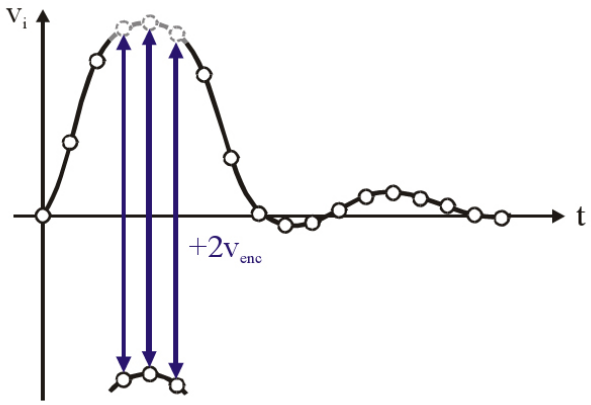


Figure 4.3: A typical temporal course of a velocity value with aliasing effect from [Reiter2007]

By definition aliasing in $v_i(t_n)$ is present if

$$\begin{aligned} v_i(t_r) - v_i(t_{r+1}) &> v_{enc} \\ v_i(t_s) - v_i(t_{s+1}) &< -v_{enc} \end{aligned} \tag{4.2}$$

which applies exactly once. Then a replacement for

$$\begin{aligned} v_i(t_n) \rightarrow v_i(t_n) + 2v_{enc} \text{ for } n = r+1, ..., s && \text{and } r > s \\ v_i(t_n) \rightarrow v_i(t_n) - 2v_{enc} \text{ for } n = s+1, ..., r && \text{and } r < s \end{aligned} \tag{4.3}$$

can be performed. In recent developments, better mechanisms for an automated aliasing correction have been invented [Reiter2007]. However, currently the above described algorithm for automatic v_{enc} adjustment is sufficient for our needs and was therefore used for further data processing.

4.2.2 Noise Suppression

The gray-value intensity of additionally produced morphological images (rephased images) correlates with the amount of entropy in the velocity encoding images (phase-contrast images). Defining a global gray-value threshold produces a binary mask which is used to suppress noisy velocity values. Due to ill defined tissue borders and varying gray-value and noise coherences different combinations of thresholds and conditional thresholds for majority decisions from different images are possible. With these ambiguities a threshold mask can be defined with additional neighboring velocity considerations as

$$Noise\ Mask = \begin{cases} 0 & \text{for pixels with } I \leq T_{abs} \\ c & \text{for pixels with } T_{abs} < I \leq T_{con} \\ 1 & \text{else} \end{cases} \tag{4.4}$$

I defines the actual signal intensity and T_{abs} and T_{con} adjustable threshold values which refer in this case to the pixel intensity value of the morphological images. The value c stands for "conditional" and is defined as uncertainty whose value is given by

$$c = \begin{cases} 0 & \text{for velocity difference to neighborhood pixels} > V_{diff}^{max} \\ 1 & \text{for velocity difference to neighborhood pixels} \leq V_{diff}^{max} \end{cases} \tag{4.5}$$

with V_{diff}^{max} as additional adjustable parameter, defining the maximal allowable velocity difference in a certain neighborhood.

4.2.3 Image Registration and Baseline Correction

The four measured images of one slice at a certain time can differ in three to four pixels at an average, dependent on the sequence and the patient. Hence a correction with a 100x100 ROI image-cross correlation is performed after loading an image series. Certainly these calculated pixel offsets are stored in the 4D-Flow Toolbox intermediate result file. For one slice the pixel translation offset can be estimated as identical for every image So the translational offset has to be calculated only for one image each time. Because of the small prospective translation (about +/- 5 pixel per direction),

a simple subsequent rigid registration is sufficient provided that the 100x100 ROI was set reasonable.

Moreover, the PC-MRI method shows two systematic errors. Additional phase shifts occur because of

- Eddy currents and
- Concomitant gradient fields.

Concomitant gradient fields can be corrected in a low order during image reconstruction with known gradient pulses [Bernstein1998], so they can be neglected further on.

For the remaining error a software based baseline correction can be performed with the 4D Flow Toolbox calculation tool using two different operating modes. A standard derivation correction can be performed in the case of "enough" [1] images and a user defined ROI correction in the other case. Details for these kinds of corrections can be learned from [Kouwenhoven1995; Reiter2006; Partanen1995].

4.3 Intermediate File Format

As so far defined the intermediate result files are an essential part of this work since further visualizations utilize this information. These files are arranged as follows: One, rather small, central information file (master file) can be used to find all other related files and the DICOM database (most likely a DICOMDIR file on a CD or harddisk). For every image slice there is one file containing headers and meta data for all image types of all times followed by drawings and all the vector data (data files).

> Example: 3D parallel mode, 20 slices, 42 times per slice. Assuming that three rephased images per slice and time exist, there is one central information file and 20 slice files. In every slice file there is one general header containing information about the contained data, followed by 42*6 image headers (3 rephased and 3 phase-contrast images per time step) containing image meta data. After that header section there is one section containing the vector data related to those images, followed by the drawings and

[1] "enough" is experientially determinated with three to four images

masks. In the final section size-varying strings, for example image-paths are stored.

A master file always contains global information and values of the complete four-dimensional velocity field in a fixed-length part. Information of the underlying image data and paths to the data files are stored in a string section with variable length. These lengths, and offsets to the strings, are stored in a second part of the master file, which size depends on the number of images and number of slices contained in an examined dataset. The file extension of a master file is ".4dm".

Data files contain information about the slice and meta information for every image of the slice. After a successful calculation of the velocity field the file also contains the corrected vector data of the velocity field. The file extension for data files is ".4dd". A header of such a data file consists of a fixed length part with global values for one image slice, followed by a header for each measured image and its correction values. Subsequently, a small header introduces the calculated velocity fields. After each field the key data for all segmentation regions and noise correction regions are stored.

The detailed binary file definitions are shown in appendix A. The header of the master file is summarized there from table A.1 to table A.2. Subsequently, the data files are defined from table A.3 to table A.8 With these information a reader for this file format can be defined.

4.4 Visualization Framework

The in section 4.2 described data can now be imported and used with different visualization applications. The 4D-Flow Toolbox already provides a tool for that purpose but its capabilities for extension and performance are absolutely restricted. It allows only an arrow plot visualization on separate slices of a volume by scrolling through the time. Even the delay when switching between a volume's slices is rather high. Based on the fact that this viewer was written from scratch without the usage of modern software paradigms, performance considerations or hardware acceleration led to the decision to apply existent powerful visualization libraries with a solid multi-platform medical viewer solution called iMEDgine [iMedgine2006] instead.

The next sections will provide an overview of the used libraries and tools and how they were adapted. Finally the extensions of iMEDgine are defined at a glance. Their

implementation and the developed visualization algorithms are described in chapter 5 in detail.

4.4.1 Tools

Numerous libraries have been used to provide an easy handling of otherwise time-consuming graphics development. Based on an Open Inventor compatible Scene-Graph library called Coin3D (in section 4.4.1.1 and [SystemsInMotion2007]) all visualizations are performed with nodes. Hardware accelerated algorithms were implemented with a shader control library called Shallows (in section 4.4.1.2 and [Shallows2007]) The medical data handling and I/O is done by the popular Insight Segmentation and Registration Toolkit [ITK2006] and some data flow issues are treated with a Scene-Graph based data flow library called Cash-Flow (section 4.4.1.3 and [Kalkusch2006]).

4.4.1.1 Coin3D

As written the Coin3D home page, this library defines "a high-level 3D graphics toolkit for developing cross-platform real-time 3D visualization and visual simulation software". Based on the well known graphics library OpenGL, rendering, interaction and import functions are encapsulated in graph-nodes. The visualization itself is therefore depending on the arrangement of these nodes in a graph. The interested reader is referred to [Hartley1998] for deeper insight into programming with an Open Inventor compatible library like Coin3D. The core data-driven design of Coin3D is illustrated in figure 4.4 and should afford a basic understanding for further definitions.

The power of this library lies in its expandability through the development of own nodes and nodekits and their easy application in scene-describing graph structures with even the use of a specialized scripting language. Since each node provides an interface to basic OpenGL commands and graphics traversal, recently introduced hardware accelerations, volume rendering techniques and data flow abilities can be encapsulated in such nodes and easily utilized in a basic scene graph as provided by each iMEDgine rendering area. Figure 4.6 shows such a scene graph in a descriptive way. The scene graph used for visualizations of cardiac flow data is in parts more similar to the taxonomy shown in figure 2.12 from chapter 2 since always a separate scene graph for morphological data and one for flow visualization algorithms can be assumed.

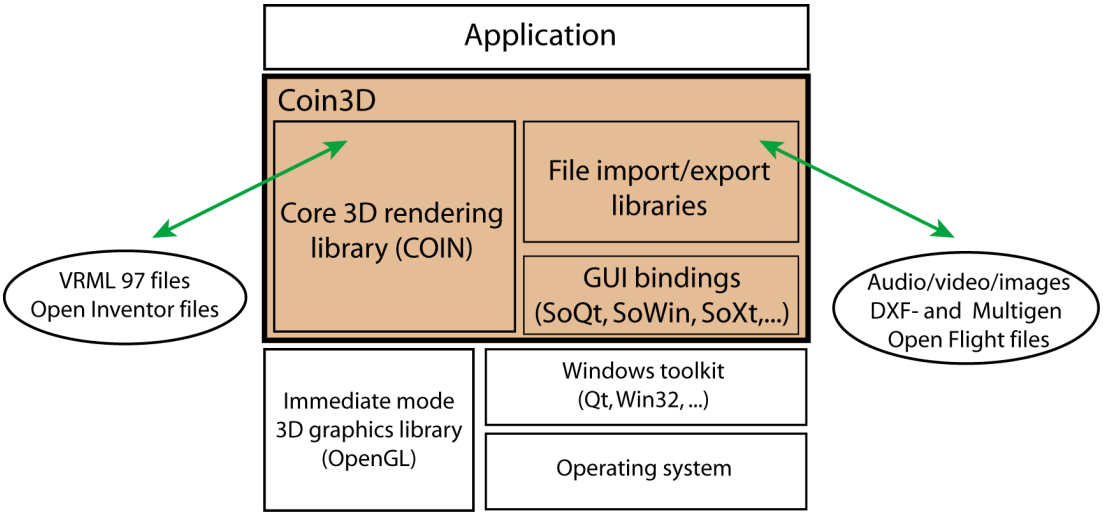


Figure 4.4: The general Coin3D architecture similar to [SystemsInMotion2007]. For compatibility reasons the implementation (chapter 5) uses the parts of Coin3D which are based on *Open Inventor Files* for I/O, *SoQt* and *Qt* for the window system and *OpenGL* as graphics library.

Volume Rendering

Three approaches for a representation of the morphological data are available. Firstly a classical slice based technique as described in section 2.1.1.1 in chapter 2 has been implemented as Inventor node. Its detailed properties are defined in section 5.3.1 in chapter 5. Secondly the Coin3D additional add-on SimVoleon package was used to provide several advanced volume rendering approaches. SimVoleon will be introduced next.

Third iMEDgine integrated a GPU Raycasting algorithm which is based on the principals as described in section 2 in chapter 2.1.1.1. This visualization was tried out and found to be slightly more efficent than the SimVoleon package and would be interesting for future research.

SimVoleon can be used with specialized nodes in the scene graph as usual. Adding a SoVolumeRender node subsequently to an optional SoTransferFunction node and a SoVolumeData node will be the smooth way to render volumetric datasets. As the names already suggest, a transfer function can be defined additionally for a SoTransferFunction node. The correct dimensioned volume data in a definable byte-order has to be stored by a SoVolumeData node. The result of this simple approach will be a

hardware assisted "3D texture mapping" as described in section 2.1.1.1 in chapter 2. If a transfer function is inserted and defined before the SoVolumeRender node, this function will be used for the projection, else a Maximum Intensity Projection or Sum Intensity Projection will result. An example for this approach is shown in figure 4.5, where an empirically defined transfer function was used.

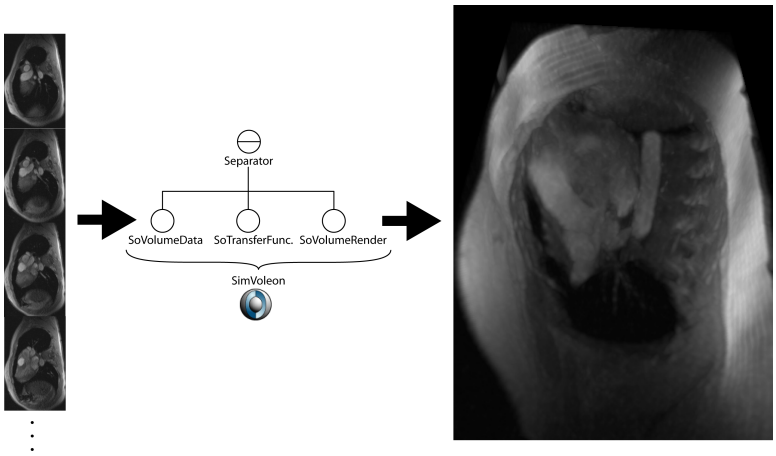


Figure 4.5: The raw image data is stored in a SimVoleon SoVolumeData node and projected with an arbitrary transfer function defined in a SoTransferFunction node. A SoVolumeRender node renders the projection; in this case the volume is slightly turned backward for about 15 degree to provide a better impression for the image on the right side.

4.4.1.2 Shallows

Shallows is a library intended to simplify the usage of GPU shader programs with a special focus on general purpose applications. The requirement of an at least Shader Model 3.0 graphics card was not seen as a reason against using Shallows because of a wide spread of Nvidia 6xxx and ATI 7800 cards or higher. Written as a cross platform C++ layer on top of OpenGL 2.0 this library supports both *Open Graphics Library Shader Language (GLSL)* and *Cg*. The two kinds of shader can be written in one shader file. All commonly known GPGPU programming concepts as described in section 2.2.1 in chapter 2 are defined by objects like the examples in the following list.

- `shallows::OffScreenBuffer(unsigned int width, unsigned int height)` and RenderTexture2D defines an off-screen frame buffer. (Similar for an on-screen render target: `shallows::OnScreenBuffer`.)

- `shallows::RenderTexture2D(texture_ptr tex)` defines textures to store (shader/render) results off screen
- `shallows::Program` and the derived `shallows::GLProgram` define a GLSL shader object. A shader object can be invoked by simple command functions like `...::run()` or `...::activate()` and initialized with the file containing the shader (or an application internal string) by `...::readFile(const char*)` or similar functions.
- Additional input for each shader can be submitted by functions like

  `...::setParamx*(const char *name,...)` [1] and for input textures

  `...::setInputTexture(const char *name, ...)` where `char* name` always refer to a string similar to the variable's/texture's name as defined in a shader as uniform variable; see next paragraph 4.4.1.2 for "uniform variables".

Basically, these four kinds of objects are the main requirements for using any shader instead of an implementation with direct OpenGL commands. Nevertheless, these classes should always be instanced with boost shared pointers [Abrahams2007] to ensure a smart and correct memory management. Certainly there are also additionally more complex functions available for special needs but with this basic equipment many algorithms can be implemented.

GLSL

GLSL is a C-like high level language to perform even complex operation on graphics hardware vertex or fragment processing units. In the next lists only the data types, some important built-in functions and always available values and these values which have to be set necessarily in each program are defined. For using more of GLSL's capabilities the interested reader is referred to [Fernandes2007; Tatarchuk2004]. First of all, listing 4.1 shows the simplest possible shader. Actually, these GLSL-shader does nothing else than passing through values.

Listing 4.1: A very simple vertex and fragment shader in Shallows notation. They will do nothing else than passing through the given input vertices and colorize interpolated fragments between with a given texture.

```
1 [Vertex shader]
2
```

[1] "x*" can refer to 1f, 1i, ... 4fv, 4iv, Matrix4fv as type qualifier.

```
3  void main(void)
4  {
5   //Texture coordinate of the vertex
6   gl_TexCoord[0] = gl_MultiTexCoord0;
7   //Position of the vertex
8   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
9  }
10
11
12 [Fragment shader]
13 uniform sampler2D aTexture; //an arbitrary input texture
14
15 void main(void)
16 {
17  //read the RGBA color of the texture's pixel
18  //at an interpolated texture position
19  vec4 texVec = texture2D(aTexture, vec2(gl_TexCoord[0]));
20  //set the color of the rendered fragment
21  gl_FragColor = texVec;
22 }
```

Available data-types

- **Standard types:** Besides `bool, int` and `float`, also vector types like `vec2, vec3` or `vec4` and matrix types like `mat2, mat3` or `mat4` and similar are available.
- **Textures:** defined with `sampler1D, sampler2D` or `sampler3D` (and even some more for cube- or shadow maps) and accessed via the built-in functions `texture1D, texture2D` or `texture3D...`
- **Arrays:** defined in the same way as in C with square brackets.
- **Type qualifiers:** Referring to the two most frequently used, `uniform` defines a variable set by the OpenGL API and `varying` a variable which is interpolated between vertex and fragment shader and therefore accessible in both.

built-in functions and values

- Vector and matrix types can be accessed element wise via a **swizzle operator**. For example a four dimensional vector accessed with `vector.xyz` or `vector.rgb` will result in the first three elements. (`w` or `a` refers to the fourth component).

The order is arbitrary and recurrences are allowed. (for example `vector.xxwy`). `{s,t,p,q}`-letters are allowed for indexing as well.

- All basic **matrix** and **vector operations** are available for the according types.
- Many standard geometric, trigonometric and exponential functions are available as described in [Fernandes2007]

built-in variables, attributes and constants

The most important of them are listed below. A complete list of them is given in [Tatarchuk2004]

- vertex shader writable variables
  - `vec4 gl_Position`: equals to the homogeneous coordinate position of the vertex and must be written.
- vertex shader readable attributes:
  - `vec4 gl_Color`: color value of the vertex;
  - `vec4 gl_Normal`: normal vector of the vertex;
  - `vec4 gl_Vertex`: coordinate of the vertex;
  - `vec4 gl_MultiTexCoord0..7`: texture coordinates on texture units 0...7.
- fragment shader writable variables:
  - `vec4 gl_FragColor`: must be written and defines the color of the output fragment;
  - `vec4 gl_FragData[0..15]`: refers to the same as `vec4 gl_FragColor` but for multiple render targets.
- fragment shader readable attributes:
  - `vec4 gl_FragCoord`: reads the position of the fragment relative to the window position (x,y,z,1/w);
  - `bool gl_FrontFacing`: reads if the fragment belongs to a front facing primitive.

- fragment shader built-in varyings:
  - `vec4 gl_FrontColor`: interpolated front color set by the surrounding vertices (cp. `vec4 gl_BackColor`; `vec4 gl_Color` is an alias for both and its value depends on the fragments' facing direction.);
  - `vec4 gl_TexCoord[x]`: refers to the interpolated texture coordinate on texture unit x.

4.4.1.3 CashFlow

Cash Flow is based on the scene graph library Coin3D [SystemsInMotion2007] as described in [Kalkusch2005]. The scene graph is extended with data-flow abilities. All attributes used to parametrize the data flow and the visualization pipeline are part of the scene graph as nodes and fields. Thus these attributes can be manipulated interactively inside the scene graph, influencing the dataflow graph [Kalkusch2006].

This framework consists of four types of nodes:

- DataNodes storing the raw data. Node symbol: 

- SelectionNodes defines access patterns to the raw data. Node symbol: 

- GridNode define topological interpretations of the raw data, storing the positions of velocity values. Node symbol: 

- RenderNode creating renderings. Node symbol: 

In contrast to [Kalkusch2006] this work does not perform direct visualization of grids. In the PC-MRI measurements, one dataset is composed of changing velocity data on a static rectangular grid. Hence, one static GridNode for the grid and one DataNode containing the velocity values for all time steps can be used. Decoupling the data for a certain time can be done via flexible Selection nodes.

4.4.2 The Visualization Backbone: iMEDgine

[iMedgine2006] is a medical image viewer and its pure features are mainly 2D visualization with different perspectives, free grouping of perspectives, fading between datasets and 3D visualization. Since this software is intended as viewer for medical volumetric

datasets, the required features to support them - such as slice navigation, zooming, panning, window / level adjustment - are implemented as well. Medical image processing, respectively the administration of morphological data is done with the Insight Segmentation and Registration Toolkit (ITK) [ITK2006] in the background. Figure 4.7 illustrates the isolated iMEDgine viewer GUI with two anatomical example datasets. Each view renders a scene-graph implemented with Coin [SystemsInMotion2007] whose main structure is shown in figure 4.6.

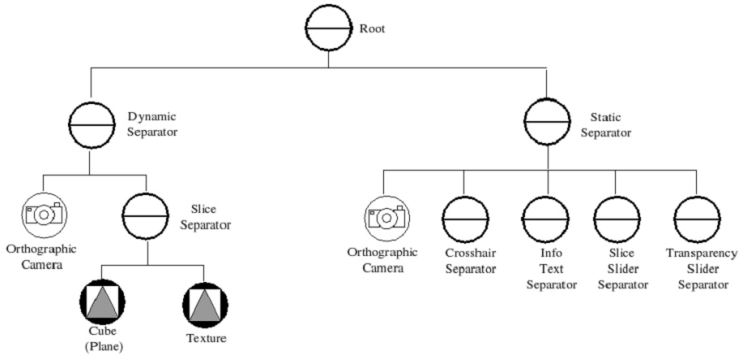


Figure 4.6: The basic scene-graph of iMedgine views. Below the "Dynamic Separator" all the image related nodes (plane, texture, volume rendering...) can be added. The "Static Separator" allows to add an additional information overlay like slice counters, cross hair, DICOM-entries etc. Image from [Streit2006].

In this thesis this basic scene graph is extended with special-purpose separator nodes such as the "Flow-Volume Separator". This allows to separate the pure gray-valued image data visualization from overlaid and embedded flow visualizations. These separators combine both Cash-Flow scene-graphs (see section 4.4.1.3) and further flow visualization nodes, i.e. hardware accelerated ones.

The use of design patterns and the well known library Qt [Trolltech2007] allows to define new datasets and corresponding views and view controlling widgets fast and comfortably. Using Coin3D scene graphs eases the development of new views through the possible scripting of a good portion of new code.

Augmenting the iMEDgine viewer with new dataset classes (DICOM and 4D-Flow Toolbox intermediate file) allowes to handle the 4D-Flow intermediate result file which is described in section 4.2. A new correlated configuration widget enables the user to choose the current temporal position and some other time-depenent adjustments for each dataset for some views. Others can be controlled directly inside the render area.

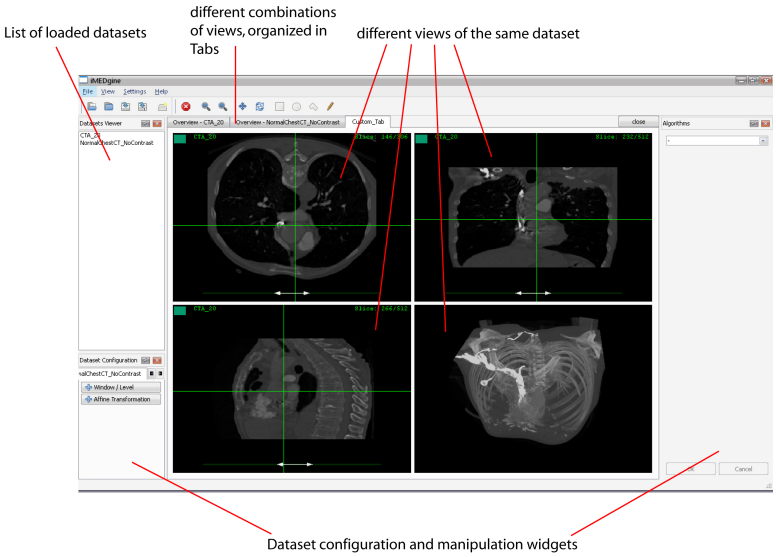
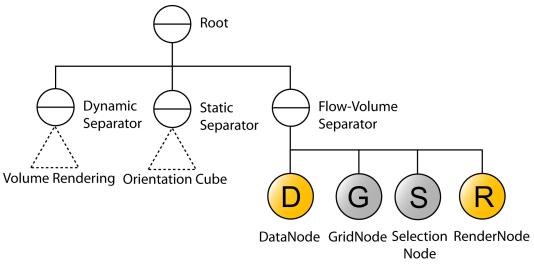
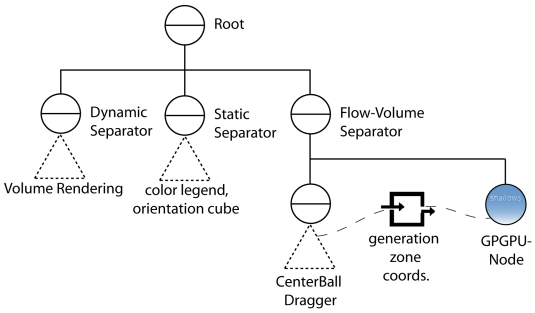


Figure 4.7: This is the standard usage configuration for the iMEDgine viewer. In this example two datasets are loaded. The different combination of views can be arranged by a view configuration dialog and will get organized in tabs. Here are three slice views (axial, sagittal and coronal at upper left, upper right and lower right positions) arranged with a volume rendered view of the dataset. The slice views are built-on two layers. In the foreground there is a green cross hair placed, which indicates the current slices shown in the other views. A dragger below the image controls the current slice and additional information is rendered as text. The latter parts are situated below the "Static Separator" in the iMEDgine basic scene graph. The slice image, respectively the volume renderer is situated below the "Dynamic Separator". Each view accommodates its own scene graph.

In addition to fixed views, an Inventor script-able view for more flexibility has been developed. Furthermore a loading, saving or configuration of view parameters and arrangements has been enabled by a xml-file format definition. Hardware accelerated or Cash-Flow supported views and morphological data representation get strictly separated and therefore completely modular in that way. New views or view-arrangements can be invented on-the-fly due to the scripting abilities of Inventor script files [1] , xml-files and GLSL-shader files. The next chapter (5) will go into detail of these iMEDgine extensions. Figure 4.4.2 gives an overview of the two simplest extended viewer scene graphs, one global Cash-Flow supported visualization and on the other local visualization with GPGPU calculations. Both can be arbitrarily extended, even during the application's run-time.



(a) The basic scene graph for Cash Flow views in iMEDgine. The iMEDgine specific separators remain for image data visualization. CashFlow nodes are arranged below a Flow-Volume Separator.



(b) In this example a Coin3D built-in "Centerball dragger" controls the zone from which a hardware shader starts its calculations (for example a seed region for a particel effect).

Figure 4.8: Two fundamentally extended scene graphs for iMEDgine viewer. (b) is in any order extendable by modifying the underlying Inventor script. Figure (a)'s visualization can be modified by varying the used render node.

[1] Inventor script files are usually denoted with an "*.iv" extension.

Chapter 5

Implementation

This chapter provides an insight into the implementation details of this work. In section 5.1 the used software packages and libraries are presented to meet the requirements from section 4.1. Therefore, figure 5.1 outlines the compilation process of our framework and summarizes all dependencies.

Subsequently, section 5.2 defines the extensions we made for the iMEDgine medical image viewer. The interfaces and structures which are necessary to access DICOM medical data and 4D-Flow Toolbox intermediate files are defined there. Furthermore, the configuration mechanisms and implemented features for different iMEDgine views are explained. In general we present UML-diagrams in these chapter with omitted methods. The corresponding full diagrams are shown in appendix B.

All developed visualization algorithms are defined in section 5.3. There we separate global visualization approaches and morphological background information visualization from hardware accelerated algorithms.

5.1 Software Environment

The main parts of the multi platform Software Development Kit (SDK) which accrued due to the presented requirements is outlined in figure 5.1. This illustration also indicates the whole compilation process with Microsoft Visual Studio 2005 for Microsoft Windows of iMEDgine and our extensions. We used the following software packages and libraries for our visualization framework:

- **CMake 2.4.7:** software configuration to provide multi-platform support,

- **iMEDgine:** medical image volume viewer for analyze and DICOM data [iMedgine2006],
- **Insight Segmentation and Registration Toolkit 2.8.1:** library for medical image data [ITK2006],
- **Qt 4.2:** cross-platform rich client widget development framework [Qt42],
- **OpenGL 2.1:** immediate mode 3D graphics library [SGI2007],
- **Coin3D 2.4.6:** scene graph library [SystemsInMotion2007],
- **SIMVoleon 2.0.2a:** volume rendering extension of Coin3D [SystemsInMotion2007],
- **SoQt 1.4.1:** interface between the Coin3D visualization library and the Qt user interface library [SoQt2007],
- **boost C++ libraries:** consistency libraries ,
- **Shallows 0.97:** GPGPU shader utility library [Shallows2007],
- **Cash-Flow 0.9:** data flow driven scene graph library [Kalkusch2005],
- **Visual Studio 2005:** Windows IDL [VisualStudio2005],

We chose the main programming languages C++ and GLSL. The used operating system is currently Microsoft Windows XP, SP2.

5.2 iMEDgine extensions

In this section, extensions added to the basic iMEDgine viewer and developed shader are described. The following short descriptions reflect the module's main tasks. The new iMEDgine modules at a glance are:

- Datasets for 4 dimensional flow- and DICOM-image data; inevitably also an interface for DICOM volumetric and flow data I/O;
- two new viewer-types; one for Cash-Flow based scene-graphs and one free definable, mainly designed for hardware accelerated visualization nodes;
- a node which serves as interface between independent viewer scene graphs and the data holded by iMEDgine;

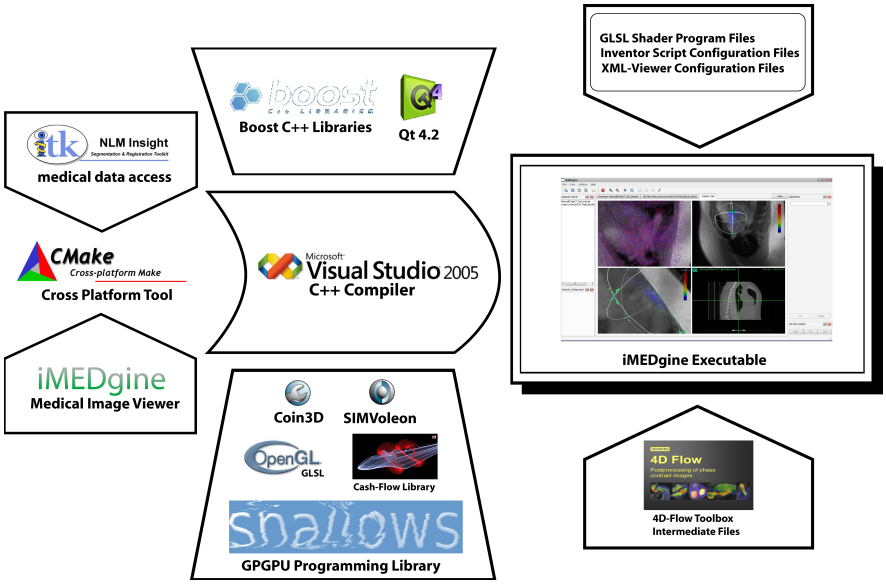


Figure 5.1: This figure gives and overview of the iMEDgine Software Development Kit (iMEDgine-SDK). The cross platform development tool *CMake* generates all necessary build files for the Insight Segmentation and Registration Toolkit and the iMEDgine framework which are required for compiling the framework with Microsoft Visual Studio 2005. Compiled with the required software libraries: boost, Qt and SoQt, Coin3D and SIMVoleon, OpenGL, Cash-Flow and Shallows, the resulting iMEDgine executable is able to process Siemens 4D-Flow Toolbox intermediate files and our own configuration and shader program files.

- several GUI extensions like controls for the dataset's I/O and a dialog for editing a color gradient. Furthermore, saving and loading abilities for view parameters are implemented.

5.2.1 Datasets

Supplying an iMEDgine-design conform reading of flow data files and additional morphological DICOM data requires new datasets and corresponding data interfaces. The simplified UML diagram in figure 5.2 shows these datasets and its inheritances. The complete class diagram can be found in appendix B.

This dataset uses a file reader related to one used in the 4D-flow toolbox. It represents the main interface to *.4dm and *.4dd files, which does not mean, that it can read them automatically. The actual reading and organizing of the files in adequate data structures has to be done separately with a flow data I/O interface. The scheme

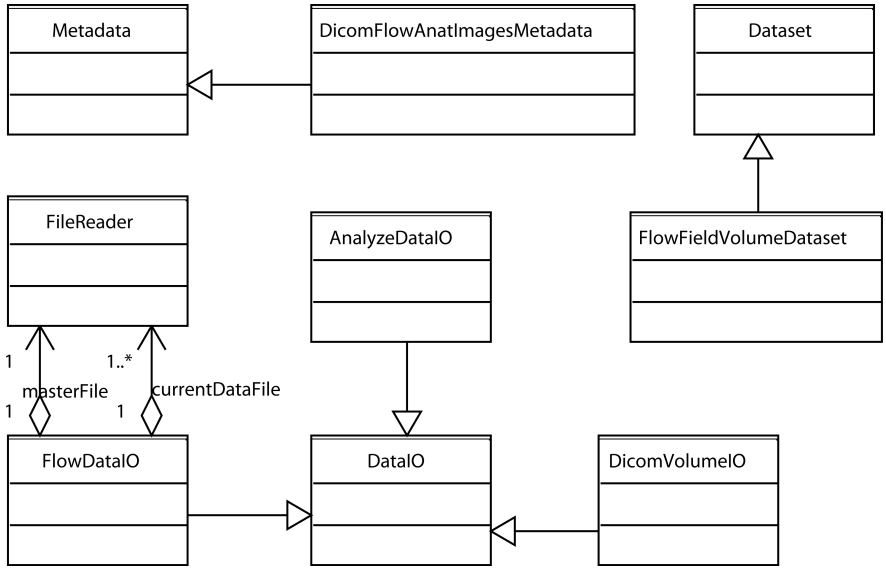


Figure 5.2: A dataset for flow data and its interfaces and inheritances as simplified UML class diagram. For the sake of clarity, all methods are cut in this representation. The full diagram can be found in figure B.1.

as presented in tables A.1 to A.9 in chapter 4 will be traversed with this class. Unfortunately, the flow data and related images are organized in temporal order instead of a required volumetric order. Thus a reorganization before storing them in a dataset class is unavoidable and may take some time. The DICOM image processing is done with the help of given paths to the images in the *.4dd files [1]. For each volume image stack a new iMEDgine DICOM volume image dataset will be read by its own DICOM volume I/O and made available by the flow field volume dataset in a dataset vector-based structure. Note that this effort only applies for anatomical (rephased) volumetric image stacks; the flow data itself is, as already described, preprocessed and numerically stored in *.4dd files. The reorganization to a volumetric order can be already done by the read strings containing the paths to the images. Since each temporal volume consists of exactly the same number of slice images in the same order, this task can be solved accordingly. The first image paths in the image file section (table A.4) of each *.4dd file refers always to the first volume and so on. The DICOM volume I/O itself utilizes the Insight Segmentation and Registration Toolkit's [ITK2006] GDCM library

[1] If the data cease to exist at the given lo6cation, an alternative path but containing the same data will be asked for.

which eases the access of the image data itself with variable *window* and *level* settings. Consequently, the image volumes returned by these datasets will already be mapped correctly by passing the related values continuously.

The flow vector field volumes are finally stored in a number of time steps sized vector structure as an array of float values. Each field is of constant size and organized as all *rows* of a slice subsequently stored for the whole volume. This enables both libraries, Cash-Flow and Shallows, to deal with this data according to the additionally available *field width, height and depth* information.

5.2.2 Viewer

Viewers represent an essential part of iMEDgine. These are render area widgets which can be arranged arbitrarily inside tabs and equipped with any kind of visualization. However, after experimenting with several configurations utilizing even one view (class) for one visualization a better solution turned out: The development of only two different kinds of iMEDgine-views was adequate. The first one uses Cash-Flow, supports its data-flow architecture and provides its classes (and nodes). The drawback with this view is, that an intern data-flow between the iMEDgine architecture and the Cash-Flow data nodes is only possible within the view itself. That means, that the dataset class has to prepare all Cash-Flow "Virtual Arrays" [1] in advance before Cash-Flow is able to deal with them. Furthermore, Cash-Flow is rather designed for the visualization of different grids than of flow data, so this view class has to administer a parallel data flow of grid-positions and flow-data positions at the same time. Nevertheless, the Cash-Flow library proved its value, with some adjustments for global field overview visualizations and grid supporting views due to its abilities for reconfigurable data access with configurable selection nodes.

A view to administer different independent hardware accelerated visualization nodes or arbitrary scene graphs is additional available. The goal is to enable a configuration not only with c++ code and inevitably recompiling the whole application but with a separate Open Inventor .iv-file. This allows changes of the scene even during run-time and arbitrary combinations of visualizations in concurrent views. Moreover handling,

[1] *"virtual arrays"* are specialized Cash-Flow arrays which define arrays within an actual array in memory. The interface to these arrays are provided by data access nodes respectively selection nodes. [Kalkusch2005]

debugging and testing of new visualization nodes becomes rather fast with these abilities. By the additional development of a "ControlDragger"' node-kit, which is actually nothing more than an 1D Dragger, definable for a certain range (for example discreet movement of the dragger between 0 and 5 in steps of 0.1), the complete controlling requirements can be defined within a view and its corresponding .iv-file. Nonetheless, this feature gets first important for a full screen or stereo projection of the scene. The thinned out UML diagram for these implemented viewer classes is presented in figure 5.3.

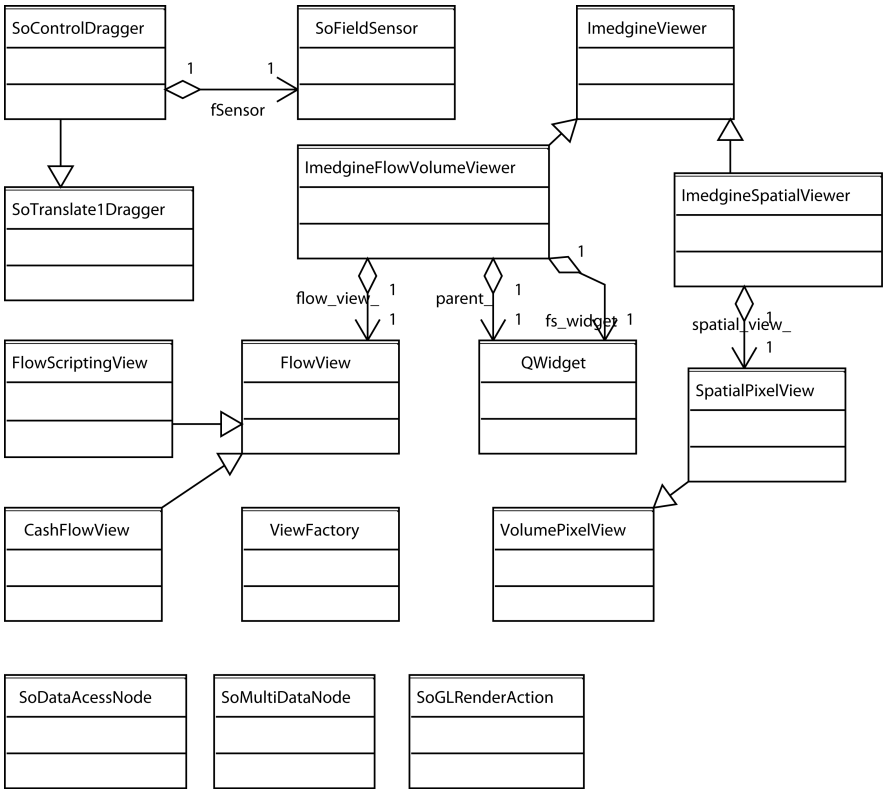


Figure 5.3: UML class diagram for viewer classes with special interest in flow scripting views and Cash-Flow views. For the sake of clarity, all methods are cut in this representation. The complete class diagram can be found in figure B.2.

Flow data viewer provide an interface between their scene graphs and the iMEDgine datasets. Each of them add a so called "View Parameter" node to the scene graph in advance, so that the flow and DICOM data can be accessed via this node by other nodes defined for example in an .iv-script. While initially this was thought as a temporary

solution further development showed, that this kind of data access is one of the most beneficial because of its speed. Section 5.2.2.2 provides a more detailed insight into this node.

5.2.2.1 Scriptable View

Since the view class for GPGPU nodes and arbitrary scripting is used more frequently than the Cash-Flow view its features will be outlined here. The view itself, following the iMEDgine design conventions, is a widget contained by an *iMEDgine-viewer*. This enables a division between rendering parts and Qt-GUI parts. However, the user will not distinguish between them when using a view. Consequently the view(er)s behavior can be summarized as follows:

- **3D-Navigation:** The navigation inside a viewer (zoom, pan, manipulating) is the same as used with standard [SystemsInMotion2007] examiner viewer (*SoQtExaminerViewer*). 'ESC' toggles between the manipulation and view mode, so that active objects like draggers can be moved. All additional features of a context menu can be accessed via a right click into the viewer.

- **view-arrangement:** Arranging different views side-by-side can be done as with all other iMEDgine views. Figure 4.7 shows such an order for standard morphological data and the usage of these concurrent views. For stereo environments the usage of only one view but divided into different tabs is recommended. Certainly, combinations of existing iMEDgine morphological data viewer and flow visualizations were particularly intended.

- **color mapping adjustment:** The mapping of color to an arbitrary view parameter can be performed by using a built-in color gradient editor. The resulting color gradient is stored in the always present view-parameter node and then be accessed by any other node. The detailed description of this dialog can be found in section 5.2.3.2.

- **altering the scene graph:** The desired scene graph (.iv-file) can be chosen with a file-selection dialog which can be found in the viewer's context menu.

- **fullscreen mode:** This feature toggles the selected view in full screen mode.

- **stereo mode:** In the case of an available Quad-Buffer graphics card, this feature enables a pure stereo view with the according equipment (for example shutter-glasses). Since there are no head-tracking interfaces implemented yet this feature can be seen as a preparation for future work as described in chapter 7.

- **control draggers:** To provide an interface for a possible use of the application in a stereo viewing environment and in full screen mode special control draggers have been developed. Figure 5.4 illustrates their application. The reason to relocate the controls into a view itself is to avoid interferences from the Qt-Interface in these view nodes.

Nevertheless, the main controlling and rendering part of a view is the selection of an applicable scene graph. Such a scene graph script has to fulfill some few constraints to be iMEDgine compliant. A script code example for such an applicable .iv-file is given in listing 5.1.

Listing 5.1: A simple .iv-file containing an applicable scene graph for a Flow-scripting-view. This example shows the usage of a stream-tube renderer with additional slice-wise rendering of the anatomical image data. Several control units are described inside. The result of this configuration is shown in figure 5.4. In general all GPGPU visualization nodes provide the same configuration fields as the one used in this listing.

```
#// This camera shows the 3D content of the scene
#// and is therefore movable and rotateable.
DEF dynamicCam SoPerspectiveCamera {
        viewportMapping ADJUST_CAMERA
        position 0 0 506.639
        orientation 0 0 1  0
        nearDistance 1
        farDistance 1000
        aspectRatio 1
        focalDistance 549.737
        heightAngle 0.78539819
}

#// This defines the interface to the iMEDgine dataset
#// and enables an easy loading and saving of view paramter.
#// SoViewParamter has to be defined correctly if the
#// visualization nodes are marked as iMEDgine view related.
#// This node gets filled by the iMEDgine framework.
DEF params SoViewParameter
        {
                name "ViewParameter"
                cam_focal_distance = USE dynamicCam.focalDistance
```

```
23                  cam_position = USE dynamicCam.position
24                  cam_rotation = USE dynamicCam.orientation
25          }
26
27  #// This camera shows the controlling parts of the scene.
28  DEF staticCam SoOrthographicCamera {
29                  focalDistance 10
30                  nearDistance 0.5
31                  farDistance 10
32                  viewportMapping LEAVE_ALONE
33  }
34
35  #// This subgraph is respensable for all controlling
36  #// parts of the scene
37  SoSeparator {
38  USE params
39  USE staticCam
40
41  SoSeparator {
42  SoTransform {
43          scaleFactor 0.1 0.1 0.1
44          translation 0 -0.90 0.3
45  }
46
47  DEF volume_control SoControlDragger {
48                     minmax 6.0
49                     range = USE params.num_volumes
50                     translation -6 0 0
51                     }
52
53  #// [...] following some more SoControlDragger
54
55  #// This shows a custumized color gradient if available.
56  SoSeparator {
57  Transform{
58          translation -0.98 0.77 0.3
59          #scaleFactor 1.0 10.0 1.0
60          rotation 0.0 1.0 0.0 1.57
61  }
62  SoImage{
63          image = USE params.color_mapping_image
64          height 80
65          width  15
66  }
67  }
68  }
69
70  #// The main scene graph contains the complete scene.
71  SoSeparator {
```

```
72
73  USE params
74
75  USE dynamicCam
76
77  #// A Centerball Dragger defines the seed-region
78  #// for the visualization
79  SoSeparator {
80  Transform{scaleFactor 50.0 50.0 50.0}
81  DEF genZone SoCenterballDragger{}
82  }
83
84     #// A GPGPU node which takes the former defined controlling
85     #// input. Note that this node is defined as iMEDgine node
86     #// since it is used with an iMEDgine dataset interface.
87       SoPPTubletStreamlineNode
88        {
89          shader_directory = USE params.shader_path
90          field_dimensions = USE params.field_dimensions
91          color_mapping_tex = USE params.color_mapping_texture
92          field_data = USE params.field_data
93          show_debug_tex 0
94          line_length 1024
95          num_lines 1024
96          iMEDgine_node 1
97          which_volume = Calculator {
98                       a 0.0 = USE volume_control.int_position
99                       expression [ "oa=a-1" ] } . oa
100            seat_region = USE genZone.center
101            seed_region_rotation = USE genZone.rotation
102            cam_position = USE dynamicCam.position
103            seed_region_width = USE width_control.float_position
104         seed_region_height = USE height_control.float_position
105         seed_region_depth = USE depth_control.float_position
106         cam_rotation = USE dynamicCam.orientation
107    }
108
109 #// Finally the simple anatomical slice renderer defined
110 #// in a separate file. Note that defining control units
111 #// with the names slice_control window_control,
112 #// volume_control, level_control and
113 #// alpha_control is necessary to use
114 #// that renderer in such a way.
115 Separator{
116         File { name "slice_renderer.iv" }
117 }
118 }
```

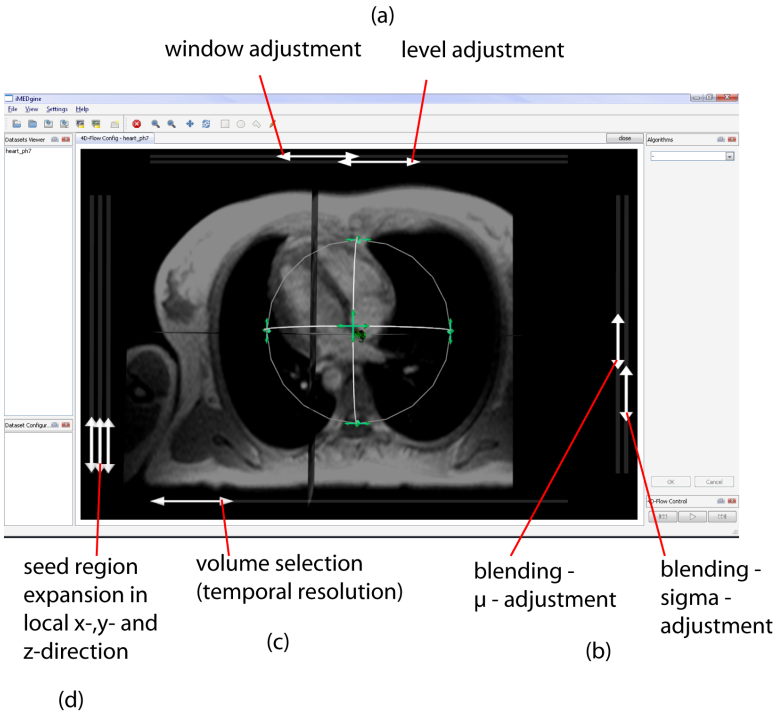


Figure 5.4: A example for the usage of control draggers. From the top clockwise the draggers around the actual image are: (a) two sliders to adjust the window and the level of the basic DICOM morphologic data which is rendered in the background. (b) Two sliders to adjust the alpha-blending of the slice-planes. (c) a slider for switching through the different volumes (time-steps). (d) another three slider to adjust the size of the seed-region of particles or lines in each direction.

5.2.2.2 View Parameter Node

This node represents an essential interface to the iMEDgine framework, so it is described separately in this section. Values are set after loading a flow dataset, so for a new view only a reference to this node has to be passed. This node can even be exported in a later described xml-format. Consequently, it is possible to restore a former workspace completely for each view, see section 5.2.3.1. The available parameters are listed and described below.

- `SoSFInt32 view_type:` Indicates the (iMEDgine) type of view. For example a scripting view is associated with "10". (These numbers are internal mapped to iMEDgine enum values).
- `SoSFVec3f cam_position`, `SoSFRotation cam_rotation`,

  `SoSFFloat cam_focal_distance:` stores the position, rotation and distance from the origin of the scene to the camera. These values are necessary to store a current perspective.
- `SoMFInt32 manipulator_positions:` a list of positions for certain seed regions. ...
- `SoSFInt32 time_position`, `SoSFInt32 num_volumes:` stores current temporal position and the overall available number of volumes.
- `SoSFString primary_dataset_path`, `SoSFString primary_dataset_key:` restoring a former used dataset in the case of loading a view configuration require these values. The key value equals the file name of the 4DFlow Toolbox master file.
- `SoMFFloat color_mapping_texture`, `SoSFImage color_mapping_image`,

  `SoMFVec4f gradient_editor_points`, `SoMFFloat gradient_points:` are all necessary parameter for color mapping: the 1D gradient texture, an image for convenience and all parameter to restore the gradient editor.
- `SoSFString field_path:` is the path to the original used field data. This is only needed for reloading a viewer configuration.
- `SoSFDataset field_data:` the interface to the raw velocity field data. The field `SoSFDataset` was developed to provide the data both as `SoMFFloat` and as reference to shallows GPGPU textures if they once have been generated. Storing the references in this node is necessary and efficient because otherwise the velocity field textures would have to be reassembled with each new view or visualization. The `SoMFFloat` field on the other hand can be accessed by Cash-Flow virtual arrays and further libraries directly. Since each view only gets a shallow copy of the data and this node, memory requirements are optimized.
- `SoSFVec3s field_dimensions:` a 3D vector, storing the field's width, height and depth.

- `SoSFString shader_path:` the path to a directory containing the shader files. This is necessary since only iMEDgine initially knows where they are, Coin3D does not. This directory is defined among others at the very first start of the application and stored in a global ini-file.
- `SoSFString scene_graph_path:` the path to the scene-graph Open Inventor script if the accessing view type is defined as script-able one.

5.2.3 User Interface Extensions

Figure 5.5 gives an overview of additional buttons and behaviors of the iMEDgine framework. The following section (5.2.3.1) defines the xml-file format for the saving and loading of view configurations. Finally section 5.2.3.2 outlines the implementation details of the developted color gradient editor.

5.2.3.1 Exporting/Importing of View Parameter

The export of a current workspace and all set parameter, view points and modifications is technically simple but powerful in routine. Even marking interesting features of the dataset by simple saving the view best showing them enables a hypothetic operator to explore and explain the data quickly. In general the most important configuration parameters for each view are the position of the camera viewing the scene, possible seed region positions and rotations, an optional defined color mapping gradient and several additional view-specific values. Since the resulting configuration file of such a snapshot is generated in XML (DOM), new view configurations can be scripted by hand. The code in 5.2 shows for example the configuration exported from figure 5.5. To keep the XML-code short no further manipulation have been applied after arranging the views. Hence all of the parameters show default values except the camera positions and time positions.

To simplify the behavior of iMEDgine directly after loading a dataset a default xml view configuration is loaded for the first overview. This provides the possibility to customize the dataset overview tab and enables a fast development of new datasets with initial overviews meeting exactly their requirements.

Listing 5.2: A viewer configuration file exported from the view arrangement from figure 5.5. Only the camera positions and time positions were varied so most parameter show default values. Note that not all parameter from the view parameter node 5.2.2.2 have to be exported to restore a complete view configuration tab. The remaining ones can be restored by using the here defined.

```
1 <!DOCTYPE ViewConfiguration>
2 <Views>
3  <View ViewType="10" > <!--//Scripting view-->
4   <Camera RA="2.98906" X="142.953"
5                        Y="146.915"
```

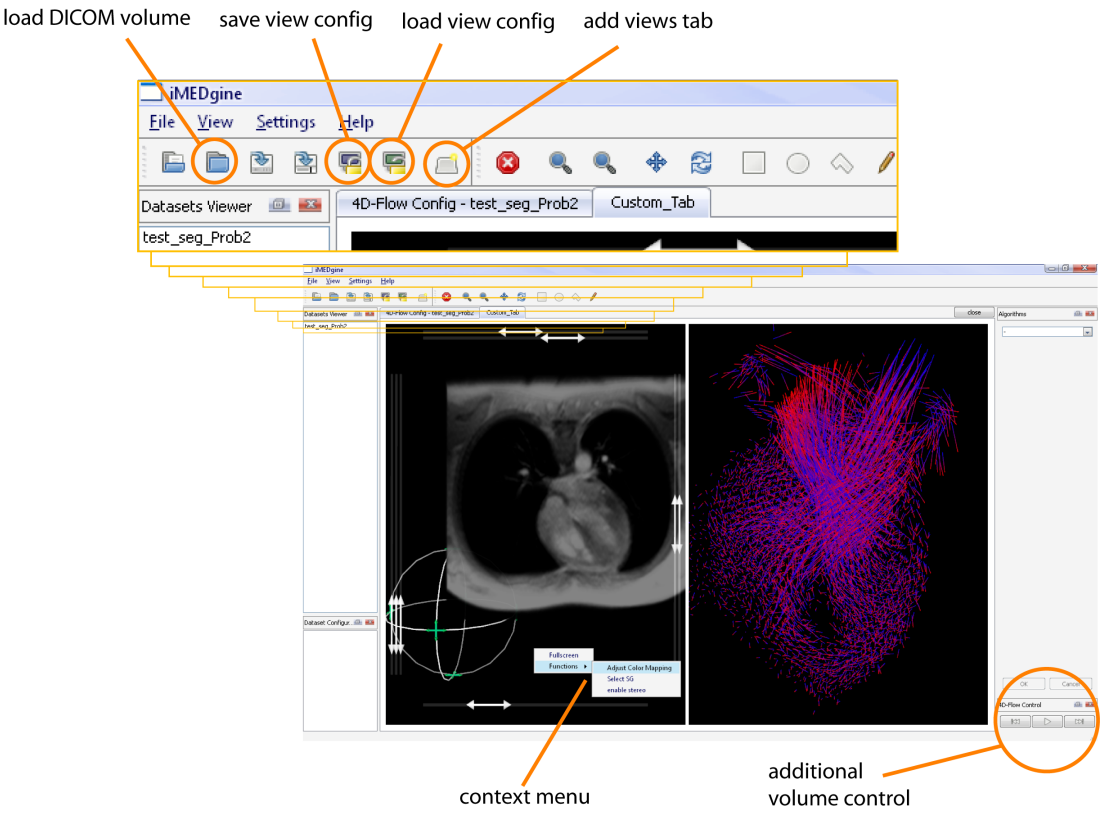


Figure 5.5: The additional implemented GUI features of iMEDgine. The image shows on the left side a scripting view with several control dragger and one seed region dragger with a slice based rendering of the anatomical volume image data described in section 5.2.2.1. The view on the right side shows a Cash-Flow scene graph with thinned out glyphs located at every tenth grid position. The current volume can be switched either with a control dragger inside a view on the additional available volume control widget on the right side. The context menu is activated with a right click in a render area of a view.

```
                        Z="-238.124"
                        RX="0.998504"
                        RY="-0.00809212"
                        RZ="0.0540836"
                        FD="272.991" />
  <Manipulators count="1" >
   <Manipulator X="0" Y="0" Z="0"
                RX="0" RY="0" RZ="0" Type="0" />
  </Manipulators>
  <ViewData>
   <TimePosition Time="2" />
   <BaseDatasetKey Key="test_seg_Prob2" />
   <BaseDatasetPath Path="/*long␣string*/" />
   <ScriptFile Path="/*long␣string*/" />
   <ColorMapping>
    <GradientStops>
     <GradientStop EditorPointR="0"
                   EditorPointA="1"
                   EditorPointB="1"
                   EditorPointG="0"
                   Point="0" />
     <GradientStop EditorPointR="0"
                   EditorPointA="1"
                   EditorPointB="0"
                   EditorPointG="1"
                   Point="0.5" />
     <GradientStop EditorPointR="1"
                   EditorPointA="1"
                   EditorPointB="0"
                   EditorPointG="0"
                   Point="1" />
    </GradientStops>
   </ColorMapping>
  </ViewData>
 </View>
 <View ViewType="7" > <!-- //Cash-Flow glyph view -->
  <Camera RA="3.14159" X="172.329"
                       Y="104.688"
                       Z="-510.047"
                       RX="0.998504"
                       RY="-0.00809212"
                       RZ="0.0540836"
                       FD="549.737" />
   <Manipulators count="0" >
    <Manipulator X="0" Y="0" Z="0"
                 RX="0" RY="0" RZ="0" Type="0" />
   </Manipulators>
   <ViewData>
    <TimePosition Time="2" />
```

```
55      <BaseDatasetKey Key="test_seg_Prob2" />
56      <BaseDatasetPath Path="/*long␣string*/" />
57      <ScriptFile Path="" />
58      <ColorMapping>
59       <GradientStops>
60        <GradientStop EditorPointR="0"
61                      EditorPointA="1"
62                      EditorPointB="1"
63                      EditorPointG="0"
64                      Point="0" />
65        <GradientStop EditorPointR="0"
66                      EditorPointA="1"
67                      EditorPointB="0"
68                      EditorPointG="1"
69                      Point="0.5" />
70        <GradientStop EditorPointR="1"
71                      EditorPointA="1"
72                      EditorPointB="0"
73                      EditorPointG="0"
74                      Point="1" />
75       </GradientStops>
76      </ColorMapping>
77     </ViewData>
78   </View>
79 </Views>
```

5.2.3.2 Color Gradient Editor

As already mentioned in section 2.1.1.2 in chapter 2 the mapping of color to an arbitrary flow parameter is an essential task for flow visualization. To define a color gradient in an intuitive and easy way a Qt 4.2 demo application [QGradientDemo2007] for gradient editing was adapted as modular dialog. The UML class diagram of the underlying widgets is shown in figure 5.6. Figure 5.7 shows the dialog and describes its usage in shortly. The output of the editor is an 1D-array of RGBA color values and corresponding supporting points for the resulting color-value function (`QGradientStops`).

5.3 Visualization nodes and subgraphs

Rather independent from the iMEDgine framework, several new shape nodes for visualizations and corresponding calculations are implemented or utilized from the Cash-Flow library. These nodes and their visualization algorithms are described in this section in detail. Section 5.3.1 and 5.3.2 implement visualizations providing a static overview of a volume. Section 5.3.3 and the following subsections illuminate dynamic visualizations based on calculations done on graphics stream processors. Their algorithms are illustrated in more detail since these methods offer a higher complexity.

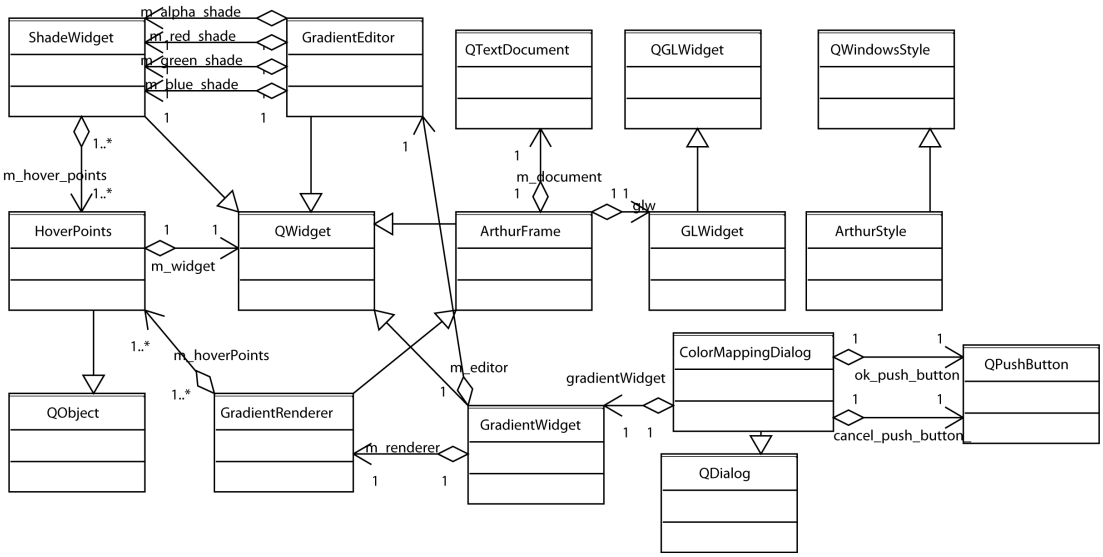


Figure 5.6: An thinned out UML class diagram showing the underlying widgets of the Color gradient editor dialog. For the sake of clarity, all methods are cut in this representation. The full diagram can be found in figure B.3.

All visualization nodes which are additionally implemented for this work can even be used with or without iMEDgine. The difference makes a boolean field called `iMEDgineNode`. If this field is defined as true a `SoViewParameter` has to be available in the scene graph so the node can get its field data, image data and other data from this parameter-node. Otherwise the programmer will have to fill an additional field `fieldData` with flow- or image data, depending on the desired visualization. An overview of those nodes is given in figure 5.8.

5.3.1 Morphological image-volume rendering

In the focus of this thesis anatomical data is only a supporting but important visualization to provide a spacial correlation of the presented flow visualizations. In general all volume rendering algorithms described in chapter 2 are now available within the iMEDgine framework, but as announced in section 2.1.1.1 the simplest method has empirically prove to be best. This technique renders slices on top of each other with different transparency values. However, without appropriate possibilities to adjust a (blending) centered slice and the transparency values even this method is not feasible. For that reason a SoShape node was implemented which does the rendering of the slices but enables the adjustment of an alpha-blending function and the translation of its peak.

The slices are shown each with different transparency values following a Gauss distribution. The position of the peak can be adjusted with the μ-value of the Gauss function, which refers to a movement of the brightest slice (or a single slice if the slope

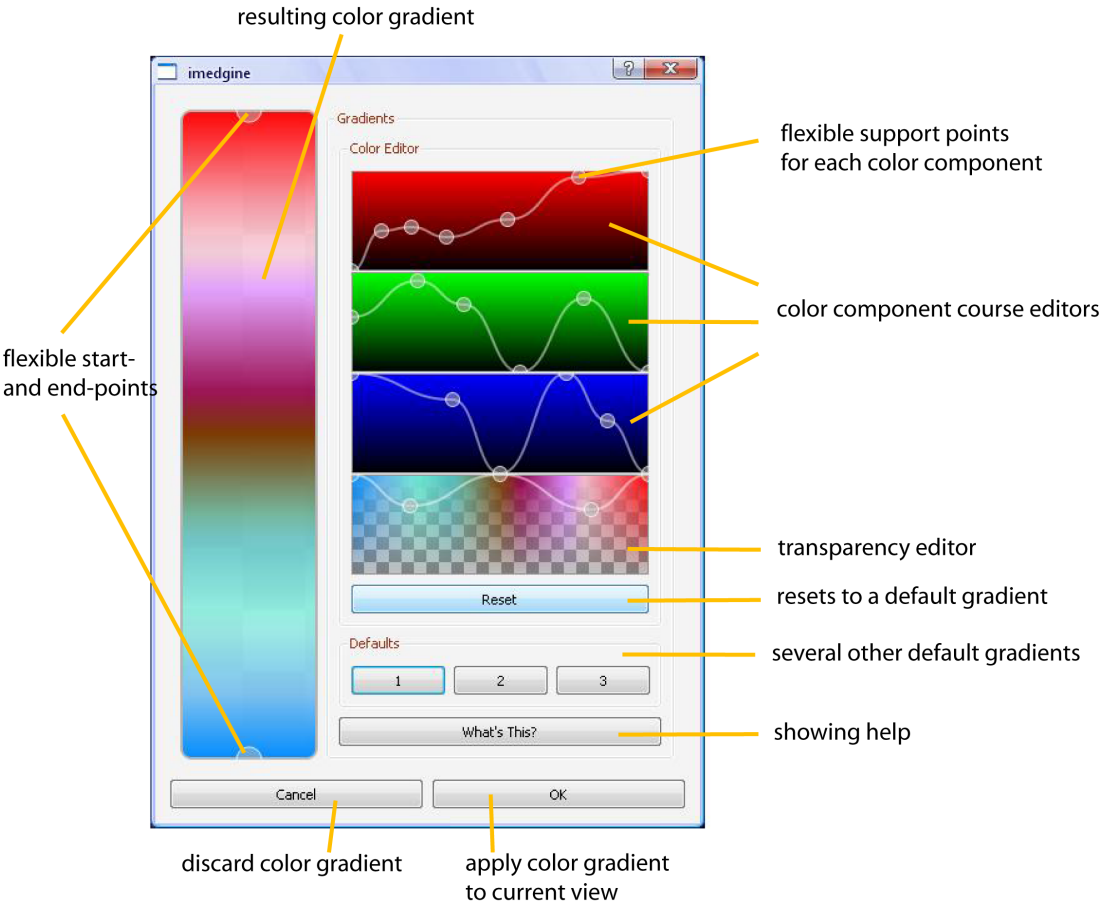


Figure 5.7: With the color gradient editor it is possible to define arbitrary supporting points for each color-component's mapping function and an optional transparency function. By applying a new gradient to a view, this gradient will serve as lookup table for a defined parameter. Else a default gradient is used. Flexible supporting points refer to the class "HoverPoints" as denoted in figure 5.6.

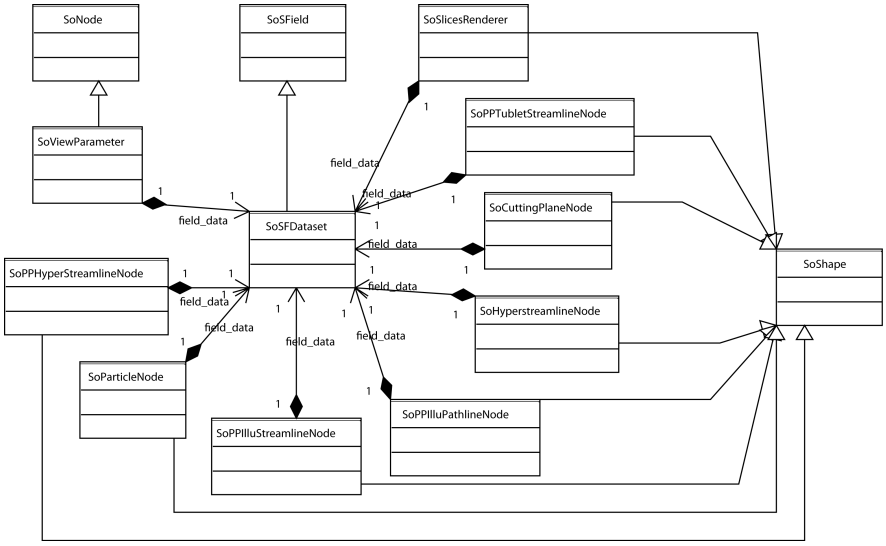


Figure 5.8: This UML class diagram gives an overview of the additional implemented visualization nodes. The full diagram can be found in figure B.4 and B.5.

is high enough). The slope of the distribution can be modified with the σ-value of the Gauss set phrase, which accords to the amount of neighboring slices seen. This principle is outlined in figure 5.9. Figure 5.3.1 illustrates this behavior and figure 5.4 describes the modification of μ and σ with control draggers.

5.3.2 Cash-Flow View

The Cash-Flow views are essentially based on the scene graph shown in figure 4.4.2. Since each volume field dataset is arranged on the same grid, a constant grid node can be defined to organize the data. Consequently, a Cash-Flow virtual array used with the data node is at the number of temporal volumes larger than the array of the grid node itself. To provide four dimensional support only the `offset` of the responsible selection has to be altered. Unfortunately, render nodes provided by the library, did not fit to the presented data and were occasionally slow since all calculations were performed on the CPU. That is the reason why only two rendering techniques performed well.

Firstly the rendering of the grid and subparts of the grid in arbitrary order is very useful to evaluate the alignment of the flow data with the image data. The grid is rendered with this technique as small green points. Subsequently visualizing subparts with Cash-Flow is easy since only the increment (`inc`) value has to be modified. The view supporting this technique got the name "grid supporting overview".

Secondly a velocity field overview with glyphs could be realized with small modifications of the Cash-Flow glyph render node. The right view rendered in figure 5.5 shows such a glyph view, using every tenth grid position. The glyphs itself are lines which are constantly colored from blue to red and a length corresponding to a tenth of the given

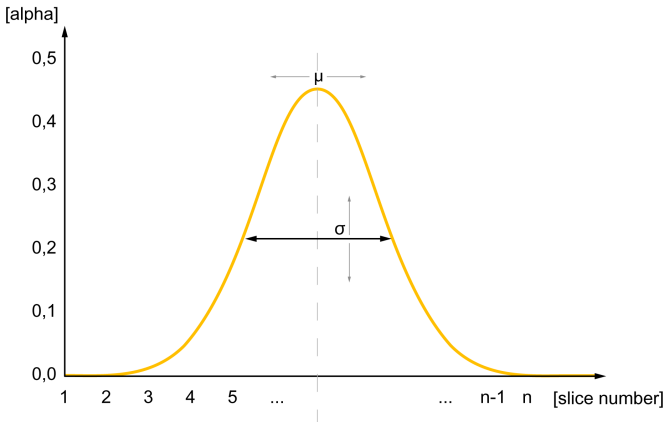


Figure 5.9: Increasing or decreasing the μ-value will result in a movement of the peak to the right or to the left. The σ-value determines the slope or the dilatation of the curve. The area under the curve must remain constant while modifying these parameters. The abscissa is defined by the measured slices. Consequently, with this curve, corresponding alpha values can be mapped to each anatomical image so that the transparency is decreasing from the focused slice outwards.

velocity value in the given direction. The constant color enables the user to get a fast and clear overview of the currently segmented volume part's flow conditions. Since no further parameter or seed regions need to be defined for this view, it was asserted to be the most intuitive usable flow-view of this work.

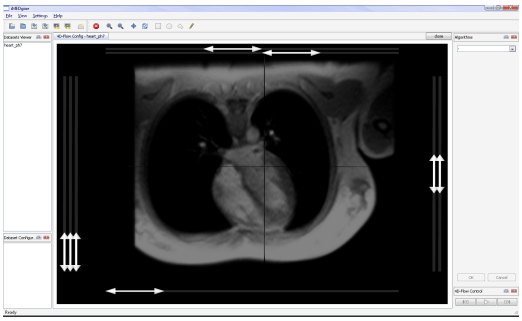
Due to the described inconveniences with dynamic visualizations and in some sense unsuitable render nodes, the focus of the implementation turned subsequently to hardware accelerated options as described in the next section 5.3.3. However more ideas how to use a data flow driven scene graph library for cardiac MRI flow data are theoretically presented in [Kainz2006] and may be partially seen as future work.

5.3.3 GPGPU Visualizations

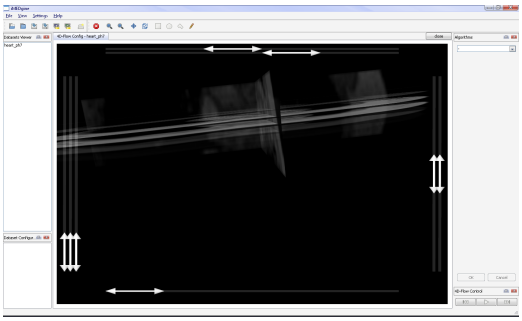
As mentioned in the introduction to this section, all of the further described nodes can either be used in a new application with additionally defined vector fields or within the iMEDgine framework, utilizing the built-in interface node (SoViewParameter) as already described for flow and image data. 2D textures are interpolated bilinear on all hardware architectures per default whereas 3D-textures are interpolated trilinear only on architectures with plenty of resources. On other hardware a standard nearest-neighbor texture lookup is performed.

5.3.3.1 Integration methods

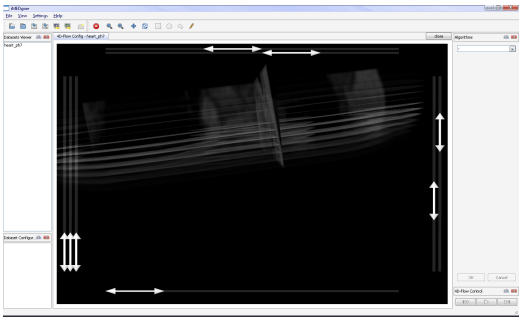
Section 2.1.1.2 in chapter 2 describes two common known integration methods. They can be anticipated here since all of the following nodes and their shader make use



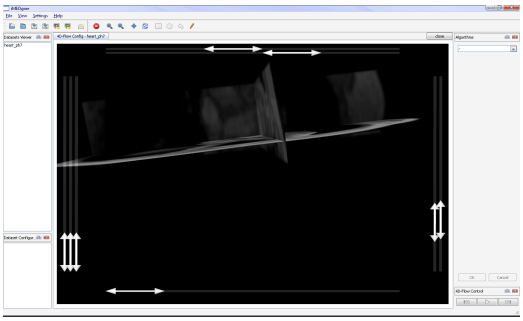
(a) A default view and adjustment of the slice renderer.



(b) A 90° turned view of the default settings.



(c) A high σ - more neighboring slices are shown with increasing transparency - and a lower μ - the brightest slice is placed in the lower part of the volume.



(d) The same μ as (c) but a very small σ makes the slope very high, so only one nearly opaque slice is shown.

Figure 5.10: Influence of the μ and σ values on a volumetric slice rendering node with Gauss distributed transparency values.

of them. Since the Runge-Kutta method uses several Euler steps for a numerical integration, the actual implementation can be adapted rather easy to a simple Euler integration i.e. for performance reasons. However, a Runge-Kutta integration of fourth order is performed per default for all following nodes. Listing 5.3 provides the two functions for Euler and RK4 methods shared by all calculation-shader. They are suited for fragment shader.

Listing 5.3: The Runge-Kutta numerical integration implemented with two functions. For the use of the 'updatePosition' function with the uncommented modifications, a pure Euler integration will result. Otherwise the function 'RK4' implements exactly the in section 2.1.1.2, chapter 2 defined Runge-Kutta integration of fourth order with a given step size and time delay. This function applies for one Runge-Kutta integration step only.

```
1
2  vec3 updatePosition(float time, vec3 oldPos)
3  {
4          //vec3 velVec = time*interpolate(oldPos); //x = v*t
5          //TRILINEAR INTERPOLATION is done with
6          //GL_LINEAR on 8800 HW!!
7          //Nearest neighbor is unfortunatly slightly
8          //faster on smaller graphics hardware
9          //x = v*t => the distance
10         // use oldPos +... *for Euler only
11         vec3 velVec = texture3D(fieldTex3D,oldPos);
12         vec3 newPos = normalize(velVec)*(time*length(velVec));
13
14         //ignore probable noise*
15         if(abs(velVec.x) < 0.1 && abs(velVec.y) < 0.1 &&
16            abs(velVec.z) < 0.1)
17         {
18          //return oldPos for Euler only
19                 newPos = (0.0,0.0,0.0);;
20         }
21  return newPos;
22 }
23
24 //perform Runge-Kutta integration fourth order
25 vec3 RK4(vec3 y, float h, float tn)
26 {
27 vec3 k1 = updatePosition(tn,y);
28 vec3 k2 = updatePosition(tn+h/2.0,y + (h/2.0)*k1);
29 vec3 k3 = updatePosition(tn+h/2.0,y + (h/2.0)*k2);
30 vec3 k4 = updatePosition(tn+h+h,k3);
31
32 vec3 newPos = y + h*(k1+2.0*(k2+k3)+k4)/6.0;
33
34 // if the position is outside of the volume
35 if((abs(newPos.x) > 1.0 ||
```

```
36      abs(newPos.y) > 1.0 ||
37      abs(newPos.z) > 1.0) ||
38     (abs(newPos.x) <= 0.0 ||
39      abs(newPos.y) <= 0.0 ||
40      abs(newPos.z) <= 0.0))
41          {
42                  newPos = y; //=oldPos
43          }
44  return newPos;
45 }
46
47 //[...]
48 void main(void)
49 {
50 //[...]
51 //for example:
52 gl_FragColor.xyz = RK4(posVec.xyz, 0.02, 10.0/textureDimensionY);
53 //[...]
54 }
```

5.3.3.2 Particles

A particle system with predefined velocity fields can be implemented on GPU-hardware with two double buffered textures for position and attribute storage and a 3D Texture for the current flow volume. An additional predefined and constant texture is used to store the random initial positions within $[-1+\epsilon, 1-\epsilon]$, where ϵ defines the initial scatter of the particles around a given coordinate point. Listing 5.4 outlines the algorithm in pseudo code.

Three subsequent shader are used to provide an iterative traversal of the algorithm. The first one looks up the current position of a particle in a velocity field-texture and updates its position depending on the found velocity value. The result is stored in a temporal texture. The second shader calculates a color for the particles with a given color-map and the particle's speeds and stores that in another temporal texture. The third shader sets as much particles to the desired positions as positions have been calculated.

Due to the iterative process of the integration of particle trajectories, one step (or position update) is performed per rendered frame. This so called multi-pass approach has already been outlined in section 2.2.2.2 in chapter 2. To complete this idea figure 5.11 illustrates this approach for the special case of a particle effect.

The shader themselves are written in *shallows notation*, so vertex shader and fragment shader are located in the same file. They are controlled by the GLRender function of a Coin3D node derived from SoShape, as shown in figure 5.8. The initialization of this node is the most time consuming part, since all textures have to be sent to the graphics hardware initially. However, this part is even the most crucial for the over all performance. For example sending particles - which all refer to a single vertex - for each

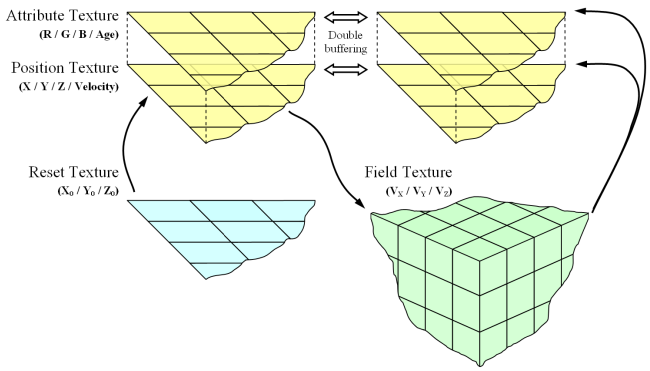


Figure 5.11: A double buffered "Position Texture" stores the current position of each particle and "Attribute Texture" the color assigned to a certain velocity or the particle. The "Reset Texture" is used if no velocity value has been found of the particle left the volume or if it exceeded its maximum lifetime. The initial positions which are stored in this texture are predefined randomly but they can even be arranged so that for example some kind of streak-line will result. The "Field Texture" stores the current flow volume for texture lookup and can be changed during rendering in case of 4D-volumes.

frame via the OpenGL API to the stream processors would cost over 50% of the frame-rate. Consequently, they get stored in a display list during the initialization step. Over all this simple particle effect performes - assuming a buffer-size of 1024x1024, which refers to over one million particles - with a frame-rate between 100 and 150 frames per second on the hardware described in section 6.1. The frame rate was evaluated with an artificial flow dataset.

Listing 5.4: Pseudo shader code for the simplest form of a GPGPU particle system applicable to a predefined 3D velocity texture.

```
1  init arbitrary positions in init=reset texture
2  init ages randomly in attribute texture
3
4  [Fragment shader] // output: attribute texture
5   uniform attribute_texture;
6   uniform position_texture;
7   uniform color_mapping_1D_texture;
8
9    if age of particle <= MAX_LIFETIME
10   gl_FragColor.xyz =
11             color_mapping_1D_texture(position_texture(u,v).a);
12   gl_FragColor.a   = attribute_texture.a + 1.0;
13
14   else
15   gl_FragColor.a = (0.0, 0.0, 1.0, 0.0); // init blue and
16                                          // age = 0
17   //end if
```

```
18 //End of fragment shader
19
20
21 [Fragment shader] // output: position texture
22         uniform seed_point_coordinates;
23         uniform attribute_texture;
24         uniform initial_pos_texure;
25         uniform position_texture;
26         uniform velocity_3D_texture;
27         uniform maximum_velocity;
28
29         if attribute_texture(u,v).a == 0.0
30         gl_FragColor = seed_point_coordinates +
31                 // + seed area rotation assumtions +
32                       initial_pos_texure(u,v);
33
34         else
35         velocity_vector =
36        velocity_3D_texture((vec3)position_texture(u,v));
37
38         //simplified. RK4 with step-size normally used:
39     gl_FragColor.xyz =
40               position_texture(u,v) + velocity_vector;
41         //for the color mapping save the speed
42     gl_FragColor.a =
43         length(velocity_vector)/maximum_velocity;
44
45         //end if
46 //End of fragment shader
47
48
49 [Vertex shader] // input: as many vertices as texels
50                 // in the position texture.
51                 // Correct assignment of the points
52                 // to texture coordinates is required.
53                 // These vertices must be stored in a
54                 // display list, to guarantee the
55                 // performance!
56  uniform position_texture;
57  uniform attribute_texture;
58
59  // all vertices initially are placed on (0.0,0.0,0.0)
60  // but have texture coordinates related to the
61  // position texels.
62  new position = gl_Vertex +
63           position_texture(gl_MultiTexCoord0.xy);
64
65  gl_Position = gl_ModelViewProjectionMatrix * new position;
66  gl_FrontColor = gl_BackColor = attribute_texture.xyz;
```

```
67 //End of vertex shader
```

Such an efficient particle effect can be easily applied for the visualization of streak lines and time lines as they are described in section 2.1.1.2 in chapter 2. The only things to modify would be in the first case the initialization of the initial positions to evenly spaced seed points placed within a plane or line and in the latter case the initial ages in the attribute textures so that all particles are always being released at the same time.

5.3.3.3 Stream lines

Streamlines show - as described in section 2.1.1.2 in chapter 2 - the trajectories of a particle influenced by a static velocity field. Consequently these lines are only valid for one separate temporal volume and will completely change when changing to another field in time. However, local structures of the flow may be explored very well by these lines.

Two approaches have been implemented to calculate particle trajectories. First a single pass shader was evaluated which subsequently updates to the lines in the first render pass with a `for`-loop. Since only with the GeForce 8x Series, a shader may be used recursively in one render pass with its output as input. This solution was chosen to guarantee a compatibility to reference systems as defined in section 6.1. The one-pass implementation on other architectures is extremely inefficient for dynamic manipulation of the scene due to a necessary recalculation of already calculated positions. In this case the access to the output texture is not possible in one pass! However, this rudimentary algorithm is not suitable for a dynamic definition of the seed region, for example by a dragger since every (complete) recalculation of the trajectory texture takes about one second up to 5 seconds. Nevertheless, this approach is for randomly but fixed distributed seed points in the volume as feasible as the multi-pass approach.

Due the focus on interactive usable flow visualization, the further line based algorithms uses a multi-pass algorithm.

The multi-pass approach utilizes a double buffered texture for the calculation of results and a vertex displacement for the actual line rendering. The main differences are that in the sample points position texture each column corresponds to the supporting points of one line and that the displaced vertices in the vertex list belong to line strips[1]. Consequently, a position initialization texture would be reduced to one dimension defining the first row of a vertex position texture. Considering that not all lines will be of an equal length and that some parts of the supporting position texture will remain empty, several other memory management approaches for exploiting the available texture memory have been considered. Finally the simplest approach was used because of the fact that each other, even texture reorganizing algorithms need as much time for the memory management as the calculation itself. Additionally, the used GF 8x architecture provides plenty of texture memory, so the position texture can be

[1] in OpenGL: GL_LINE_STRIP vertex order.

chosen large enough to provide sufficient texture columns for adequate lines. Listing 5.5 gives an insight into two of the used shader; one for the calculation which is only invoked if this is necessary and one to display the results.

Listing 5.5: Pseudo shader code for the simplest form of a GPGPU stream line calculation.

```
init arbitrary positions in init_1D_texture

[Fragment shader] //output: position texture
uniform init_1D_texture;
uniform seed region parameters;
uniform positon_texture;
uniform moving; //indicates if the seed region moves
uniform velocity_3D_texture;

 texCoord = gl_TexCoord[0].xy
 initVec = init_1D_texture(u,v);
 // + considerations for the rotation and dilation
 // of the seed reagion

 if texCoord.y < 1.0 //the start point of the line

   gl_FragColor.xyz = initVec.xyz;
   gl_FragColor.a = 0.0;

 else

         texCoord.y -= 1.0; //get the previous supporting point
         posVec =  positon_texture(texCoord.xy)

 //end if


 if moving > 0 // seed region is moving
               // no calculations are necessary

   gl_FragColor.xyz = initVec.xyz;
         gl_FragColor.a = 0.0;
 else

   // perform one integration step.
   // field texture is accessed in RK4
   gl_FragColor = RK4(posVec.xyz, step-size, time);

 //end if
}
//End of fragment shader

[Vertex shader] // input: as many vertices organized in
```

```
                // line strips as available in the
                // position texture on position (0,0,0).
                // Correct assignment of the points
                // to texture coordinates is required.
                // These lines must be stored in a
                // display list, to guarantee the performance!

uniform positon_texture;

void main(void)
{
//[...]
gl_Position = gl_ModelViewProjectionMatrix *
              positon_texture(gl_MultiTexCoord0.xy)

}
//End of vertex shader


[Fragment shader] // arbitrary input and algorithms
                  // for coloring the lines
//[...]
//End of fragment shader
```

Either mapping of color to an arbitrary parameter of the lines or an illumination as described in section 2.1.1.2 and figure 2.14(b) in chapter 2 or both is possible for the remaining fragment shader from listing 5.5. To give an example, listing 5.6 shows the fragment shader implementation of the equations 2.6 and 2.7 from the just referred section to illuminate actual diameter-less lines.

Listing 5.6: Pseudo fragment shader code for the additional "illumination" of arbitrary diameter-less lines. This approach would even work for a particle effect.

```
[Fragment shader]
uniform positon_texture;
uniform V; // norm. camera direction
uniform L; // norm. light direction

void main(void)
{
        prev_position = positon_texture(u-1.0,v);
        next_position = positon_texture(u+1.0,v);
        T = normalize((next-prev)); // tangent

        LN = sqrt(1.0-pow(dot(L,T),2.0));
        VR = LN*sqrt(1.0-pow(dot(V,T),2.0))-dot(L,T)*dot(V,T);
        I = 0.1 + 0.1*LN + 0.8*pow(VR,64.0);

  // a color may be set within the vertex shader,
  // so the fixed function pipeline will iterpolate
```

```
18    // between different ones.
19    // This color can now be illuminated per fragment with I.
20    gl_FragColor = gl_Color * I;
21  }
22  //End of fragment shader
```

5.3.3.4 Path lines

Calculation of path lines is built upon the calculation of streamlines. This line type has to consider the fourth dimension as well since they represent a particle's trajectory over time. The simplest way to generate a path line is to assemble it with parts of streamlines from each volume. This method is error-prone but efficient to implement by simple changing the volumes over time during the calculation of streamlines as described in section 5.3.3.3. Assuming that a given four dimensional dataset consists of t volumes and the position buffer texture is of the size $n \times m$ the volume number has to be incremented each $floor(\frac{m}{t})$ frame. Since these lines are now constant when the user for example changes the image volume, an additional indicator should be available for convenience. Coloring parts of the lines with a signal color has proved to show the user in which volume which part of the line was generated. Figure 5.12 shows that behavior on an example.

Certainly all illumination and further coloring techniques can be used with these lines in the same way as with streamlines. The remaining thing with this algorithm is to evaluate whether an interpolation in temporal direction - which means an about quadratic increase of computational costs - would produce a smaller error or not. These considerations are intended for future work.

5.3.3.5 Stream tubes

When path lines and stream lines calculations are done even a more realistic visualization, so called stream tubes are possible. The basic idea is described in detail in section 2.1.1.2 in chapter 2.

Generating a real mesh of vertices around an actual calculated line would result in a vast amount of vertex calculations and consequently in a very low performance. To handle this problem a so called impostor rendering technique was used. This trick allows to render reasonably realistic tubes with only twice as many vertices as used with simple line visualizations. Instead of a line strip a quad strip[1] is stored in a display list and then subsequently displaced to the correct positions. The visualization algorithm can be performed in one vertex shader and one fragment shader which is illustrated in figure 5.13.

The main task for the vertex shader is to displace the two(!) vertices corresponding to one supporting point position always perpendicular to the viewing direction. The points of the quad strip itself can be distinguished by different initial positions so

[1] *GL_QUAD_STRIP* vertex definition for OpenGL)

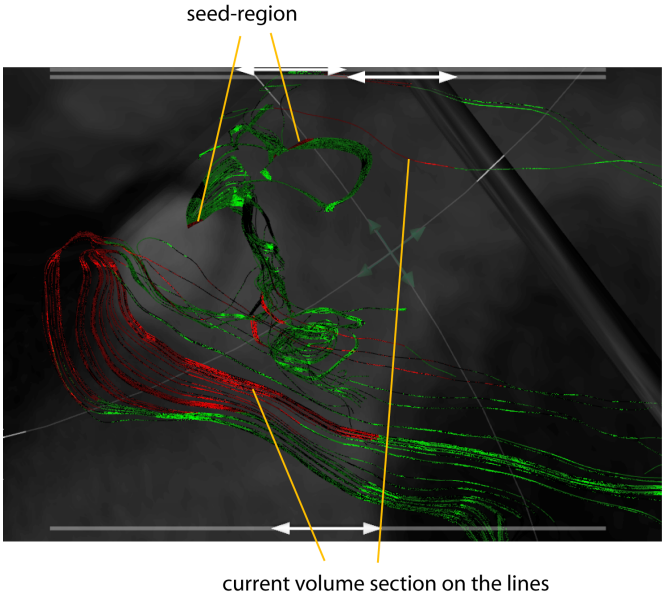


Figure 5.12: This Figure shows path lines with a red colored one-dimensional seed-region and red-colored sections of the lines which are placed in this part related to the volume currently shown in the background by the morphological slice renderer in the background.

that always the same side of the strip is placed below the actual line and the other above. Due to the fact that a line strip is not a tube when looking directly in line direction a terminator at the start and the end of each tube has to be added. This might be a pre-shaded circle rendered on an impostor-quad perpendicular to the line's tangent direction or a sphere or anything else. The start and the end point of each line can be easily determinated since they always refer to the first, respectively the last texture coordinate of each column in the already calculated supporting points texture. Consequently, the terminator structures only have to be assigned to these coordinates.

For realistic lighting and coloring of the tube, the interpolations abilities of built in shader varyings can be utilized. A smooth color course can be established by setting the variable `gl_FrontColor` for each vertex to a value looked up in a former introduced color gradient for some arbitrary parameter. Moreover the correct reflection of light is important. To give an impression of a dilated tube an additional varying for the vertex normals has to be defined. By setting the normals always in the same direction as the vertices have been displaced relatively to the line, correct spherical interpolated normals will result for each fragment between the borders of the strip. The blue arrows in figure 5.13 try to explain these coherences. Phong-shading with a fragment shader is consequently possible directly. Due to a well defined normal vector for each fragment, equation 2.6 can now be solved directly without a maximization criterion as used for the illumination of lines. This approach still performs with about 60 to 120 frames per second on the reference systems. The line length and number were each defined with

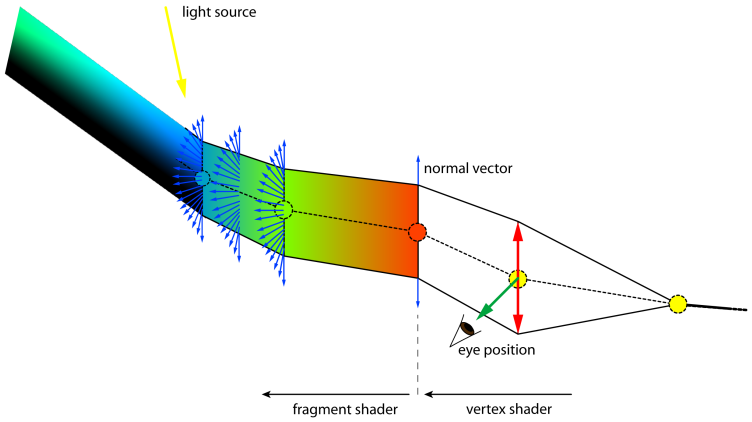


Figure 5.13: The development of stream tubes based on a pre-calculated line texture with stream- or path line supporting points. A vertex shader places the pre-defined vertices of a quad strip relative to available supporting points and defines an arbitrary color and a normal vector perpendicular to the viewing direction. With the interpolated color and normal vector values a fragment shader can perform a Phong based even realistic illumination.

1024 and the frame rate was evaluated with one certain artificial flow dataset.

Some modifications of this algorithm would lead to stream ribbons (see section 2.1.1.2 and figure 2.14(a) in chapter 2). In this case the vertex displacement relative to the line may not occur perpendicular to the viewing direction but dependent on the surrounding field's rotor. A vertex' normal vector perpendicular to the ribbon itself would complete this modification.

5.3.3.6 Cutting-Plane Techniques

A simple but effective technique to represent parts of the field is the insertion of different kinds of planes in a volumetric velocity field. This plane can consist of many vertices organized in a triangulated mesh quad (*mesh-plane)* or of a simple quad bordered by four vertices (*quad-plane)*. Both techniques can be used with different benefits.

An obvious parameter to map onto this plane would be the color coded magnitude of the velocity value in perpendicular direction to the plane. The sampling of the volume with mesh vertices would provide the advantage of an additional linear interpolation within the plane if, for example, the complete linear interpolation of the velocity 3D texture is not possible due to performance reasons and a nearest neighbor interpolation is used for 3D textures instead. In this case the sampling of the volume is made with a vertex shader directly with the vertices' coordinates and the interpolation with the built-in varying `gl_Color`. In the quad-plane based approach the sampling can be done by mapping the processed texture coordinates between the four vertices into the velocity volume texture.

The result will be in both cases the same at a first sight. The differences are given if

more complex operations are performed for the plane. Such operations may be required since with a simple mapping onto a 2D plane a third dimension and its information gets lost. For example, the magnitude of the flow through a certain vessel can be investigated but not the directional effects.

With a mesh-plane it is possible to displace its vertices in direction of the sampled velocity value. The result will be a height profile, indicating the strength and direction of the flow. A subsequent assignment of the direction of the flow to the vertex as normal vector will even provide a Phong illumination of that crinkly plane.

The usage of the velocity direction as normal vector can also be applied to a new approach for quad-planes. The drawback of the former vertex displacing algorithm is that one the one hand the plane will occlude itself and on the other hand the vertex processing of a mesh is more expensive than the processing of only four. Consequently, a related algorithm but implemented with a fragment shader can use the flow directions within the plane as normal map for the illumination. For computer games this technique is called "bump mapping" because it lets actual flat surface appear with a profile. If the user knows that the velocity direction is used as normal vector, he can easily interpret a three dimensional information in a two dimensional plane. The crib for example a head-lighted scene will then be: "The more it reflects, the more the flow indicates towards me". The drawback of this method is that it requires this crib as meta knowledge.

Certainly, with planes much more sophistical visualizations may be implemented and some such ideas are outlined in chapter 7. However, the goal of them would go farther than the focus of this work since most of them would suggest a plane as kind of sensor array placed in a hemodynamic system. The results of such approaches would rather all provide a measured quantity than an instructive visualization.

5.3.3.7 Concurrent Combinations

The benefit of a free definable view as presented is obvious. All visualization nodes can be combined, even with different seed regions. Eventually the buffer sizes have to be reduced due to performance. This may be also necessary to preserve the overview of the visualizations since for example a million particles with ten-thousands of arbitrary lines would only produce visual clutter. It was empirically found that at most two different visualizations per seed region are meaningful. More than one dragger in a scene also appears distracting. A combination of side by side views will be more satisfying for such a concurrent visualization. More promising control approaches are therefore presented in section 7.4 in chapter 7.

Chapter 6

Experiments

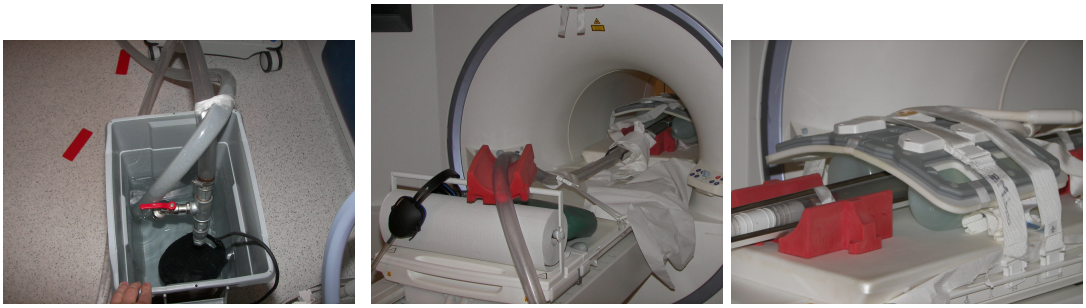
Several attendant PC-MRI measurements have been accomplished. Gathered datasets from humans and a phantom are described in section 6.2. Additionally the already introduced libraries and iMEDgine which supports multiple development platforms, a Windows and Visual Studio 2005 based environment as described in section 6.1 was built up. Finally section 6.3 will show screen-shots of different visualizations by means of instructive parts of selected datasets and section 6.4 concludes with render performance measurements.

6.1 Development and Test Systems

For development and demonstrations purposes a PC-system was assembled with the following specifications:

- EVGA GF 8800GTS 640MB DDR3 320Bit PCIe 16x/ Nvidia graphics card,
- AMD Athlon64 X2 4200+ 2.2GHZ EE pI Socket AM2 @ 2.5Ghz,
- Western Digital S-ATA2 320GB 7200rpm 16MB cache, 3,5" hard-drive,
- ASUS M2N-E SAM2 HT2000 nForce 570 Main board,
- Corsair DDR2 3072MB 6400 PC800 64MX8 NON-ECC system memory,
- LG Flatron L194WT 19" wide screen monitor.

As reference system a Sony Vaio S5M/S, Pentium M740 with Nvidia GeForce Go 6400 and 1024MB system memory was used. The program **Fraps v. 2.8.2**[Fraps2007] was used to benchmark frame rates and to take screen-shots and screen movies during run time.



(a) The flow is emulated with a submersible pump and controlled by a valve. The end of the tube closes the circuit in the same tub.

(b) The narrowing is placed for the measurements in the iso-center of the magnetic field.

(c) Two bottles filled with water enable for the body-coil an equal distance to the tube.

Figure 6.1: The measurement setup at the radiology department of the Landesklinikum Graz for a flow phantom with artificial narrowing.

6.2 Test Datasets

To evaluate the qualities of the proposed visualization algorithms ten datasets of real human blood flow in different vessels and the human heart. In addition two artificial flow datasets from a phantom were measured at the radiology department of the Landesklinikum Graz and at the Diagnostikzentum Graz.

First two datasets covering the whole heart were investigated. It turned out that these dataset indeed produce beautiful visualizations but for a first evaluation they were too complex. Furthermore, the effort to segment vessels in these datasets with the 4D-Flow Toolbox was extraordinary high. Two additional datasets covering the left ventricular outflow tract (LVOT) were found to be complex as well but the scenes were more comprehensible. Unfortunately these datasets often provided to few measured slices, so the examined voxels were rather anisotropic. Finally, six datasets containing only one or two slices but showing significant vessel structures with dominant in-plane flow behavior were studied. They turned out to be rather comprehensible and showed clear results but the major context of the flow was only given by the underlying morphological structures.

Second, a phantom which tries to emulate a constant flow through a flexible tube with an artificial stenosis was measured. The flow was generated by a submersible pump and a valve placed behind. An artificial narrowing was simulated by a drilled screw cap placed on a cut off PVC-bottle in the middle of the tube and the magnetic field's iso-center. Figure 6.2 shows the measurement setup at the radiology department of the Landesklinikum Graz.

However a complete evaluation of the visualization algorithms and their proof of correctness will occur in future work. Section 7.5 in chapter 7 gives an outlook on that issue. Section 6.3 shows some representative images from the resulting visualizations.

6.3 Results

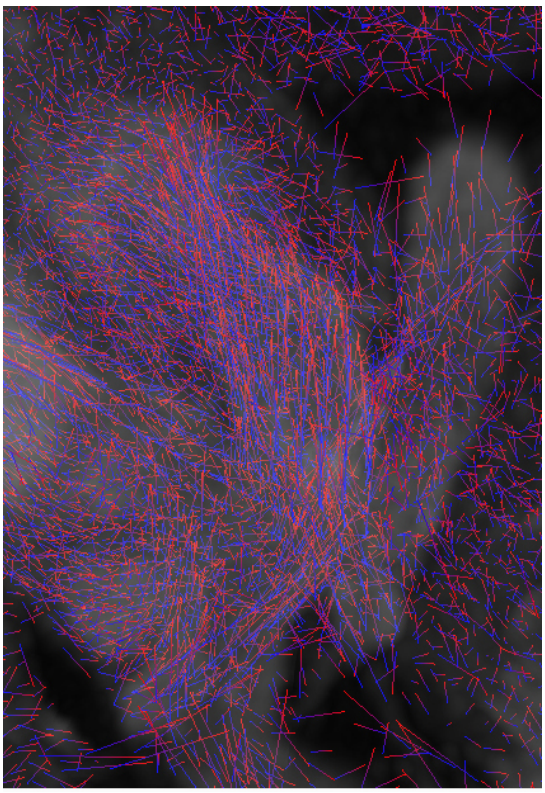
To complete this chapter some images resulting from the so far described algorithms and datasets can be presented in this section. The following figures show several more or less simple flow structures since more complex ones would not be feasible for these static images due to missing interactivity. Referring to that, it can be underlined that the more complexity a velocity field contains the more important interactivity gets, even if it is only the ability to rotate, pan and zoom the scene. Due to these inconveniences not always the same but the most instructive perspective was chosen for similar datasets. The diagrams 6.8 to 6.9 in section 6.4 subsequently compares the main techniques due to their performance in frames per seconds on the two reference systems which were presented in section 6.1.

- **Cash-Flow and glyph-based** visualizations are shown in figure 6.2.
- Different applications for **particles** are shown in figure 6.3.
- **Stream lines** and **path lines** are shown in figure 6.4.
- Different types of **cutting planes** are shown in figure 6.5.
- **Stream tubes** are shown in figure 6.6.
- Examples for a **combination of visualizations** are shown in figure 6.7.

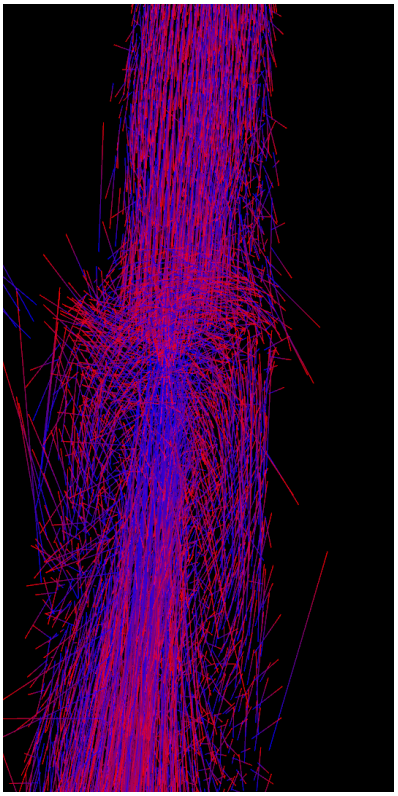
6.4 Scalability

Diagram 6.8 to 6.9 show that a glyph renderer is faster when less glyphs are rendered. In this case we varied the render positions between rendering every fifth velocity value to rendering every 20^{th} position. Since this algorithms is mainly performed on the CPU and every frame all positions are passed to the graphics cards, a frame rate higher than 60 could not be expected and also a not so high difference between the reference systems.

The performance of GPU based visualizations highly depends on the amount of rendered elements and therefore on the used buffer sizes. The buffer size $n \times m$ determinates in the case of a particle based visualization the number of possible rendered particles by $n * m$ and in the case of lines n determinates the number of lines and m the supporting-point count of each line. We tested quadratic buffer sizes from 256 to 1024 pixels in each direction. Rectangular buffer textures are possible as well, but perform with the same results as quadratic ones until one size exceeds the next possible power of 2 (for example 256, 512, 1024,...). Consequently, a quadratic buffer size leads to an optimal result. Since all buffer positions are definitely processed each render pass, techniques for an early line termination, for example when the line moves out of the velocity field, would not increase the performance.



(a) Viewing a human heart from above with the pulmonary artery in the front and the vena cava superior in the right handed background. Additionally the morphological data is rendered by a SimVoleon sub-tree.



(b) The flow phantom dataset shows the artificial stenosis viewed slightly against the flow direction.

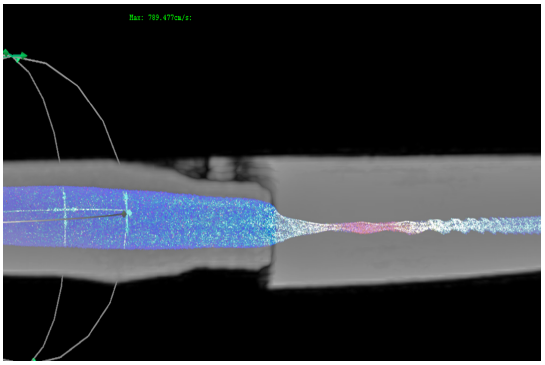
Figure 6.2: Point based glyph overview of the human heart of subject number two (a) and a flow-phantom view (b) displayed with a Cash-Flow render node. Both are containing 15 measured slices.

On the GeForce 8800 graphics card all algorithms showed that they are suitable for real-time applications except a flexible cutting plane with 1024x1024 vertices. But this visualization subjectively gives the same information with 512x512 vertices and a bigger buffer size may not be necessary. Figure 6.9 additionally compares the performances with concurrently used volume rendering utilizing a SimVoleon render node to a visualization without any morphological data rendering. Figure 6.8, 6.10 and 6.11 compare the pure flow visualizations with ones using the slice-rendering node as presented in section 5.3.1. As expected a more advanced volume rendering approach decreases the achievable frame rate. A cutting plane based on a bump mapping approach is in these figures compared noncompetitive since this technique only needs four vertices and performs a direct texture lookup in its fragment shader. The used dataset was always the same - 256x104 pixels x 11 slices x 2 time steps - flow phantom with emulated ECG

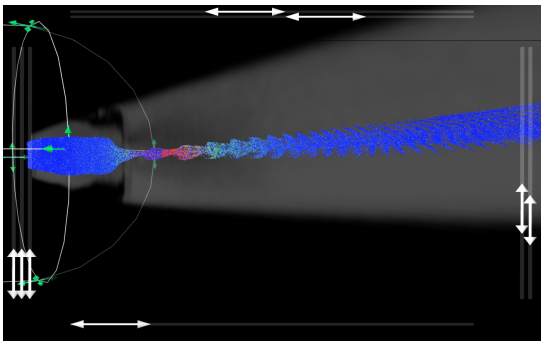
activity.

The highest possible frame rate without any rendering is 2050 fps on the GF 8800 architecture and 530 fps on the reference system. Referring to diagram 6.12 the frame rate decreases for concurrently rendered GPU based visualizations nearly on an exponential curve.

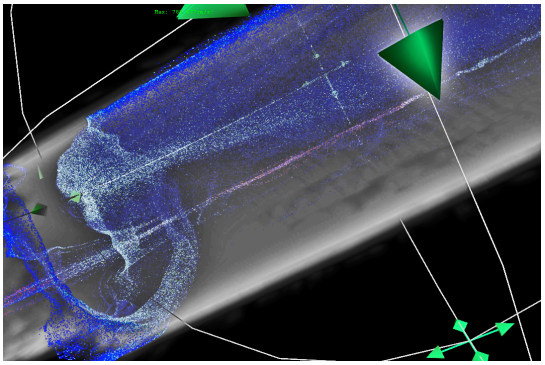
For stream line based visualizations the limit of switching through the time has to be considered as well. All lines will need exactly m render frames to fully evaluate for the use of a $n \times m$ buffer for each time step. Due to our interactive implementation, the user need not wait for a full line evaluation to get to a certain time position. The course of the lines can be estimated even after the calculations of a few sampling points and the time step can be changed independently from the line calculation.



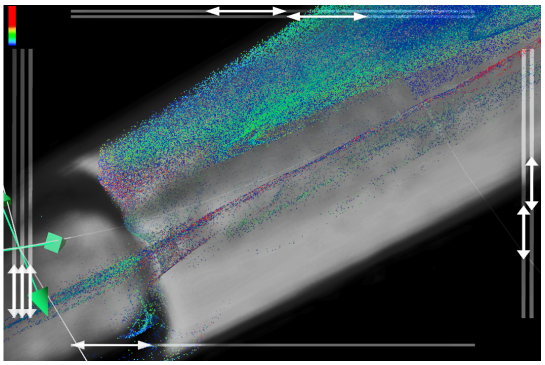
(a) Particles with overlaid SimVoleon volume rendering.



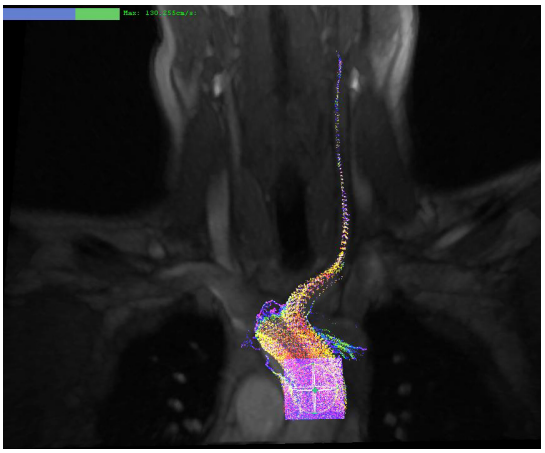
(b) Particles with a slice based rendering.



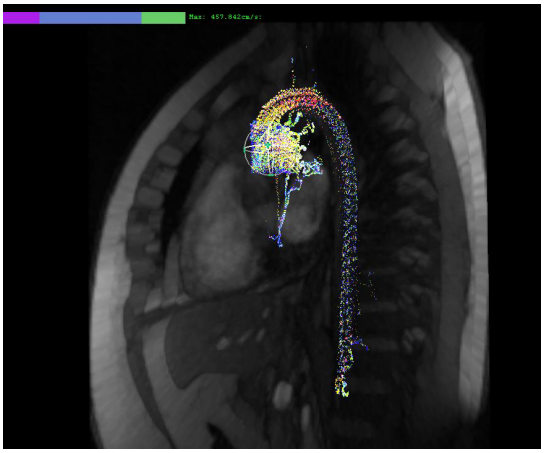
(c) Details of the stenosis with volume rendering. The seed region was moved to the border of the tube.



(d) Details of the stenosis with slice based background and a different color-mapping gradient. The seed region was moved to the border of the tube.

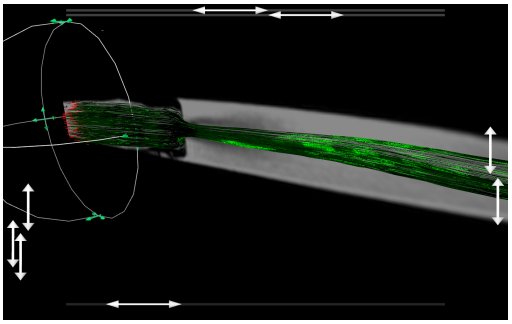


(e) A particle effect applied to an one slice dataset of the author's carotid artery bifurcation.

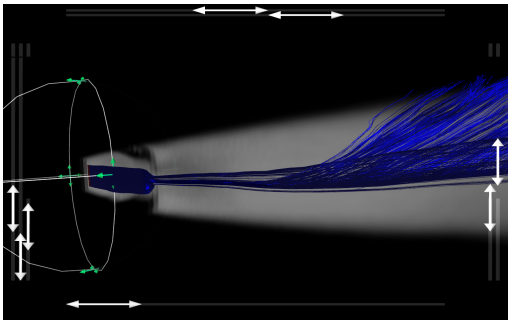


(f) The same effect as in (e) applied to an one-slice dataset of the author's aortic arch.

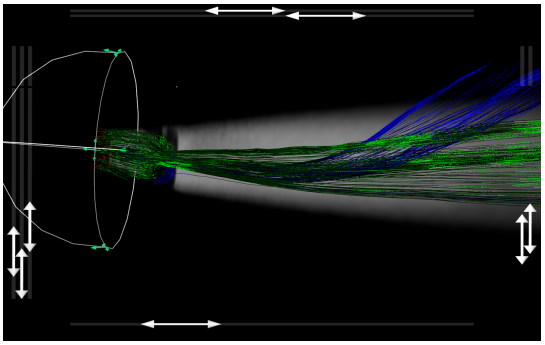
Figure 6.3: Particle based sparse representation of the artificial stenosis dataset. The left two images use a SimVoleon sub-tree for volume rendering, the right ones the slice based rendering approach. Note the back-flow funnel in images (c) and (d). (e) and (f) show an according effect applied to real blood flow.



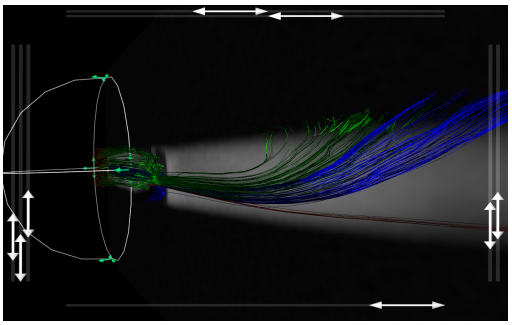
(a) Stream lines for the first time step.



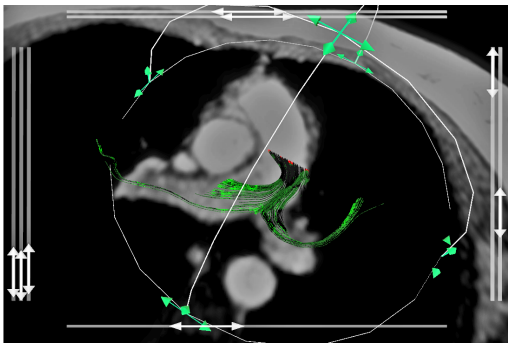
(b) Path lines.



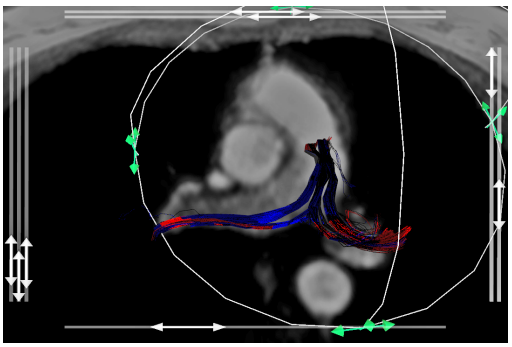
(c) Concurrently rendered stream lines and path lines for the last time step.



(d) Concurrently rendered stream lines and path lines for the first time step.

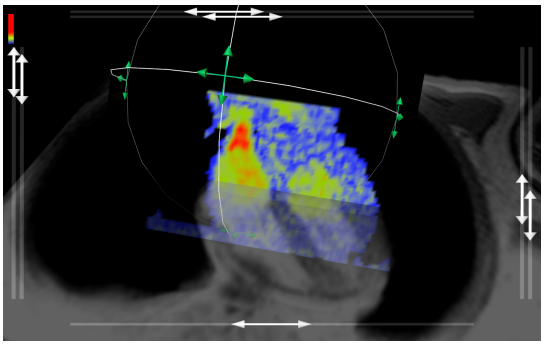


(e) Arteria pulmonalis bifurcation with streamlines at time step six.

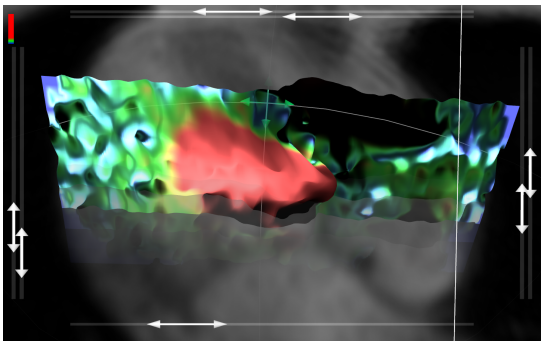


(f) Arteria pulmonalis bifurcation with pathlines. The parts which refer to the time of figure (e) are marked red.

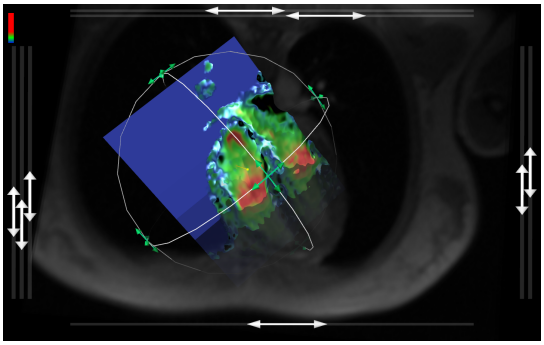
Figure 6.4: Comparing illuminated stream lines and path lines. Note that figure (c) and (d) illustrates that the artificial flow inside the flow phantom does not specify a constant flow as expected in advance. (e) and (f) visualize the bifurcation of the arteria pulmonalis of a human heart of subject number two.



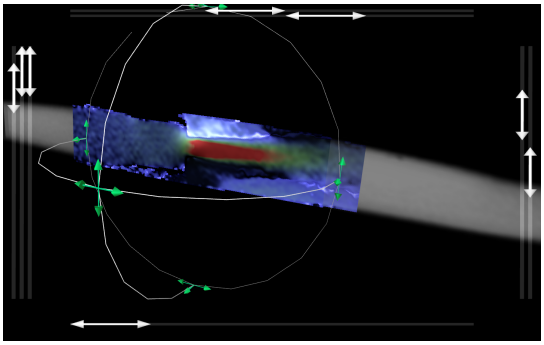
(a) A standard mapping of velocity magnitude to a color gradient on the plane texture.



(b) A mesh-plane with vertex displacement technique and velocity magnitude to gradient mapping. Note that the plane is occluding itself.

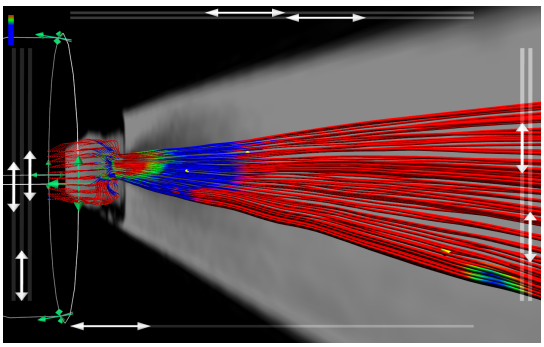


(c) A cutting plane with bump-mapping technique where the velocity directions corresponds to the normal vector of each texel of the plane.

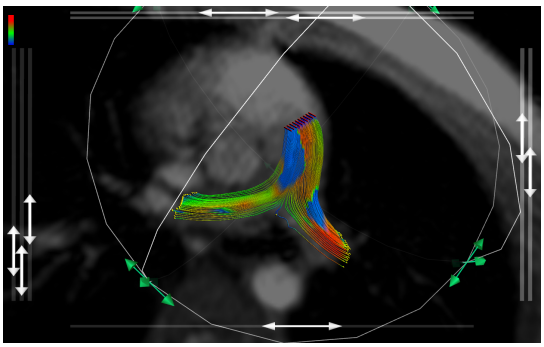


(d) The same cutting plane as used with figure (c) applied to an artificial stenosis dataset.

Figure 6.5: Comparing different cutting plane visualizations for the same dataset of the whole human heart of subject number two. The positions of the planes were chosen in a way to provide a good survey for the characteristics of the different approaches.

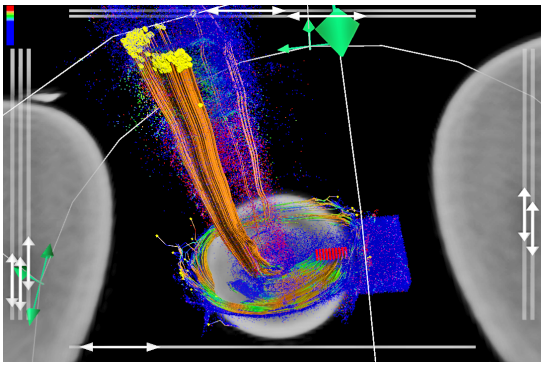


(a) Stream tubes applied to an artificial stenosis dataset.

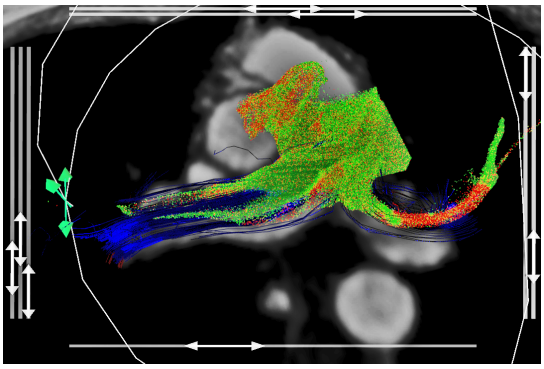


(b) Stream tubes applied to the bifurcation of the arteria pulmonalis of the human heart of subject number two.

Figure 6.6: Stream tubes applied to calculated stream lines.



(a) Stream tubes and particles with the same seed region applied to an artificial stenosis dataset measured transversal with a seed region near the actual stenosis. The viewing direction is slightly opposed to the flow direction.



(b) A combination of path lines and particles applied to the already presented bifurcation of an arteria pulmonalis of subject number two.

Figure 6.7: Two examples for a concurrent combination of different visualizations. This can be achieved by inserting the according nodes to a new Inventor script as described in section 5.2.2.

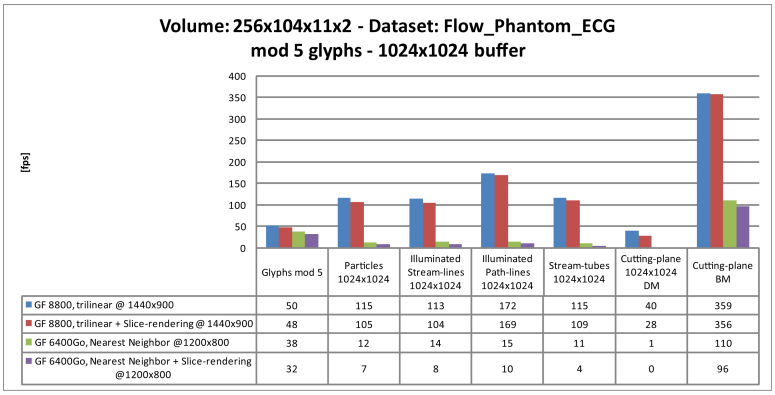


Figure 6.8: Comparing the frame rates of glyphs rendered at every fifth position with GPU visualizations at a buffer size of 1024x1024 which equals 1.048.576 permanent available particles or 1024 lines with each with 1024 sampling points. The last two are cutting plane visualizations, where the first abbreviation *DM* stands for displacement mapping and the second, *BM*, for bump mapping.

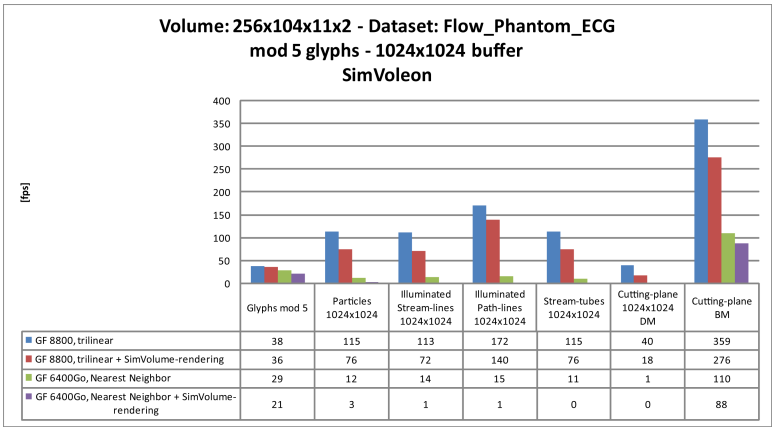
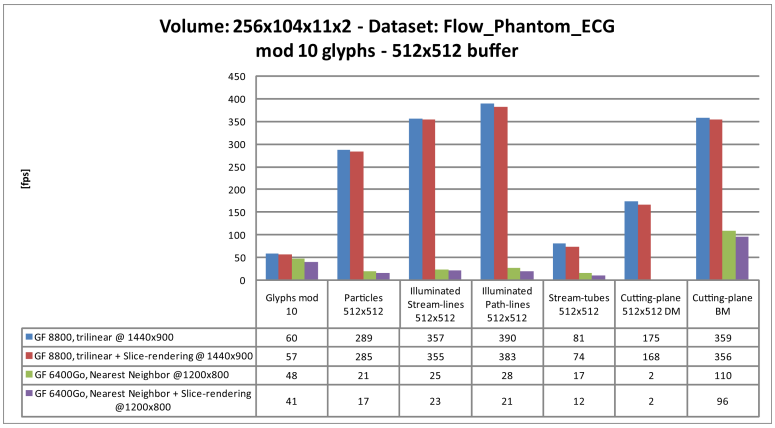
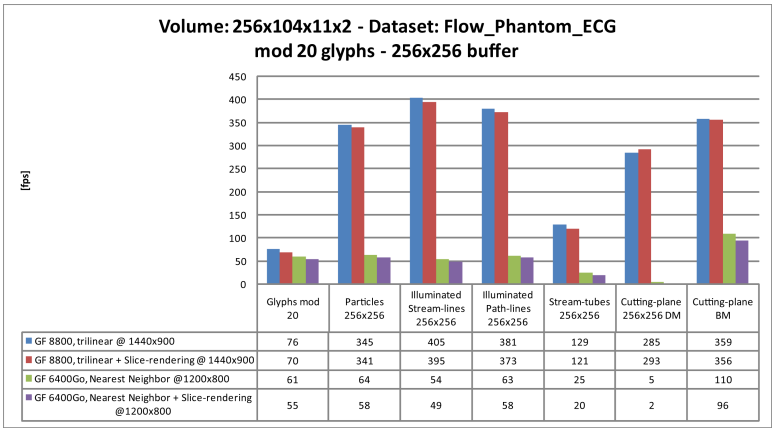


Figure 6.9: Comparing the frame rates of glyphs rendered at every fifth position with GPU visualizations at a buffer size of 1024x1024. The last two are cutting plane visualizations, where the first abbreviation *DM* stands for displacement mapping and the second, *BM*, for bump mapping. Instead of the slice render approach for additional morphological data rendering a SimVoleon volume render sub-tree was used.



	Glyphs mod 10	Particles 512x512	Illuminated Stream-lines 512x512	Illuminated Path-lines 512x512	Stream-tubes 512x512	Cutting-plane 512x512 DM	Cutting-plane BM
GF 8800, trilinear @ 1440x900	60	289	357	390	81	175	359
GF 8800, trilinear + Slice-rendering @ 1440x900	57	285	355	383	74	168	356
GF 6400Go, Nearest Neighbor @1200x800	48	21	25	28	17	2	110
GF 6400Go, Nearest Neighbor + Slice-rendering @1200x800	41	17	23	21	12	2	96

Figure 6.10: Comparing the frame rates of glyphs rendered at every tenth position with GPU visualizations at a buffer size of 512x512 which equals 262.144 particles or 512 lines with each with 512 sampling points. The last two are cutting plane visualizations, where the first abbreviation *DM* stands for displacement mapping and the second, *BM*, for bump mapping.



	Glyphs mod 20	Particles 256x256	Illuminated Stream-lines 256x256	Illuminated Path-lines 256x256	Stream-tubes 256x256	Cutting-plane 256x256 DM	Cutting-plane BM
GF 8800, trilinear @ 1440x900	76	345	405	381	129	285	359
GF 8800, trilinear + Slice-rendering @ 1440x900	70	341	395	373	121	293	356
GF 6400Go, Nearest Neighbor @1200x800	61	64	54	63	25	5	110
GF 6400Go, Nearest Neighbor + Slice-rendering @1200x800	55	58	49	58	20	2	96

Figure 6.11: Comparing the frame rates of glyphs rendered at every 20^{th} position with GPU visualizations at a buffer size of 256x256 which equals 65.536 particles or 256 lines with each with 256 sampling points. The last two are cutting plane visualizations, where the first abbreviation *DM* stands for displacement mapping and the second, *BM*, for bump mapping.

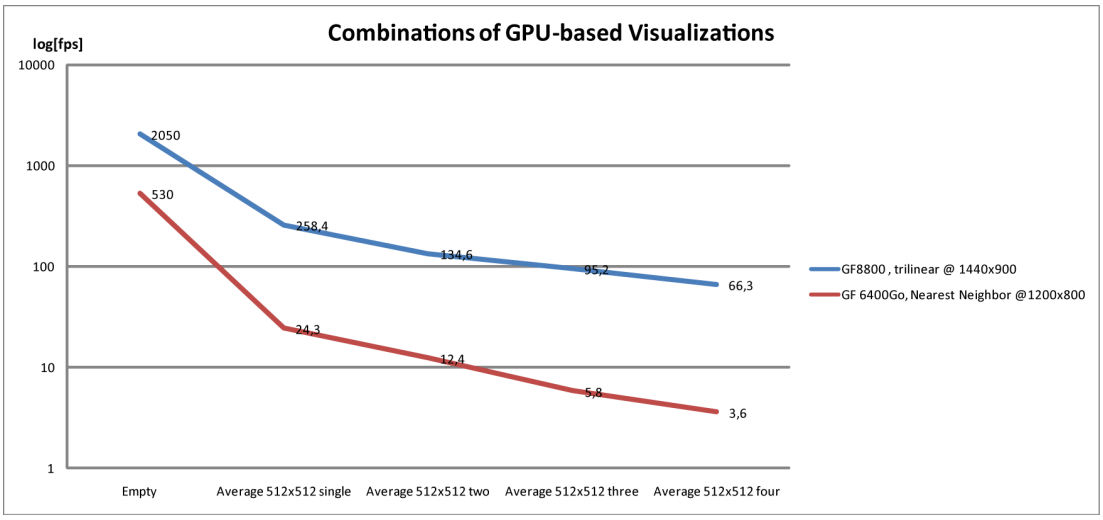


Figure 6.12: Comparing the average frame rates from particles, illuminated stream lines and path lines and stream tubes at a buffer size of 512x512. Due to the logaritmic scale of the ordinate, the frame rates decrease related to an exponential curve.

Chapter 7

Future Work

This chapter gives an overview of the vast possibilities for future work. Referring to figure 2.12 from chapter 2, section 7.2 itemizes some possible improvements of the already implemented new visualizations. Some further derivable quantities of the measurement method presented in chapter 3 are outlined in section 7.3. Usability and controllability of the algorithms are treated in section 7.4 with a special focus on augmented and virtual reality ideas. Finally section 7.5 deals with one of the most important questions for future developments: "Are the attempted efforts expedient for clinical routine at all?".

7.1 Measurement Method and Preprocessing

The measurement method itself provides much more than the quantification of velocity in the blood. Theoretically all moving tissues and fluids in the human body can be investigated. For example a measurement of the movement of the myocardium or the flow of synovial fluid could be useful as well. As mentioned in chapter 3 a method to pre-estimate the v_{enc} value and a sequence which can perform with respiration of the patients would be desirable too.

The Siemens 4D-Flow Toolbox, respectively the calculation tool is currently a nice and powerful tool to process the raw MRI image data fast and accurate. Nevertheless a segmentation of interesting or rather flow-undefined regions by hand is a quite exhausting and time consuming task. Consequently, the following improvements are proposed:

- An automatic segmentation of the myocardium respectively of these regions which are definitely valid flow values. This would be a vision task, so these deficiencies were not illuminated closer in this work on visualization. However, if such a segmentation would perform accurate the segmentation borders itself could be used for a direct visualization of the tissue as surface.
- Better algorithms for an automatic aliasing correction are available because the course of the flow velocity values is not as linear as indicated in figure 4.3 in chap-

ter 4. The curve is actually much more hackly so that the currently implemented algorithm does not perform well.

- The implemented baseline correction seems not to work correctly. Its algorithm has to be investigated more closely.
- The calculated velocities have to be evaluated over all with concurrent performed catheter based measurements or at least mathematically with calculations of a stationary and well defined flow phantom.

7.2 Visualization Techniques

In this work two aspects of the visualization taxonomy presented in chapter 2 were implemented as shown in figure 7.1. Certainly a complete evaluation of all known and some new visualizations would be worthwhile but also very time consuming. Hence the future features listed in this section give a short overview of some ideas which arose during the development process.

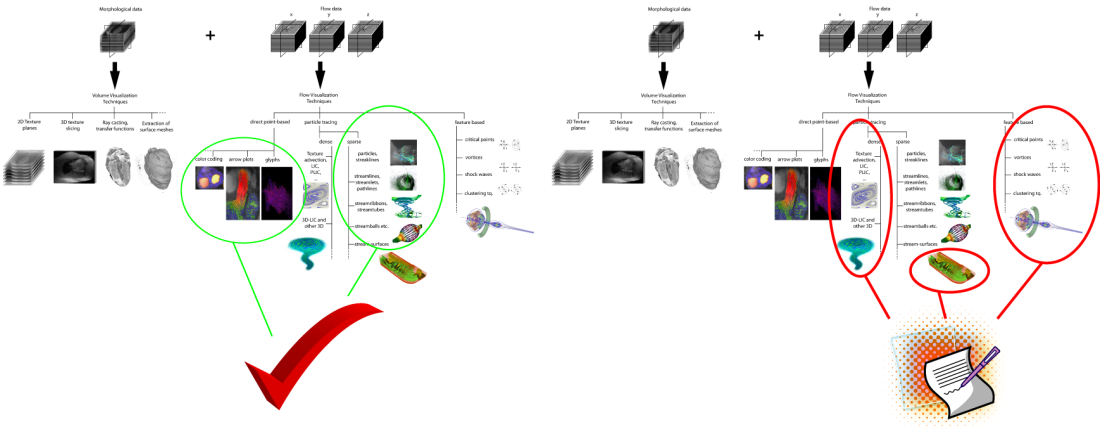


Figure 7.1: The investigated areas of the in chapter 2 presented taxonomy are shown on the left. The remaining parts for future work are shown on the right side.

First some possible improvements of the now available visualizations are mentioned. Subsequently some preferences for new visualization algorithms are listed.

- In the case of line based visualizations simple arrows or glyphs could be placed at the calculated supporting point positions to provide a hint for actual flow direction even with a smart chosen distance. Due to the simplicity of this modification the same performance should be reached.
- Currently, in most cases the velocity magnitude was color mapped onto the flow visualization structures. Further quantities should be evaluated to prove their

possibilities. For example the bend, rotor or a kind of clustering of lines which developed in related streams could be appropriately colored.

- For path lines and other time dependent calculations an interpolation of the velocity values between the time steps should be compared to the actual algorithm. Such a smooth change over time should reduce the integration error of the trajectories.

- A GPGPU 3D-UFLIC approach should be tried out to evaluate its performance and usability for the representation of the given flow fields.

- Further clustering approaches can be invented to give a fast and accurate overview of the important structures in the velocity field. Since this could theoretically be applied to an arbitrary visualization, the focus for this task would be an algorithm to filter these important areas. Further evaluations to show how an important region is defined are required for that issue as well.

- With a calculation of the rotor of the surrounding field and an additional knowledge of pressure differences as described in section 7.3, a visualization related to diameter-changing and torsion-showing "Hyperstreamlines" from [Reina2006] could be considered.

- For future research the most important property of the measured flow may be the identification and representation of vortices. Due to their still proved diagnostic use for the cardiac efficiency a detailed investigation of algorithms which can identify vortices would be important. Work has been done for the detection of two dimensional flow vortices, so the major focus would be a detection and tracking of vortices in four dimensional velocity fields.

7.3 Derived Magnitudes

Another task would be the mathematical extraction of further information from the velocity field. These quantities could subsequently be used for further visualizations or mapping strategies but with an increase of the required knowledge.

- Maybe the simplest quantity is to gather the cardiac output. One has to determine the heart-beat-volume of at least the left ventricle and the heart rate. The heart rate can be recorded during the MRI-measurement. The heart-beat-volume would be easy to calculate if the maximum and minimum volume of the ventricle during the series are known. The difficultly is to determine the volume of the ventricle.

- The calculation of relative pressure differences can theoretically be performed with an evaluation of the Navier-Stokes equation along different paths. To guarantee the performance the different paths approach furthermore offers an implementation on a SIMD architecture, consequently a GPU. The knowledge about these

pressure differences allows an evaluation of for example surface pressures and tissue stress. The calculation could be performed locally with "sensor planes" as mentioned in chapter 5.3.3.6 or in a preprocessing step for the whole volume but also with a certain support of a GPU. First experiments have been done based on the work of [Ebbers2001; Buyens2003] but have not finished yet.

- In several sub-parts of the flow one could think about attending the viscosity and the fringe effects of the blood for the movement of some special particles. Given that the blood flow inside the heart is a laminar one, this challenge is feasible. Since this part will be delicate, several closer studies about the physics of blood flow will have to be done before.

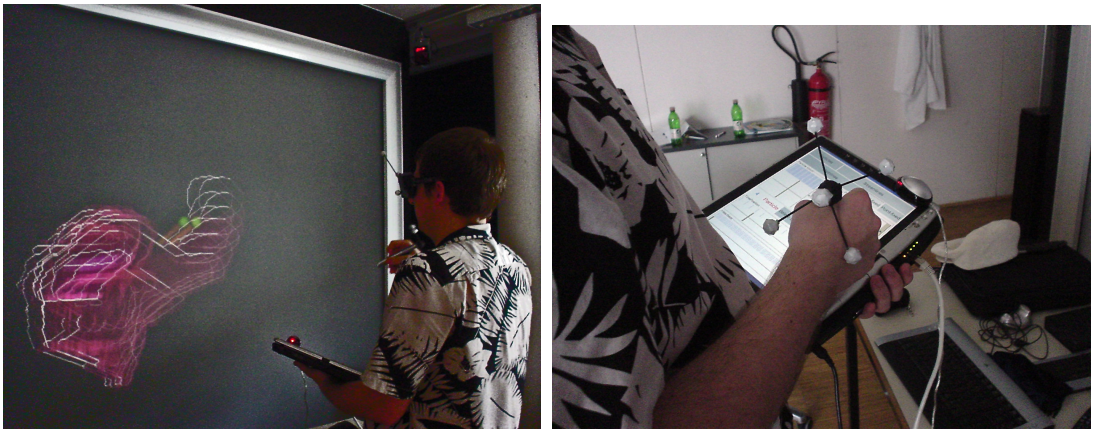
7.4 Virtual Reality Visualization Environments

One of the most interesting findings is the fact that the control of the presented visualizations is a still inconvenient. Dealing with four dimensional data, controlled by a two dimensional interface can be of course confusing. In previous work already an augmented reality setup as shown in figure 7.4 was investigated to cope with these problems. In general this setup enabled an injection of particles in a simple head tracked stereo representation of the left ventricle with an infrared 3D tracked pencil. At that time the focus of this work done in the course of the lecture on "Virtual Reality" at the technical university of Graz, [Schmalstieg2006a], lied on the investigation of the possibilities of a Studierstube [Schmalstieg2002] supported environment for this kind of data. For future work a viewer with an interface to Studierstube should absolutely be considered. A setup as shown in figure 7.4 may indeed be too large and expensive for an use in clinical routine but Studierstube supports a vast amount of display devices and tracking systems. Hence small and intuitive setups could be invented. A first step in this direction has already be gone since iMEDgine script-able views (section 5.2.2.1) support a simple stereo rendering if the corresponding hardware is available. Further ideas for an virtual reality control environment are summarized in the next list. However, these techniques have to be evaluated permanently for their usability and acceptance with medical staff.

- First of all several different control mechanisms of the perspectives itself have to be investigated. For example flying through volumes instead of zooming and panning could be useful to present the data to an audience.

- As already mentioned, controlling the seed regions of interactive visualizations would be simpler with at least a kind of 3D interface. Due to the lack of tracking systems at clinical work stations a first approach for that could be a space-mouse. Later on more sophistical research and evaluations with for example different "cheap tracked things" assigned to certain visualizations may be accomplished.

- The large amount of control options in a view may be addressed in a better way



(a) Stereo projection with head tracking and a wireless input device.

(b) A tablet PC as control unit.

Figure 7.2: A setup in the ICG Vis-Center with Studierstube augmented reality environment, stereo projection and a tablet PC as control device. The seed region was moved by an ART tracked wireless pen device. This setup was developed in the course of the lecture "Virtual Reality" in summer term 2006 [Schmalstieg2006a].

than with control draggers in the views itself. Maybe an external control device would be useful.

- Certainly, even larger setups with several display devices which give different information of the datasets can be tried out. However, it has to be considered that large setups are expensive and the available space for diagnostics is always restricted. Nonetheless, providing different information with different devices in an intuitive and fast way should be one of the main focuses of further developments.

7.5 Clinical Evaluations

Finally most of the made assumptions for the measurement and further calculations are not proved completly. To allow a use of this kind of measurement and its interpretations in a clinical routine studies with healthy and probably diseased subjects should be performed even with a comparing measurement of the flow and pressure states with catheter based methods. A collateral evaluation of the most instructive and diagnostic useful presentation of flow data for physicians should be accomplished as well.

Chapter 8

Conclusions

We investigated the visualization capabilities of four-dimensional cardiovascular flow as measurable by magnetic resonance phase-contrast imaging. This thesis summarizes the steps from the measurement process to a preprocessing toolbox to a scene-graph based medical image viewer which we extended with advanced GPU-accelerated visualization techniques.

Due to an accurate and intuitive workflow we were able to present the flow data and its corresponding morphological image data directly after the measurement with fast configurable and combinable views. To give a possible usage of these visualizations for diagnostics, we had to aim for interactive manipulations of the scenes and hence for highly efficient rendering algorithms. Consequently, we tested programmable graphics hardware with different performance and a data-flow scene-graph library called CashFlow. Additionally, three volume rendering approaches to present the morphological background data were discussed with our medical partners. To them, rendering the measured images directly with a Gauss-distribution weighted transparency value on top of each other appeared to be the most intuitive morphological visualization.

For flow visualizations we concentrated on sparse representations of the velocity fields. These, mostly particle trajectory based approaches, appeared to be more instructive to our medical partners and were more suitable for acceleration techniques in contrast to dense, texture based algorithms. We implemented glyph and color mapped arrow plots, different kinds of direct particle rendering and stream lines and path lines, both with illumination techniques and a stream tube impostor render upgrade, applicable to particle trajectories. A color mapping gradient editor was developed to ease the definition of transfer functions to characterize different flow parameter.

To evaluate the accuracy of the implemented flow visualization techniques we built a flow phantom with an artificial narrowing to obtain a clear and concise steady flow over time. We could demonstrate that all developed algorithms yield to a comprehensible result. Due to a not achieved steady flow, a mathematical proof of the measured flow values and resulting visualizations was not possible yet. This issue should get feasible after several improvements of the flow phantom.

Beside the improvement of known sparse flow visualizations, we developed a framework with free arrangeable and runtime definable render areas. Commonly known Inventor script files can be used to define a customized visualization using hardware accelerated calculation and visualization algorithms. Furthermore, we can export all parameter of a view arrangement to a XML-based configuration file so that a custom workspace can be restored at any time.

Our medical partners preferred in the end a basic but new intersection plane visualization technique which uses the direction of the velocity values as normal vector for a plane's texels intersecting the flow volume. Onto the plane itself the magnitude of the velocity value is color mapped. Consequently, it was possible to combine 3D information in a 2D plane by using a headlight and the basic knowledge that "The more it reflects, the more it flows towards the observer.". This visualization technique was also the fastest we have implemented and is therefore combinable with all other visualizations with nearly no decrease of performance.

Jarke J. Van Wijk wrote in [Wijk2005] that there is a growing gap between the research community and its prospective users and that researchers have to search for and enumerate possible actions of users after using their prospective tools. If such actions cannot be found or defined the value of visualization is doubtful. Both was kept in mind during the development of the presented application. A close collaboration with the radiology department of the Landesklinikum Graz and the Diagnostikzentrum Graz turned out the requirements for a usage of these visualizations in clinical routine. Furthermore possible visualization supported diagnostic methods were discussed but needs still to be evaluated with matching patient studies. A cooperation with the Siemens Medical Solutions department enabled this project to result in a complete workflow as shown in chapter 4. Especially for such a complex flowing system as given in a human heart, no automated diagnostic systems exist so far. Hence, the investigation of new algorithms to detect health-related flow patterns will be of special interest in future work. An additional visualization of interesting parts can be of value for faster diagnostics and would consequently reduce the diagnostic costs or could give hints for more accurate measurement parameters.

Acknowledgements

Cordial thanks to my parents who actually made this thesis possible. Their motivation and encouragement brought me to the large amount of knowledge I currently own. Thank you, Pia for your patience and love during the six years of my course of studies. You prevented me from resigning and always believed in me.

A special thank to Gert and Ursula Reiter whom no hour was too late to measure anything imaginable at the Landesklinikum Graz. Without the contribution of Martin Urschler we would not have worked together for about three years in the meantime. Also a million thanks to you, Martin for this fruitful collaboration.

Not to forget Dieter Schmalstieg and Horst Bischof who first enabled this project. They helped me with the official cooperation procedure with Siemens and our medical partners. Dieter Schmalstieg contributed additionally to his charge as supervisor with several concepts from his unexhaustible pool of ideas. Furthermore thanks to my second supervisor, Michael Kalkusch for his active assistance for the software design and the preparation of the Cash-flow library; and even for the good portion of hardening. Furthermore, Denis Kalkofen who was so kind to give me advice and additional hints as well. Additionally three cheers for Marc Streit, who helped me with the finalization and motivation during the writing and implementation process.

Finally a special thank to my friends, who always provided enough beer and boozed up discussions to forget the stress and exertions of the gray everyday student life. I am looking forward to celebrate my final degree with you!

Appendix A

Intermediate File Format Definition

In this chapter the detailed file format is defined for files which are generated by the Siemens MED 4D-Flow Toolbox. These files contain all necessary information for a visualization of PC-MRI datasets. Artifact corrected field data is stored in *data files* as their headers are described in section A.2. *Master files* organize these data files as shown in section A.1.

A.1 Master File

This central file consists of two sections, a header and a string section. The file extension for master files is ".4dm". The structure of the header is given in table A.1.

The total length of this fixed-length part is 40 bytes. Directly after that part an array with numSlices offset/length pairs is stored to indicate position and length of the paths to the data files. Every pair looks like table A.2.

A.2 Data Files

Data files contain information about the slice and meta information for every image of the slice. The file extension for data files is ".4dd". The header of each file looks like the description of table A.3.

The header's fixed-length part has a size of 60 bytes. After the header section the image headers are stored.

Offset	Name	Type	Description
0x00	magic	int	Magic word to recognize the file as master file. Always contains the value 0x4df1c0de
0x04	headerLen	short int	Length of the fixed-length part of the header. In case the length is larger than expected (for example when reading a newer file version) the reader should skip the extra fields. Version 1 has a headerLen of 40 bytes.
0x06	mode	short int	Mode of the DICOM data (PARALLEL_1D = 0, PARALLEL_2D = 1, PARALLEL_3D = 2, GRID_3D = 3)
0x08	version	unsigned char	Version of this file format. This specification describes version 1
0x09	compatVersion	unsigned char	The version the reader at least has to understand to be able to parse that archive. Set to 1 in this version.
0x0A	dicomPathOffset	short int	The offset to the DICOM path string calculated from the beginning of the string section (bytes)
0x0C	dicomPathLen	short int	The length (in Unicode characters) of the DICOM path string.
0x0E	numSlices	short int	The number of slices associated with this master file. Equivalent to the number of data files.
0x10	stringSectionOffset	int	Offset to the start of the string section.
0x14	maxVelocity	float	The maximum velocity magnitude (in cm/s) in all the phase-contrast data files. This field is used to calculate the proper mapping from velocities to colors in the visualization part
0x18	preferredBlock	short int	Index to the preferred block. The images of this block are related to the vector field data. This field is set to 0 in all parallel modes.
0x1A	dicomPatientIndex	unsigned short	Index of the patient inside the DICOM archive
0x1C	dicomNumStudies	unsigned int	Number of DICOM studies
0x20	dicomNumImages	unsigned int	Number of DICOM images
0x24	checksum	int	CRC32 checksum of this header (excluding the checksum, of course)

Table A.1: The fixed-length part of the master file header.

Offset	Name	Type	Description
0x00	offset	short int	Offset (in bytes) to the start of the path to a data file. Those paths are relative to the location of the master file. Normally these strings will just contain the filename, as the whole archive is stored inside a single directory. Note that those offsets are stored relative to stringSectionOffset.
0x02	length	short int	Length of the path string in Unicode characters.

Table A.2: These are the offset/length pairs to indicate position and length of the paths to the data files.

Depending on the measurment mode there will be up to six images (therefore also up to six image headers) stored per time. Every single one will look like table A.4.

Subsequently, there are numVectorFields fields stored, every single one exactly $numRows * numColumns * sizeof(vector)$ long. They are stored without any padding bytes in between. A short header introduces each field as shown in table A.5.

All contours referenced from the images in this slice are stored in the drawings section. It contains a set of contours. Due to the variable lengths of drawings the fixed-length header of every drawing contains a length field to allow skipping of drawings. Note that the start of all drawings is aligned to address divisible by 8 without remainder. Table A.6 defines such an entry.

The regions are stored right after this header. Difference regions where two polygons define one common region have to be considered as well. This slightly complicates the matter of table A.7.

Following this region header the polygon vertices are stored (first the inside vertices, then the outside vertices). In this case it is enough to store polygon vertices as floating point row/column values, as the regions always lie in the image plane. Every vertex will be stored as shown in table A.8. The last vertex in this list will be connected to the first to close the polygon.

All noise correction masks referenced from the images in this slice are stored in the masks section. Every mask field contains an unique mask key for the correct image-association. The creation values (0x04 - 0x10) are stored to support the same settings for the noise correction dialog by reloading in the calculation tool. The mask header is 12 bytes long. Following the header from table A.9 the mask is stored. Due to the binary character of the masks, the mask will exactly need the space of the image size. (Each pixel one bit). If every image in a 3D-parallel mode was noise corrected, there will be an $Images/3$ mask count. Otherwise the number of masks is equal to the number of images.

Offset	Name	Type	Description
0x00	magic	int	Magic word to recognize the file as data file. Always contains the value 0x4df1da7a.
0x04	headerLen	int	Length of the fixed-length part of the header. In case the length is larger than expected (for example when reading a newer file version) the reader should skip the extra fields. Version 1 has a headerLen of 52 bytes. The version of the archive is stored in the master file.
0x08	imageSectionOffset	int	Offset to the start of the image section.
0x0C	numImages	int	The number of image headers following this file header.
0x10	vectorFieldOffset	int	Offset to the start of the vector data section. Set to 0 if no vector data is present.
0x14	numVectorFields	unsigned short	Number of vector fields stored in the vector field section. Only != 0 if a vector field is stored.
0x16	numLoD	unsigned short	Number of LoD layers per field stored in the velocity field section. Only != 0 if a vector field is stored.
0x18	numIntersections	int	Number of intersections stored in 3D grid mode. Only != 0 if a vector field is stored and we are in grid mode.
0x1C	drawingSectionOffset	int	Offset to the start of the drawing section.
0x20	numDrawings	int	Number of drawings stored in the drawings section.
0x24	stringSectionOffset	int	Offset to the start of the string section.
0x28	masterFilePathOffset	int	Offset to the start of the path-string of the master file relative to the start of the string section (bytes). The string always contains a relative path, most likely just a filename, as the master file is in the same directory.
0x2C	masterFilePathLen	int	Length of this path string in Unicode characters.
0x30	maskSectionOffset	int	Offset to the start of the mask section
0x34	numMasks	int	Number of masks stored in the mask section
0x38	checksum	int	CRC 32 Checksum over this header (excluding the checksum, of course).

Table A.3: The header of each data file.

Offset	Name	Type	Description
0x00	headerLen	int	Length of the image header.
0x04	normalVecX	float	x Coordinate of the ImageKey's normal vector
0x08	normalVecY	float	y Coordinate of the ImageKey's normal vector
0x0C	normalVecZ	float	z Coordinate of the ImageKey's normal vector
0x10	positionX	float	x Coordinate of the ImageKey's position vector (patient coordinate system)
0x14	positionY	float	y Coordinate of the ImageKey's position vector (patient coordinate system)
0x18	positionZ	float	z Coordinate of the ImageKey's position vector (patient coordinate system)
0x1C	triggerTime	float	Trigger time/index of the Image (also cached in the ImageKey). Note that this doesn't really correspond to the DICOM trigger time field. It basically is an index calculated using the DICOM trigger time field.
0x20	imageType	int	Type of the image (REPHASED = 0, THROUGH_PLANE = 1, IN_PLANE_V_FH = 2, IN_PLANE_V_AP = 3, IN_PLANE_V_RL = 4) acquisationTime float Image Acquisation Time, but decimal-cutted, because of no need of the ms. (Image Type is already given)
0x24	windowCenter	float	Center of the Window
0x28	windowWidth	float	Width of the Window
0x2C	zoomFactor	float	Current image zoom factor
0x30	xOffset	float	Current image x offset
0x34	yOffset	float	Current image y offset
0x38	venc	float	Encoding velocity (in cm/s)
0x3C	vadj	float	Aliasing adjustment velocity (in cm/s)
0x40	c1	float	Baseline correction parameter c1
0x44	c2	float	Baseline correction parameter c2
0x48	c3	float	Baseline correction parameter c3
0x4C	c4	float	Baseline correction parameter c4
0x50	c5	float	Baseline correction parameter c5
0x54	c6	float	Baseline correction parameter c6
0x58	vbl	float	Baseline correction parameter vbl
0x5C	drawingKey	unsigned int	Key to the image's drawing (unique)

0x60	xImOffset	signed char	Offset of the current image in pixels and x-direction for image registration corresponding to the thp or rl Image.
0x62	yImOffset	signed char	Offset of the current image in pixels and y-direction for image registration corresponding to the thp or rl Image.
0x64	ImageType	int	Determines the kind of REPHASED Image. 0 = PC, 1 = THP, 2 = AP, 3 = FH, 4 = RL
0x68	noiseLevel	unsigned int	The estimated noise level of the Image
0x6A	maskKey	unsigned int	Key to the image's mask.
0x6C	registered Image	bool	If this is the registered base image, the flag is true
0x6D	dicomFilenameOffset	unsigned int	Offset (in bytes) to the start of the DICOM filename of this image relative to the start of the string section. Contains an absolute path to the DICOM file.
0x71	dicomFilenameLen	unsigned int	Length of this path string in Unicode characters.
0x75	checksum	int	CRC 32 Checksum over this header (excluding the checksum, of course).

Table A.4: The image header. The length of this header is 117 bytes.

Offset	Name	Type	Description
0x00	numColumns	unsigned short	Number of columns for this layer
0x02	numRows	unsigned short	Number of rows for this layer

Table A.5: The velocity field header.

Offset	Name	Type	Description
0x00	drawingKey	int	The key the images use to refer to this drawing. An unique integer to be used as key in the drawing map.
0x04	drawingLength	int	Length of the whole drawing data structure (without trailing padding bytes due to alignment!).
0x08	numRegions	short int	The number of regions stored in this drawing.
0x0A	drawingShared	short int	Boolean flag. Any value != 0 means the drawing is shared.

Table A.6: The drawings entry.

Offset	Name	Type	Description
0x00	type	short int	Region type (BASELINE = 0, SEGMENTATION = 1, SEGMENTATION_DIFF= 2)
0x02	numInVertices	short int	Number of vertices defining the inside polygon for difference regions. This is also used to store the number of vertices for the other regions.
0x04	numOutVertices	short int	Number of vertices defining the outside polygon. Only used for difference regions, in all other cases this should be set to 0.
0x06	padding	short int	Padding bytes, set to 0.

Table A.7: The regions entry.

Offset	Name	Type	Description
0x00	columnCoordinate	float	Floating point column coordinate of this vertex.
0x04	rowCoordinate	float	Floating point row coordinate of this vertex.

Table A.8: The vertex entry.

Offset	Name	Type	Description
0x00	maskKey	int	The key the images use to refer to this mask. An unique integer to be used as key in the mask map.
0x04	xMaskSize	int	If in gird mode only between the outboard intersections is masked, this field stored the size of the correction mask in x-direction. Else this value is equal to x-size of one image.
0x08	yMaskSize	int	If in gird mode only between the outboard intersections is masked, this field stored the size of the correction mask in y-direction. Else this value is equal to y-size of one image.
0x0C	padding	short	Padding bytes, set to 0

Table A.9: The noise correction mask header.

Appendix B

UML Class Diagrams In Detail

This chapter supplements chapter 5 with more or less complete UML class diagrams.

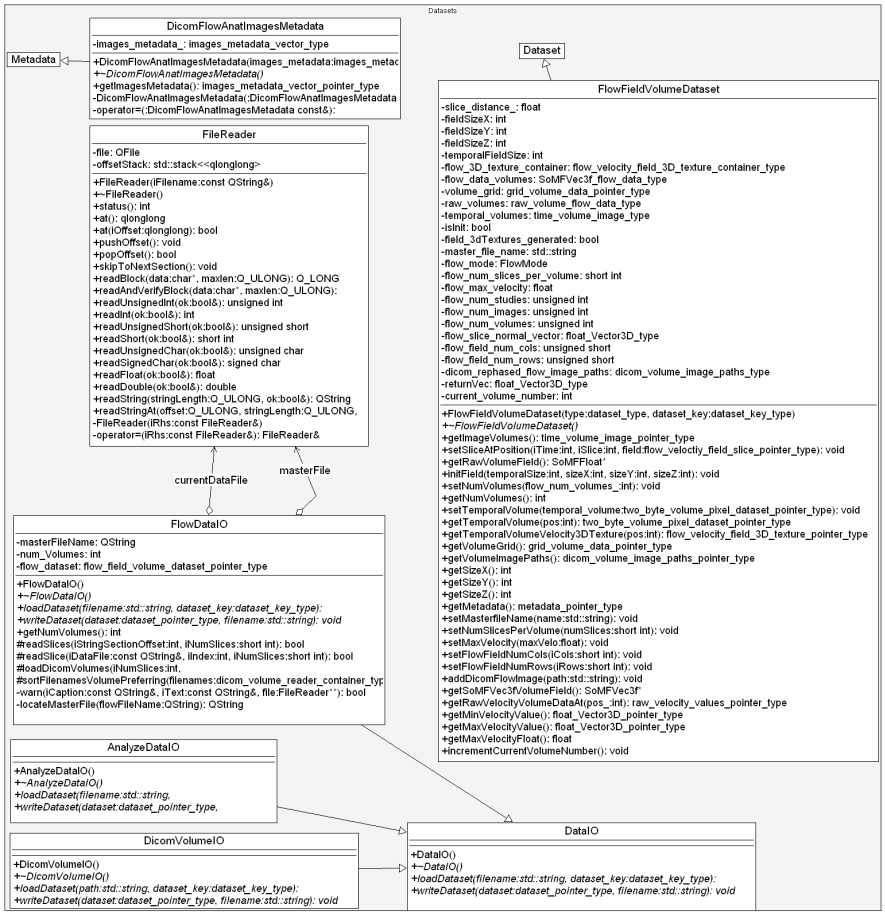


Figure B.1: A dataset for flow data and its interfaces and inheritances as part of an UML class diagram.

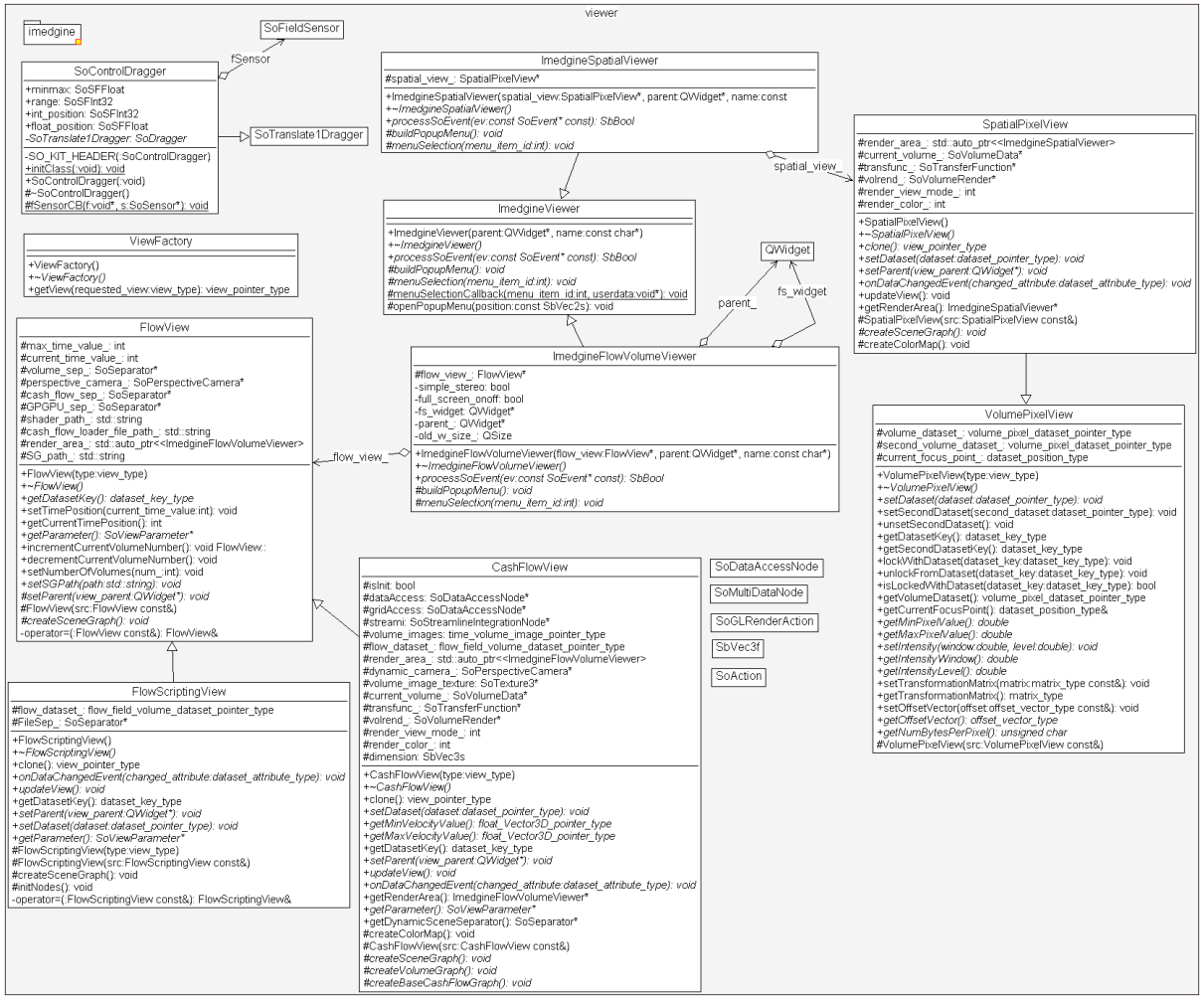


Figure B.2: UML class diagram for Viewer classes with special interest in flow scripting- and Cash-flow Views.

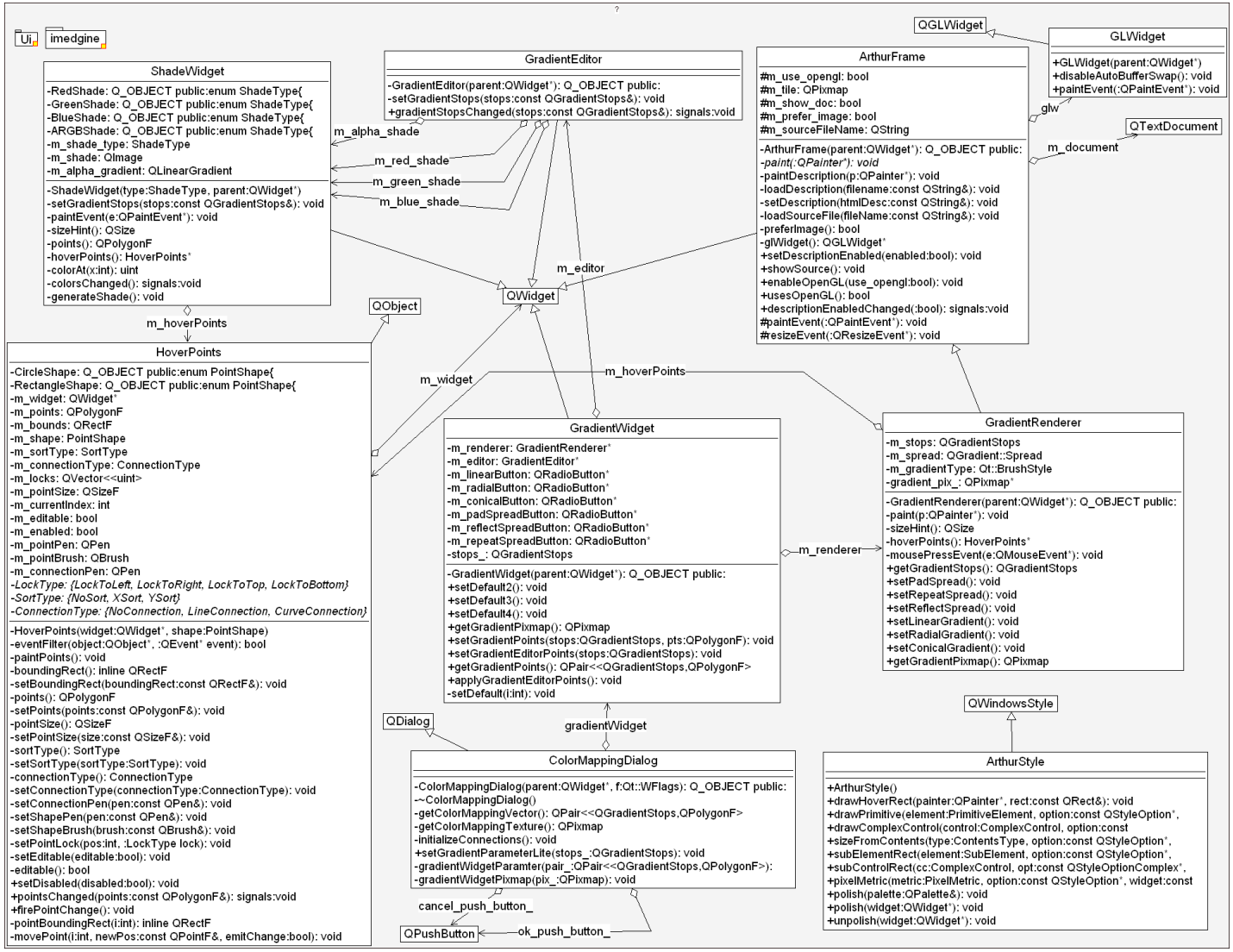


Figure B.3: An thinned out UML class diagram showing the underlying widgets of the Color gradient editor dialog.

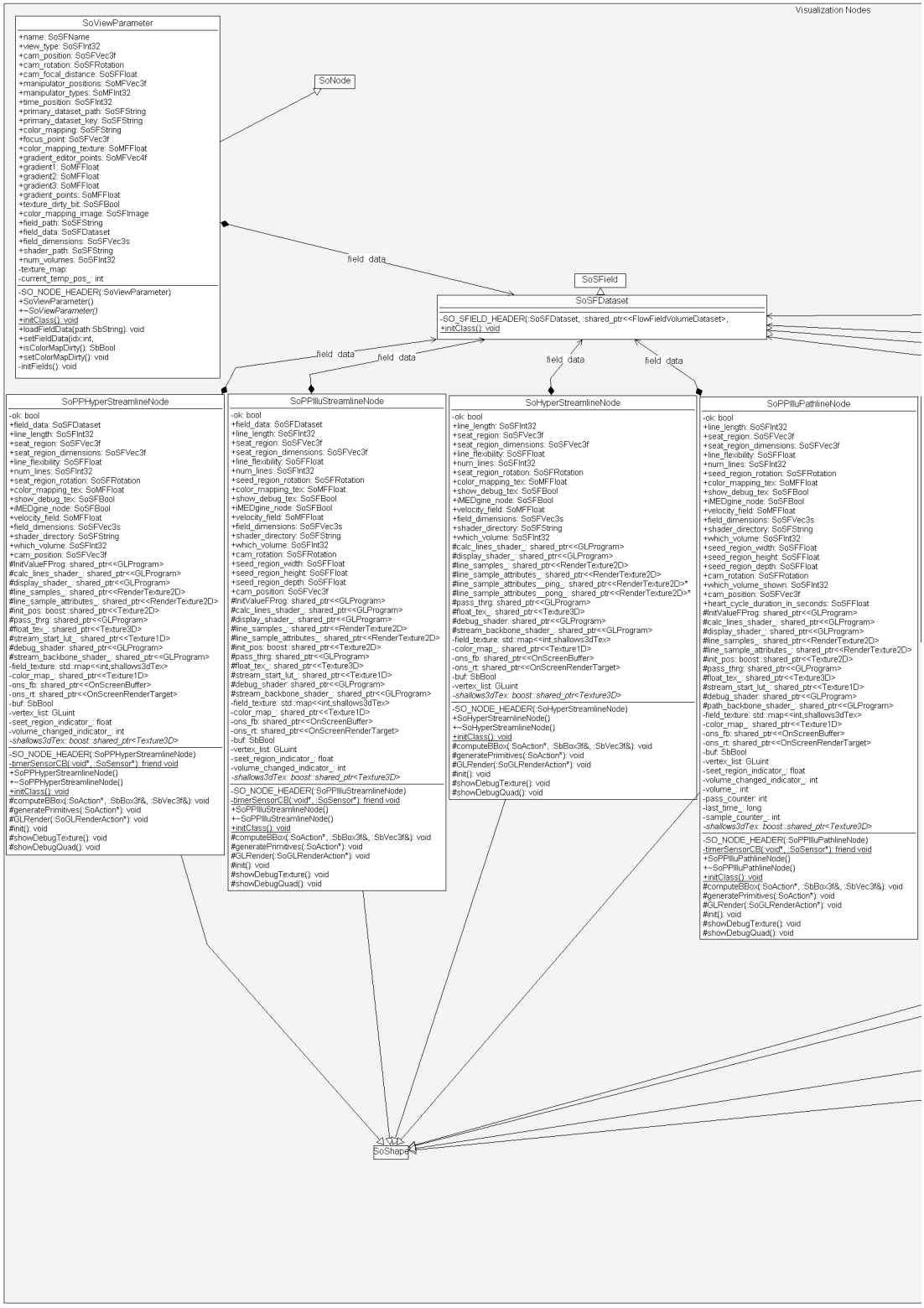


Figure B.4: The left side of figure B.5

Figure B.5: This UML class diagram gives an overview of the additional implemented visualization nodes. The left part of this diagram is given in figure B.4.

Glossary

Notation	Description	
B_0	A strong homogeneous magnetic field which cause a small total magnetization of a spin-system. The order of magnitude is about 0.5...3$[Tesla]$ for clinical usage and up to 7$[Tesla]$ in experimental setups.	42
v_{enc}	**V**elocity **ENC**oding value. A reference velocity value which has to be determined in advance for flow-sensitive PC-MRA to acquire a correct mapping from the phase angle of moving spins to the range of values.	58
3D texture	In contrast to a 2D texture with coordinates u, v a three dimensional texture consists of n-2D textures providing a third index w. Modern graphics hardware support these types of texture. Their Visualization is not trivial and many approaches to that issue exist.	15
alpha value	is a computer graphics expression for the amount of transparency of a pixel.	10
ALU	Arithmetic-Logic Unit. A part most microprocessors.	30

Notation	Description	
bipolar gradient pulse	A bipolar gradient pulse is one in which the gradient is turned on in one direction for a period of time then turned on in the opposite direction for an equivalent amount of time. A positive bipolar gradient pulse has the positive lobe first and a negative bipolar gradient pulse has the negative lobe first. The area under the first lobe of the gradient pulse must equal that of the second. A bipolar gradient pulse has no net effect on stationary spins. Spins which have a velocity component in the direction of the gradient will be effected by the bipolar gradient pulse.	56
cash flow	a scientific visualization toolkit developed by Michael Kalkusch [Kalkusch2006]	78
coronal	medical direction; viewing from front or behind	36
CPU	**C**entral **P**rocessing **U**nit. This is a programmable microprocessor device in a personal computer and others devices for general purpose computations.	34
DICOM	**D**igital **I**maging and **Co**mmunications in **M**edicine. A file format to store medical image data.	61
discharge flow-through	The flow through a certain area during a certain time.	36
DoF	Degrees of Freedom	36
ECG	**E**lectro**c**ardio**g**ram. A curve recorded from the electrical activity of the heart over time.	53
endocardium	is the innermost layer of cells in the heart	36
epicardium	describes the outer layer of heart tissue	36
FID	Free Induction Decay. Can be measured after the disorganization of a magnetization vector by a RF-pulse with parallel to B_0 placed coils.	42
fragment	A candidate for a pixel shown on screen after passing the pixel-pipeline.	30

Notation	Description	
GPU	**G**raphics **P**rocessing **U**nit. A dedicated microprocessor for computer graphics.	29
gradient field	Basically a gradient field is a field which modulates a constant field B so that the magnetic flux density is lowered in one direction and raised in the opposite direction. The effect of a gradient field in NMR is that the Larmor frequency for magnetized matter is changed depending on the slope of the gradient. In Example a gradient is used to select a certain slice of a Volume. Applying a RF-pulse with concurrently switched on gradient in z-direction will only excite that slice for which the resonance condition is fulfilled.	45
Hermitian	A Hermitian signal is symmetric and can be transformed to a Signal with Im = 0 with the inverse flourier transformation. The latter one is only one of the benefits of a Hermitian signal.	45
I/O	is a synonym for Input - Output behavior.	72
Iso-Surface	A triangle mesh surface whose supporting points are placed at identically intensity values (Iso-value) in a volumetric dataset. To obtain these points the popular Marching Cubes algorithm can be utilized for example.	19
Iso-value	A threshold value, for example to find the border between two different tissues in a volumetric dataset. The popular Marching Cubes algorithm, for example, can be utilized to envelop all positions with the defined Iso-value with a surface triagle mesh.	19
LIC	**L**ine **I**ntegral **C**onvolution. A Technique to convolute a vector field with a noise texture.	27
LVOT	**L**eft **V**entricular **O**utflow **T**ract	59
magnetization vector	A helping construct to characterize the total magnetization of a spin-system placed in a homogeneous magnetic field.	42

Notation	Description	
MIP	**M**aximum **I**ntensity **P**rojection referrs to a certain transfer function in direct volume rendering techniques and concurrently to the whole volume rendering process when used for angiography. This transfer function returns the maximum occurring value sampled from a volumetric dataset along a certain ray and is therefor especially suited for contrast enhanced vessel scans.	16
MR	Magnetic Resonance	37
MRI	Magnetic Resonance Imaging is based on the principals of NMR. The method was called 'Magnetic Resonance Imaging' instead of 'Nuclear Magnetic Resonance Imaging' (NMRI) because of the negative connotations associated with the word nuclear in the late 1970's.	36
myocardium	is the muscular tissue of the heart	36
NMR	Nuclear Magnetic Resonance	36
NMRI	Nuclear Magnetic Resonance Imaging	36
Nucleus	The core of an atom, consisting of neutrons and protons. plu.: Nuclei	36
Open Inventor	Open Inventor, originally IRIS Inventor, is a C++ object oriented retained mode 3D graphics API designed by SG to provide a higher layer of programming for OpenGL. Its main goals are better programmer convenience and efficiency.[Hartley1998]	72
PACS	**P**icture **A**rchiving and **C**ommunications **S**ystem. An image archiving- and communication system based on PCs and networks used in many hospitals.	61

Notation	Description	
parallel mode	This term is used for a special type of PC-MRI measurement. In contrast to a grid-mode acquisition, where three images with each through plane components are measured and subsequently intersected, a parallel mode phase-contrast image triplet consists of two in plane velocity encoding components and one through plane for three dimensional datasets (3D-parallel mode)	63
particle tracing	Methods for calculations on further flow field visualizations is based on the movement of massless particles injected into a velocity field.	21
pathological	Pathology is the study of the processes underlying disease and other forms of illness, harmful abnormality, or dysfunction	36
pulse sequence	A pulse sequence is a preselected set of defined RF and gradient pulses, usually repeated many times during a scan, wherein the time interval between pulses and the amplitude and shape of the gradient waveforms will control NMR signal reception and affect the characteristics of the MR images'. from [MR-TIP]	45
Rendering	is the process of generating an image from a model, by means of computer programs.	34
Resonance	Exchange of energy between two systems at a certain frequency.	36
RF-pulse	**R**adio-**F**equency-**Pulse**. A time varying magnetic field to excite magnetized matter - or rotate a magnetization vector.	41
RGBA	is the abbreviation for a **R**ed-**G**reen-**B**lue-**A**lpha values quadruple.	95
saggital	medical direction; viewing from the left or right side	36
SAR	**S**pecific **A**bsorption **R**ate. An integral magnitude which refers to the absorbed energy of tissue per time. SAR must not be higher than a warming of tissue more than $1°C$ would be provoked.	46

Notation	Description	
shader	A small program which is mostly compiled during runtime of the main application and executed on the GPU.	30
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata. Describes a processor which is able to rapidly perform a similar operation on multiple inputs.	31
spoiling table	Rapidly applied gradients with different strengths to destroy cohesive magnetization.	48
taxonomy	is the practice and science of classification.	21
TE	Echo Time. The Time at which an echo signal reaches its maximum.	45
texel	This term refers to a pixel in a texture with certain texture coordinates in one-, two- or three dimensions.	105
transfer function	A function which influences the integration along a ray through a volumetric dataset to obtain a certain projection of the volume. Depending on this function different tissues in medical datasets can be completely suppressed or highlighted during visualization.	16
transversal	medical direction; viewing from above or below	36

List of Figures

List of Tables

References

[Abragam1989] A. Abragam. *Principles of Nuclear Magnetism*. Oxford University Press, **1989**. 35, 42

[Abrahams2007] Dave Abrahams, Darin Adler, Ed Brey, Herv Brnnimann, Fernando Cacciola, Greg Colvin, Beman Dawes, Joel de Guzman, Peter Dimov, and Robert Ramey et al. *Boost c++ libraries*. http://www.boost.org/, **2007**. Boost provides free peer-reviewed portable C++ source libraries. 74

[Albrecht1996] Peter Albrecht. *The runge-kutta theory in a nutshell. SIAM J. Numer. Anal.*, 33(5):pages 1712–1735, **1996**. ISSN 0036-1429. 23

[Arnold1998] M. Arnold. *Half-explicit runge-kutta methods with explicit stages for differential-algebraic systems of index 2. BIT Numerical Mathematics*, 38:pages 415 – 438, **1998**. LIDO-Berichtsjahr=1999,;. 24

[Banks1994] D.C. Banks. *Illumination in diverse codimensions*. In *SIGGRAPH '94*, pages 327–334. ACM Press, **1994**. 27, 28

[Barlow1956] H.B. Barlow. *Retinal noise and absolute threshold. J Opt Soc Am.*, 46(8):pages 634–639, **1956**. PMID: 13346424 [PubMed - indexed for MEDLINE]. 10

[Barten1992] Peter G. J. Barten. *Physical model for the contrast sensitivity of the human eye. Proceedings of SPIE*, 1666:pages 57–72, **1992**. 10, 19

[Fraps2007] beepa P/L ACN 106 989 815. *Fraps, real-time video caputre and bechmarking*. http://www.fraps.com/, **2007**. 113

[Bloch1946] Felix Bloch. *Nuclear induction. Physical Review*, 70(7 And 8):pages 460–475, **1946**. 35

[Brill1994] M. Brill, W. Djatschin, M. Hagen, S. V. Klimenko, and H.-C. Rodrian. *Streamball techniques for flow visualization. IEEE Proceedings Visualization '94*, -:pages 225–231, **1994**. 22, 28

[Bruckner2007] S. Bruckner and M.E. Groeller. *Style transfer functions for illustrative volume rendering.* In D.Cohen-Or and P.Slavk (editors), *EUROGRAPHICS 2007*, volume 26. **2007**. 13, 14

[Buyens2003] Fanny Buyens, Odile Jolivet, Alain de cesare, Jacques Bittoun, and Alain Herment. *Calculation of left ventricular relative pressure distribution in mri using acceleration data. magnetic resonance in medicine*, 53:pages 877–884, **2003**. 128

[Cabral1993] Brian Cabral and Leith Casey Leedom. *Imaging vector fields using line integral convolution.* In *SIGGRAPH: ACM Special Interest Group on Computer Graphics and Interactive Techniques International Conference on Computer Graphics and Interactive Techniques archive Proceedings of the 20th annual conference on Computer graphics and interactive technique*, pages 263 – 270. ACM Press, **1993**. 27, 29

[Mostbeck1992] Gerhard H. Mostbeck Gary R. Caputo and Charles B. Higgins. *Mr measurement of blood flow in the cardiovascular system. AJR, American Roentgen Ray Society*, 159:pages 453–461, **1992**. 36

[Chernyaev1995] E. Chernyaev. *Marching cubes 33: Construction of topologically correct isosurfaces*, **1995**. 18

[VisualStudio2005] Microsoft Cooperation. *Microsoft visual studio 2005.* http://msdn2.microsoft.com/de-de/vstudio/default.aspx, **2005**. IDL. 82

[Cg2007] Nvidia Cooperation. *Cg Language Specification.* Nvidia, **2007**. 29

[Nvidia2005] Nvidia Developer. *Nvidia gpu programming guide.* www.nvidia.com, **2005**. 32

[Dunne1990] S. Dunne, S. Napel, and B. Rutt. *Fast reprojection of volume data.* In *Proc. First Conf. Visualizationin Biomedical Computing*, pages 11–18. Atlanta, GA, **1990**. 17

[Ebbers2001] Tino Ebbers, Lars Wigstrm, Ann F. Bolger, Jan Engvall, and Matts Karlsson. *Estimation of relative cardiovascular pressure using time-resolved three-dimensional phase contrast mri. Magnetic resonance in medicing*, 45:pages 872–879, **2001**. 128

[ClincalMRI2006] Robert R. Edelman, John R. Hesselink, Michael B. Zlatkin, and John V. Crues III. (editors). *CLINICAL MAGNETIC RESONANCE IMAGING*. Sauners Elsevier Inc., **2006**. 1, 16, 17, 35, 54

[Edelstein1987] W.A. Edelstein, J.F. Schenck, O.M. Mueller, and C.E. Hayes. *Radio frequency field coil for nmr*. U.S. Patent 4,680,548, **1987**. 35

[Engel2002] K. Engel and T. Ertl. *Interactive high-quality volume rendering with flexible consumer graphics hardware*. In *Eurographics '02*. **2002**. 17

[Engel2006] Klaus Dieter Engel. *Real-time volume graphics*. Peters, Wellesley, Mass., **2006**. ISBN 1-56881-266-3. 12, 15, 16, 17

[Euler1768] Leonhard Euler. *Institutionum calculi integralis volumen primum in quo methodus integrandi a primis principiis usque ad integrationem aequationum differentialium primi gradus pertractatur.*, volume 2. St. Petersburg Academy, **1768**. 23

[Fernandes2007] Antnio Ramires Fernandes. *Opengl shading language tutorial*. http://www.lighthouse3d.com/opengl/glsl/index.php?intro, **2007**. In this tutorial shader programming using GLSL will be covered. Shader are a hot topic and 3D games have shown that they can be put to good use to get remarkable effects. This tutorial aims at providing an introduction to the world of shaders. 74, 76

[FernGPU2004] Randima Fernando. *GPU gems*. Addision-Wesley, Boston [u.a.], **2004**. ISBN 0-321-22832-4. 29

[Frits1993] Theo van Walsum Frits H. Post. *FLUID FLOW VISUALIZATION*. Springer Verlag, Berlin, **1993**. 22

[Laub1998] M. Drobnitzky G. Laub, J. Gaa. *Magnetic resonance angiography techniques. electromedica 66*, 2:pages 68–75, **1998**. 36

[GbR2007] ComputerBase Medien GbR. *Geforce 8 architektur und benchmarks*. http://www.computerbase.de/artikel/hardware/grafikkarten/, **2006**. In german. 29

[Gee2005] A.G. Gee, M. Yu, and G.G Grinstein. *Dynamic and interactive dimensional anchors for spring-based visualizations*. Technical report, Computer Science, University of Massachussetts Lowell., **2005**. 6

[iMedgine2006] T. Gross, M. Streit, M. Urschler, A. Lex, M. Reininger, and C. Koerner. *http://imedgine.org/*. Internet, **2006**. Developted at the Institute for computer graphics and vision. 70, 77, 82

[Haase1986] A. Haase, J. Frahm, and D. Matthaei. *Flash imaging. rapid nmr imaging using low flip angle pulses. Magnetic Resonance in Medicine*, 67:pages 258–266, **1986**. 35

[Shallows2007] Trond R. Hagen, Jon Mikkelsen Hjelmervik, and Johan S. Seland. *Shallows making gpgpu programming fast and easy.* Website, Sourceforge, **2007**. Http://shallows.sourceforge.net/. 71, 82

[Hartley1998] Robert Hartley. *Open inventor. Linux J.*, 1998(53es):page 2, **1998**. ISSN 1075-3583. 71, 149

[Hauser2003] Helwig Hauser. *Third lecture on visualization, tu wien, vrvis.* Lecture, http://www.vrvis.at/via/staff/hauser/, **2003**. Http://www.vrvis.at/via/staff/hauser/. 12

[Hauser2005-2006] Helwig Hauser. *Lecture on visualization.* Lecture TU Wien, WS 2.0, 186.004, **2005 - 2006**. 25

[Hauser2002] Helwig Hauser, Robert S. Laramee, and Helmut Doleisch. *State-of-the-art report 2002 in flow visualization*, **2002**. 6

[Hauser2001] Helwig Hauser, Lukas Mroz, Gian Italo Bischi, and Eduard Grller. *Two-level volume rendering. IEEE Trans. Vis. Comput. Graph.*, 7(3):pages 242–252, **2001**. 16

[He1996] Taosong He, Lichan Hong, Arie E. Kaufman, and Hanspeter Pfister. *Generation of transfer functions with stochastic search techniques.* In *IEEE Visualization*, pages 227–234. **1996**. 13

[Heidrich1995] Wolfgang Heidrich, Michael McCool, and John Stevens. *Interactive maximum projection volume rendering.* In Gregory M. Nielson and Deborah Silver (editors), *Visualization '95*, pages 11–18. IEEE Computer Society Press, **1995**. 16

[Hinshaw1983] W.S. Hinshaw and A.H. Lent. *An introduction to nmr imaging: From the bloch equation to the imaging equation. Proceedings IEEE*, 71:pages 338–350, **1983**. 36

[SoQt2007] Systems in Motion. *Soqt library.* http://www.coin3d.org/lib/soqt, **2007**. Provides the glue between Systems in Motion's Coin high-level 3D visualization library and Troll Tech's Qt 2D user interface library. 82

[SystemsInMotion2007] Systems in Motion AS. *Coin, the core 3d rendering library.* Website, **2007**. 2, 71, 72, 77, 78, 82, 87

[Qt42] Trolltech inc. *Qt: Cross-platform rich client development framework, version 4.2.* http://trolltech.com/products/qt, **2007**. 82

[Interrante1998] C. Interrante, V. Grosch. *Visualizing 3d flow. Computer Graphics and Applications, IEEE*, 18(4):pages 49–53, **1998**. 29

[SGI2007] OpenGL is a registered trademark of SGI. *Opengl, immediate mode 3d graphics library.* http://www.opengl.org/, **2007**. 82

[Jakob2006] P.M. Jakob. *Klassische 'spin-gymnastik'.* Lecture, **2006**. Lehrstuhl fr Experimentalphysik, Universitt Wrzburg. 35

[Hornak2006] Ph.D. Joseph P. Hornak. *The Basics of MRI.* Rochester Institute of Technology, **2006**. 35

[Kainz2006] Bernhard Kainz. *Heartbeatbox white paper.* http://www.icg.tu-graz.ac.at/Members/club_kainz/WhitePaper/download, **2006**. 100

[Kalkusch2005] Michael Kalkusch. *A Visualization Framework for 3D Flow Data.* Master's thesis, TU Vienna, **2005**. 2, 9, 26, 77, 82, 85

[GrabnerKalkusch2006] Michael Kalkusch and Markus Grabner. *Lecture on computer graphics "ausgewaehlte kapitel aus computer grafik" ws 2006/2007.* Lecture, **2006**. 25, 33

[Kalkusch2006] Michael Kalkusch and Dieter Schmalstieg. *Extending the scene graph with a dataflow visualization system.* In *VRST06*, 13. **2006**. 71, 77, 147

[Kerwin2007] Thomas Kerwin. *Medical image processing.* http://www.cse.ohio-state.edu/ kerwin/itk-medical.html, **2007**. Images from a MRI dataset of a sheep's heart. 16, 22

[GLSL2004] John Kessenich, Dave Baldwin, and Randi Rost. *THE OPENGL SHADING LANGUAGE*, revision 59 edition, **2004**. 29

[Kipfer2004] Peter Kipfer, Mark Segal, and Rüdiger Westermann. *Uberflow: a gpu-based particle engine.* In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 115–122. ACM Press, New York, NY, USA, **2004**. ISBN 3-905673-15-0. 23

[Kniss2002a] Joe Kniss, Charles Hansen, Michel Grenier, and Tom Robinson. *Volume rendering multivariate data to visualize meteorological simulations: a case study.* In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 189–ff. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, **2002**. ISBN 1-58113-536-X. 13

[Kniss2001] Joe Kniss, Gordon L. Kindlmann, and Charles D. Hansen. *Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets.* In Thomas Ertl, Kenneth I. Joy, and Amitabh Varshney (editors), *IEEE Visualization*. IEEE Computer Society, **2001**. 13, 14

[Kniss2002] Joe Kniss, Gordon L. Kindlmann, and Charles D. Hansen. *Multidimensional transfer functions for interactive volume rendering. IEEE Trans. Vis. Comput. Graph.*, 8(3):pages 270–285, **2002**. 15

[Kouwenhoven1995] Marc Kouwenhoven, Mark B. M. Hofman, and Michiel Sprenger. *Motion induced phase shifts in mr: Acceleration effects in quantitative flow measurements - a reconsideration. Magnetic Resonace in Medicine*, 33:pages 766–777, **1995**. 69

[Siemens2003] Randall Kroeker. *Siemens Application Notes: Triggered-Retrogated Cine With Arrhythmia Rejection, (Works-in-Progress), Magnetom Numaris 4 A21B, 01/03.* Siemens MR R&D, Orlando Lon Simonetti Siemens MR R&D Chicago Orlando.simonetti@siemens.com, **2003**. 53

[Krueger2005] Jens Krueger, Peter Kipfer, Polina Kondratieva, and Rudiger Westermann. *A particle system for interactive visualization of 3d flows. IEEE Transactions on Visualization and Computer Graphics*, 11:pages 744 – 756, **2005**. 23

[Kutta1901] M.W. Kutta. *Beitrag zur näherungsweisen Integration oder Differentialgleichungen. Zeit. Math. u. Phys.*, 46:pages 435–453, **1901**. 23

[QGradientDemo2007] Trolltech Labs. *Gradients demo source code.* http://doc.trolltech.com/4.2/demos-gradients.html, **2007**. 96

[Lacroute1994] Philippe Lacroute and Marc Levoy. *Fast volume rendering using a shear-warp factorization of the viewing transformation.* In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages

451–458. ACM Press, New York, NY, USA, **1994**. ISBN 0897916670. 17

[Laur1991] David Laur and Pat Hanrahan. *Hierarchical splatting: a progressive refinement algorithm for volume rendering.* In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 285–288. ACM Press, New York, NY, USA, **1991**. ISBN 0-89791-436-8. 17

[Levoy1988] M. Levoy. *Display of surfaces from volume data. Computer Graphics and Applications, IEEE*, 8:pages 29–37, **1988**. 15

[Levoy1990] Marc Levoy. *Efficient ray tracing of volume data. ACM Trans. Graph.*, 9(3):pages 245–261, **1990**. 15

[Levoy1990a] Marc Levoy. *Volume rendering by adaptive refinement. The Visual Computer*, 6(1):pages 2–7, **1990**. 15

[Liang1999] Z.-P. Liang and E. M. Haacke. *Magnetic resonance imaging. Chapt. in Encyclopedia of Electrical and Electronics Engineering (Webster, ed.; John Wiley & Sons)*, 2:pages 412–426, **1999**. 35, 41

[Liu2002] Z. Liu and R. Moorhead. *Auflic: An accelerated algorithm for unsteady flow line integral convolution*, **2002**. 27

[Lorensen1987] William E. Lorensen and Harvey E. Cline. *Marching cubes: A high resolution 3d surface construction algorithm. Computer Graphics (SIGGRAPH 87 Proceedings)*, 21(4):pages 163–169, **1987**. 18

[Lu1999] Zhong-Lin Lu and Barbara Anne Dosher. *Characterizing human perceptual inefficiencies with equivalent internal noise. J Opt Soc Am A Opt Image Sci Vis., JOSA A*, 16:pages 764–778, **1999**. 10

[Bernstein1998] Bernstein MA, Zhou XJ, Polzin JA, King KF, Ganin A Pelc NJ, and Glover GH. *Concomitant gradient terms in phase contrast mr: Analysis and correction. Magnetic Resonance in Medicine*, 39:pages 300–308, **1998**. 69

[Malzbender1993] Tom Malzbender. *Fourier volume rendering. ACM Trans. Graph.*, 12(3):pages 233–250, **1993**. ISSN 0730-0301. 17

[Markl2006] Michael Markl. *flow in mri - introduction & application.* In *ESMRMB 'Perfusion and Flow', September 2006.* **2006**. 36

[Markl2003] Michael Markl, Marcus T. Alley, Frandics P. Chan, Kris L. Wedding, Mary T. Draney, Chris J. Elkins, David W. Parker, Ryan Wicker, Charles A. Taylor, Robert J. Herfkens, et al. *Time-resolved three-dimensional phase-contrast mri. Journal of Magnetic resonance imaging*, 17:pages 499–506, **2003**. 36

[Max1995] Nelson L. Max. *Optical models for direct volume rendering. IEEE Trans. Vis. Comput. Graph.*, 1(2):pages 99–108, **1995**. 15

[Moeller1996] Torsten Mller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. *Classification and local error estimation of interpolation and derivative filters for volume rendering.* In *VVS*, pages 71–. **1996**. 17

[Montani1994] Claudio Montani, Riccardo Scateni, and Roberto Scopigno. *A modified look-up table for implicit disambiguation of marching cubes. The Visual Computer*, 10(6):pages 353–355, **1994**. 18

[MR-TIP] MR-TIP. *Magnetic resonance - technology information portal.* Website, **2007**. Http://www.mr-tip.com/. 35, 53, 150

[NEMA2003] NEMA. *Digital imaging and communications in medicine (dicom) , nema standards publications ps 3.4-2003.* Technical report, National Electrical Manufacturers Association (NEMA), Rosslyn, Virginia 22209 USA, **2003**. 10

[DICOM2007] http://medical.nema.org/ NEMA. *The dicom format.* website, **2007**. 10

[Nielson1991] G.M. Nielson and B. Hamann. *The asymptotic decider: resolving the ambiguity in marching cubes. IEEE Conference on Visualization '91, Proceedings.*, -:pages 83–91, 413, **1991**. 18

[Nvidia2006] Nvidia. *Nvidia geforce 8800 architecture technical brief.* Technical brief, Worlds First Unified DirectX 10 GPU Delivering Unparalleled Performance and Image Quality, **2006**. NVIDIA GeForce 8800 GPU Architecture Overview. 30

[Eindhoven2002] Technical University of Eindhoven. *Sequoiaview.* Website, **2002**. Www.win.tue.nl/sequoiaview. 6, 7

[ITK2006] National Library of Medicine of the National Institutes of Health. *Insight segmentation and registration toolkit (itk).* open-source, Berkely-style license, **1999-2003**. Seen on http://www.itk.org, 11.2006. 65, 71, 78, 82, 84

[Partanen1995] Ritva Vanninen Keijo Koivisto Harri Tulla Hannu Manninen Kaarina Partanen. *Hemodynamic effects of carotid endarterectomy by magnetic resonance flow quantification. Stroke, American Heart Association, Inc.*, 26:pages 84–89, **1995**. 69

[Pascucci2004] V. Pascucci. *Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping.* In *Joint Eurographics - IEEE TVCG Symposium on Visualization (VisSym),*, pages 293–300. **2004**. 28

[Peeters2006] Tim Peeters, Anna Vilanova, Gustav Strijkers, and Haar Romeny. *Visualization of the fibrous structure of the heart.* In *VMV 2006, Aachen.* **2006**. 27, 28

[Pfister2001] Hanspeter Pfister, William E. Lorensen, Chandrajit L. Bajaj, Gordon L. Kindlmann, William J. Schroeder, Lisa Sobierajski Avila, Ken Martin, Raghu Machiraju, and Jinho Lee. *The transfer function bake-off. IEEE Computer Graphics and Applications*, 21(3):pages 16–22, **2001**. 12

[Pharr2005] Matt Pharr. *GPU Gems 2.* Addison-Wesley, **2005**. 29, 30, 32, 33

[Purcell1946] E.M. Purcell, H.C. Torray, and R.V. Pound. *Resonance absorbtion by nuclear magnetic moments in a solid. Physical Review*, 69(1-2):pages 37–38, **1946**. 35

[Reina2006] G. Reina, K. Bidmon, F. Enders, P. Hastreiter, and T. Ertl. *Gpu-based hyperstreamlines for diffusion tensor imaging.* pages 35–42, **2006**. 127

[Reiter2007] Dr. Gert Reiter. *Personal communication*, **2007**. MRI. 67, 68, 153

[Reiter2006] Gert Reiter, Ursula Reiter, Bernhard Kainz, Andreas Greiser, Horst Bischof, and Rainer Rienmller. *Mr vector field measurement and visualization of normal and pathological time-resolved three-dimensional cardiovascular blood flow patterns.* In *Poster presentation.* **2006**. 1, 22, 62, 66, 69

[Rezk-Salama1999] Christof Rezk-Salama, Peter Hastreiter, Teitzel Christian, and Thomas Ertl. *Interactive exploration of volume line integral convolution based on 3D-texture mapping.* In David Ebert, Markus Gross, and Bernd Hamann (editors), *IEEE Visualization '99*, pages 233–240. San Francisco, **1999**. 27

[Roth1982] Scott D. Roth. *Ray casting for modeling Solids. j-CGIP*, 18(2):pages 109–144, **1982**. 15

[Rothman2004] S. L. G. Rothman, R. B. Geehr, E. L. Kier, and H. B. Hoffman. *Multiplanar reconstruction as an aid in ct diagnosis. Neuroradiology*, 16:pages 596–597, **2004**. 11

[Rottger2003] Stefan Rttger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Straer. *Smart hardware-accelerated volume rendering.* In *VisSym*. Eurographics Association, **2003**. 15

[Runge1895] C. Runge. *Über die numerische Auflösung von Differentialgleichungen. Mathematische Annalen*, 46(2):pages 167–178, **1895**. 23

[Scharsach2005] Henning Scharsach. *Advanced gpu raycasting.* In *CESCG 2005*. **2005**. 15

[Schmalstieg2002] Dieter Schmalstieg, Anton L. Fuhrmann, Gerd Hesina, Zsolt Szalavri, L. Miguel Encarnao, Michael Gervautz, and Werner Purgathofer. *The studierstube augmented reality project. Presence*, 11(1):pages 33–54, **2002**. 128

[Schmalstieg2006a] Dieter Schmalstieg, Markus Grabner, and Michael Kalkusch. *Lecture on virtual reality.* Lecture SS2006, Institut fr Maschinelles Sehen und Darstellen (ICG), Technical University of Graz, **2006**. 128, 129

[Schulz1999] M. Schulz, F. Reck, W. Bertelheimer, and T. Ertl. *Interactive visualization of fluid dynamics simulations in locally refined cartesian grids.* In *Visualization '99. Proceedings*, pages 413–553. IEEE, **1999**. 28

[Scott1962] G. G. Scott. *Review of gyromagnetic ratio experiments. Review of Modern Physics*, 34(1):pages 102–112, **1962**. 35

[Shekhar2003] Raj Shekhar and Vladimir Zagrodsky. *Cine mpr: Interactive multiplanar reformatting of four-dimensional cardiac data using hardware-accelerated texture mapping.* In *IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE*, volume 7. **2003**. 11

[Shen1998] Han-Wei Shen and D. L. Kao. *A new line integral convolution algorithm for visualizing time-varying flow fields. IEEE Transactions on Visualization and Computer Graphics*, 4(2):pages 98–108, **1998**. 27

[UserLib2004] Siemens. *Mr user's library & tools.* C++ Library, **2004**. 65

[Slichter1990] C.P. Slichter. *Principles of Magnetic Resonance*. Springer-Verlag, **1990**. 35

[Stalling1997] D. Stalling, M. Zockler, and H.C. Hege. *Fast display of illuminated field lines. Visualization and Computer Graphics, IEEE Transactions on*, 3(2):pages 118–128, **1997**. 27

[Stollberger2007] R. Stollberger. *Nmr imaging and spectroscopy*. Lecture TuGraz, **2007**. 35, 36

[Streit2006] M. Streit and T. Gross. *imedgine presentation*. Institute of computer graphics and vision (ICG), TUGraz, Workshop, **2006**. 78

[Tatarchuk2004] Natalya Tatarchuk and Bill Licea-Kane. *Glsl real-time shader development*. In Wolfgang Engel (editor), *ShaderX3: Advanced Rendering Techniques in DirectX and OpenGL*. Charles River Media, Cambridge, MA, **2004**. 74, 76

[Thompson2002] Chris J. Thompson, Sahngyun Hahn, and Mark Oskin. *Using modern graphics architectures for general-purpose computing: a framework and analysis.* In *MICRO*, pages 306–317. ACM/IEEE, **2002**. ISBN 0-7695-1859-1. 32

[Tory2004] Melanie Tory and Torsten Moeller. *Rethinking visualization: A high-level taxonomy. IEEE Symposium on Information Visualization (INFOVIS'04)*, 1:pages 151–158, **2004**. 7, 8

[Treece1999] G. M. Treece, R. W. Prager, and A. H. Gee. *Regularised marching tetrahedra: improved iso-surface extraction. Computers & Graphics*, 23(4):pages 583–598, **1999**. 19

[Tricoche2004] X. Tricoche, C. Garth, G. Kindlmann, E. Deines, G. Scheuermann, M. Ruetten, and C. Hansen. *Visualization of intricate flow structures for vortex breakdown analysis. Visualization, 2004. IEEE*, 10-15 Oct.:pages 187–194, **2004**. 22

[Trolltech2007] Trolltech. *Qt4 http://www.trolltech.com/*. Website, **2007**. 78

[Upson1989] C. Upson, T.A. Faulhaber Jr., D. Kamins, D. Laidlaw, D. Schlegel J. Vroom, R. Gurwitz, and A. van Dam. *The application visualization system: a computational environment for scientific visualization. Computer Graphics and Applications, IEEE*, 9:pages 30–42, **1989**. 9

[Wijk2005] J.J. van Wijk. *Views on visualization. IEEE Transactions on Visualization and Computer Graphics*, 12(4):pages 421–433, **2006**. Best paper award. 6, 131

[Vivek1999] andAlexPang Vivek Verma, David Kao. *Plic: bridging the gap between streamlines and lic*. In *Visualization '99. Proceedings*. **1999**. 27, 29

[Voigt2002] Robert Voigt. *An Extended Scatterplot Matrix and Case Studies in Information Visualization*. Classification and definition of terms, Hochschule Magdeburg-Stendal, **2002**. 6

[Weiskopf2007] Daniel Weiskopf. *GPU-Based Interactive Visualization Techniques*. Springer-Verlag Berlin Heidelberg, **2007**. 2, 20, 22, 23, 24

[Zockler1996] Malte Zockler, Detlev Stalling, and Hans-Christian Hege. *Interactive visualization of 3d-vector field using illuminated stream lines*. In *Visualization '96. Proceedings.*, pages 107 – 113, 474. IEEE, **1996**. 26

Index