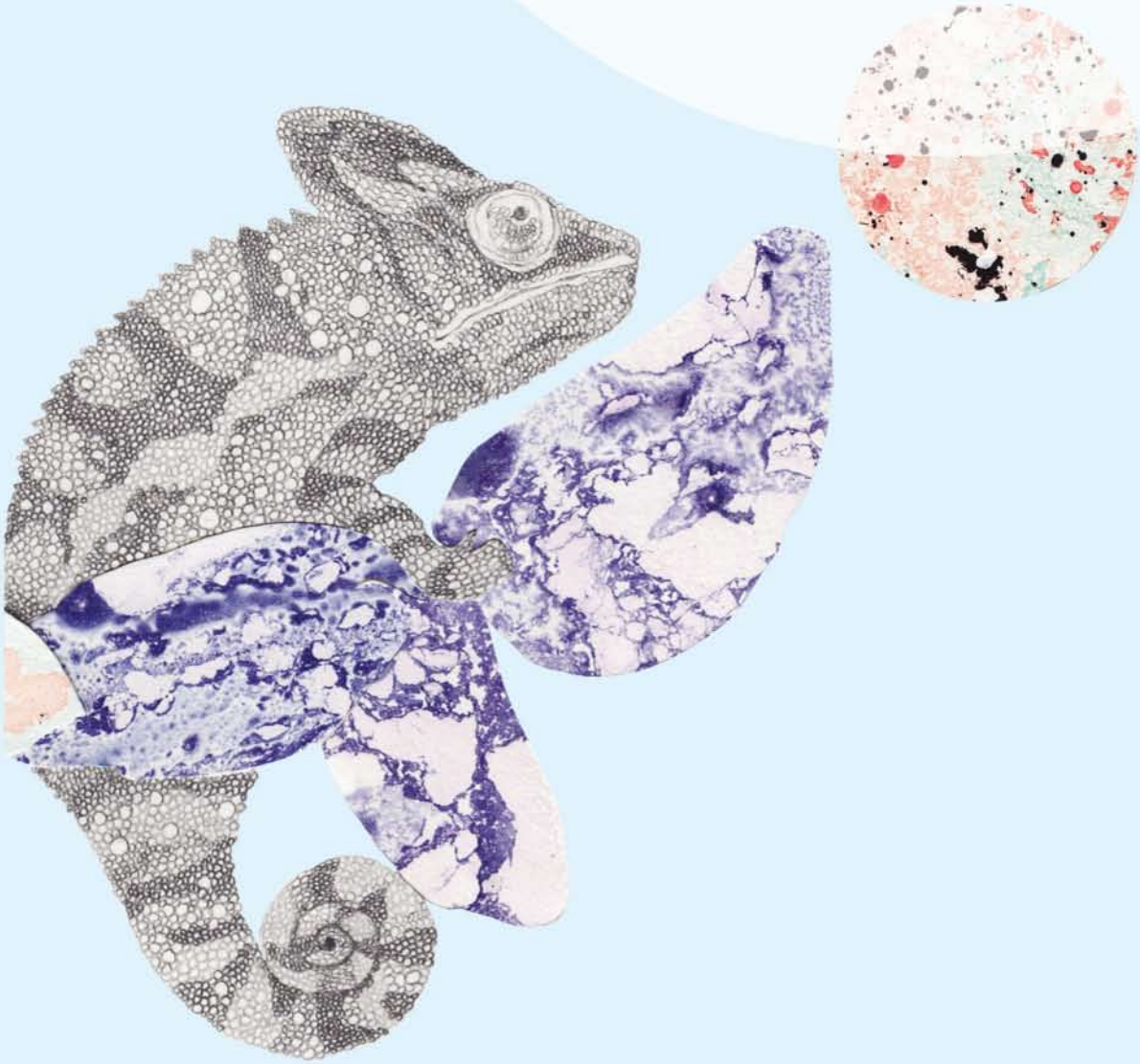
Visualization of Variation and Variability



Stef Busking

Visualization of Variation and Variability

Stef Busking

About the cover

Massive thanks to my sister, Jennifer, for creating the drawing on the cover. The two chameleons symbolize the difficulty of a standard side-by-side comparison, especially when the subjects are dynamic and not all differences are relevant.

Visualization of Variation and Variability

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties
in het openbaar te verdedigen op vrijdag 12 december om 12:30 uur

door

Stef BUSKING

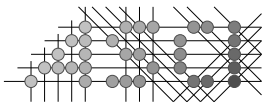
Master of Science in Computer Science & Engineering
Technische Universiteit Eindhoven
geboren te Vlissingen.

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. F.W. Jansen

Copromotor:
Dr. C.P. Botha

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. F.W. Jansen	Technische Universiteit Delft, promotor
Dr. C.P. Botha	vrml. Technische Universiteit Delft
Prof. dr. ir. B.P.F. Lelieveldt	Technische Universiteit Delft
Prof. dr. ir. J.J. van Wijk	Technische Universiteit Eindhoven
Prof. dr. J.B.T.M. Roerdink	University of Groningen
Prof. Dr.-Ing. Bernhard Preim	Universität Magdeburg
Dr. A. Vilanova	Technische Universiteit Delft



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 317.

Netherlands Organisation for Scientific Research

This project was supported by the Netherlands Organization for Scientific Research (NWO) as project number 643.100.503 “Multi-Field Medical Visualization”.

ISBN 978-94-6186-404-8

Production streamlined by Bas Busking grafisch ontwerper bno
Printed by Proefschriftmaken.nl / Uitgeverij BOXPress

To Lixin, Noah and Liya

Preface

This thesis describes research performed at the Computer Graphics and CAD / CAM group of the Delft University of Technology, as part of the "Interactive, Multi-Field Data Visualizations for Medical Applications" (MFMV) project. This project was supported by the Netherlands Organization for Scientific Research (NWO) and was part of the VIEW programme.

Stef Busking

Contents

CHAPTER 1

Introduction

The goal of visualization is providing insight into often complex data. As technology progresses, so do our means of acquiring more varied and detailed data. In medicine, for instance, modern imaging equipment can produce datasets consisting of multiple modalities, such as CT, MRI and fMRI. Studies often require data to be obtained from large numbers of individuals, both patients and healthy controls. Motion, growth and other developments can be studied by acquiring data on the same subjects at multiple time steps. Some techniques produce high-dimensional data, often by using a large number of parameters to model physical characteristics, such as object shapes. Even where data is not complex in itself, fusion of datasets creates ever growing combinations. Multi-field datasets, for example, are formed when existing data is combined through registration or when it is extended by adding from an endless variety of derived quantities, depending on the problem, approach and interests of the researcher.

The main purpose of introducing such complexity is that there are often interesting similarities or differences between the values from all these sources. The main problem is in discovering and analyzing such patterns. In general, a notion of *comparability* exists, specifying aspects of the data thought to represent different instances of the same class or phenomenon. For instance, a certain subvolume in an MRI volume can be identified as representing the brain, or groups of pixels in different digital images can be identified as belonging to the same object. While these represent in many ways "the same thing", the data for different instances of these features likely demonstrate different values. In the following, we define *variation* to refer to these different values as demonstrated by the datasets under consideration. In

understanding such differences, one often generalizes these instances to form a model of variation, which we will refer to as *variability*: the possibility, likelihood and extent of variations.

Comparison is understood in the following as the act of analyzing variation or variability based on two or more specific instances of the data. These can represent different positions within an individual, different individuals or even different populations. A special case of comparison is relating a single instance to an aggregation of instances, for example comparing a single shape to a model describing all possible shapes in a given population. An important part of comparison is determining which parts of the data are to be considered comparable, by creating a model of correspondence between the instances involved. This process is further detailed in section 1.1.

We will use the terms *difference* and *similarity* as opposites, referring to a measure of how far or close two instances are to being considered equal in terms of their corresponding values. *Change* is a specific class of difference which implies a difference within the same entity/spatial locality when compared over multiple points in time. It is often also used to indicate modeling such differences as a function of time. Various other terms exist which refer to instances of these. For example, growth is a type of change referring to an increase in size over time, and alignment is a type of similarity describing the direction or orientation of subjects in space.

Clearly, there is an enormous range of possible comparisons. In this thesis we will focus on spatial phenomena, such as object shapes, fluid motion and medical volume data. This means instances and variation can be described in terms of spaces, the dimensions thereof and the scale of features extracted from the data. Difference and similarity can be measured in terms of distances, which are scalar values usually composed from the differences in value measured along the dimensions (both physical and abstract) under consideration. While time is often used to define the entities involved in a comparison, it plays an implicit role in this thesis. Many comparison scenarios concerning events in time are covered, but time is never treated differently from any other source of variation. Two time points are treated the same as two different individuals, and time can be seen as a dimension in a feature space (see section 1.2.2) like any other attribute. We do limit ourselves to comparison of static entities, comparison of dynamic phenomena (e.g., motion) is left for future work. Where dynamic is mentioned in the context of this thesis it refers to the interactivity of the visualizations, not the data itself.

The goal of this research was to investigate how visualization can be used to provide insight into variation and variability. We focus on interactive techniques, based on the assumption that such techniques can provide insight into a dataset more readily than static images or non-interactive animation. We cover the range of comparisons from variability within one entity through variability between two or more entities to variability within a population. Comparison between populations is left as future work. Rather than trying to cover every possible type of comparison and creating overly generalized solutions we have selected specific instances of comparison,

introduced in section 1.3. For these, we present suitable visualizations to help identify and understand the data.

In the following sections, we first discuss the task of comparison on an abstract level. We identify different types of comparison in section 1.1, and present our main tools for exploring variation and variability in section 1.2. We conclude with short summaries of the remaining chapters of this thesis, which explore various instances of comparison and comparative visualization in detail.

It is customary to include a discussion of related work in the initial chapter. However, due to the disparate nature of the topics discussed in the chapters of this thesis, most related work is discussed in the relevant chapters. The research presented here has been executed in the context of the research project "Interactive, Multi-Field Data Visualization Techniques for Medical Applications". As such, research in the medical application area has been a major influence, not least including the work done by the other two PhD candidates on the project, J. Blaas [Bla10] and V. Prčkovska [Prc10]. Another important influence to our work has been the work by H. Doleisch, H. Hauser et al. in the area of Interactive Visual Analysis [ODHW12].

1.1 Comparison

Before we can discuss our solutions for comparison in specific contexts, it is important to understand our view on comparison at an abstract level. In most dictionaries, the verb *to compare* is defined as either representing things as similar, or as the act of analyzing things specifically to discover their similarities and/or differences. Given the definitions in the previous section we therefore use *comparison* to refer to *the mapping and understanding of variability.*

Comparison has a dual focus. On one side it is about understanding similarity, which leads to finding commonalities between the data instances. For example, similarity in shapes of features extracted from data from multiple patients could indicate a specific medical condition. On the other side, such common parts are rarely constant. It is therefore often also critical to understand the nature of differences around these commonalities, in order to be able to distinguish between entities matching the model and those that do not, the so-called outliers.

Difference is a relation between two entities. It can be described in a multitude of ways, such as the magnitude and direction of a deformation, the translation and rotation of two objects in their surrounding environment or the scale and amplitude of a texture pattern. Understanding differences often requires expressing them in terms of the different ways in which the instances under consideration can vary. A *decomposition* of variability can provide insight that only quantification could not.

Closely related to comparison is what we have named *the relevance problem*: usually, especially with real-world datasets, there are many more differences than we are actually interested in, such as noise, misalignment and conditions during the acquisition process. In medical imaging, the process of object alignment is called

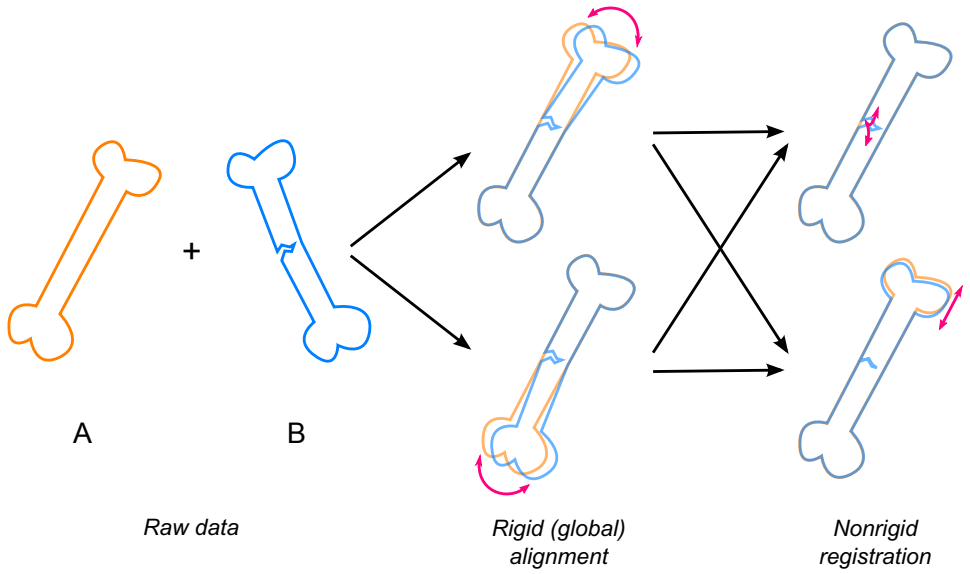


Figure 1.1: An example of domain matching, illustrating the relevance problem.

registration. We use the term *domain matching* to refer to the more generic case of bringing data into a common domain where comparison can take place. The goal of domain matching is therefore to remove all differences not relevant to the comparison. In practice, this is not possible (at least not 100%), as relevance is a highly application-specific concept which usually requires specific domain knowledge to determine and is in some cases (e.g., when exploring unknown phenomena) variable or even undefinable in advance of the comparison.

Figure 1.1 gives a simplified example of the relevance problem in relation to domain matching. Two input datasets are given, representing some bone at different points in time. As the B instance has been deformed, attempting to align the objects using only global transformations (middle) can yield different results – and possibly conclusions – about the same data. However, the rotation of part of the bone seen in these results is likely not relevant to the issue being studied. A domain matching process which allows local deformations (within limits, a process also known as deformable, nonrigid or elastic registration) can be applied to remove this difference from the data. However, this too can have different results. A deformable registration using the ends of the bones as a reference will show a hole in the middle (top right), while aligning the edges of this hole will show a slight reduction in the length of the bone (bottom right). Again, which of these differences is relevant depends on the goals of the user performing the comparison. Tweaking the registration process to yield one

result over the other often requires domain-specific knowledge and input.

Because of the difficulty in solving the relevance problem, most current approaches to comparison are application-specific, ad-hoc solutions. Obviously, generic techniques are preferable, as they can be reused for different applications. However, in order to understand differences and similarities, feedback must be given in terms of the application domain, for instance using visual representations specific to the data types, modalities or to the application field. In chapter 3, for example, we illustrate the value of integrating anatomical features as context into our (otherwise rather abstract) deformation field visualizations. Likewise, in chapter 4, we provide a view visualizing a selection in terms of the original components of the dataset, to allow users to better interpret that selection.

In this thesis, we use visualization to support comparison, as a means of gaining insight into variability. We investigate and present solutions for several applications, but also consider the wider applicability of the techniques used. We use three principal tools, discussed in section 1.2: comparative visualization, feature spaces and interactive exploration.

1.1.1 Types of comparison

Comparison can occur at a large range of scales, from points within a dataset through individual objects to comparing entire populations. The term *entity-level comparison* can be used to emphasize that a comparison concerns distinct entities. Closely related tasks are *population analysis*, which considers the higher level structure of a population, and *entity-population comparison*, which uses population analysis to relate specific individuals to the variability discovered in the population.

Actual comparative visualization applications may encompass all of these tasks. For instance, pairwise and one-to-many comparisons are often used as tools in population studies, of which examples are given in chapter 2 and chapter 5.

Entity-level comparison

Entity-level comparison is concerned with a limited number (usually two) of concrete individual entities. Such comparisons tend to focus on first establishing correspondence and then analyzing differences. Visualizations for this type of comparison are therefore aimed at showing relevant differences, given some definition of relevance for the problem domain.

Example questions to be answered for this type of task:

1. Domain matching: how can correspondence be established between the entities?
2. Filtering: What are all differences?
3. Analysis: How can these differences be explained? How can they be decomposed into the separate contributions from different causes?

Population analysis and entity-population comparison

Population-level analyses move away from analyzing individuals and instead focus on an aggregation of a (usually large) number of individuals as a continuous whole. Individual entities are now treated as samples of a larger population, which is often modeled in terms of a continuous subspace of the variability domain. The distribution of individuals in such a subspace can be modeled and analyzed statistically. Therefore, these tasks tend to focus on identifying structural patterns within this space, such as clusters, trends, biases or outliers. Individuals are generally only considered in relation to the population or structures therein. In other words, these comparisons (and the corresponding visualizations) are often mainly about identifying and analyzing interesting similarities.

Example questions to be answered for this type of task:

1. Domain matching: how can correspondence be established between the entities? Given such a notion of correspondence, is there a mathematical model which fits the entities under consideration?

2. Filtering: Can the model be decomposed to ignore irrelevant differences? Can the model be reduced (e.g., in terms of complexity and parameter space) while preserving relevant similarities?

3. Analysis: What are the trends within the model? Are there distinct clusters? Any outliers? How can these be explained in terms of the problem domain?

A related concept (though outside of the scope of this thesis) is *correlation*, which can be understood as analyzing similarity in variability structure rather than in physical aspects. This is often relevant when comparing entire populations.

1.2 Tools for understanding variability

The key approach for understanding variability explored in this thesis is interactive visual analysis [ODHW12] (IVA). We see this approach as an important first step in gaining insight into a new dataset, before quantitative analysis can be performed. Visual analysis can help locate features (patterns of interest) without initially requiring a great amount of understanding of the data. Based on the observable characteristics of features, interactive visual analysis can also be used as a stepping stone towards selecting and applying traditional modeling approaches and more focused feature extraction techniques.

As the name implies, important aspects of IVA are visualization and interactivity, as well as the use of a tight user feedback loop to control not only the visualization parameters but also the data selection, filtering and mapping methods used. The following sections will introduce the main techniques used in this thesis, which are often applied in an IVA approach.

1.2.1 Comparative visualization

The visualization part of IVA can take many forms. In this thesis, we focus on comparative visualizations, as they seem uniquely suited for data exploration and allowing comparison in context.

The term comparative visualization has been used for nearly any use of visualization in comparisons. For this section, we limit the definition to visualization solutions for entity-level comparison. While such solutions can be used in population-level comparisons, the goal of the visualization is always to compare concrete individuals. Such individuals can be actual measurements, but could also be constructed or synthesized, such as the mean of a population. Visualizations of more global aspects of the population, such as the characteristics of its distribution, fall under statistical and/or uncertainty visualization (see section 1.2.2).

Comparative visualization is a wide area of research, which includes many different applications, but also approaches [PP95]. As mentioned, a large part of these are ad-hoc solutions for a specific comparison problem. Side-by-side comparisons are common, which is quite interesting given the potential perceptual problems inherent in this approach due to the reliance on memory, such as *change blindness* [THM+10].

A pipeline for comparative visualization

In order to place the wide variety of approaches into a common context, we defined a pipeline for comparative visualization. This pipeline is based on the one presented by Pagendarm and Post [PP95], but expands the point of comparison.

The traditional visualization pipeline [HM90] forms the basis of our comparative visualization pipeline (figure 1.2). This pipeline consists of data acquisition, filtering (including feature extraction), mapping and rendering stages. In our comparative visualization pipeline, the output of each of these steps can serve as input to a comparison or "merge".

We refer to the point at which the traditional pipeline provides input to a merge as the *merge point*. Pagendarm and Post previously defined *data-level* comparison (merge point at the start of the pipeline) as well as *image-level* comparison (merge point located at the rendering stage). Verma and Pang [VP04] expanded their terminology by adding *feature-level* comparison, with the merge point located after the filtering stage. This leaves one more option: a merge occurring after the mapping stage, which we will call *representation-level* comparison.

The merge is a parallel process, which somewhat mirrors the steps of the traditional pipeline. The difference is that the merge can take input from any of the visualization pipeline steps, its steps may be distributed over the traditional pipeline in different ways, and the composited result could be fed back into the traditional pipeline at any point depending on its format. The steps of the merge are: domain matching, comparison filtering, comparison mapping and composition.

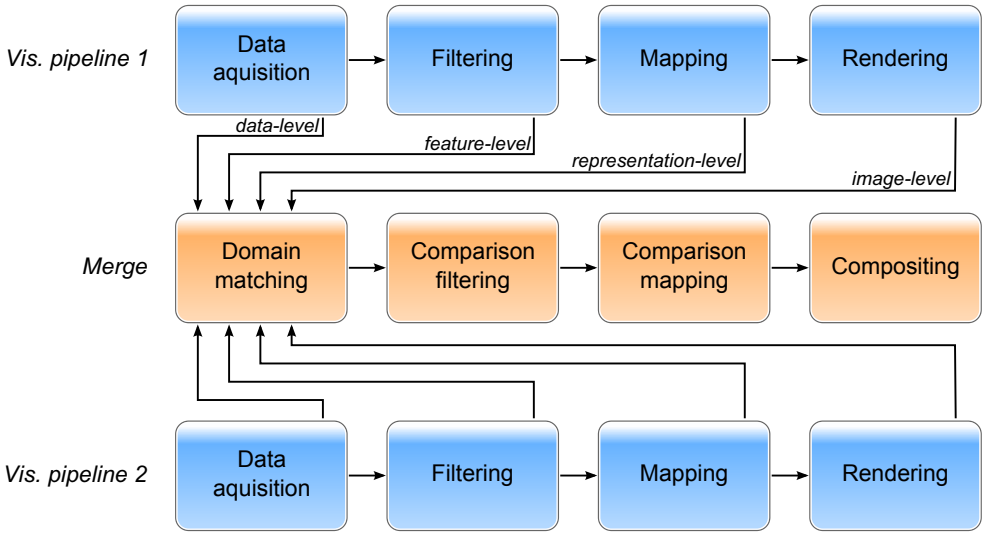


Figure 1.2: The comparative visualization pipeline.

Domain matching is where data is made comparable. This may include conversion of data to a common representation (e.g., normalization), as well as registration processes that are used to align dataset domains.

Comparison filtering is where the actual differences (or similarities, correlations etc.) are determined. Differences are typically features or derived values extracted from the registered data. There is a similarity between domain matching and comparison filtering: in both cases a transformation from one domain to the other needs to be determined. The difference is that in the first this is used to eliminate irrelevant differences, and in the second to extract relevant features. Finding the separation between these two stages is therefore the same as solving the relevance problem defined in section 1.1.

Comparison mapping mirrors the mapping stage in the traditional pipeline, and serves to assign visual representations to the differences extracted in the previous step. This could occur as part of the normal mapping process, but can also happen at other stages of the pipeline.

Composition is where the visual representations of differences are merged back into the traditional pipeline. Often, a focus+context approach is used where these differences are combined with visual representations that provide context to the comparison [CMS99]. Usually, composition occurs as part of rendering at the image level, but composition at other levels is also possible. For instance, difference feature geometry could be merged with the geometry resulting from the mapping stage, or a composition of data and difference volumes could be used in volume rendering.

Guidelines for comparative visualizations

Using this pipeline, we can define some important qualities of a good comparative visualization.

The visualization should be *explicit* whenever possible. That is, it should show relevant differences and/or similarities directly rather than leaving it up to the viewer to discover these. Visualizations that omit the comparison filtering and comparison mapping stages are said to be *implicit*, the most obvious examples of which are side-by-side visualization, superposition (e.g., "checkerboarding") and similar approaches. In extreme cases, even domain matching is omitted. The fact that side-by-side comparison, or "spot the differences", is a popular puzzle should speak to its lack of efficiency.

When comparative visualization is applied to a new problem, it may not be known initially which differences are going to be present and relevant for the specific application. In this case, targeted explicit comparative visualization is not possible. A good alternative here is to make the visualizations as *direct* as possible (i.e., keep them close to the structures of the original data). This way, researchers can iterate quickly on hypotheses and can more easily map discoveries in the visualization back to the problem domain. In terms of the pipeline, this means keeping the mapping stages simple.

A *focus and context* technique [Dol07] can be especially helpful in many comparative visualization scenarios. Like direct visualization, they enable quick mappings between the visualization representations and the problem domain. Focus and context techniques are usually implemented in the composition phase of the pipeline.

1.2.2 Feature spaces

Features spaces are a concept often encountered in the context of machine vision and pattern recognition. A feature space is a multi-dimensional space formed by considering each measurable attribute of the individuals in some population as a dimension. Individual objects are represented as points within this space, simply by combining their attributes into a single *feature vector*. Following from this definition, we can define variability as a distribution within the feature space: basically, the ways in which things can potentially differ are represented by the dimensions in a feature space. The extent of these differences can be measured as distances within the space, or modeled using statistical distributions. Feature spaces can therefore be used for entity comparison, but may be most useful for population analysis.

Depending on the field of research in which they are used, some concepts discussed in this thesis can have multiple names and vice versa. For this reason we will first present some definitions.

A *field* is an assignment of values to each point in a mathematical space, also called the *domain* of the field. A *multi-field* is a combination of several fields sharing a common domain. In feature extraction and tracking research, *features* are usually

defined to be objects, local behaviors or other patterns of interest in a dataset, whereas in many other fields they are the dimensions of a feature space or components of a multi-valued dataset. In order to resolve this ambiguity, we refer to the former as *feature objects*, while the components of a multi-field are referred to as *attributes*. This includes both components present in the original data as well as any derived quantities.

A multi-field can be seen in terms of its associated feature space, where each attribute of the field is considered as a dimension. This can include the original dimensions of the domain on which the field was defined: the feature space is a unification of the domain and range of the field. A sampling of the field can be transformed into a set of high-dimensional points in the feature space. In other cases, the feature space is often defined by considering one or more parameterizations of the entities under consideration. The dimensions of the feature space then correspond to the parameters of these parameterizations.

In general, an arbitrary number of different quantities describing some entity can be used as axes or dimensions in a high-dimensional space. The selection of suitable dimensions for a feature space is usually determined by the problem under consideration. The dimensions of the feature space are therefore often heterogeneous in nature, which means analysis approaches should take differences in scale and (possibly non-linear) relations between dimensions into account when attempting to discover and interpret interesting patterns within the space.

Nevertheless, for many cases simple geometric approaches such as distance metrics or projections can lead to important observations regarding the structure of a given population in its feature space. A well-known example is the common scatter plot visualization, where individuals are plotted as points by taking two dimensions of the feature space as axes. The scatter plot approach can be generalized to a larger number of axes in a number of ways. Plots for each pair of dimensions can be arranged in a scatter plot matrix, or more than two dimensions can be visualized in the same plot through projections. For instance, in chapter 5 we demonstrate an interactive implementation of star coordinates [Kan00,Kan01]. In chapter 4, we base our measure of dissimilarity on the standard Euclidean distance and use this to visualize patterns of similarity within a multi-field dataset. As demonstrated in these chapters, an important advantage of such methods is that the simplicity of their computation often enables interactivity.

Many feature space-based approaches have been developed in the context of pattern recognition and machine learning. These include techniques to transform and/or project a feature space to an alternative, possibly lower-dimensional basis, such as *principal component analysis* [Pea01] or *multidimensional scaling* [BG05]. Other techniques aim to classify individuals into separate groups or *clusters* based on the similarities in their attributes, or by partitioning their feature spaces into distinct areas [JMF99].

Finally, in statistics, analysis of sets of points in high-dimensional spaces is a

common topic. Many of these techniques start with modeling a population by fitting some statistical distribution (e.g., a high-dimensional Gaussian) based on the locations of known points in the feature space.

As feature spaces are applied in multiple fields of research, so are their visualizations. (Generalized) scatter plots and other high-dimensional techniques exist for more-or-less direct visualization of the points and distances within a feature space [WB97]. Indirect visualizations include results from applying a classification or clustering technique. In the statistical area, many visualizations have been proposed to show uncertainty and statistical distributions [Cle93].

1.2.3 Interactive exploration

The importance of visual continuity in visualization is well-known, and follows from research on change blindness and the Gestalt principles [Pal99]. As comparative visualization is all about discovering how one entity relates to another, continuity in such visualizations is even more important. While animation is a useful tool to provide such continuity, we advocate taking one step further by making interaction an essential part of the process. Such user-in-the-loop approaches, often referred to as *interactive visual analysis* [ODHW12] have been shown to enable quick exploration of data as well as fast iteration when interesting feature objects have been discovered and are to be analyzed.

On the technical side, such approaches call for rendering techniques that perform at interactive speeds. Image-based approaches are often better able to deliver the required performance, as their complexity is usually more strongly related to the size of the image than to the complexity of the scene being rendered. Furthermore, recent advances in GPU acceleration for graphics as well as general purpose computation enable ever more visualizations to be rendered in real-time, allowing for direct interaction rather than a slow "render, examine, adjust" loop.

Good interactive exploration approaches follow the Visual Information Seeking Mantra [Shn96]: overview, zoom and filter, details on demand. An important aspect of an interactive visualization approach is therefore the way in which users can quickly express their interest within the visualization. Object selection, mouse-driven exploration (i.e., interaction through simply pointing or hovering the mouse), and (possibly importance-driven) focus and context techniques are essential. The use of illustrative rendering can often help to suppress unnecessary detail, in order to make the visualization easier to comprehend. Contour lines, for instance, can be used to highlight specific objects or shapes, and cut-away views often introduce less visual complexity than using transparency on otherwise occluding surfaces. Van Pelt et al. recently used the zoom level itself as a trigger for switching the visualizations between overview and details-on-demand mode [vPGL+14].

Another approach which we have found to work well for interactive comparative visualization applications is the use of multiple linked views [GRW+00, Rob07]. As mentioned in section 1.2.1 it is important for a user to easily map findings and their

context between the visualization and problem domains. Keeping visualizations as direct as possible is one way to accomplish this, but an alternative is to use the linked views approach and let the application perform the mapping of selections, feature objects and/or positions between the various domains. For instance, when hovering the mouse over any of the views, the object under or position of the cursor could be mapped to the visualizations shown in the other views and rendered appropriately. This way, an application can combine both generic and application-specific visualizations, without the potential loss of relatability to the problem that a generic visualization might cause. In chapter 5, for instance, we allow the user to simply hover the mouse over a view representing a high-dimensional feature space of object shapes. We then use a different view to show the 3D shape represented by their cursor position, allowing straightforward exploration of this high-dimensional space.

Our focus in this thesis is in presenting new approaches for the interactive visualization of variability in different contexts. This means that in the techniques presented, performance is an issue only as far as achieving frame rates suitable for interactivity, and is therefore not necessarily maximized. Likewise, we apply our visualizations only to datasets which fit in (main or GPU) memory. A large body of work exists on out-of-core processing and rendering techniques, but application of these to our visualizations is left outside the scope of this discussion. Our applications and their usability are evaluated by illustrating specific use cases, and in the case of chapter 2, an expert evaluation.

1.3 Research questions and contributions

The contributions of this thesis consist of a number of solutions to exploring several aspects of variability in an interactive way. While these solutions have been applied to specific problems, they are usually more generally applicable. In parallel with developing these solutions, we have developed a software framework on which our prototype implementations have been built (see chapter 6).

In particular, we aim to answer the following research questions:

- How can we use an IVA approach for visualizing and exploring object shapes in a direct way? (chapter 2, chapter 5)
- How can we allow a user to interactively manipulate the composition of a visualization in order to clearly visualize variation and variability in their context? (chapter 3, chapter 4)
- How can we assist a user in an interactive application in order to overcome the relevance problem? That is, how can we interactively allow a user to indicate relevance and have the visualization react to this? (chapter 3, chapter 5, chapter 4)

The following sections give short summaries of each of the following chapters. In these, we state the specific contributions made in the chapter and describe how the chapter relates to the other parts of the thesis.

Chapter 2: Image-based rendering of intersecting surfaces for dynamic comparative visualization

Previously published as: Stef Busking, Charl P. Botha, Luca Ferrarini, Julien Milles, Frits H. Post, *Image-based rendering of intersecting surfaces for dynamic comparative visualization*, The Visual Computer 27(5), 2011, p. 247–363. [BBF+11]

- Entity-level comparison of two 3D surfaces
- Layered rendering

This chapter discusses direct comparison of 3D surfaces. We present a new implementation of the intersecting surfaces visualization [WT05] which can be applied to dynamic objects at interactive rates. Our implementation also contains several enhancements, addressing limitations of the original technique.

To test the effectiveness of the intersecting surfaces technique and our enhancements on dynamic objects, we apply our implementation to the visualization of statistical shape models. These are parameterized models of shape representing a population of similar objects. In our implementation, these models can be deformed interactively in order to explore the shape variability within the population. We also performed an expert evaluation of our visualization and enhancements with researchers who use these models in their daily work.

Finally, this chapter introduces the layered rendering pipeline, on which our implementation is based. This pipeline and derivatives of it were used for various implementations in the context of this thesis. A more thorough description of the pipeline is given in chapter 6.

Chapter 3: Direct visualization of deformation in volumes

Previously published as: Stef Busking, Charl P. Botha, Frits H. Post, *Direct visualization of deformation in volumes*, Eurographics / IEEE-VGTC Symposium on Visualization, Computer Graphics Forum 28(3), 2009, p. 799–806. [BBP09]

- Indirect comparison of two 3D volumes through direct, explicit visualization of domain mapping

In medical visualization, surfaces such as those used in the previous chapter are often derived from volume data. In this chapter we present a technique for comparing such volume datasets directly. We base our technique on non-rigid registration, a

domain-matching technique which allows (but attempts to minimize) local deformations in order to align volume datasets. We directly visualize the deformation field resulting from this registration, rather than transforming the volumes to a common domain. Due to its nature as a transformation from one volume to the other, this field captures all differences between the volumes.

We present a visualization of several specific features in the deformation field. Changes in volume, which are often of special interest in the medical context, are visualized by volume rendering. Context is provided by a contour visualization of structures in the scalar volumes, as well as by showing areas of high-magnitude deformation using texture convection, a flow visualization technique. We explore the use of importance-based composition as a technique for integrating focus and context in an interactive way.

Chapter 4: Example-based interactive illustration of multi-field datasets

Previously published as: Stef Busking, Charl P. Botha, Frits H. Post, *Example-based interactive illustration of multi-field datasets*, in Computers & Graphics 34(6), 2010, p. 719–728. [BBP10b]

- Entity-level comparison of points within a single multi-field
- Feature space as a basis for similarity

In the previous chapter, we highlighted areas with specific characteristics within a volume dataset. The work presented in this chapter generalizes that concept, while simultaneously introducing feature spaces as a powerful mechanism for dealing with multi-variate data.

By considering distance in feature space as a measure for dissimilarity, we allow example-based exploration of arbitrary multi-field datasets. Basically, we allow a user to simply click a point within the volume and instantly see all areas with similar characteristics, as well as which characteristics those are. We apply basic transformations such as non-uniform scaling and projection to the feature space to allow the user to fine-tune the application to their notion of similarity, by emphasizing or de-emphasizing characteristic and/or user-selected attributes of the selected points over others.

Chapter 5: Dynamic multi-view exploration of shape spaces

Previously published as: Stef Busking, Charl P. Botha, Frits H. Post, *Dynamic multi-view exploration of shape spaces*, in Eurographics / IEEE-VGTC Symposium on Visualization, Computer Graphics Forum 29(3), 2010, p. 973–982. [BBP10a]

- Interactive exploration of 3D shape models using multiple linked views

- Feature space visualization with immediate 3D shape feedback

Combining the feature space concept with the statistical shape models used in chapter 2, we present an interactive multi-view approach to exploring the parameter spaces of such models. A shape space is formed by considering all parameters of a given shape model as dimensions. Essentially, this results in a high-dimensional feature space.

We visualize the model in this space using a dynamic scatter plot projection of the high-dimensional points representing the objects in the shape population. This view is strongly linked to a 3D visualization of the object as represented by any point within the shape space. The user can simply point within the scatter plot and the application will use interpolation between the population shapes to interactively visualize the corresponding 3D object shape. Both views include relevant feedback on the statistical properties of the shape model, in order to explore such information both on the population-level (per object) and on the local level (per point).

Furthermore, we introduce a hybrid view in order to visualize and explore linear features such as trends within the shape space. This view combines two spatial dimensions with a third direction taken from the high-dimensional shape space, essentially combining various shapes along a trajectory through the shape space. Several different visualization configurations are presented, each providing a different view into the high-dimensional structure of the population and their interpretation as shape characteristics.

Chapter 6: NQVTK

- GPU-accelerated visualization framework

- Layered rendering / raycasting

This chapter discusses the software framework which has grown from and was used as the basis of all prototype implementations of techniques discussed in this thesis. Recent advances in graphics hardware have resulted in highly programmable GPUs. NQVTK is a framework enabling easy GPU-based prototyping of visualizations and rendering algorithms. It was built to easily interface with VTK [SML04] for data loading and processing. Using the NQVTK data-to-GPU framework, volume and mesh-based datasets can be efficiently transferred to the GPU and accessed directly within the code of a GPU shader program.

NQVTK also implements several rendering frameworks, including layered rendering (used in chapter 2) and layered ray casting (used in chapter 4). These are supported by several utility classes, providing simple scene graph functionality, access to advanced OpenGL functionality and a basic interaction framework.

Chapter 7: Discussion

The preceding chapters each discussed a specific application of visualization of variation and variability. The final chapter attempts to bring these together, identifying essential elements of such visualizations and addressing the research questions posed in this chapter.

The chapter concludes by generalizing the findings from the different cases explored in the thesis, and presents some views on moving forward from the work presented here.

CHAPTER 2

Image-Based Rendering of Intersecting Surfaces for Dynamic Comparative Visualization

Abstract

Nested or intersecting surfaces are proven techniques for visualizing shape differences between static 3D objects. In this chapter we present an image-based formulation for these techniques that extends their use to dynamic scenarios, in which surfaces can be manipulated or even deformed interactively. The formulation is based on our new layered rendering pipeline, a generic image-based approach for rendering nested surfaces based on depth peeling and deferred shading.

We use layered rendering to enhance the intersecting surfaces visualization. In addition to enabling interactive performance, our enhancements address several limitations of the original technique. Contours remove ambiguity regarding the shape of intersections. Local distances between the surfaces can be visualized at any point using either depth fogging or distance fields: Depth fogging is used as a cue for the distance between two surfaces in the viewing direction, whereas closest-point distance measures are visualized interactively by evaluating one surface's distance field on the other surface. Furthermore, we use these measures to define a three-way surface segmentation, which visualizes regions of growth, shrinkage, and no change of a test surface compared with a reference surface.

Finally, we demonstrate an application of our techniques in the visualization of statistical shape models. We evaluate our techniques based on feedback provided by medical image analysis researchers, who are experts in working with such models.

2.1 Introduction

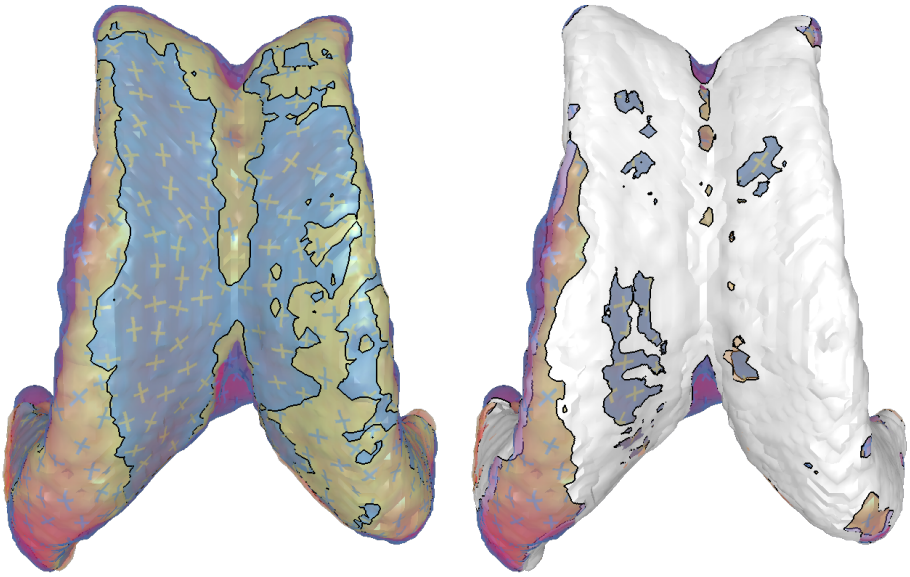
In this chapter, we examine one class of solutions to the problem of comparing the shapes of 3D surfaces. Comparison of data plays an important role in many areas of scientific research. Visualization can be useful to support comparative data analysis. The most common approach to comparative visualization of surfaces is a simple side-by-side display (with similar viewing conditions) of the two surfaces under consideration. Such an approach relies on memory to compare details of the surfaces, and local distances between surfaces are hard to estimate. We identify the following requirements for an effective comparative visualization:

- Differences should be made *explicit*, alerting the user to the presence and nature of all differences present.
- Visualization of differences should be *local*, showing not only the presence but also the precise location and extent of all differences. In medical applications, for instance, local information is required for understanding differences between patients or studying changes in specific biological structures over time.
- The visualization should be able to show *relevant* differences and hide irrelevant ones.

As an example of what is meant by relevance, consider the alignment of surfaces prior to comparison. Inaccuracies in the registration process can result in misalignment of the resulting surfaces. This misalignment, however, is usually not relevant to the researcher's problem. An ideal visualization could automatically distinguish between relevant and irrelevant differences, and show only the former. However, the notion of relevance is a highly application-dependent property, which can generally only be decided by the researcher. The use of user-feedback and *interactivity* is therefore essential to deal with this issue. Various applications can benefit from interactive visualization:

- User interaction or guidance in the registration process.
- Local registration for exploring differences between specific parts of the objects under consideration, ignoring more global differences.
- Analysis and comparison of dynamic or deformable surfaces.

In this chapter we present a visualization for the comparison of 3D surfaces. Our visualization is based on the proven *intersecting surfaces* technique, first introduced and evaluated by Weigle et al. [WT05]. Our contribution consists of three aspects: We present an alternative, image-based implementation of this technique, which enables interactive performance even when manipulating alignment or dealing with dynamic



(a) No thresholding (b) Differences thresholded at 1 mm

Figure 2.1: Comparative visualizations using our techniques of two partial brain ventricle surfaces segmented from MRI data. The initial visualization (left) clearly shows some symmetry in the differences, but our 3-way visualization (right) reveals an area of considerable change.

objects. Furthermore, we present enhancements designed to address specific limitations of the existing technique. Finally, we present a case study, evaluating the suitability of intersecting surfaces and our enhancements for the visualization of statistical shape models [CTCG95].

Figure 2.1 shows the comparison of two segmentations of the same brain ventricle MRI scan using our techniques. Such a visualization may give important information on the characteristics of a new segmentation algorithm. The yellow surface represents the baseline segmentation, while the blue surface shows a different segmentation. This means blue areas and yellow glyphs represent areas not covered by the new segmentation, while yellow areas and blue glyphs represent areas covered by this segmentation which are not in the baseline segmentation. Differences are remarkably symmetric in overall shape, but the lack of coloring due to fog indicates distances are small and contain little local variation. This may indicate these could be due to variations in the segmentation process. In figure 2.1(b) we applied a threshold at a distance of 1 mm, which reveals an area of considerable difference in the shape of the segmented ventricles.

Our implementation of the techniques described in this chapter is available as part of the open-source NQVTK library (`http://nqvtk.googlecode.com/`).

2.2 Related work

In this section we first define the position of our techniques in the comparative visualization process. We use this as a framework to present and discuss work related to our techniques, including the intersecting surfaces visualizations by Weigle et al., on which our approach is based.

2.2.1 Domain matching and comparison

Comparison can occur at any stage of the standard visualization pipeline [PP95]. However, the process always requires datasets to be aligned before differences can be determined and/or visualized. Many solutions exist for this *domain matching* step, the details of which are mostly application-dependent. In part this is because the processes of aligning data and extracting differences cannot always be cleanly separated. For instance, a body of work exists which applies non-rigid registration to volume data [RSDA02, BBP09] or to features extracted from it [SRD+97]. The deformation field resulting from this process is then analyzed to determine differences. This deformation field will typically also contain irrelevant differences due to imperfect alignment, such as tissue deformations caused by patient movement rather than by pathological processes.

Our techniques can be applied after domain matching in order to directly visualize the remaining differences between two surfaces. This enables such differences to be studied in detail, but can also give insight into the quality of the matching itself. We aim to be independent of the choice of domain matching technique. We only assume that a good (possibly application-specific) solution for this stage is available: i.e., one that does not remove any of the relevant differences and preferably leaves a minimum of irrelevant differences. As perfect matching (i.e., separation of relevant and irrelevant differences) is often impossible, our techniques should provide enough information to assess the relevance of the differences.

As stated in section 2.1, our visualization of such differences should be explicit and local. This means a suitable technique should determine and extract localized differences after the matching step rather than simply visualizing the aligned datasets. These differences should then be mapped to clear elements in the resulting visualization. Many existing comparison techniques are implicit in that they skip this extraction step, which means the act of comparison is left to the user.

2.2.2 Comparative visualization of 3D surfaces

Numerous measures have been proposed that can be used to express the difference (or similarity) in shape between two surfaces; most notably in the area of image retrieval [WBO97, dGS99, MdF05] and shape retrieval [Vel01, LHGQ06]. However, most of these measures, such as the commonly used Hausdorff distance, only express similarity at a global level.

The local visualization of differences can yield important insights which might not be obvious from global measurements. However, only a few techniques have been presented for comparing shape locally [MIA+03, GGGZ05]. Local distance measures often require establishing some form of *correspondence* between the surfaces (i.e., domain matching). A commonly used method for visualizing local distance measures is to display these on one of the two surfaces using an appropriate color map [MIA+03, LST03, PNN06]. This has the disadvantage that only one of the surfaces is shown; the shape of the second surface is not obvious.

A way to overcome this is to make one or both surfaces transparent and overlay them in the visualization. Such a *nested surfaces* approach (used by, e.g., Tory et al. [TMA01]) shows all context information. Similar approaches have been proposed in uncertainty visualization (see, e.g., Johnson and Sanderson [JS03]). However, in these visualizations the identification of differences is left to the user. This is complicated by the fact that overlaying multiple transparent and potentially intersecting surfaces results in an image which is not always clear to a user. In particular, understanding the shapes of transparent surfaces and classifying surfaces as being either in front or behind other surfaces can be difficult perceptual tasks.

Attempts have been made to resolve these issues. Textures are commonly applied to improve shape perception of transparent surfaces [Rhe96]. Interrante et al. [IFP97] proposed using stroke textures based on the directions of principal curvatures. Bair and House [BH07] investigated the use of several types of grid textures, and performed user studies on their effects on perception of surface shape.

Weigle et al. [WT05,Wei06], proposed the use of *constructive solid geometry* (CSG) operations to solve the perceptual problem of inside/outside classification. Their intersecting surfaces technique, described in detail in the next section, forms the basis for the visualization presented in this chapter. User studies performed by Weigle et al. have shown these visualizations to be effective for the comparison and understanding of surface shapes.

2.2.3 Base visualization

Because of its proven effectiveness, we use the intersecting surfaces technique by Weigle et al. as the basis for our visualization. In this technique, differences between two surfaces are extracted using CSG operations. The intersection of the closed objects formed by the surfaces represents the volume in common between both objects, and is rendered as an opaque object. The remaining parts of the two surfaces represent

differences, and are rendered transparently in order to show the intersection behind them. This solves the inside/outside classification problem, as parts of each surface inside of the other are always opaque and outside parts are always transparent.

As suggested by Interrante [IFP97], Weigle's visualization includes glyphs aligned with the surface curvature on the transparent parts of the surfaces to better illustrate surface shape. In order to visualize local distances between the two surfaces, Weigle's method relies on shadows cast by these glyphs. As an alternative to shadows, point correspondence glyphs can be used [Wei06]. These are line segments which connect corresponding points on the two objects that are being compared.

In their extensive user studies, Weigle et al. compared their visualization to a number of previously existing techniques [WT05, Wei06]. The studies show that the intersecting surface visualizations, both with shadows and with point correspondence glyphs, are more effective than the existing techniques. The contribution of this chapter includes improvements which address specific limitations of Weigle's technique.

2.3 Enhanced intersecting surfaces

In this section we present our enhanced visualization for the comparison of 3D surfaces. We first identify and discuss limitations of the existing technique, and use these to introduce and motivate our enhancements: an image based implementation for increased flexibility and interactive performance, dynamic intersection contours, integration of distance cues and suppression of irrelevant differences.

2.3.1 Limitations of the original visualization

In their work, Weigle et al. identified a number of shortcomings of their intersecting surfaces implementation. We summarize these issues below, and add two more issues (5 and 6) found when applying our requirements, described in section 2.1:

1. Folded surfaces and other forms of self-occlusion can hide differences. Weigle proposed interactive control of the camera as a possible solution, which is also possible in our techniques. However, similar issues occur when glyphs are placed near intersections, which may cause confusion regarding the shape of those intersections (see figure 2.2).

2. Due to their fixed spacing, surface glyphs can only illustrate shape at a fixed scale. Because of this, small-scale details can be missed. Weigle proposed making glyph spacing adaptive to local shape characteristics. However, we found that the sparse nature of the glyphs also means that the distance between surfaces can only be estimated around glyph/shadow pairs, or around point-correspondence glyphs.

3. The visualizations are not useful when objects are too different, or are not aligned properly during domain matching. In this case, Weigle recommends that explicit correspondence information be included in the visualization.
4. The approach is not easily extended to more than two surfaces.
5. In the base visualization, differences are shown regardless of their relevance, with no way to distinguish between relevant and irrelevant differences.
6. Importantly, Weigle et al. did not apply their techniques to the comparison of dynamic surfaces and the original implementation therefore does not support scenarios where the objects are not static in shape or in their relative positions and orientations.

2.3.2 Enhancements

While simultaneous comparison of more than two surfaces is considered outside the scope of this work, we present enhancements to the base visualization that address each of the remaining issues in the list above. The following sections discuss the motivation for and design of these enhancements. The details of their implementation can be found in section 2.4.

Image-based implementation

In this chapter, we present an image-based implementation of the intersecting surfaces visualization. Our implementation provides interactive control of the viewpoint, suggested by Weigle as a possible solution to the first issue. It also enables the visualization to be applied to dynamically deforming objects with interactive performance (issue 6), as the intersection is not computed geometrically. This in turn enables new applications for the visualizations. For instance, occlusion issues could also be solved by a dynamic peel-away approach. Additionally, our techniques could enable interactive domain matching tools, or interactive local alignment of corresponding features. Such tools could help in dealing with objects with significant global differences but local similarities, such as described in issue 3. While we consider such solutions future work and do not discuss these further in this chapter, we do present an example of dynamic objects in our case study (section 2.5).

In section 2.4.1 we present a generic rendering pipeline on which we base our image-based implementation. This pipeline enables combinations of CSG rendering and transparent surfaces in a straightforward and extensible way, which can also be useful in other visualization scenarios.

Intersection contours

As described in issue 1 and shown in figure 2.2, the complex appearance of the intersecting surfaces can sometimes cause confusion. Specifically, occlusions might be

interpreted as intersections and vice versa (figure 2.2(a)). Glyph placement can also cause confusion when this occurs on the intersection curve, potentially distorting the shape of intersections or hiding them completely (figure 2.2(b)). If the intersections between the surfaces are complex, the visualization becomes more cluttered. In this case, such issues are likely to become problematic, as small but possibly relevant differences may be missed.

Explicitly marking the intersection curves in the visualization removes any ambiguity in these cases. Contouring is a commonly used technique in illustrative rendering. Lines can be drawn to emphasize important boundaries in a visualization. While it may seem more natural to enhance occlusions rather than intersections, such contours are view-dependent and can therefore make the visualizations harder to understand. Furthermore, in our case the intersections between objects are actual features of interest, and contouring them serves the dual purpose of highlighting these features. Similarly, changing the color of the outer surface (and glyphs) would solve problems as shown in figure 2.2(b). However, as color is used to identify surfaces, such a change would complicate the visualization. Contours provide a solution that works independent of the choice of surface coloring and texturing. The intersection curves of the surfaces provide clear contours for the "difference" areas in the visualization. They also enable clear distinction between intersection and occlusion.

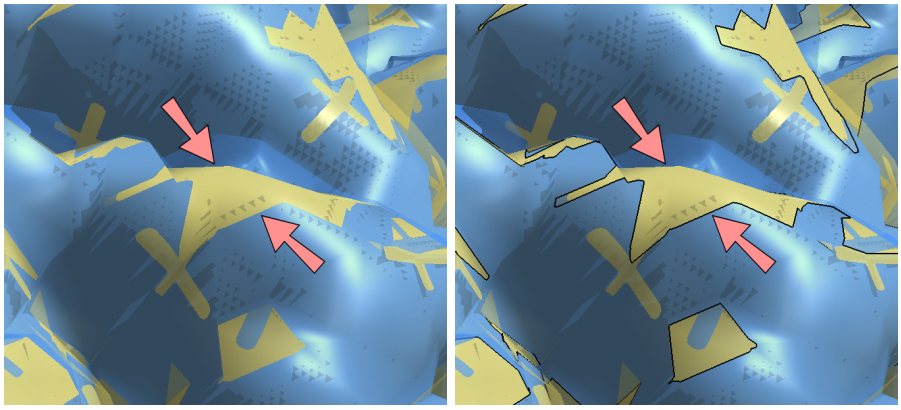
Local distance cues

In addition to the well-known difficulties of illustrating the shape of transparent surfaces, the second issue is also caused by local distance information being visualized only sparsely in the original intersecting surfaces technique. Such information is important for understanding the shape and size of differences, and can also help in understanding the shape of the objects themselves.

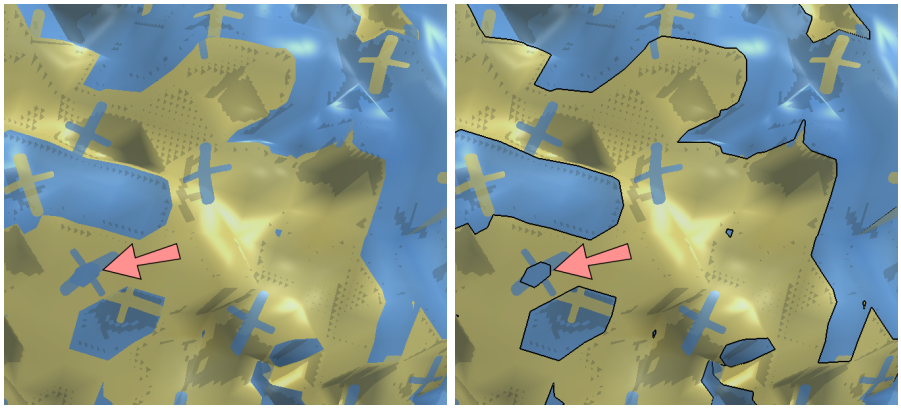
In Weigle's technique, the user has to rely on shadows cast by glyphs in order to judge local distances between the surfaces. However, on many complex surfaces it may not be straightforward to match up glyphs with their shadows. It may also be the case that there simply is no glyph present in an area of interest. One example of problematic shape is shown in figure 2.3. In this case, areas with increased depth (an example is shown in the cross-section, figure 2.3(c)) are not immediately obvious in the visualization (figure 2.3(a)). When using glyphs, a smart placement of these glyphs (and their shadows) might solve such problems. However, this could lead to an uneven distribution of glyphs over the surface, which causes other perceptual issues.

We integrate two forms of local distance cues to alert the user to these areas in an intuitive way. Specifically, we simulate fogging between the inner and outer surfaces as a viewpoint-dependent distance cue and integrate closest-point distance for viewpoint-independent feedback. These enhancements also provide a way to visualize the shape of the surfaces at any scale, albeit relative to each other.

In fogging, we add a third color between surfaces, the amount of which is computed in terms of the distance between these objects along the viewing direction and a

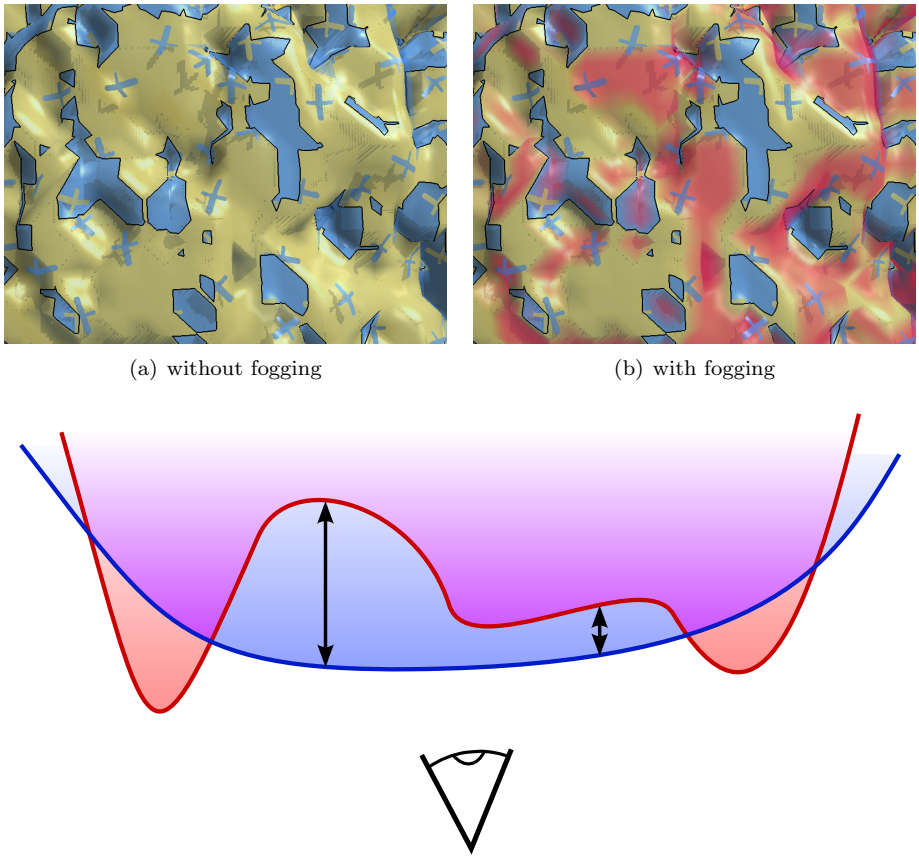


(a) occlusion (top left arrow) and intersection (bottom right)



(b) small intersection hidden due to glyph placement

Figure 2.2: Without contours, intersections between the blue surface and the yellow surface are not always obvious. Marking all intersections differentiates them from occlusions (a) and highlights smaller features (b), which might otherwise be hidden.

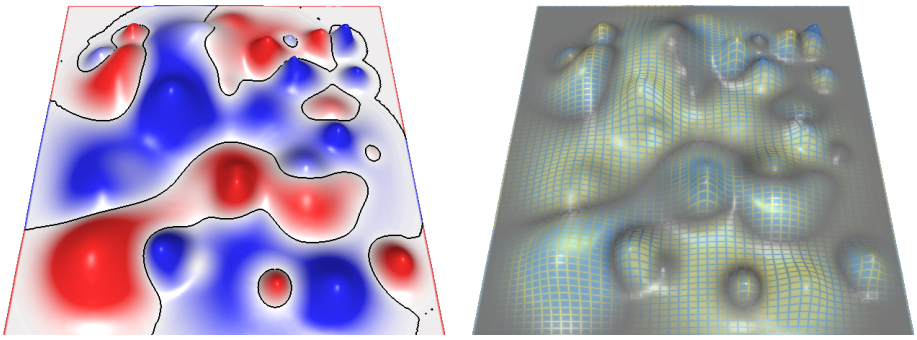


(a) without fogging

(b) with fogging

(c) cross section showing area with varying depth

Figure 2.3: The choice of viewing direction and placement of glyphs may hide important information about the size and shape of differences. By including local distance feedback in the visualization (here using red fog added between the surfaces), larger differences are made more obvious.



(a) Distance fields can be used to color each surface by its distance to the other, even when surfaces are moved relative to each other.

(b) Mapping distance to saturation of the surface colors reduces the prominence of small differences, but causes perceptual issues due to the blurry appearance.

Figure 2.4: Distance field based visualization of distance between surfaces.

user-controllable density parameter. The effect is that the areas of difference between the objects are highlighted more for larger distances. As can be seen in figure 2.3(b), differences in depth between surfaces can be seen clearly after fog is added. Interactive manipulation of fog density can help a user to compare relative depths of these differences. This interactivity also removes the problem that fog obscures the shape of the inner surface, as the fog can simply be removed or made less dense in order to examine surface shape after differences have been identified.

As fogging is view-dependent, results may change when the viewpoint is moved. We integrate closest-point distance as a view-independent alternative (figure 2.4), and enable coloring of either surface based on this distance. Our implementation, described in section 2.4.3, achieves interactive performance even when objects are moved relative to each other, meaning that this technique could also be used in an interactive domain matching solution such as described earlier.

Relevance filtering

Issue 5 originates directly from our requirements. If objects differ in some areas and are identical in others, the intersecting surfaces visualization clearly shows such differences. In real-world data, however, noise and other inaccuracies are often a concern, which can cause many small differences between objects. In intersecting surfaces, each of these will be shown with similar visual impact as other differences, cluttering the resulting images and distracting users from those other, possibly more interesting differences.

We use the distance measures introduced in the previous section in order to allow users to suppress smaller differences in the visualization. We experimented with

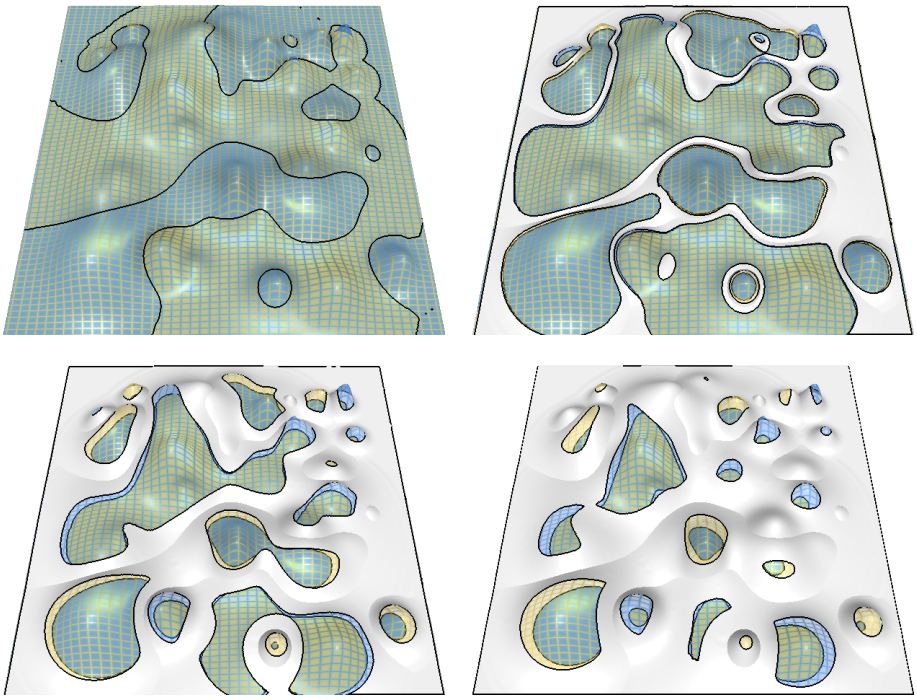


Figure 2.5: Using increasing thresholds to hide irrelevant differences.

mapping distance to the saturation of the surface color (figure 2.4(b)). This has the effect of both surfaces fading to gray as the distance between them gets smaller. Small differences such as those caused by noise will therefore also appear gray, rather than as the sudden change of object color they cause in the original visualization. Unfortunately, changing saturation over the surface negatively affects the ability to perceive object shapes.

A simpler solution is thresholding the local distance measure. This way, we allow users to suppress differences that are considered too small to be relevant. Essentially, this has the effect of widening the intersection contours into areas of insignificant differences. We render such areas in an opaque neutral white color. The result, shown in figure 2.5 and figure 2.1, is a less cluttered visualization where the user is less distracted by small fluctuations between the surfaces. The threshold value can be controlled interactively by the user, to control the distance from which differences are considered relevant.

One limitation of the current solution is that while in white areas the objects are considered similar, the two objects are still rendered as separate surfaces. This effect can be seen in the bottom two images in figure 2.5, where thresholds on the inner

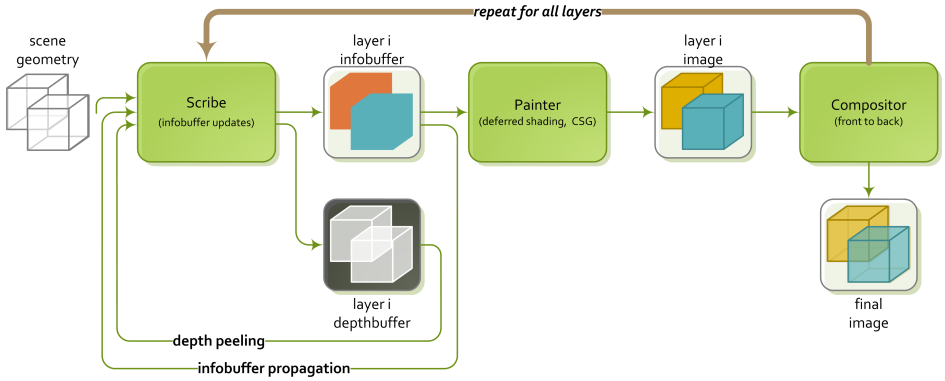


Figure 2.6: The layered rendering pipeline.

surface show up through the holes representing significant differences. Especially for higher threshold values, this can cause the already complex visualizations to be harder to interpret. In future work we aim to address this limitation by using techniques from illustrative rendering in order to reduce the visual complexity of the thresholded areas.

2.4 Implementation

In this section we discuss our image-based implementation of the intersecting surfaces visualizations and of the enhancements described in the previous section.

While not the focus of this work, we implemented both opacity-modulating glyph textures (as used by Weigle et al. [WT05, Wei06]) and grid textures (see Bair et al. [BH07]) to help illustrate the shape of transparent surfaces. The integration of such techniques in the layered rendering pipeline (described next) is considered straightforward. Similarly, our implementation uses standard shadow mapping [Wil78] to make the opacity-modulating textures cast shadows on the inner surface, as recommended by Weigle et al. [WT05, Wei06].

2.4.1 The layered rendering pipeline

We introduce the *layered rendering pipeline* (figure 2.6) as a generic basis on which we build our image-based implementation of the intersecting surfaces technique. Layered rendering combines depth peeling, deferred shading and CSG in order to create an approach for surface rendering similar to ray casting, but at interactive rates. In the intersecting surfaces visualization, this pipeline enables interactive performance even when objects are deformed. It also facilitates the addition of further enhancements which address limitations of the original visualization.

As stated, we would like to be able to deal with objects of high geometric detail, and support objects which are dynamic or interactively positioned relative to each other. This requirement means that a geometric approach to computing the CSG intersection, as used by Weigle et al. [WT05], is infeasible due to the complexity of such a computation. We therefore investigated possibilities for an image-based approach.

While image-based CSG solutions exist [GMTF89, Wie96], the layered rendering pipeline is an extension of depth peeling [Mam89,Die96,Eve01]: All image-based CSG approaches require multiple passes over the geometry, each extracting separate layers of geometry distinguished by the number of surfaces occluding them. Depth peeling, however, also ensures that the surfaces are extracted (and rendered) in strict front-to-back order, which is required in order to properly render any transparent surfaces. We extend the pipeline with a stage in which deferred shading [DWS$^+$88, ST90] is applied to each layer extracted from the geometry. Furthermore, we extend deferred shading to allow propagation of information between layers.

Weigle [Wei06] described an image-based inside / outside classification approach based on depth peeling. Weigle's algorithm, however, is limited to two surfaces and is not a full CSG algorithm; any fragment appearing deeper than a fragment already labeled interior is also labeled interior. This makes, for example, clipping the geometry resulting from the CSG operation or rendering the intersection with transparency difficult to achieve. Other combinations of depth peeling and image-based CSG have been presented by Guennebaud et al. [GBP06], used to handle transparent objects in point-based rendering, and Nienhaus et al. [NKD], who presented a method for illustrative rendering of CSG models. These combinations were designed specifically for their applications, however, and are not easily extended.

The layered rendering pipeline takes inspiration from ray casting. The ray casting algorithm takes an image-order approach, where rays are cast from each image pixel. These rays traverse the scene and intersect any objects within. Due to this image-order approach, achieving a correct ordering for compositing transparent surfaces is inherent in the algorithm. Similarly, CSG operations are implemented easily by simply keeping track of object intersections along a ray.

The design of the layered rendering pipeline is based on several observations. Firstly, rendering an object using traditional rasterization essentially provides the intersection points of all rays cast from the image simultaneously for a single layer of geometry. Secondly, the depth peeling algorithm can be used as-is to process all intersections in the scene in front-to-back order.

The layered rendering pipeline, therefore (see figure 2.6), consists of repeated execution of two stages in order to process the layers in the scene. The *scribe* stage performs depth peeling using a dual-depth buffer approach [Die96] and stores information about the current layer which is required for shading in an *info-buffer*. The information in this buffer is created by updating the info-buffer from the previous layer with information from the geometry of the current layer. This way such in-

formation can be propagated through the scene in a way similar to ray-casting. The *painter* stage then compares this information with that of the previous layer(s) in order to create the image for this layer. Similar to deferred shading, the painter stage does not require the scene geometry to be drawn, as all information required for shading is present in the info-buffers. The resulting images are composited using front-to-back blending to create the final image.

The total number of layers could be very large, depending on the depth complexity of the scene being rendered. However, because later layers may not noticeably influence the final image, rendering can be terminated after either a set number of layers is reached, or when the number of pixels in a layer is below some fraction of the total image size. This number is determined using occlusion tests available in modern hardware.

In the following, we refer to the info-buffer for layer l_i as B_i. The data in B_i describes (for each pixel) properties either of the corresponding layer l_i, or of the depth slab s_i, which is the volume between l_i and l_{i+1}. An important part of B_i is the *in/out mask*, a bit mask which indicates for each pixel and for each of the objects in the scene whether s_i is inside or outside of the object. We assume all objects are closed, non-self-intersecting surfaces, although our approach still works with objects that extend into infinity in the viewing direction (e.g., landscape surfaces such as figure 2.5, viewed from above). In the scribe stage, the bit corresponding to the visible surface for each pixel is set to 1 if that surface is front-facing, and to 0 if it is back-facing. Other bits are left unchanged and are copied from B_{i-1}. The mask for B_0 needs to be initialized according to the camera's position w.r.t. the objects in the scene. However, it can generally be assumed in our visualizations that the camera is positioned outside of all objects.

The in/out mask provides a simple way to perform arbitrary CSG operations on the objects in the scene, as well as clipping objects to arbitrary geometry. Additionally, it solves a common problem in depth peeling regarding coplanar surfaces. In our implementation, the mask is stored in one of the 8-bit RGBA channels of the info-buffer texture. This means we can track up to 8 objects at a time, which is more than sufficient for the purposes of the intersecting surfaces visualization.

CSG rendering

The base visualization only uses the CSG intersection of objects. In future extensions, however, other CSG operations could be useful as well. For example, such operations could be used as part of area-of-interest selection in a focus+context visualization.

CSG rendering is as simple as applying the Boolean-valued CSG expression to the in/out masks for both B_i and B_{i-1} during the painter stage. If the resulting values are different for some pixels, that part of l_i is a surface of the object resulting from the CSG operation, and can be rendered as such (see figure 2.7).

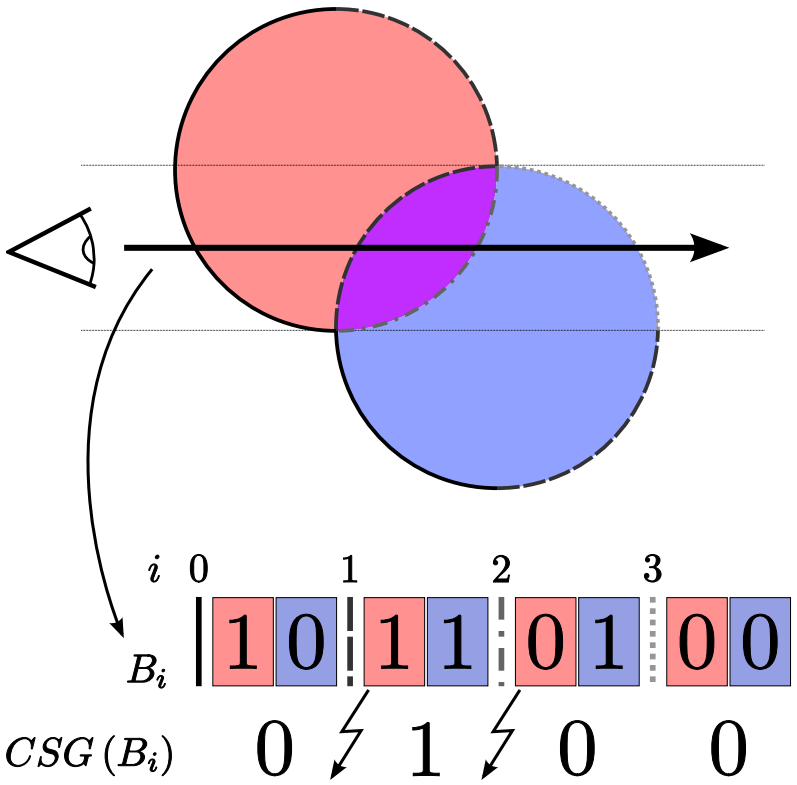


Figure 2.7: Layered rendering for the CSG intersection of two spheres. In-out masks are shown for each depth slab for positions along the arrow. Changes in $CSG(B_i)$ correspond to surfaces of the CSG result.

Clipping

One advantage of this approach is that it results in the CSG surfaces on all depth layers rather than only a single layer. For instance, the Goldfeather algorithm [GMTF89] only provides the first depth layer of the CSG object. Similarly, Weigle's image-based classification technique only locates the first intersection surface along the viewing direction. Having correct CSG surfaces on all layers means it is easy to implement clipping techniques in the painter-stage. These can be used, for instance, to enable viewing of interior structures of the CSG model. Furthermore, because we maintain an in-out mask between layers, the objects in the scene can be clipped against arbitrary geometry. Clipping against arbitrary closed objects is done trivially during the painter stage by discarding all fragments for which the bit corresponding to the clipping object is set in the in-out mask (indicating they are part of surfaces located inside this object).

Coplanarity

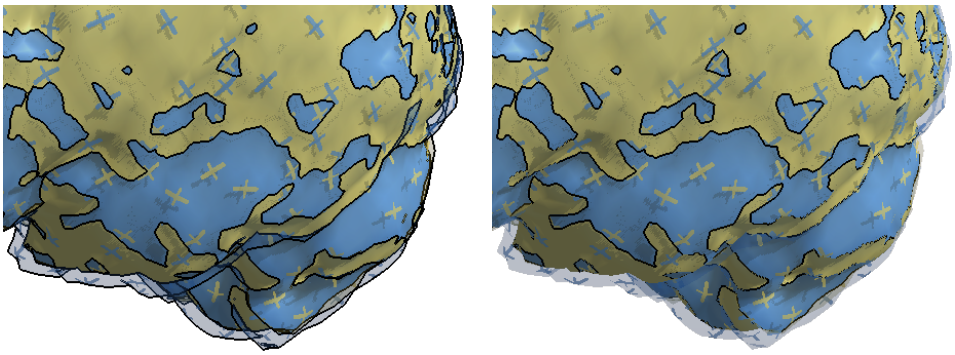
One important limitation of the depth peeling algorithm is that it does not deal well with coplanar surfaces. As only depth information is used to distinguish between surfaces in peeling, in the case of coplanar surfaces only one of the surfaces will be detected. In addition to causing visual artifacts when surfaces are not rendered, this will cause problems for a CSG implementation such as the one described above.

Our layered rendering approach offers an elegant solution to the problem. The problem stems from the fact that normal depth peeling only allows fragments with a depth value *larger* than the previous layer. In our *coplanarity peeling* algorithm, identical depth values are also allowed, as long as the in-out mask resulting from the previous layer indicates we have not rendered these before. Our assumption that surfaces are not self-intersecting enables us to perform only a single "greater or equal" test against the previous layer, rather than testing for equality separately, thereby simplifying the implementation.

2.4.2 Intersection contours

Like the intersection objects themselves, intersection curves can be computed geometrically. Such computations, however, are complicated, hard to generalize and unsuited to interactive visualization. We use our layered rendering pipeline to create an image-based derivation for these contours.

In illustrative rendering, a common technique is to derive object silhouettes and feature contours from discontinuities in images containing depth values or surface normals respectively [ST90]. Similarly, we use the info-buffer to record, for each pixel, the identity of the object which contributed that pixel for the current layer. During the painter-stage, we perform simple edge detection on this buffer by comparing pixels to



(a) all detected contours (b) filtered contours

Figure 2.8: Using depth comparison to filter the contours: silhouettes are removed while intersection contours remain.

their immediate 4-connected neighbors. This approach, however, leads to more than just the intersection contours (see figure 2.8(a)).

In order to filter this set of contours, we also store the depth values for each pixel in the layer. Our reasoning is as follows: if two neighboring pixels are on opposite sides of a true intersection contour, their depth values will be similar as well. While this assumption does not hold in general, we benefit from the fact that in most comparison scenarios objects are similar. If depth values on opposite sides of a detected edge fall within an epsilon range of each other, the edge is rendered in the image (figure 2.8(b)). The value of this epsilon depends on the scale of objects in the scene and is currently determined empirically on a per-application basis.

2.4.3 Integrating local distance information

As stated in section 2.3.2, we integrated two types of local distance cues. Their implementations again take advantage of the layered rendering pipeline, and are described in the next sections.

Fogging

When using ray casting to render fog, colors and opacities are accumulated along the rays while they travel through fog volumes. Similarly, we determine the amount of fog between two surfaces as a function of the distance between those surfaces. In our layered rendering pipeline, this means we can simply compare the depth values for the current layer l_i to those of the previous layer l_{i-1} in the painter stage, in order to determine the distance for each pixel. Based on this distance, we compute the accumulated fog opacity for the depth slab s_{i-1} and then blend the fog over the

current layer. Distance can be mapped to opacity linearly, however, as multiple slabs may contribute to an area of fog, an exponential mapping yields better results.

As in CSG rendering, a Boolean expression can be applied to the in-out mask (in B_{i-1}) in order to specify the (combinations of) objects in which fogging should be applied. An alternative to fogging is distortion (e.g., defocussing, blurring) of the layer image due to refraction or translucent materials. To implement this, the layer should be rendered normally, after which the resulting image can be distorted in accordance with previous layers in order to produce the desired effect.

Distance fields

As discussed in section 2.2, many shape (dis)similarity measures have been proposed in the literature. The values resulting from these measures change whenever either object is manipulated by the user. Recomputing the measure for each point on both surfaces may not be feasible in an interactive system. We therefore propose a solution based on distance fields.

A distance field is a sampling of a distance function, which maps positions in 3D space to the distance of that position to the surface. The simplest such function uses the Euclidean distance metric between a position in space and the position of the closest point on the surface. In the case of a closed surface, such a distance function could be signed, i.e., the sign of the distance could indicate whether a point is inside or outside of the object. Although creating such fields is computationally expensive, they only need to be computed once per object.

We use the Closest Point Transform algorithm by Mauch [Mau00] to precompute (signed) distance fields for our surfaces. These distance fields are made available during rendering as 3D textures. When objects are rendered during the scribe stage of the pipeline, we sample the distance fields in order to obtain distances between the objects. To accomplish this, 3D locations corresponding to each surface pixel need to be transformed to the local coordinate frame of the other object, after which the corresponding 3D texture can be sampled. This effectively results in the distances of each point to the closest point on the other object. Additionally, objects can easily be manipulated in this approach; the corresponding transformations simply need to be taken into account when determining lookup locations in the distance fields.

Although this technique technically does not require the use of the info-buffer, we still benefit from the basic depth peeling and deferred shading options inherent in the layered rendering formulation, as well as allowing the visualization of the distances to be handled separately from their measurement.

2.4.4 Relevance

Implementing the relevance thresholding technique in our framework is as simple as adding a threshold test after sampling the distance field in the scribe-stage. We replace our object identification in the layer information buffer by a classification.

This classification can take one of three values. If for some fragment the distance is smaller than the threshold, the fragment is classified as being "equal". Otherwise, the fragment is classified as significantly belonging to one of the objects.

During the painter-stage, we render "equal" fragments as opaque in a neutral color. Our contour detection approach can be performed on the new classification buffer (rather than on the object identification buffer) in order to emphasize the relevant areas.

2.4.5 Performance

We created a prototype implementation of the techniques presented in this chapter in C++, using VTK for data processing and OpenGL for rendering. The OpenGL shading language (GLSL) was used to implement all GPU-based algorithms, which require at least NVIDIA GeForce 7 graphics hardware (or equivalent). The resulting system allows for interactive rendering of the visualization, as well as interactive manipulation of the surfaces involved in the comparison, the viewpoint and the parameters of the various techniques.

Our layered rendering implementation makes full use of deferred shading. For this, our layer information buffer consists of three 32-bit render targets. The first two contain color and surface normals for use in deferred shading. The third is used to store surface classifications, the in-out-mask and surface depth values. As only this third buffer needs to be propagated between layers, the memory costs of the algorithm are those of five 32-bit images at the display resolution (see section 2.4.1). The use of distance fields adds to this requirement; we used two 256^3-byte distance volumes, which provide sufficient accuracy for the purposes of our visualizations.

Tests were performed on a single core of an AMD Athlon 2.6 GHz dual core machine with 3GB RAM and NVIDIA GeForce 8800 graphics. Images were rendered with shadow mapping enabled at $1\,000 \times 700$ resolution, resulting in frame rates between 10 and 50 frames per second for four depth layers (depending on surface complexity), which is sufficient for most intersecting surfaces visualizations. As noted, the use of shadow mapping requires scenes to be rendered twice per frame: once from the light's viewpoint and once from the camera. This means a doubling of frame rates can be achieved when shadow mapping is not required.

We used the brain ventricle segmentations (figure 2.1, 79 668 triangles), an artificial height field dataset (figure 2.5, 1 044 484 triangles) and the ventricle shape models (figure 2.10, 3 788 triangles) for our measurements. Both the number of triangles and the number of depth layers linearly influence the rendering performance. This is expected, as geometry is rendered once for each layer. Decreasing the maximum number of layers increases performance, but at the cost of image quality. A lower number of layers may cause display artifacts when large numbers of transparent surfaces overlap each other. In the case of the intersecting surfaces visualization, depth complexity of the scene rarely exceeds 4, due to the fact that the inner object is rendered opaque. However, the use of clipping techniques (as described in section 2.4.1) may cause an

increase of the number of layers required. This is because clipping is performed in the painter stages, meaning the hidden geometry and the clipping object itself both contribute to the depth complexity of the scene in the scribe stages.

The k-buffer architecture proposed by Bavoil et al. [BCL+07] could be used to greatly reduce the number of passes required to perform depth peeling. However, currently no hardware implementations of this architecture exist.

Depth peeling based algorithms are usually criticized for their high memory requirements. We note, however, that in our techniques only information from the current and previous layer is required in order to render any given layer. Therefore, we require only two buffers in memory at all times. Furthermore, rather than storing each layer, we update the final image on the fly. This brings the total memory requirements of the layered rendering algorithm down to the memory required for two info-buffers and one final image. As only a subset of information needs to be propagated between layers, memory requirements can be further reduced.

The interactive performance achieved by our image-based implementation of the intersecting surfaces visualization enables new uses of such visualizations. As the intersection does not have to be computed explicitly, the objects involved in the comparison do not need to be static. The next section illustrates an application of the comparison of dynamic surfaces.

2.5 Case study

We performed an expert evaluation study of the techniques presented in this chapter. This section describes our collaboration with two researchers in medical image analysis, who are experts in creating and analyzing statistical shape models. In this study, we applied our techniques to visualize real-world statistical shape model data provided by the researchers.

The researchers are also the third and fourth authors of the article on which this chapter is based [BBF+11]. They contributed to this work by providing information on the problem domain and by giving feedback on our prototypes, but did not work on the implementation itself. In this context, their role as test users and evaluation subjects was not compromised.

2.5.1 Statistical shape models

In medical research, statistical shape modeling [CTCG95] is an important tool for understanding both differences in and variability of shapes of anatomical features. A shape model is created by first identifying corresponding points on all individuals in a population. Principal component analysis is then applied to determine the average shape of the population, as well as the principal modes of variation, ranked from most to least significant.

For this case study, the researchers provided us with a pair of statistical shape models used in research of Alzheimer disease (AD). The shape models represent the variation of the shape of the brain ventricles within two populations: a group of AD patients and a control group of healthy patients. It has been shown [FPO+06] that brain ventricles enlarge significantly due to the loss of brain tissue caused by AD.

Shape models are a special case of comparison, for which a known correspondence between points on both meshes is not only available, but in some cases highly relevant to the comparison. Currently, such correspondence is visualized by the researchers by showing a single surface combined with *point correspondence glyphs* (i.e., lines or arrows) pointing from this surface to where the other surface would be. Additionally, the researchers frequently use color mapping to visualize measures on a single surface.

2.5.2 Evaluation method

The evaluation consisted of two meetings, during which we discussed our visualizations with the researchers. The first meeting was an informal introduction of our visualization, consisting of a demonstration of our techniques and a discussion of shape model visualization in general. We used the information gathered at this meeting to improve our prototype implementation, which was used to formally evaluate our techniques in the second meeting. Additionally, we gathered information on currently used visualization techniques, and used the feedback to formulate a set of three shape research scenarios in which our visualizations could be applied.

1. Exploring shape differences between two static 3D surfaces. This scenario has been studied in detail by Weigle et al. [WT05].

2. Exploring variation of a single shape model. This includes both comparing new individuals to an existing model as well as examining variation within the model with respect to the mean shape. For this scenario as well as the next, the user can interactively control the shape model parameters to explore different shapes.

3. Validation (error-checking) of a shape model. A shape model is created by moving the vertices of a mesh to match different individuals in a population. If these vertices move along the surface of the model to match two individuals, the model may indicate a difference where in fact there is none. The models are validated by checking for such errors.

The comparison of two different shape models is another important topic for shape model visualization. However, this requires statistical information to be included in the visualization and is therefore considered future work.

We defined eight visualizations, shown in figure 2.9, for exploring shape differences. The combinations were selected to be able to evaluate the strengths of each technique separately while still creating meaningful visualizations. To achieve this, we included both single surface and intersecting surfaces visualizations, intersecting surfaces with

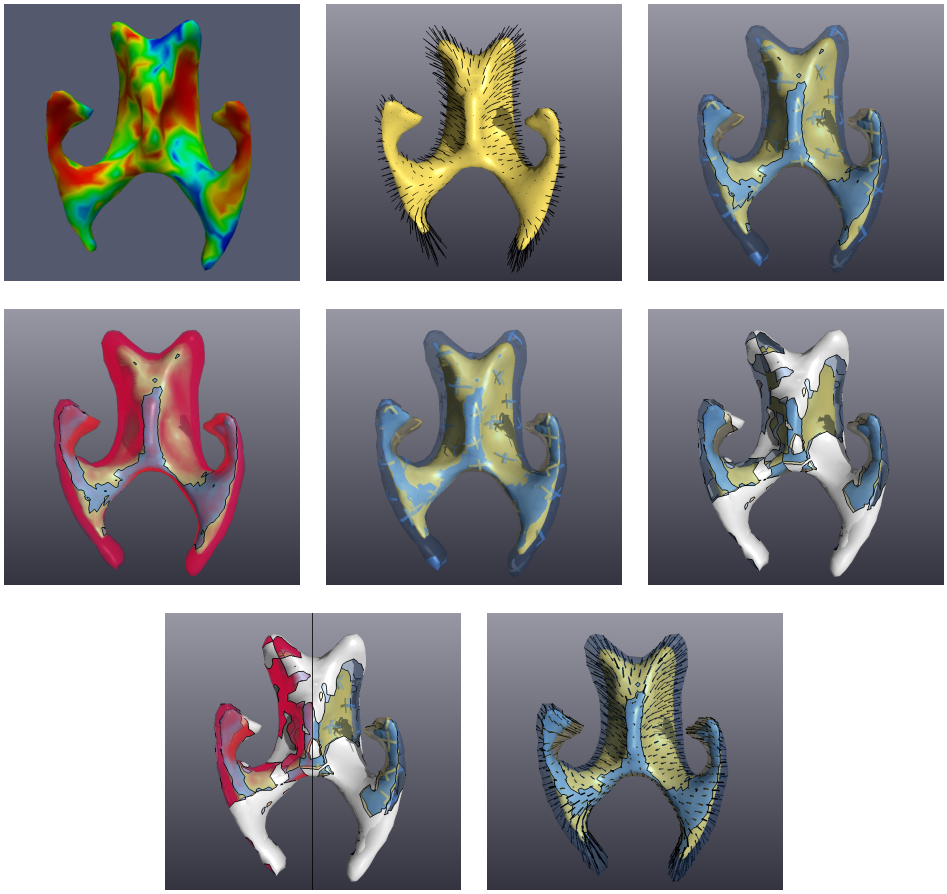


Figure 2.9: Visualization configurations for surface comparison used in the expert evaluation.

fogging or with shadow-casting glyphs, and combinations with or without relevance thresholding. To compare the proposed enhancements to currently accepted visualization techniques, we also included basic color mapping, point-correspondence glyphs and Weigle's original intersecting surfaces visualization (without our enhancements) in the evaluation. During the second meeting, the researchers were asked to grade these visualizations for each scenario.

We used a Likert scale to grade agreement with the statement "Visualization X is suitable for use in scenario Y", where X and Y formed all combinations of visualizations and scenarios. Secondly, the researchers were asked to rank the visualizations in order of preference for each scenario. As a final question, we asked the researchers to create their preferred visualization from the techniques in the prototype application.

In order to avoid distortion of the results, the researchers were explicitly warned about possible biases in using a Likert scale and told to evaluate each visualization independently of the others for the first question, leaving comparison for the ranking question. We limited the evaluation to the interactive use of each visualization, as such use is the primary aim of our contributions.

2.5.3 Software setup

We performed the case study using a specially prepared version of our software, which could be toggled between each of the predefined combinations of techniques. For the purposes of this study, we added an implementation of point correspondence glyphs, similar to the glyphs currently in use by the researchers (shown in figure 2.10) and those used by Weigle [Wei06].

In analyzing shape models, the researchers are most frequently interested in the statistical properties of the model. However, there are use cases in which direct shape comparisons can be useful. The interactivity provided by our techniques enables the intersecting surfaces technique to be used on dynamically deforming surfaces. We adapted our software to allow for interactive deformation of the shape model mean surfaces along each of the modes of variation, in order to explore the variability of the statistical model. The researchers were allowed free use of this functionality for the evaluation of each technique in the latter two scenarios.

Based on feedback received from the researchers at the initial meeting, we enhanced the user interface of the prototype application for the formal evaluation by adding quantitative feedback about visualization parameters. We also added linked views showing the separate shapes involved in the comparison, with linked cursors between all views. While the researchers were asked to only use the intersecting surfaces visualization for comparison, they indicated the simple views were helpful in learning how to interpret the intersecting surfaces visualizations.

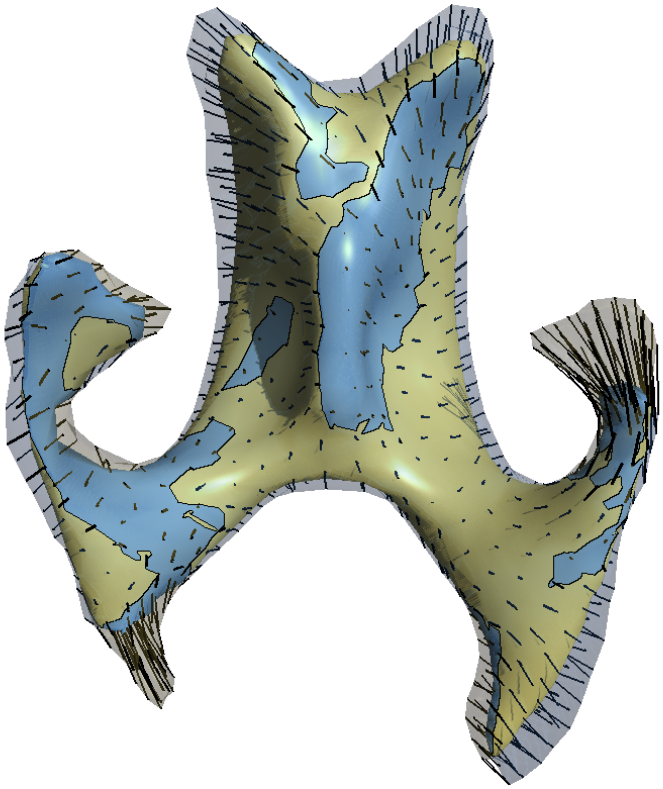


Figure 2.10: Point-correspondence glyphs connect corresponding points on the mean and deformed shape model surfaces. Here they are used to show the third principal mode of variation in the control group shape model.

2.5.4 Evaluation results

After evaluating each visualization, the experts were asked to explain their grades. The Likert scale grades and rankings served to provide important structure to this discussion. This section summarizes the feedback gathered from this process.

In all scenarios, the researchers preferred the intersecting surfaces visualizations over the single-surface visualizations. In a single-surface visualization, it is hard to determine the shape of the second surface. The researchers did note that for certain specific questions, such as examining statistical relevance of deformations, a color-mapped surface provides enough information.

Fogging and shadow-casting surface glyphs: Both fogging and glyphs are preferred over plain intersecting surfaces, as without them it is very hard to see the 3D structure of and separation between both surfaces. However, the researchers had some concerns regarding the fog, causing them to prefer the glyphs. First, fogging adds visual complexity; opacity is not commonly used in current visualization techniques to indicate distance, and the fog color adds a third color to the visualization. Secondly, glyphs also served to emphasize the shape of the outer surface. Their removal, coupled with the fact that the fog obscures the inner surface makes the shape of both surfaces harder to understand. The fog color can be useful, however, to quickly direct attention to areas with large difference. Other techniques can then be used to explore surface shapes in detail.

Intersection contours: When using the shadow-casting surface glyphs, the researchers noted that intersection contours are important to help distinguish between glyphs and intersections. The presence of these contours also helped the researchers to better understand the intersecting surface visualizations in general.

Relevance: The researchers see the benefit of suppressing irrelevant differences using a threshold. However, they indicated several points where the current visualization should be improved. First, the visual complexity of the current implementation is too high. Thresholding being performed on both surfaces separately clutters rather than suppresses these areas in the visualization. Because of this clutter, visualizations which included the thresholding technique were ranked very low in the evaluation. More work should be done to reduce the visual complexity of the masked areas, e.g., using illustrative techniques. The presence of a third color (white) is acceptable in these visualizations, as the color has a well-defined meaning (areas with no relevant differences). Second, comparing thresholding to color mapping, the researchers indicated a need for better visualization of the values in non-masked areas. Additionally, a color scale should be used to highlight particularly large or significant differences in the unmasked areas.

Point-correspondence information: While not normally available for general surface-to-surface comparison, information on point correspondence is essential for shape model validation (scenario 3), and should therefore always be included in these visualizations. The point-correspondence glyphs fulfill this requirement, but can be enhanced in several ways. Color mapping the glyphs by their length can help to show the distance between corresponding points. Similar to surface thresholding by distance, an option should also be added to filter glyphs that are too short to be relevant by thresholding, and the user should be able to toggle their display interactively.

One thing to note is that for single-surface visualizations, glyphs showing point-correspondence may point inside of the surface. This makes such areas impossible to distinguish from areas of no change, unless the surface is made transparent (thereby causing other perceptual issues). The intersecting surfaces visualizations always show all point-correspondence glyphs. It was mainly for this reason that this was a preferred combination for the researchers.

2.5.5 Lessons learned

A recurring concern during the evaluation was the high visual complexity of the intersecting surfaces visualizations, especially when deforming surfaces are involved. Despite this, intersecting surfaces were ranked higher than single surface visualizations for all scenarios. Overall, the researchers feel that the visualizations can be easily interpreted after some use. However, the initial learning curve is rather steep, and the additional complexity introduced by our fogging and thresholding techniques added to this. For exploration of the variation in a shape model, the researchers considered information about exact differences less important than for the other scenarios. This means techniques like thresholding, color mapping and the point correspondence glyphs are best omitted in this scenario, in order to reduce visual complexity.

Scenario 3 also showed the importance of tuning visualizations to application-specific requirements. The preferred visualization chosen by both researchers consisted of a basic combination of intersecting surfaces with contours and shadow-casting surface glyphs to aid perception of the shape of the transparent surfaces. However, due to the nature of statistical shape models, this visualization should also include point-corresponding glyphs, which can be colored by distance, filtered by relevance and toggled on or off by the user.

2.5.6 Summary and outlook

In this case study, we applied our techniques to real-world data and problems to gauge their effectiveness. Our expert evaluation agrees with most of our own findings as well as those from Weigle's studies. However, it also indicates that more work should be done on reducing the visual complexity of the depth cueing (fogging) and relevance masking (thresholding) techniques, as well as of the intersecting surfaces technique

in general. Additionally, user studies should be performed at a larger scale to more thoroughly evaluate the techniques.

We intend to continue the collaboration in future work to apply our techniques to shape model comparison and population studies.

2.6 Conclusions and future work

We have presented techniques for the interactive visualization and exploration of surface differences using intersecting surfaces. Our main contributions are:

- A new image-based formulation of the proven intersecting surfaces visualization, based on our new layered rendering pipeline. Our formulation provides interactive performance, even when the objects being compared are deformed or moved relative to each other. This interactivity enables intersecting surface techniques to be applied in new domains such as shape model visualization.

- Enhancements to the original intersecting surfaces technique. These include object intersection contours, as well as the integration of local difference measures in the form of view-dependent depth cues and view-independent distance fields. Furthermore, we extend the intersecting surfaces visualization to a three-way segmentation of surface differences, where the visual impact of insignificant differences is reduced.

- A case study, exploring the usefulness of intersecting surfaces techniques for the visualization and comparison of statistical shape models.

Our layered rendering formulation offers a flexible framework for various rendering techniques which are otherwise not straightforward to implement. The approach has strong parallels with ray casting, in that information is propagated from the viewer into and through the geometric scene. This enables image-based implementation of various ray-casting-like algorithms for layered surfaces, such as CSG operations and fogging inside objects. However, as the method is still based on rasterization, performance is much higher than that of a ray casting technique and the technique can take full advantage of hardware acceleration. Furthermore, due to the formulations sharing a common framework, techniques can be combined in a straightforward way.

Based on the layered rendering formulation, our techniques create an interactive visualization for the comparison of surfaces. As indicated by Weigle [Wei06], this interactivity is an important aid for understanding the shapes of objects being compared. Interactivity during object manipulations such as relative movement and / or deformation also enables new applications for the intersecting surfaces visualization, such as the visualization of statistical shape models explored in our case study.

Our visualization addresses the requirements defined in section 2.1: The visualization of differences is local and explicit, and we presented methods for the suppression

of irrelevant differences in the visualization. The distance cues and relevance filtering techniques can be used with various distance metrics, as long as it is possible to create a distance field for such a metric.

Furthermore, our enhanced intersecting surfaces techniques address most limitations of the original technique: Contour lines remove ambiguity between intersections, occlusions and surface glyphs, distance cues add additional inter-surface distance information at any point on the surfaces, and relevance filtering hides irrelevant information. Regarding the visualization of highly different objects or dealing with occlusion, we provide both suggestions and (through our framework) opportunities for solving these in future work.

The resulting visualization can provide an overview of differences for any given pair of surfaces. This can assist a researcher in selecting areas of interest and/or choosing further techniques to apply to quantitatively analyze these differences. Our evaluation has shown that intersecting surfaces can also be a useful technique for comparing dynamic objects, provided that: 1) The visualization is properly adapted to the problem domain, e.g., by adding correspondence feedback in the case of our shape models. 2) The visual complexity of the visualizations is kept as low as possible. E.g., rather than combining the various techniques and enhancements, a better option is to allow switching between them one by one based on user interests.

2.6.1 Future work

Further improvements can be made in the use of texture patterns to show local surface shape as well as differences. In particular, if a surface is deformed, comparison of the deformed pattern to the original may yield more insight into the nature of the deformation. Texture-based techniques, possibly shown on cross-sections of the data, could help to illustrate these changes.

In our case study we applied our techniques to the visualization of statistical shape models. The statistical aspects of these models are currently missing from the visualizations. In work performed subsequent to that presented in this chapter [BBP10a] (see chapter 5), we presented an application for the visualization of high-dimensional shape spaces. We plan to integrate the intersecting surfaces techniques into our software in the future, as well as to extend these visualizations to include feedback on the statistical properties of the model.

Given two shape models for different populations, statistically significant differences can be determined between the models themselves [FPO+06]. By including such statistical information, our visualizations could help to link such statistical differences to physical differences in shape. This would aid both in understanding the causes and in validating the statistical models.

Finally, there is a need for intuitive methods for the interactive exploration of differences within the data. Our use of layered rendering already enables objects to be manipulated interactively. We are looking for ways to guide this interaction in

order to select and eliminate certain differences which are not relevant to the application domain. Additionally, we plan to integrate linked cross-sectional views and probing tools to provide quantitative statements about differences. Combined, these techniques will enable better exploration and visualization of relevant differences.

Acknowledgments

This research is supported by the Netherlands Organization for Scientific Research (NWO), project number 643.100.503 “Multi-Field Medical Visualization”.

CHAPTER 3

Direct Visualization of Deformation in Volumes

Abstract

Deformation is a topic of interest in many disciplines. In particular in medical research, deformations of surfaces and even entire volumetric structures are of interest. Clear visualization of such deformations can lead to important insight into growth processes and progression of disease.

We present new techniques for direct focus+context visualization of deformation fields representing transformations between pairs of volumetric datasets. Typically, such fields are computed by performing a non-rigid registration between two data volumes. Our visualization is based on direct volume rendering and uses the GPU to compute and interactively visualize features of these deformation fields in real-time. We integrate visualization of the deformation field with visualization of the scalar volume affected by the deformations. Furthermore, we present a novel use of texturing in volume rendered visualizations to show additional properties of the vector field on surfaces in the volume.

3.1 Introduction

Comparisons play an important role in many scientific areas. When comparing one object to another, differences between the two can be interpreted as deformations which transform one object to the other. In a medical context, such deformations often correlate directly to growth processes or the progression of diseases. For this reason,

analysis of deformations has become an important technique for medical researchers to understand these processes.

The analysis of deformation-fields with the purpose of studying morphological changes is called deformation-based morphometry, or DBM [AHF+98]. DBM is primarily promoted in brain-imaging research, where it is especially popular as it can be used to detect morphological differences over an entire brain. This is used, for example, to analyze all differences between subject brains and a standard brain in order to determine image-based characteristics that are associated with schizophrenia [GNB+01].

Another example illustrating the importance of volumetric changes in medical data is the study of rheumatoid arthritis (RA) and ostheoarthritis (OA). These are joint diseases that affect bone and cartilage in different ways. 3D MRI is increasingly being used to study the progression of these diseases over time. The diseases inflict progressively more damage on the affected joints in the form of, for example, bone erosions, bony outgrowths called osteophytes and changes in the cartilage. These characteristics are measured in MRI datasets [MSC+99, KKS+07] and are used to track the progression of the disease.

In all cases where 3D deformation fields have been used to study morphological changes (mostly brain imaging), only rudimentary visualization techniques have been applied. In general, aggregative metrics such as Jacobian-based volume change are calculated and used to typify differences. With suitable visualization techniques, differences can be studied in far more detail. In other areas where 3D image-based changes are being studied but deformation-fields are not yet being used, such as RA and OA progression, suitable visualization techniques should stimulate their introduction and hence facilitate these studies as well.

In this chapter, we present new techniques for the direct focus+context visualization of 3D deformation fields, aimed at facilitating the study of volumetric change in medical research. More specifically, they have been designed for use during the early exploration and hypothesis-generation stage of the medical research pipeline. Due to the nature of deformation fields, an ideal visualization of such data should provide insight into three things:

- the local behavior of the deformation in specific areas,
- the context of the deformations, in the form of the scalar volume that is being deformed,
- the effects of this deformation on this scalar volume

Our visualization approach fulfills these requirements: It enables the researcher to explore the deformation field in terms of areas with specific characteristics, such as growth or loss of tissue. Our techniques place these features within the context of the anatomical features present in the scalar volume, and also visualize how these features and the volume itself are affected by the deformations. In the next sections,

we first discuss existing techniques for analyzing and visualizing deformation fields. In section 3.3 we present the details of our visualization. Finally, we present and discuss results obtained by applying our visualization to both synthetic and clinical data, and give directions for future research.

3.2 Related work

3.2.1 Deformation analysis

The currently accepted standard for analyzing deformation between 3D images (e.g., CT, MRI) uses non-rigid registration (e.g., [ABH+06, RCA+06]). Registration is the process of deriving a transformation under certain constraints, which aligns images to bring them into the same coordinate frame. A rigid registration limits this transformation to, for instance, an affine transformation, while non-rigid registration allows the registration to match free-form deformations. The most commonly used class of non-rigid transformations are B-splines [RSH+99, KSP07].

After registration, the deformation field which represents the transformation from one image's coordinate frame to the other holds important information about the differences between the two images. Traditionally, this deformation field is analyzed using statistical methods (e.g., [PHS+08]).

3.2.2 Vector field visualization

While direct visualization can provide valuable insight, e.g. for selecting analysis methods and areas of interest, few papers have explored visualization of deformation fields. As deformation fields are vector fields, they are often visualized using techniques from flow visualization. For instance, Tittgemeyer et al. [TWK02] proposed a visualization based on an estimation of critical points within the vector field. Color-mapped surfaces are used to show the local deformations relative to certain important features within the scalar volume. Riddle et al. [RLF+04] used 2D color-mapped Jacobians to show changes in tissue volume on separate slices of the dataset.

Our visualization of deformation fields is based on direct volume rendering (DVR). DVR is commonly used to visualize scalar volumes. Volume rendering has been used in combination with methods such as LIC [HA04] in order to visualize vector fields. However, such methods require a pre-processing stage to generate the LIC volumes. Similarly, Crawfis et al. [CMBC93] applied LIC, volumetric extensions of streamlines called flow volumes and other techniques to create scalar "density" volumes based on the vector fields. These were then rendered using traditional methods.

Direct rendering of the vector fields has not received much attention. In earlier work, Crawfis and Max [CM92] presented an approach where the vector image is filtered with an oriented kernel during rendering, thereby visualizing the vector direction as an oriented line segment. Frühauf [Frü93] visualized vector fields with DVR

by mapping vector direction and magnitude directly to colors and opacities. Both approaches suffer from problems due to occlusion. In fact, dense 3D vector field visualization is considered an unsolved problem. Additionally, traditional flow and vector field visualization techniques are often not suitable for deformation visualization, as they do not provide enough context information to understand the deformations.

Compared to general vector fields, deformation fields such as those encountered in medical research have several properties which could be taken advantage of to improve their visualization. Depending on the nature of the data and problem under consideration, certain types of deformation may be considered more important than others. For example, changes in the amount of tissue are likely to be of greater interest to a researcher than movement of the patient between scans. Additionally, such features are likely to be local (e.g., growth of tumors or development of lesions). Because of this, visualizations of these features are less likely to suffer from occlusion problems.

Other techniques for analysis and visualization of deformation have been presented based on mechanical modeling and tensor fields representing stress and strain [HFH+04]. Computation of such fields operates under certain assumptions and implies an extra level of abstraction. Our visualization does not require pre-processing beyond the initial computation of the deformation field. Furthermore, we use techniques from importance-based visualization and non-photorealistic rendering to create a sparse but meaningful visualization of the field and its context.

3.3 Visualizing deformation

In this section, we present a focus+context visualization of deformation fields. While the process of obtaining the deformation field is outside the scope of this work, this usually involves applying registration twice. Here, the first registration is used to remove differences from the data which are not considered relevant to the problem being studied. For example, an affine registration can remove a difference in orientation of the patient between two scans. The second, non-rigid registration captures the remaining differences between the images in a deformation field. We share the assumption made in DBM that the deformation field for a large part reflects the actual physiological changes in the patient.

Figure 3.1 shows the components making up our visualization. The focus of the visualization consists of a direct rendering of features derived from the deformation field. We define measures that allow the user to highlight areas of the volume where the deformation has certain characteristics, e.g., growth or loss of tissue. Context is provided by simultaneous sparse volume rendering of the scalar volume. We distinguish between static and dynamic context in order to show both the surrounding anatomical features as well as the effects of the deformation. We present both interactive and static enhancements to the context visualization that show these effects.

The focus and context visualizations are integrated into a single visualization, such

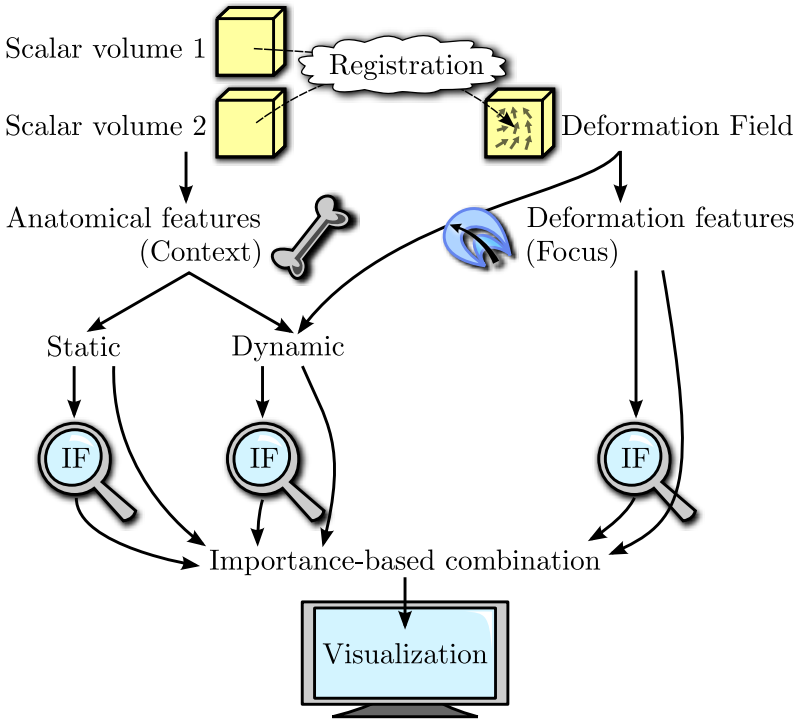
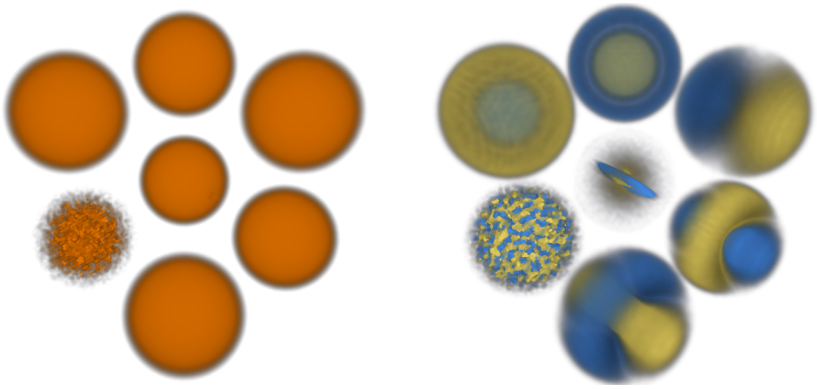


Figure 3.1: Visualizing deformation fields



(a) Magnitude (b) Growth

Figure 3.2: Visualizations of a synthetic 64^3 vector field. The left figure only shows vector magnitude (in orange), while the right figure shows our growth measure (yellow for negative values, blue for positive), giving a more detailed representation of deformation.

that the resulting combination presents relevant deformation features in an easily identifiable way *within* the context of the deforming volume. For this purpose, we define *interest functions*, which assign importance to features in each part of the visualization. Finally, we discuss the use of texture in direct volume rendering as a channel to provide additional information about the deformation vector field.

3.3.1 Deformation measures

The deformation fields our visualization deals with are given as vector-valued volumes. Traditional volume rendering uses a transfer function to map values in a volume to colors and opacities. Independently, however, the components of the vector field mean little. Furthermore, the design of transfer functions (let alone general three-dimensional transfer functions) is highly non-trivial. When examining deformations, a researcher is often interested in areas where the deformation has certain characteristics. Therefore, we define scalar valued measures that capture these characteristics. By visualizing these measures using volume rendering, we are able to highlight areas of the volume corresponding to the characteristics that are of interest to the user, without requiring explicit segmentation.

The most straightforward example of this is areas where the magnitude of the deformation exceeds a certain threshold. A direct volume rendering of vector magnitude in a synthetic deformation field is shown in figure 3.2(a). While this clearly shows areas of significant deformation, the nature of the deformation is not visible.

In analyzing vector fields, the Jacobian matrix J of the field is often used to gain insight into the local behavior of the field. The Jacobian is a matrix consisting of the three first-order partial derivatives of the field. The absolute value of its determinant, $\|J\|$, indicates the local volume change, and is therefore especially useful for analyzing deformations in medical data, as this directly corresponds to growth or loss of tissue. Based on the Jacobian determinant, we define the *growth* measure g:

$$g = \begin{cases} \|J\| - 1 & \text{for} \quad \|J\| < 1 \\ 1 - \frac{1}{\|J\|} & \text{otherwise} \end{cases} \tag{3.1}$$

Negative growth indicates loss of tissue. The measure is symmetric, and in the range $[-1, 1]$, with $g = 0$ meaning no change in volume, $g = -1$ meaning total loss of volume, and $g = 1$ meaning infinite growth. Its absolute value represents local tissue change. We therefore visualize the growth measure by using this absolute value to determine opacity. Features are color coded to indicate whether they represent growth or shrinkage. Figure 3.2(b) shows a direct volume rendering of growth features in a synthetic dataset.

Because the Jacobian is a derivative of the field, we can introduce the concept of scale [Wit83]. Approximation of derivatives is usually sensitive to noise in the data. By computing the Jacobian (and the measures derived from it) at a higher scale, the impact of such noise can be removed. Moreover, manipulation of this scale allows the user to easily filter the features in the dataset based on their scale. For this reason, computation of the Jacobian is done on the GPU, allowing for near-interactive manipulation of the scale parameter. We implemented derivative approximation using central differencing, where scale determines the spacing between samples. We also implemented derivative approximation based on convolution with a Gaussian derivative kernel. This yields higher quality approximations of the derivative with a smoothly adjustable scale, at the cost of performance.

While the growth measure is effective in showing areas where change in volume occurs, it fails to highlight areas of possibly significant deformation where the volume remains the same (e.g. translations). The simpler magnitude visualization shown in figure 3.2(a) does show these areas. However, these often occlude areas where the growth measure is significantly large, as well as potentially useful context structures (see section 3.3.2). To solve this problem, we use our importance-based filtering techniques presented in section 3.3.3 to combine the two visualizations, assigning higher importance to the growth features. This essentially highlights areas where deformation occurs without volume change. To avoid occluding growth features, these areas are not visualized directly. Instead, the magnitude measure is used in section 3.3.3 to determine areas where dynamic context information should be shown, which is sparse enough not to cause occlusion problems.

It should be noted that most non-rigid registration techniques result in a deformation field with vectors which for each point in a so-called *fixed image* represent the difference in position to the corresponding point in the *moving image*. Applying the

growth measure to such a field assumes the semantics of going from the fixed image to the moving image. That is, interpreted directly, the field describes a transformation which "deforms" the fixed image. Determining the inverse transformation is often impossible, however, due to the symmetry of the growth measure, its sign can simply be reversed to reverse these semantics.

3.3.2 Providing context

Deformation features such as those visualized in the previous section are meaningless to a medical researcher unless they can be related to anatomical features. In our visualization, such context information is provided by a scalar volume. Often, this will be one of the volumes used to compute the deformation field. Assuming the registration is perfect, the second volume can be derived from the first by applying the deformation. Therefore, we only need a single volume to provide context information.

In visualizing the context volume, we distinguish between two kinds of structures. *Static context* consists of structures not significantly affected by the deformation. *Dynamic context* consists of those structures which are deformed significantly, as well as the effects of the deformation on the volume itself. We use separate *modes* to visualize both types of context.

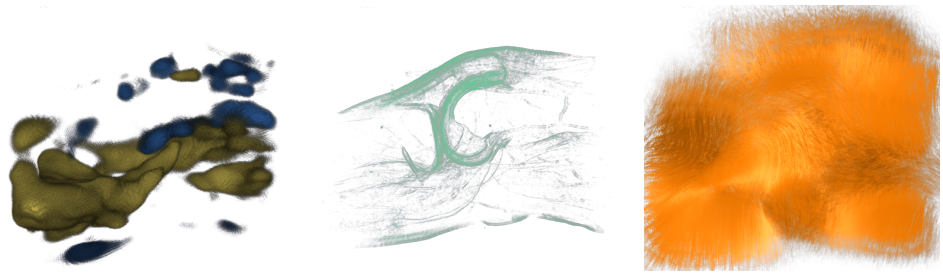
In order to minimize occlusion issues, we propose a sparse visualization of the context volume. As our visualization is aimed at medical researchers, we can assume advanced knowledge about the anatomical structures present in the volume. Therefore, it is not as important to show all information present in the scalar volume; such information can be better obtained using traditional volume visualization techniques. Because the deformations affect the shape of (anatomical) features within the volume, we focus on showing just these shapes.

Because values in CT or MRI scans are relatively uniform within separate tissues, high gradients often indicate boundaries between different types of tissue. Furthermore, the gradient can be used as an estimation of the normal of the boundary surface. To provide static context information, we only show parts of the boundary surface where this gradient is near-perpendicular to the viewing direction. This is achieved by volume rendering a *contourness* measure c, which is defined as follows:

$$c = s \left\| \vec{n} \right\| \left(1 - \max \left(1, \alpha \left| \frac{\vec{n}}{\left\| \vec{n} \right\|} \cdot \vec{e} \right| \right) \right) \tag{3.2}$$

where s is the scalar value at the current position, $\vec{n}$ is the scalar volume gradient, $\vec{e}$ is the unit vector pointing towards the camera and $\alpha \geq 1$ is a parameter controlling the sharpness of the contours. This results in a silhouette representation of the surfaces within the volume, shown in figure 3.3(b).

Visualizing dynamic context consists of two parts. First, we show the effects of the deformation on the structures shown in the static context mode. To achieve this, we can simply use the deformation field multiplied with a user-controllable value to



(a) focus (b) static context (c) dynamic context

Figure 3.3: Deformation visualization elements

offset sampling positions. This allows the user to interactively morph between the volume before and after deformation. We exaggerate deformations by allowing values outside of the range $[0, 1]$. The resulting caricaturistic visualization [RVG06] can make small changes in the volume more apparent. For example, the silhouette contours in figure 3.3(b) can be smoothly morphed between the two datasets visualized, or even show exaggerated deformations, thus emphasizing their effects.

While morphing works well in an interactive setting, it requires the use of animation, which is not always available when communicating results. For this reason, we also explored static methods for visualizing the effects of the deformation. As a static alternative to morphing we overlay the contours from the deformed volume on those from the original volume. We use different colors for both sets of contours, so they can easily be distinguished.

The second part of our dynamic context visualization is a visualization of the effects of the deformation on the volume itself. For this, we use a dense vector field visualization technique similar to spot noise [vW91]. For each position $\mathbf{p}$ we sample a precomputed volume containing uniform noise at multiple positions $\mathbf{p} + \delta\vec{v}$, where $\vec{v}$ is the deformation vector at $\mathbf{p}$ and δ ranges from 0 to a user-configurable maximum. This essentially smears out the noise volume along the vectors in the deformation field (see figure 3.3(c)).

3.3.3 Integrated visualization

The visualization described so far has three *modes*, examples of which are shown in figure 3.3:

Focus: visualizes features in the deformation field.

Static context: visualizes parts of the scalar volume unaffected by the deformation.

Dynamic context: visualizes the effects of the deformation on the volume.

These three modes could be used separately to visualize a given dataset. However, in that case relations between features seen in the different modes are not always apparent. We therefore integrate the modes into a single visualization. Our approach is to render all three modes simultaneously and combine results for each step through the volumes.

For each position in the volume, we select the mode that is most relevant for showing that particular point. We do this by defining *interest functions*, which assign importance to each point in the volume. The domain of these functions can be any value computed for this point in any of the modes. However, here we define interest functions which create a meaningful visualization, in which it is easy to distinguish areas with specific flow characteristics.

For the focus mode, the objects of interest are the areas where growth or shrinkage is high. We therefore use the absolute value of growth (equation 3.1) as the domain of the interest function. Defining the interest function similar to the opacity transfer function for this mode leads to good visibility of the features in the visualization. The interest function could be extended slightly outside the range of opacities to create an empty shell around growth features. This makes them stand out more in the resulting image.

As our static context consists of thin and very sparse contours, we can afford to assign these high importance. This way, they will always be visible, even if other features occupy the same space. Importance is again assigned similar to opacity, based on the measure defined in equation 3.2. A distinction between dynamic and static context can easily be made based on the local magnitude of the deformation field. We therefore use magnitude as the domain for the dynamic context interest function. We assign dynamic context features lower importance than focus features. This way, we highlight areas where deformation occurs without a change in the amount of volume / tissue.

During rendering we compute the values of these three interest functions for each step through the volume. The mode with the highest importance is selected and applied, resulting in a color and opacity. Finally, these colors and opacities are composited as in traditional direct volume rendering. In our prototype implementation, the interest functions consist of simple linear interpolations from 0 to 1 between double thresholds on the function's domain. These thresholds can be manipulated interactively by the user (as well as the normal transfer functions for each mode), allowing for easy manipulation of the relative importance of features in the visualization (see figure 3.4)

One important issue to consider when combining multiple modes in a single visualization is that it may not be clear to which mode a given feature belongs. To solve this issue, we use color only to distinguish between features with different semantics. Furthermore, the different modes each have their own visual style, to make them easily distinguishable.

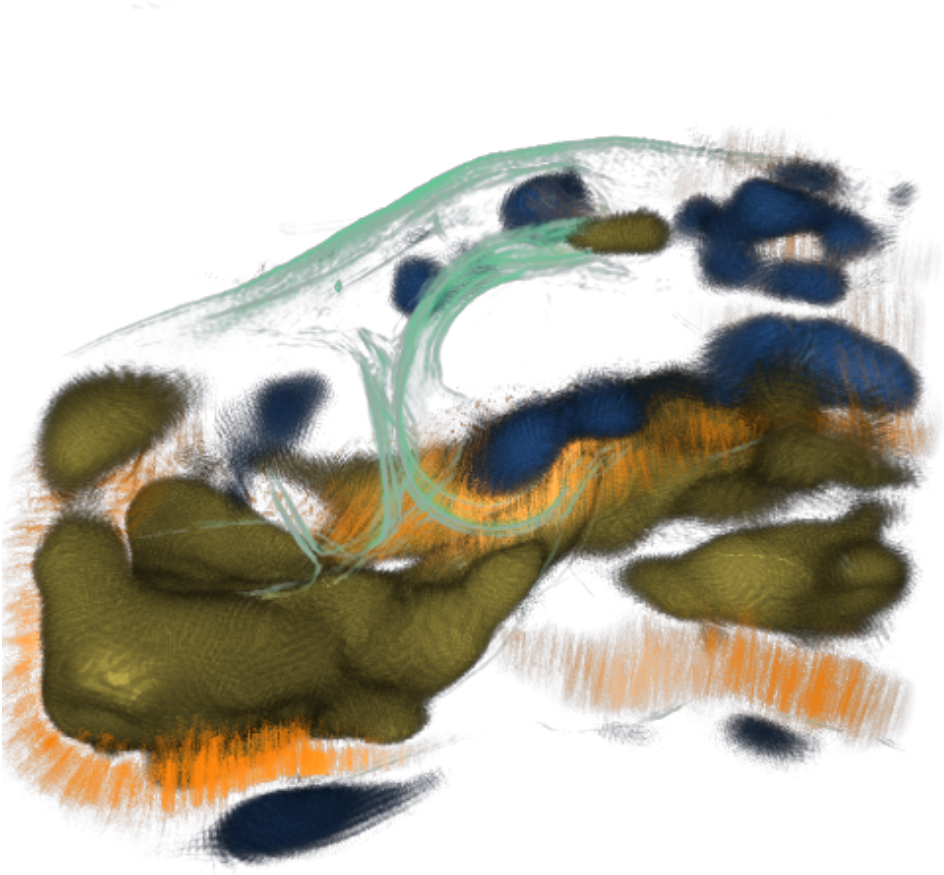


Figure 3.4: Combined visualization of focus and context features in a $512 \times 512 \times 80$ knee dataset. Jacobians were computed with a kernel radius of $4 \times 4 \times 0.6$ voxels. Growth features are shown for $\|g\| > 0.14$.

3.3.4 GPU-based ray casting of scalar and vector fields

We base our techniques on our GPU-accelerated multi-volume ray caster, which uses a similar technique to that presented by Krüger and Westermann [KW03]. In short, we use a depth peeling approach to render proxy geometry for each volume one layer of geometry at a time. A fragment shader is used to perform ray casting between the layers. We keep track of active volumes for each layer, thereby enabling simultaneous multi-volume ray casting. The complete rendering process for two volumes is shown in figure 3.5.

Rays are cast for each layer from the positions on the previous layer to those on the current one, using a fixed step size (see figure 3.5). Active volumes are determined, loaded as textures, and sampled for each position along the ray by transforming the current position into their local coordinate frame. Both the measures described in section 3.3.1 and the visualizations described in section 3.3.2 are computed in real-time from the vector and scalar volumes on the GPU. The resulting colors and opacities are combined using the importance filtering techniques described in section 3.3.3, and then composited to form the resulting color and opacity for the layer. To reduce the visual impact of artifacts caused by the fixed step size, we offset the sampling positions of the rays in neighboring pixels by fractions of the step size. While the result has a dithered appearance, it leads to better visibility of structures within the volume, even for larger step sizes, which in turn lead to better performance.

Optionally, gradients can be computed in real-time using central differencing. These can then be used for lighting, allowing for better perception of shapes in the resulting visualization. Due to the computational complexity, however, computing gradients of growth features may be too expensive for real-time rendering. A simple alternative, used in the figures in this chapter, is to use a limb darkening effect similar to that used in [HA04]. Limb darkening can be achieved by using a softer transfer function to create a darker halo around surfaces, providing the effect of shading without needing to compute surface normals.

A different technique we explored is using streamline shading [Frü93] to visualize deformation direction. However, this technique only works well on thin, line-like objects oriented in the direction of the deformation. As the features in our focus mode generally form blob-like surfaces, streamline lighting creates a confusing appearance and does not provide intuitive insight into the deformation direction. For this reason, we apply this form of lighting only to the smears in our dynamic context visualization, which do have a clear linear structure.

3.3.5 Using texture

While texture hardware is often used for volume rendering, actual texturing is not commonly used in volume visualization due to the complexities of mapping two-dimensional images on arbitrary surfaces. However, due to the simplistic and (at least at larger scales) relatively smooth nature of the surfaces in our visualization,

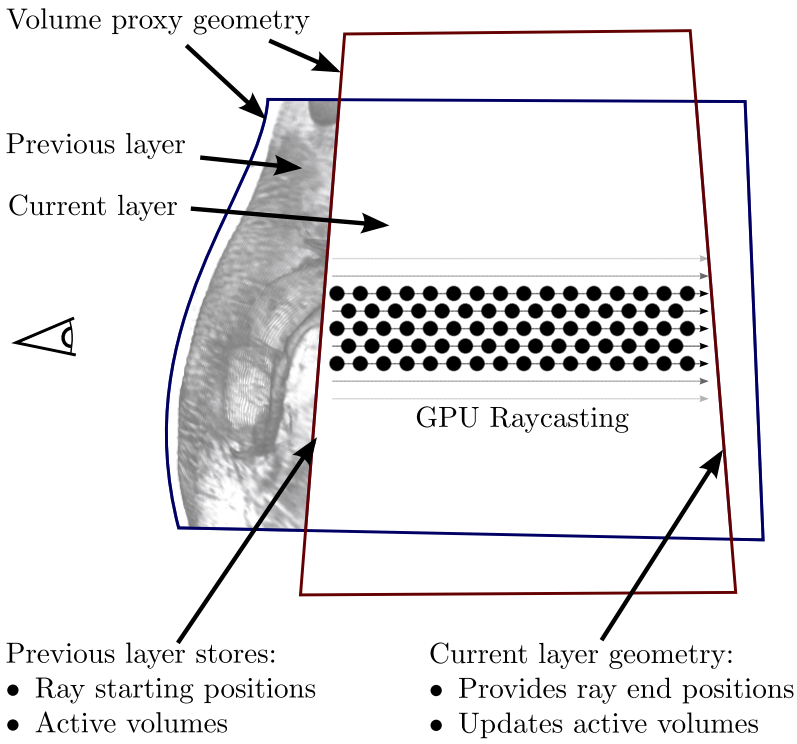
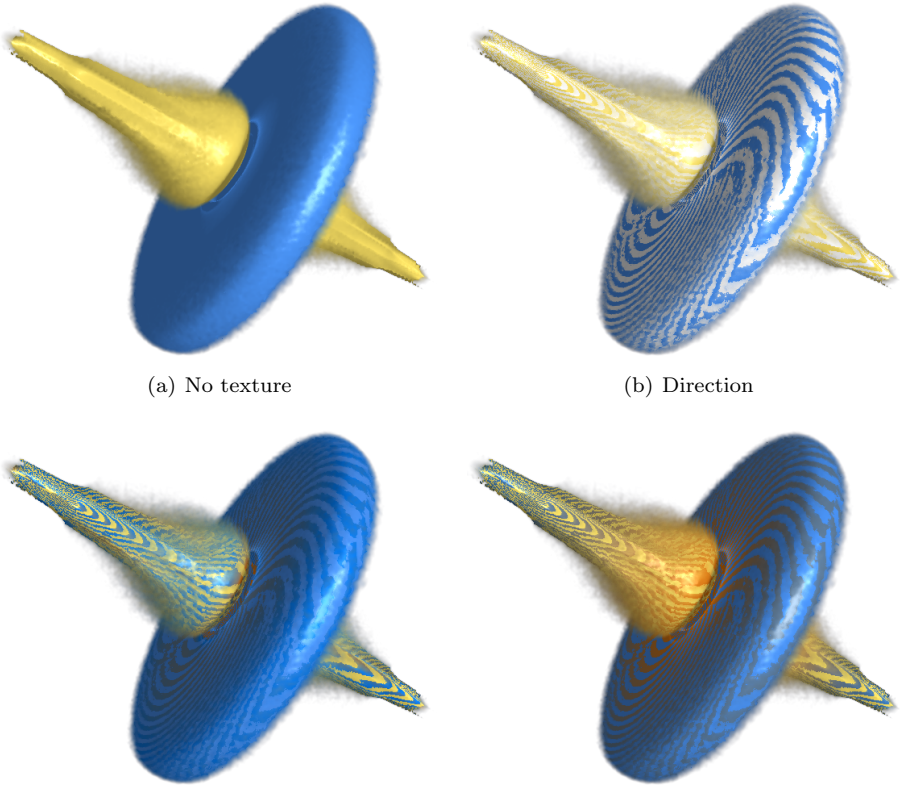


Figure 3.5: Using a layered approach for GPU-based multi-volume ray casting



(a) No texture (b) Direction

(c) Perpendicularity (d) Magnitude

Figure 3.6: Enhancing the visualization with texture reveals various properties of the deformation field, such as a rotational component in this synthetic saddle point deformation (128^3). Colors in the last two figures go from blue (low) to orange (high).

and due to our choice of single colors for different features, texture forms an available channel for presenting additional information to the user.

We use texture to indicate the local direction of the deformation vector field. Because this direction is obviously three-dimensional in nature, we split the vector in a surface-tangential and a perpendicular component. The tangential direction is visualized using an oriented stripe texture. The angle between the vector and the surface is visualized in the color of the stripes. Similarly, color can be used to visualize vector magnitude. Figure 3.6 shows the same feature both without and with several variations of our texturing technique.

Because the surfaces of features might be noisy at lower scales, computing the texture directly on the tangent plane results in a noisy image where the direction may not be clearly visible. Instead, we project the deformation vector to the axis-aligned plane most similar to the tangent plane, and compute the texture based on this vector. Next, a striping pattern is applied along the vector perpendicular to the projected deformation vector in the plane. The width of the stripes can be changed by the user to adapt the texture to vector fields with different smoothness.

3.4 Results

The algorithms described in this chapter were implemented in C++ using OpenGL and the GLSL shading language for GPU programming. The visualization runs with interactive performance on Nvidia GeForce GTX 280 hardware, achieving an average frame rate of 10 fps on the visualization shown in figure 3.7. Applying texture or lighting to the focus features is more computationally intensive, as this requires estimation of the surface normal for these features. Given current trends in GPU capabilities, however, performance of these techniques is likely to improve significantly with next generation hardware. The current implementation is limited to volumes and deformation fields that fit in GPU memory. Out-of-core techniques could be used to remove this limitation.

Results from applying our visualizations to artificially generated deformation fields are shown throughout section 3.3. These vector fields were generated by additive combination of simple vector fields representing rotations, saddles, translations, sources and/or sinks [WH91]. While we applied our visualization to datasets from an OA study (shown in figure 3.3 and figure 3.4), these images are intended only to illustrate our techniques as we have yet to work with domain experts to validate these results. In addition to these datasets, we used a pair of MRI images, taken 4 months apart, of the brain of a patient suffering from Multiple Sclerosis. MS leads to the appearance of lesions in the white matter of the brain. The data shows a high number of such lesions, some of which grow, shrink, appear or disappear between the two scans. The images had previously been normalized. As is common in deformation analysis research, we used elastix [KSP07] to apply a standard B-spline non-rigid registration

between the two images in order to obtain a deformation field. We then applied our techniques to visualize this field together with one of the original MRI images.

Our visualization, shown in figure 3.7, shows that changes in the brain clearly appear as growth and shrinkage features. Most of these features are yellow, which corresponds to contraction or loss of tissue. Experts hypothesize that the increasing number of lesions leads to a reduction in the volume of the brain ventricles. The yellow features in the center of the brain seem to be in line with this hypothesis, although further analysis is required to confirm this. The visualization also shows a large deformation outside the brain in the lower left corner, which is probably due to misalignment. The yellow feature on the far left is an artifact, resulting from padding the dataset.

3.5 Conclusions and future work

We presented new techniques for the focus+context visualization of deformation in volumes, useful in studying development and disease progression in medicine. Our techniques directly visualize deformation fields in terms of the nature and effects of the deformations. In summary, we identify the following contributions:

- direct visualization of features derived from deformation fields, allowing exploration of the fields on a higher semantic level than traditional vector visualizations,
- direct volume rendering of the underlying scalar volume, to show both *context* and *effect* of the deformations,
- importance-based integration of both visualization techniques into a single image,
- the use of texture to present additional information about the deformation field in relation to the features shown in the visualization.

While our focus in this research was on medical datasets, the techniques presented in this chapter can be applied in other areas as well. In future work, we intend to refine these techniques and work with domain experts, most notably in OA research, to apply our approach to real data. For this, new types of measures may need to be developed to capture characteristics of interest to these experts. Additionally, we aim to extend our filtering techniques to allow a user to select features based on various criteria other than scale. Finally, we intend to extend our techniques to allow comparison of two or more deformation fields.

Acknowledgments

We are grateful to Dr. G. Kloppenburg and Dr. I. Watt of respectively the Reumatology and Radiology departments of the Leiden University Medical Center for

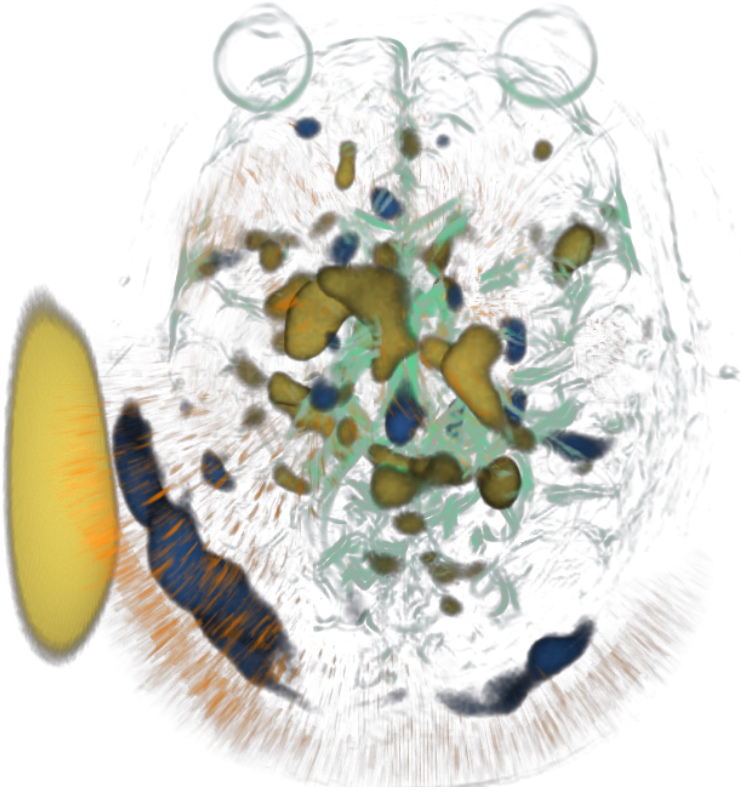


Figure 3.7: Visualization of deformation due to changing white matter lesions in a brain MRI dataset of $181 \times 217 \times 179$ voxels. Jacobians were computed with a kernel radius of 2 voxels. Growth features are shown for $\|g\| > 0.09$.

their valuable input and OA research datasets. This research is supported by the Netherlands Organisation for Scientific Research (NWO), project number 643.100.503 “Multi-Field Medical Visualization”.

CHAPTER 4

Example-based interactive illustration of multi-field datasets

Abstract

Multi-fields are widely used in areas ranging from physical simulations to medical imaging. Illustrative visualization techniques can help to effectively communicate features of interest found in a given field. Current techniques for multi-field visualization are mostly focused on showing subsets of local attributes such as single values or vector directions, e.g., using colors, texture, streamlines or glyphs. Instead, we present an approach based on highlighting areas with similar characteristics, considering all attributes of the field.

Our approach is example-based and interactive. A user simply selects a point within the field, upon which the system automatically derives the characteristic combination of attributes for that point. Our system then automatically creates a visualization highlighting areas within the field which are similar to the example point with respect to these characteristics. The visualizations are presented using sparse, illustrative techniques, using contours and colors to clearly delineate and identify separate areas. Users can interact with the visualizations in real-time, by moving the example point or, optionally, by changing the characteristics or adjusting other parameters used to determine similarity.

4.1 Introduction

Detection and extraction of features is an important step in the process of visualizing any dataset. Features are patterns of interest within the data. As interest in and specification of such patterns vary between applications, features are often segmented from the data using application-specific approaches.

With this work, our contribution is the following: We present a method whereby user interest in any multi-field dataset can be indicated by simply pointing somewhere in the dataset. Based on a real-time analysis of the high-dimensional data attributes at this example point and an optional point of contrast, the system then determines and illustrates all similar regions as feature objects. The user can interactively manipulate existing or add new feature objects, resulting in illustrations such as shown in figure 4.1. This enables an explorative approach to the specification and investigation of features in multi-field datasets.

Our techniques can be used to create an *illustration-by-exploration* approach: Rather than creating a single visualization that shows all features, we create sparse, user-guided visualizations of specific features, where an overview of the entire dataset and the context of features is obtained through exploration rather than being given in a single, potentially cluttered image. Illustrative rendering techniques are well-suited to visualize features found through such exploration, due to their sparseness and ease of interpretation.

These techniques can supplement existing visualization and segmentation techniques by offering a quick and intuitive way to explore a given dataset, and assist in locating and examining the characteristics of features in the dataset. This information can then be used by existing techniques to extract and / or quantify such features.

In the following, we first discuss work related to multi-field visualization and our proposed approach. We then present our interactive exploration approach, followed by a detailed description of the feature extraction and rendering techniques involved. We illustrate our approach with several example visualizations of extended vector and tensor fields. Finally, we present our conclusions and directions for future work.

The complete implementation of our techniques and the prototype application have been released as open source software. Windows binaries and cross-platform source code are available at `http://multifieldexplorer.googlecode.com/`.

4.2 Related work

Fields are functions that are defined on the spatial domain. Multi-fields or multi-field datasets simply refer to datasets consisting of two or more fields on the same spatial domain. This definition includes data consisting of any combination of multiple scalar fields, vectors and tensors. For an extensive overview on the visualization of this type of data, we refer the reader to the survey by Fuchs et al. [FH09]. In the remainder of this section, we give a compact summary of current work on multi-field

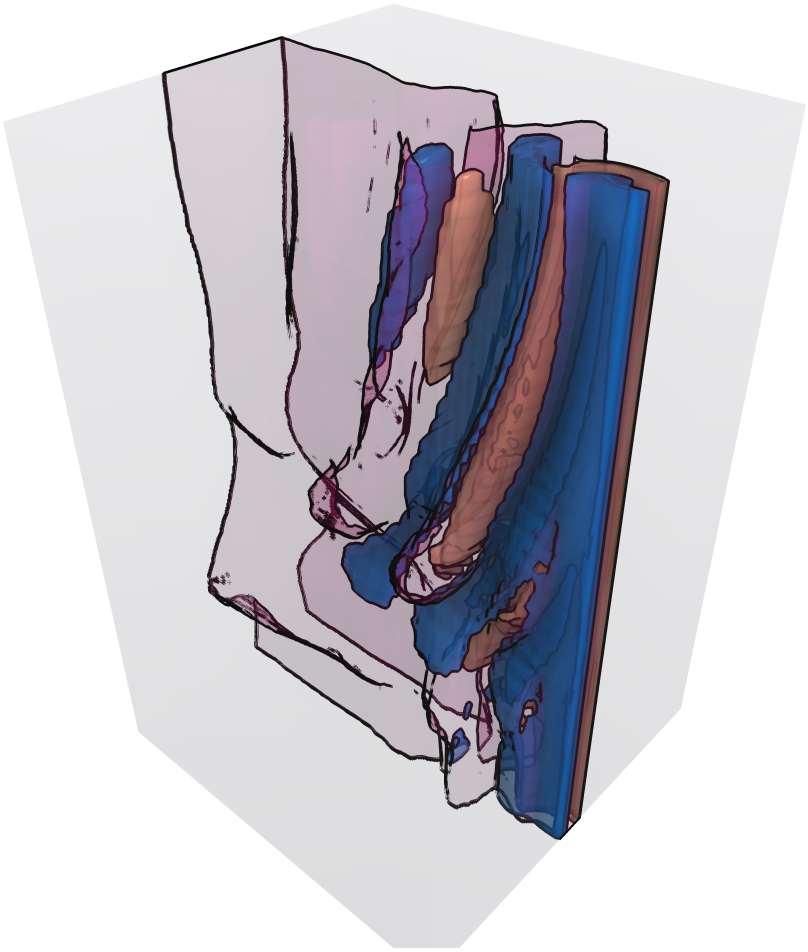


Figure 4.1: Illustrative visualization, created using our approach, of a multi-field representing the flow behind a tapered cylinder, placed vertically on the right side just outside the field. The field consists of the velocity vector field and its derivatives. The illustration highlights the alternating vortices characteristic for such flow, selected by curl similarity to a point in one of them. The transparent surface shows the area where flow velocity is altered by the cylinder and was selected by contrasting the velocity behind the cylinder with that at the edge of the field.

data visualization, grouping and ordering techniques according to their level of data abstraction.

In feature-tracking, features are objects, local behaviors or other patterns of interest in a dataset, whereas in many other fields they are the dimensions of a feature space or components of an image. In order to resolve this ambiguity, we refer to the former as *feature objects*, while the components of a multi-field are referred to as *attributes*.

Current techniques for vector field visualization often focus on showing directions [LHD+03]. In tensor visualization, second order symmetric tensors can be represented by ellipsoids or super-quadrics [Kin04], or reduced to their principal directions and visualized as a vector field. Analogously, many techniques for multi-scalar visualization are also based on representing each sample point with a single visual element. For example, in multi-scalar volume rendering, the transfer function is extended to map from the multi-dimensional input space to the RGBA output space [KKH02]. These techniques all attempt to visualize multi-field data as directly as possible.

Visualization of quantities derived from a field can often show more about the field than a visualization of only the *direct attributes* that are part of the field. For example, the inclusion of image gradients has been shown to lead to better transfer functions [KKH02], and Busking et al. [BBP09] visualized deformation vector fields by highlighting areas of significant change in volumes based on a measure derived from the field's Jacobian. However, different derived quantities may be required for different applications. Van Walsum et al. [vWP94] presented a framework which allowed users to select and visualize arbitrary quantities derived from vector fields. Their system also allowed users to combine the resulting feature objects using Boolean set operations. Similar compositing techniques were applied to the visualization of multi-field and time-varying data by Woodring and Shen [SW06], although only direct attributes of the fields were used in their work.

A multi-field can also be seen in terms of its feature space, where each attribute is a dimension. Henze [Hen98] presented a visualization approach using multiple views of "linked derived spaces", which are essentially projections of this feature space. Kniss et al. [KKH02] presented a dual-domain interaction approach for exploring multi-field volume data, where interactions in the spatial domain are mapped to a linked view of the feature space. This view can then be used to manipulate transfer functions (defined on this feature space), which are in turn used for direct volume rendering of the volume data in the spatial domain. This idea was further developed by linking multiple scatter plots of the feature space with a direct volume rendering showing selected subsets of the data [GRW+00, DGH03, BBP07].

Finally, feature extraction techniques attempt to reduce complexity in the data by extracting physically meaningful patterns, hence lifting the resultant visualization to a higher level of abstraction [PVH+03]. This is also the goal of visualization techniques based on topology [HTSW03], clustering [GPR+04, MLD05] or the extraction of physical properties, such as vorticity. Vortex extraction, the most common type

of feature extraction technique for vector fields, is often based on a combination of physical and mathematical criteria [PVH+03].

The efficacy of topology techniques on vector and tensor data is highly dependent on the presence and configuration of respectively critical and degenerate points. Results of clustering are often too dense for illustration. Furthermore, control is limited to selecting clustering heuristics, which may be non-intuitive. Both techniques are also computationally intensive, which makes them less suitable for interactive use. Extraction of physical properties is highly domain-specific.

Ma proposed using machine learning techniques, neural networks for example, to extract meaningful physical feature objects from the multi-field data, based on user interaction [Ma07]. This is a good attempt towards automatic generation of feature detection criteria, but is difficult to control and not directly and transparently linked to field attributes.

In this chapter, we combine the direct visualization of arbitrary combinations of field attributes with a feature space approach and example-based interaction. Rather than visualizing attributes directly, we introduce the use of *local similarity* as the basis for feature selection. We use example-based, user-adjustable similarity measures computed in feature space to visualize feature objects consisting of all points within a dataset that are similar to a user-defined example. This way, rather than specifying the characteristics of feature objects directly (e.g., extracting an iso-surface of attribute X at value Y), the user can simply point at interesting areas in the data.

In our work, illustration plays an important role in visually summarizing the data. Illustrative visualization is often used to give a deliberately unbalanced view of a complex scene, by emphasizing objects of interest, de-emphasizing or suppressing other objects, and showing context only for orientation. Various techniques have been proposed to integrate such visualizations of focus+context in a single image, including cut-away views or importance-driven visualization [VKG04]. One could also see example-based feature specification as a flexible and interactive method of specifying an importance field. For our work, however, we take advantage of the interactivity of our techniques to create a strongly selective approach, where the objects shown are only those selected by the user. Context should be provided either by interactive exploration, the addition of other user-specified objects, or by integration of our techniques with existing, possibly domain-specific techniques.

In terms of the example-based specification of feature objects, our work is related to the stroke-based transfer function specification of Ropinski et al., where the user specifies interesting edges by indicating them directly in the rendering by means of a stroke, resulting in the automatic specification of a transfer function that emphasizes the visual appearance of all stroked edges [RPSH08]. Our approach is differentiated by catering for multi-fields and utilizing the direct calculation of similarity versus indirect application through a transfer function.

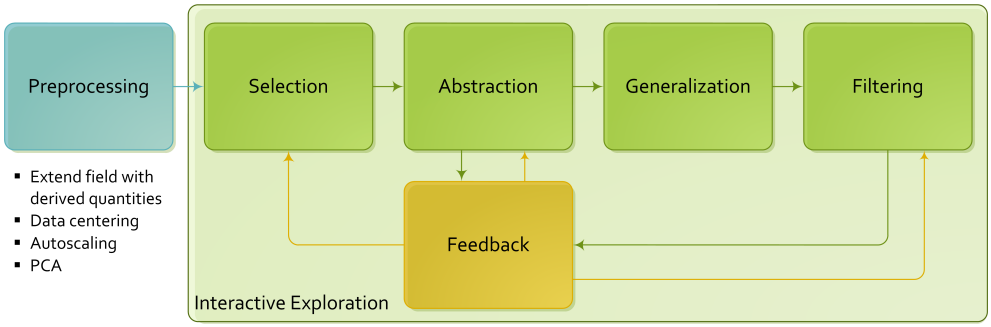


Figure 4.2: The workflow in our interactive illustration system. Defining a feature object consists of selecting an example point in the field (selection). The values at this point are used to automatically select characteristics of interest (abstraction). These are then used to generate a similarity measure (generalization), from which the feature objects are extracted (filtering). The system is interactive, allowing a user to provide feedback and immediately observe the results in the visualization.

4.3 Interactive exploration

We aim for *interactivity*, as this enables our visualization to be used as a tool for interactive exploration as well as for creating illustrations that are effective in conveying meaningful feature objects identified during exploration. Globally, the work-flow of a user using our system to illustrate a field is as shown in figure 4.2:

1. As a *preprocessing* step, a multi-field is constructed from the input field, containing any number of attributes. For example, a vector field can be extended by adding derived quantities such as magnitude, divergence or curl. Adding such attributes allows any feature objects characterized by these attributes to be identified and visualized by our system. It also allows the system to better describe such feature objects to the user by presenting the defining characteristics in terms of these attributes.

2. *Selection:* the user selects a point of interest within the field, or a pair of points which is to be compared. The values at the first point determine the example on which the feature object is based. The second point, if given, determines values that should be outside the feature object.

3. *Abstraction:* the system automatically determines characteristic attributes for the given point or pair of points and presents these to the user.

4. *Generalization:* Based on these characteristics the system determines an appropriate similarity measure. This measure combined with the example point defines a similarity field over the dataset.

5. *Filtering:* the resulting feature object, consisting of all points in the volume that are similar to the selected point with respect to the chosen characteristics, is visualized interactively by thresholding the similarity field.

6. *Feedback:* moving the selected point of interest updates the visualization in real-time, allowing interactive exploration of the data. The user can also refine the feature object by manipulating the selection of characteristics, or by selecting from a number of visual styles to add additional semantics to the illustration.

The process can be repeated to add any number of additional feature objects to the illustration. In this way, a user can select multiple feature objects by using multiple example points, each with its own selection of characteristics and visual styles, thereby creating an illustration combining all feature objects of interest.

Section 4.4 describes our techniques for example-based selection of meaningful feature objects. These objects are visualized interactively using illustrative styles, which are detailed in section 4.5.

4.4 Example-based selection of feature objects

Our illustrative exploration approach is based on extracting similarity-based feature objects from the data. These objects are defined as those areas which are similar to a user-defined point with respect to a combination of user-defined characteristics. We use a similarity measure defined on the points of the multi-field in combination with a pair of user-selectable thresholds to derive such areas. By adjusting the thresholds, the user can decide when points are considered "similar enough" to be included in the feature object.

Our approach is based on the notion of feature spaces. A sampling of any multi-field consisting of N variables can also be interpreted as a set of points in a *feature space* of N dimensions. Section 4.4.1 describes our definition of similarity in such a feature space. The methods for creating a compact feature space in which these similarity measures can be applied are described in section 4.4.2. Finally, section 4.4.3 describes how we use user-defined examples to derive appropriate parameters in order to enable our interactive exploration approach.

4.4.1 Similarity

Various similarity metrics for multi-fields or vector fields have been proposed in literature [DX08,STR03]. Most of these, however, only compute global similarity between whole fields. Furthermore, some metrics are expensive to compute.

As noted above, we base our similarity metric on the feature space representation of the multi-field. Similarity (or rather dissimilarity) is equivalent to distance within this feature space. We transform the original feature space using projection and scaling in order to provide user control and example-based refinement of the similarity measure.

To keep our application simple and interactive, we use the p-norm to measure distances in this transformed space:

$$d(\mathbf{a}, \mathbf{b}) = \left(\sum_{i=0}^{N} |a_i - b_i|^p \right)^{\frac{1}{p}}$$

Here, $\mathbf{a}$ and $\mathbf{b}$ are feature vectors consisting of the values of all attributes of the multi-field, representing the two points that are being compared, after applying the feature space transformations described below. The parameter $p \geq 1$, can be used to change the type of norm computed. This is generally set to 2 for the Euclidean norm.

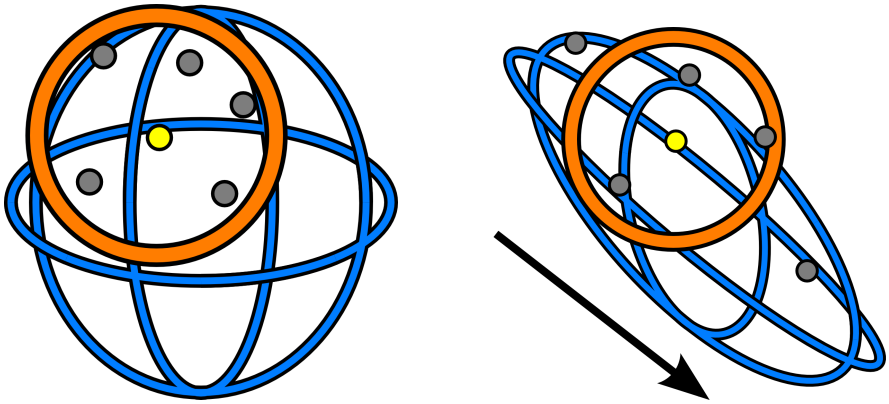
Projection

Not all attributes may be of interest when determining similarity. In fact, a user may want similarity to be invariant with respect to certain attributes. If those attributes directly correspond to dimensions in the feature space it suffices to simply omit the respective values from the similarity computation. However, we reduce the dimensionality of the feature space before visualization using principal component analysis (as described in the next section). In this case attributes can be eliminated by *projection* onto a subspace containing only the remaining attributes. To achieve this, the transformation between the original and reduced feature spaces is stored during preprocessing. The projection of a feature space point is computed by transforming the point's feature vector back to the original space, setting the values corresponding to the attributes that are to be ignored to zero and transforming the result back to the reduced feature space.

Non-uniform scaling

Furthermore, we enhance the similarity computation by introducing a *bias vector*. The bias vector is a vector in feature space, determined automatically based on user input (see section 4.4.3), along which this space can be stretched before determining similarity. The effect is that changes proportional to the linear combination of attributes specified in the vector have more influence on the value of the similarity measure than other changes, as illustrated in figure 4.3. Stretching is implemented as simple linear scaling along the bias vector with user-adjustable scaling factor s. Adjusting the scaling factor allows the similarity measure to be shifted between neutral ($s = 1$) or any amount of bias ($s > 1$). Values $s < 1$ are also allowed; the resulting similarity measure favors similarity in ways uncharacteristic of the selected example. Such measures may be used to uncover patterns normally obscured by trends in the most characteristic attributes for a point.

The scaling operation is made volume-preserving with respect to the user-selected feature subspace by pre-scaling the entire space uniformly by factor $s_u = s^{-1/(N-1)}$, then scaling along the bias vector by $s_n = s/s_u$. Here N is the number of dimensions



(a) Normal feature space (b) Biased feature space

Figure 4.3: Non-uniform scaling of the feature space. The orange circle represents the thresholded similarity measure, and the small circles represent the example (yellow) and other feature vectors. Stretching the feature space makes the measure more sensitive to changes along the bias vector (represented by the black arrow) and less sensitive to other changes.

of the original feature space remaining after projection. This way, given a uniform distribution of points in the feature space, feature objects will include approximately the same number of points regardless of the value of s. In the case where s approaches infinity, the similarity measure becomes equivalent to distance along the bias vector.

4.4.2 Creating the feature space

Multi-fields can potentially have large numbers of attributes, which directly affects performance and memory requirements in an interactive application. Furthermore, feature space dimensions may be heterogeneous in scale, or contain statistically dependent behavior.

In order for our techniques to find interesting relations, however, it is good to have a large number of attributes between which such relations may be found. Additionally, feedback (e.g., on user selections and bias vectors) is given and projections are defined in terms of these attributes. This means a good selection of attributes can help a domain expert to understand features found in the dataset in terms of the application domain. For this reason we pre-compute the feature space for a given multi-field in four steps:

1. Build the feature space from the points in the original multi-field. Optionally add various derived quantities as attributes of the field, such as local derivatives (e.g., gradients) or application-specific measures relevant to the user's interest.

2. Center the feature space on the origin by computing the mean value over the data for each dimension and subtracting this mean vector from each point.

3. Scale the centered feature space such that the variance over each dimension is 1 by dividing each dimension's values by their standard deviation.

4. Perform principal component analysis (PCA) on the scaled space, then transform the data to a basis of eigenvectors in order to reduce the dimensionality while maintaining as much data variance as possible.

PCA creates a basis of eigenvectors in order of how much they account for the variability in the original data. This means dimensionality can be reduced by selecting only a subset of the first N eigenvectors. However, if the original feature space is heterogeneous in nature, the PCA may favor larger scale dimensions. To alleviate this, we use *autoscaling* (i.e., division by the standard deviation for each dimension) to pre-scale the feature space dimensions, such that PCA is based on correlation rather than covariance. Pre-treatment techniques such as autoscaling can strongly influence the results of the PCA. The selection of an appropriate technique is therefore highly application-dependent. Van den Berg et al. [vHW$^+$06] give a good overview of autoscaling and alternative techniques, and their effects on the results of the PCA.

PCA is a commonly used technique in pattern recognition for the purposes of reduction and decorrelation of high-dimensional data. While our experiments (see section 4.6.2) have shown that the resulting feature spaces enable good selections of features within the data, alternative approaches exist as well. Heimann and Meinzer [HM09] give a good overview of alternative techniques for dimensional reduction in the context of statistical shape models, which could also be adapted to this application.

4.4.3 Example-based similarity

Feature objects are extracted by thresholding the similarity field created by comparing each point in the data with the values at the user-defined example location. We use the bias vector to steer the similarity measure to not only consider the values at this location, but also their relative proportions. The bias vector is automatically determined based on the *example vector* of values at that location and a second *contrast vector*. We provide two options for selecting the second vector:

- The contrast vector can be set to the mean feature vector computed over the entire field. This way, deviations from the mean are emphasized as characteristic attributes for any user-selected point.

- The contrast vector can be sampled at a user-defined location. For instance, by selecting a location outside of the objects of interest in the field, characteristic attributes can be determined based on comparison to background values.

Alternatively, such a selection can be used to emphasize the differences in characteristics between different feature objects in the field by placing example and contrast points in the objects to be compared.

While the resulting bias vector is normalized before being used to stretch the feature space, its original length is used (together with the stretch factor) to adjust the user-defined thresholds. This way, a threshold of 1 will lead to the feature object including all points with similarity values up to the difference between the two points that were selected.

In summary, given an arbitrary point $\vec{p}$, example feature vector $\mathbf{x}$ (sampled at a user-defined location) and contrast feature vector $\mathbf{c}$ (which is either the mean of the data or the feature vector at a second user-defined position), the corresponding feature object is extracted as follows:

1. Sample the multi-field to obtain the feature vector $\mathbf{p}$ corresponding to $\vec{p}$.
2. Project feature vectors $\mathbf{p}$, $\mathbf{x}$ and $\mathbf{c}$ to a subspace which eliminates a user-defined subset of "irrelevant" attributes.
3. Transform the projected feature vectors $\mathbf{p}'$ and $\mathbf{x}'$ to the stretched space given by the bias vector $\mathbf{v} = \mathbf{x} - \mathbf{c}$ and a user-defined bias factor s.
4. Compute the p-norm distance between the resulting vectors $\mathbf{p}''$ and $\mathbf{x}''$.
5. Compare the resulting value to the user-defined thresholds to classify point $\vec{p}$ as being inside or outside of the feature object.

4.5 Illustrative visualization

Our objects of interest are areas similar to the user-defined examples. We visualize each of these feature objects using an iso-surface of the similarity measure at a user-defined threshold. As we aim for our visualization to be used in interactive exploration scenarios, we use a sparse visualization style which is easy to comprehend, but provides enough detail to help the user gain insight into the data. Our style is inspired by schematic drawings as used in biology and engineering, which make heavy use of contours and simple textures to delineate and annotate feature objects in an image.

In order to enable interactive exploration and to avoid the complexities of surface extraction, we render the iso-surfaces directly using a ray casting approach. Using current generation GPU hardware, ray casting approaches can be implemented with real-time performance.

We use a GPU ray casting algorithm based on the technique introduced by Krüger and Westermann [KW03]. Rendering involves using a rasterized bounding volume to determine ray entry and exit points. These are then used to trace each ray in parallel using a fragment shader, stepping through the multi-field in fixed intervals. At each

point along a ray, we sample from volumes containing the preprocessed field values to obtain the feature vector for that point. The similarity measures for all feature objects are then evaluated as described in section 4.4 in order to find intersections with the feature objects' surfaces. If multiple surface intersections are found, the positions of these intersections are refined and then sorted. The front-most intersection determines the surface visualized for that ray.

We use a deferred shading approach [ST90] in order to create our illustrative visualization style. Rather than performing surface shading directly during ray casting, our ray casting pass outputs surface information to a G-buffer, including feature object number, surface normal and the position of the ray-surface intersection. The deferred shading pass uses only the information in this buffer to shade the corresponding pixels. Furthermore, in order to support the use of transparent surfaces, we use depth-peeling [Die96] in order to apply the deferred shading enhancements to each of the surfaces encountered along the rays. The results of each deferred shading pass are composited to create the final image.

The deferred shading pass is used to create the illustrative style of our visualizations. The use of transparent surfaces can help show the interior structure of feature objects, but multiple nested transparent surfaces are often hard to interpret. To alleviate this, object contours are enhanced using image-based detection of silhouettes, surface intersections and sharp edges. This enables the viewer to easily distinguish the shapes of feature objects even if transparent surfaces are used. Fully transparent surfaces (showing only contours) can be used as well, and may be helpful for adding contextual information to a visualization.

By default, simple colors are used to identify the feature objects in the image. Additionally, users can select from a number of screen-space texture patterns to be overlaid on the feature object surfaces. These textures can be used as a form of annotation, for example, a pattern consisting of "plus" glyphs can be used to indicate expansion or growth, while "minus" glyphs can indicate decreases in local volume.

Furthermore, we apply screen-space ambient occlusion (SSAO) to simulate global illumination (Ritschel et al. give a good overview of related work in this area [RGS09]). It has been argued [WB08] that correct global illumination improves perception of surface shape and depth in visualizations, and helps the user better understand the 3D positions of feature objects relative to each other. As it is straight-forward to integrate existing SSAO techniques in a deferred shading pipeline, their description falls outside the scope of this chapter.

4.6 Results

We implemented our techniques in C++, using OpenGL and GLSL for all GPU-based algorithms. A user interface was created (see figure 4.4) which, in addition to our visualization of the feature objects, includes several controls to manipulate the parameters of each feature object.

A 2D slice view of the field allows users to specify example points within the field. The example position is updated interactively by simply hovering over the slice viewer, allowing for easy exploration of the field. The values for the field's attributes at the selected point are visualized using a set of colored rectangles, where values are mapped to colors relative to the mean value over the field. In case the field has been reduced in dimensionality (such as described in section 4.4.2), the stored transformation is used to convert values back to the original feature space before visualization. The normalized bias vector resulting from the user's selection is visualized in a similar way. Clicking the individual rectangles in this view allows the user to further manipulate the definition of similarity by selecting the corresponding attributes to be ignored for this feature object's similarity measure.

4.6.1 Performance

As our techniques are aimed at interactive exploration, we measured their performance using several datasets. Using a current generation GPU (NVIDIA GeForce GTX 280), performance is generally suitable for interactive exploration. As our techniques are based on ray casting, performance is strongly dependent on the resolution of the vector field under consideration. Using a dataset of $512 \times 512 \times 80$ (figure 4.6(b)), average frame rates were around 12 frames per second when rendering at 600×600 resolution. Image resolution does not influence performance as strongly, as modern GPUs are highly parallelized. Performance and data sizes can likely be improved by integrating existing optimization and large-data-handling techniques from volume rendering literature.

As discussed in section 4.5, we used depth peeling to enable transparent surfaces to be used. This adds a separate shading pass for each layer in the scene, as well as requiring ray casting to proceed beyond the first surface hit, with both leading to decreased performance. With up to 6 layers of transparent feature object surfaces, performance of the same dataset was around 7 fps, which is still suitable for interactive use. If no transparent surfaces are used, depth peeling is not required.

Our current implementation is limited to data sets that fit in GPU memory. This means both the dataset size in voxels as well as the number of attributes of the (PCA-reduced) multi-fields are limited. However, it would be straightforward to adapt current work on volume rendering for large datasets (e.g., Kniss et al. [KMM+01]) to our approach, in order to enable larger volumes to be explored interactively. Similarly, our implementation of the pre-processing stage currently uses an in-core approach. Processing of the largest dataset used in this chapter took around 20 minutes and up to 7 GB of RAM on a recent machine. However, unless the user needs to refine the selection of attributes, this only needs to be done once per dataset.

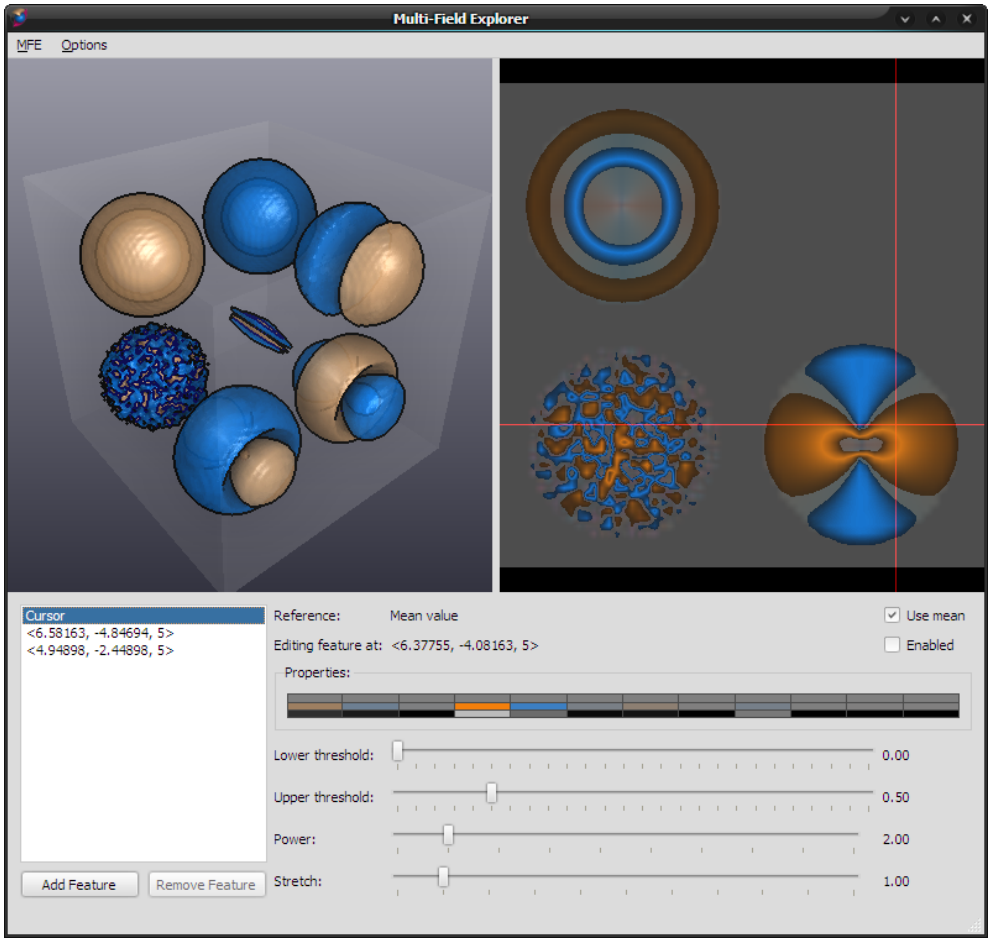


Figure 4.4: The prototype application visualizing a synthetic dataset in real-time ($64 \times 64 \times 64$ points). The interface consists of our visualization (top left), a slice viewer for selecting example points (top right), a list of all current feature objects (bottom left), colored rectangles representing the values in the contrast vector, the values at the current point and the resulting bias vector (middle) and a set of sliders for manipulating the thresholds and bias factor for the current feature object. In this case, two feature objects have been defined with different example points.

4.6.2 Examples

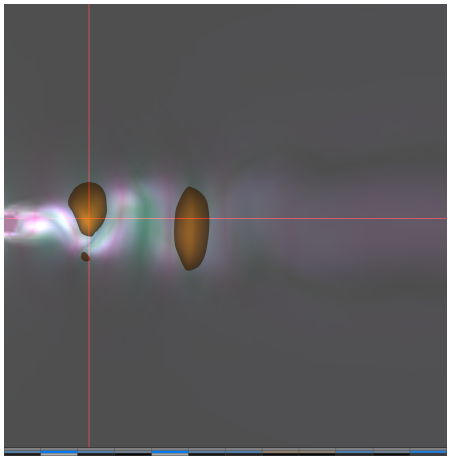
We visualized several example datasets in order to demonstrate our approach. These datasets from different application domains contain well-known features, which are typically extracted and shown using application-specific techniques. This section shows how our techniques can be used to quickly find and visualize such features in an application-independent manner, thereby demonstrating the flexibility of our approach and validating its usefulness as a first step in exploring multi-field datasets.

Our first example is a computational fluid dynamics dataset representing the flow behind a tapered cylinder. The full dataset is time-dependent and can be obtained online from NASA [JL91]. As our techniques do not yet handle time-dependent data, we used a single frame from the dataset. Additionally, we cropped and re-sampled the (originally $64 \times 64 \times 32$ curvilinear) dataset to a regular grid of $128 \times 128 \times 204$ points. We extended this vector field dataset into a multi-field during pre-processing by adding a simple set of derived attributes: In addition to the vector data, we added the normalized vector, the vector magnitude, and several vector calculus derivatives of the field. The derivatives were all computed from the Jacobian of the field: the trace of the Jacobian matrix is the divergence of the field, while its determinant indicates changes in volume. We also included curl, a vector-valued attribute describing local rotation. The Jacobian can be computed at a user-defined scale, which enables filtering of small-scale details such as noise.

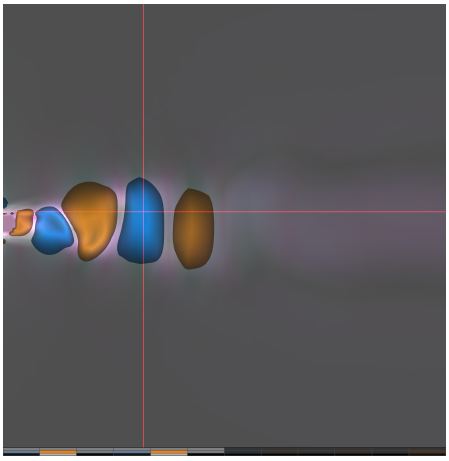
At certain flow speeds, such as the one used in the simulation which created the dataset, a pattern of vortices forms behind the cylinder. These vortices alternate between clockwise and counter-clockwise directions. We used our techniques to illustrate this flow behavior.

By using a single feature object and simply moving the example point in the area behind the cylinder, two opposing patterns can be located. Figure 4.5(a) shows one of these feature objects. The visualization of the example feature vector for this object, shown below the figure, indicates that the directional components of the field (the first six attributes) are highly characteristic of the selected example point. Masking out the non-directional attributes of the field and increasing the bias factor to further emphasize the characteristic attributes reveals more of the pattern. The visualization of the example vector also shows that directions in the second feature object, shown in figure 4.5(b) are opposite to the first.

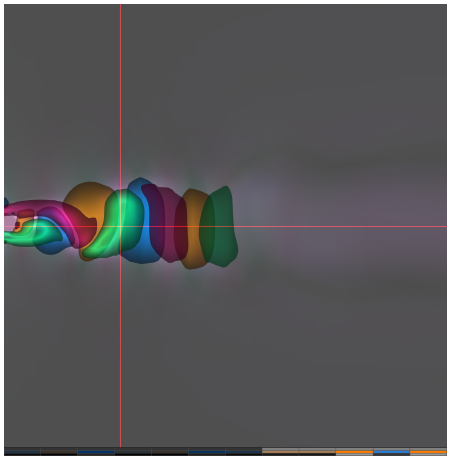
As the yellow and blue areas indicate regions with directions opposite to each other, we can assume the vortices exist between these areas. To highlight these, we use a pair of feature objects with the similarity measure made invariant to direction, and example points set between the yellow and blue feature objects. Figure 4.5(c) shows how these new feature objects (pink and green) capture a similar recurring pattern between the yellow and blue objects. The visualization of the example vector confirms that these are the vortices, as curl (represented by the rightmost three rectangles) is the characteristic attribute for the new objects. Again, values for these attributes are opposite to each other, indicating rotation in opposite directions. Figure 4.5(d) shows



(a) Single feature object



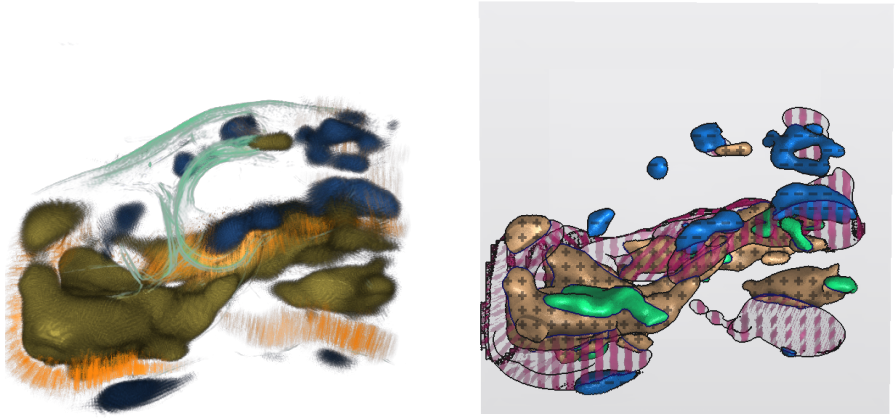
(b) Opposing flow directions



(c) Vortices

(d) 3D visualization

Figure 4.5: Using our techniques to explore the flow behind a tapered cylinder. Slice views show steps in the exploration process. The crosshairs indicate example positions, the colored rectangles below each image visualize (from top to bottom) the contrast and example feature vectors as well as the bias vector for the corresponding feature object.



(a) Direct visualization of deformation [BBP09].

(b) Similarity-based feature objects.

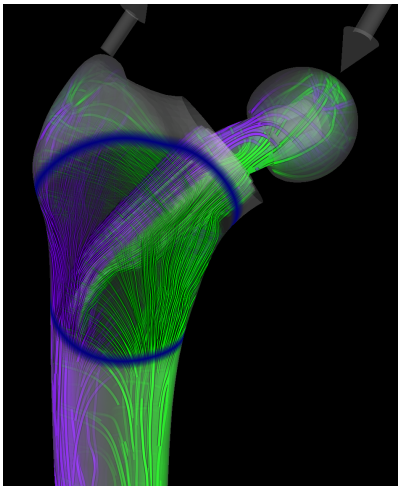
Figure 4.6: Visualizations of a deformation field ($512 \times 512 \times 80$ points) representing changes between two MRI scans of a human knee.

the final visualization of the extracted feature objects, revealing the 3D structure of these vortex patterns. Figure 4.1 shows another visualization of these vortices, with an added feature object representing the area where the flow velocity is altered by the presence of the cylinder.

Figure 4.6 shows visualizations of a vector field resulting from the non-rigid registration of two MRI scans of a human knee. The use of such deformation fields is common in medical image analysis. Busking et al. [BBP09] created direct visualizations of such fields by visualizing a growth measure derived from the Jacobian determinant of the field (figure 4.6(a)). A similarity measure based on a subspace containing only this growth attribute can capture the same feature objects (figure 4.6(b)). Furthermore, by exploring the data using feature objects with different feature-subspaces, areas were found with low magnitude but a significant increase in volume. These were highlighted in green using a feature object based on both magnitude and growth.

Missing from our current implementation is a visualization of relevant context information. However, the techniques could easily be combined with a visualization of contours such as used in the earlier work mentioned above (shown in green in figure 4.6(a)). Context information could also be visualized with our feature objects if the relevant data is included in the multi-field.

Figure 4.7 shows a visualization of a dataset used for hip joint replacement planning. This dataset consists of two tensor fields representing simulated stress and strain resulting from pressure being applied to the implant and femur bone. Dick et al. [DGBW09] recently presented techniques for visualizing such fields interactively



(a) Stress tensor streamlines [DGBW09].

(b) Similarity-based feature objects.

Figure 4.7: Visualizations of a dataset ($85 \times 79 \times 101$ points) consisting of simulated stress and strain tensors in a human femur with a hip joint replacement implant.

with streamlines following the first principal component of the stress tensor, shown in figure 4.7(a). Figure 4.7(b) shows similar structures highlighted using our techniques, applied directly to the tensor field. The visualization clearly shows the opposing tension (yellow) and compression (blue) parts of the implant (shown in purple and green in figure 4.7(a)), as well as the similarly opposing effects on the sides of the bone itself (shown in pink and green).

4.7 Conclusions

Our contribution in this chapter is a new method for the interactive visualization and exploration of multi-fields. Our technique is based on the direct visualization of similarity, defined as distance in a feature space, where the similarity measure is automatically derived from user-specified examples.

While our prototype implementation offered the concepts presented as the *only* option for exploring a dataset, we expect similarity-based techniques for exploration and visualization to be integrated with traditional direct visualization of field attributes. This way, traditional visualizations can guide a user in locating points of interest. Our techniques then automatically highlight similar areas within the dataset, the details of which the user can further explore using different visualization techniques.

Our techniques lead to an intuitive approach to interactive exploration, as little

interaction (only point selection) is required in order to explore a given field. Furthermore, the concepts and techniques presented are generic, and can be applied to the visualization of any type of multi-field.

In this work we have presented a proof of concept for a new idea, supported by several application examples demonstrating the utility of the proposed method. When these concepts have been integrated into a real visualization system for such an application, that would present a good opportunity to perform user-oriented evaluation.

While our techniques can be applied to any type of multi-field, extension to time-varying data is more involved. Feature tracking techniques could be applied to track user-defined features over time. New visual representations should be developed to visualize the evolution over time of any detected feature objects. Stompel et al. [SLM02] presented various illustrative techniques for enhancing the visualization of features in multi-field and time-varying datasets, which could be applied to our feature objects for this purpose.

The techniques proposed could be applied to segmentation of features from a given dataset. Boolean combinations of feature objects, as used by van Walsum et al. [vWP94] and Woodring and Shen [SW06], could be used to specify areas to segment. More advanced filtering techniques could also be added, including the detection of connected components to distinguish between separate parts of the same feature object. Alternative similarity metrics should also be investigated. A variety of metrics in combination with a system for combining feature objects would create a powerful system for segmentation by example.

The definition of examples could also be extended. For example, tracing techniques like streamlines are frequently used to visualize vector fields, while brushing techniques are frequently used to make selections in other example-based systems. Such selections consisting of multiple points could be used as examples in our approach by, e.g., averaging attributes over the set, or by considering similarity of points as compared to the example point closest to the point being evaluated. This way, our approach can be integrated with application-specific techniques for feature selection. For instance, in a medical brain dataset, areas with similar characteristics could be selected around fibers traced from DTI data.

Finally, we plan to integrate our approach with other visualization techniques. In addition to highlighting areas using our current rendering techniques, similarity-based feature objects could also be used to indicate importance or user interest in an importance-based visualization approach.

Acknowledgments

We are grateful to Dr. C. Dick of the Computer Graphics and Visualization group, Technische Universität München, for providing the hip joint replacement datasets. This research is supported by the Netherlands Organization for Scientific Research (NWO), project number 643.100.503 “Multi-Field Medical Visualization”.

CHAPTER 5

Dynamic Multi-View Exploration of Shape Spaces

Abstract

Statistical shape modeling is a widely used technique for the representation and analysis of the shapes and shape variations present in a population. A statistical shape model models the distribution in a high dimensional shape space, where each shape is represented by a single point.

We present a design study on the intuitive exploration and visualization of shape spaces and shape models. Our approach focuses on the dual-space nature of these spaces. The high-dimensional shape space represents the population, whereas object space represents the shape of the 3D object associated with a point in shape space.

A 3D object view provides local details for a single shape. The high dimensional points in shape space are visualized using a 2D scatter plot projection, the axes of which can be manipulated interactively. This results in a dynamic scatter plot, with the further extension that each point is visualized as a small version of the object shape that it represents. We further enhance the population-object duality with a new type of view aimed at shape comparison. This new "shape evolution view" visualizes shape variability along a single trajectory in shape space, and serves as a link between the two spaces described above.

Our three-view exploration concept places a strong emphasis on linked interaction between all spaces. Moving the cursor over the scatter plot or evolution views, shapes are dynamically interpolated and shown in the object view. Conversely, camera manipulation in the object view affects the object visualizations in the other views. We present a GPU-accelerated implementation, and show the effectiveness of the three-

view approach using a number of real-world cases. In these, we demonstrate how this multi-view approach can be used to visually explore important aspects of a statistical shape model, including specificity, compactness and reconstruction error.

5.1 Introduction

Shape spaces are continuous higher dimensional domains where each position represents a complete 2 or 3-dimensional object surface [Ken84]. Statistical shape models (SSMs) represent whole populations of objects by modeling them as distributions in shape space. The most well-known application of this type of modeling is the Active Shape Model (ASM) [CTCG95]. An ASM models the distribution in shape space with its principal components, and is used to locate similar shapes in volume datasets.

Shape spaces and statistical shape models are an important concept in image segmentation, object classification and recognition, the study of anatomical variation and many other areas where the understanding or modeling of the variability in a whole population of shapes is required. Shape models can be used to represent existing (input) shapes, or they can be used to synthesize new shapes similar to the input shapes.

Currently, visualization of shape spaces is done using straight-forward methods, such as showing object shapes regularly sampled on an axis in shape space, or showing a scatter plot of the input shapes over the first two principal axes. A more comprehensive visualization approach would contribute significantly to the understanding of shape variability and of the behavior and quality of statistical shape models. The need for better visualization is further accentuated by cognitive demands made by the high-dimensional nature of shape space. In spite of these observations, there are currently no examples in literature of visualization applications that focus on comprehensive shape space exploration.

In this chapter, we present just such a visualization application that enables the visual exploration of shape spaces and shape models. For such a tool, we identify the following requirements, which are further detailed in section 5.3:

1. The visualizations should provide insight into the high-dimensional structure of the population in shape space.

2. The tool should visualize the statistical shape model and its relation to the population. The fidelity and characteristics of the shape model should be visually verifiable.

3. Information should be presented at both the global (population) and the local (object) level whenever possible.

In the following, we first discuss work related to the visualization of shape models and high-dimensional spaces. We then discuss these requirements in detail and present our contributions, which are:

- A strongly-linked multi-view approach to visual shape model exploration and validation (section 5.4).
- The shape evolution view: a new type of view in-between shape space and object space allowing direct visual comparison and visualization of a single shape-space direction or trajectory (section 5.4.3).
- A GPU-accelerated implementation of natural neighbor interpolation, allowing real-time continuous exploration of the population in shape space by smoothly interpolating between individuals (section 5.5.2).

A prototype implementation of our multi-view approach was created as described in section 5.5. Using this implementation, we demonstrate the effectiveness of our approach in a number of real-world cases. Finally, we present conclusions and directions for future work.

5.2 Related work

Our work focuses on the visualization of the variation in a population of shapes by means of shape spaces, where a shape space is defined as the continuous higher dimensional space in which each input shape is represented by a single point [Ken84]. The most well-known use of this type of shape space is probably the Active Shape Model, an SSM used for segmentation, where the main modes of variation are found using principal component analysis [CTCG95].

Shape models have been extensively used for image segmentation both in medicine and other applications. These include studying anatomical shape differences between populations [GGSK05, FPO+06], visualizing organ shape variation in 3D anatomical atlases [HH06] and studying evolutionary morphological changes [WAA+05].

In all applications, direct visualization of shape variability coupled with the flexible visual exploration of shape space would contribute to a better understanding of the shape variation, in both the local and global sense, and hence the characteristics of the shape model that is being used. However, examples in literature of comprehensive visualization approaches for shape spaces are scarce. General techniques for high-dimensional visualization can be used to visualize a population of objects in shape space. An overview of techniques for high-dimensional visualization is given by Wong and Bergeron [WB97], while a more recent overview of multivariate techniques is given by Fuchs and Hauser [FH09]. However, such techniques do not take into account the fact that each high dimensional point also represents a two or three-dimensional shape, the local details of which are often important.

Shape variation can be shown at a local level with color mapping, or by placing ellipsoids at all points on the average or mean shape, representing the Gaussian variation of each point in shape space [FPO+06, FAP+07]. The distribution in an SSM shape space can be shown, for the first two modes at least, with a simple scatter

plot, whereas the shape variation over a particular mode can be shown with a number of shapes, regularly sampled over that mode, shown side-by-side [CTCG95], or by animating object shape changes over the main modes of variation. Lamecker et al. extended this basic approach by also animating between the mean shape and the training shapes, although little detail is given on exactly how this is done [LSL+04]. A good overview of other mesh morphing techniques, although not in the context of shape spaces, is given by Alexa [Ale02].

Kilian et al. used shape spaces to interpolate between and extrapolate from input shapes [KMP07]. Their method can also be used to explore the shape space spanned by a number of input models by allowing interaction, primarily curve-drawing, on a specially constructed 2D polygon with the input models at its vertices. Our approach differs from that of Kilian et al. in three ways: 1: We project all shape space points on a 2D plane that can be smoothly and arbitrarily positioned in shape space, inspired by the method of Blaas et al. [BBP07]. 2: Our method supports real-time and smooth visualization of the shape changes represented by interaction on the shape space projection, without any pre-processing. 3: We make use of a coordinated multiple view approach to integrate three different visualization techniques for exploring the shape space.

After the publication of this work, Hermann et al. presented new interactive approaches for exploring shape spaces. By interactively deriving models of subspaces rather than computing the full shape model up front, the user is less influenced by non-linearities in the actual shape space [HSK11]. They have also recently presented an approach using per-point covariance tensors to guide exploration of the full shape using interactive local deformations [HSSK14].

5.3 Assumptions and Requirements

Creation of a shape model involves first establishing a point correspondence relation between a population of input shapes. Next, these shapes are parameterized and a statistical model is fitted to the resulting distribution in a high-dimensional shape space. Heimann and Meinzer [HM09] give a good overview of the various techniques that exist for the creation of such shape models. For this work, we assume a shape model has already been created. In particular, we use active shape models based on landmarks obtained using the GAMEs algorithm [FOP+07] to demonstrate our approach. However, our techniques can be applied to any shape model that satisfies the following assumptions:

- Each shape can be represented as a point in an N-dimensional shape space. Shapes in an active shape model use a fixed set of landmarks; shapes in shape space are defined as the vector of coordinates for all landmarks.

- Shape space points can be interpolated to form new shapes. We assume linear

interpolation in shape space is valid. If non-linear spaces are used, the interpolator used should take this non-linearity into account.

- The distribution of points in shape space is modeled using a statistical distribution. As in ASMs, we use PCA to derive the parameters of a multivariate Gaussian distribution. Other statistical distributions could be used, provided that their contours can be plotted in arbitrary linear projections of the shape space.

In the following, we examine the three requirements defined in the introduction in detail, based on these assumptions:

1. Provide insight into the high-dimensional structure of the population in shape space. Statistical shape modeling is based on the assumption that the population being modeled follows a certain distribution in shape space, such as the multivariate Gaussian distribution in our models. Visualization of this high dimensional space can help to visually verify such an assumption. Furthermore, it can allow a user to spot clusters or outliers within the population. Dimensional reduction techniques such as PCA can help in selecting appropriate projections for visualizing this space.

2. Visualize the shape model and its relation to the population. As noted, SSMs model a statistical distribution in shape space. This means shapes can be described in terms of their likelihood with respect to the model. Additionally, a number of important measures can be computed describing the performance of a given shape model with respect to modeling a given population [DTA+03, FPO+06]:

Compactness describes the accuracy with which a given number of model parameters can be used to model a population. The *reconstruction error* describes the mismatch between the training shapes and their approximations given by the model. The *generalization error* is similar to the reconstruction error, but approximates the error reconstructing unseen shapes in the population. Finally, *specificity* describes the difference between the variation in the population and that in the shape space generated by the model.

These measures were defined originally to yield a single number per shape model [DTA+03]. We have adapted the definitions to show the local behavior of the measures on a per-shape or even per-point basis, in order to explain the global result for a given shape model.

3. Present information at both a global and a local level. Shape spaces have a dual nature in that they can be seen both as a high-dimensional space, where shapes are single points, and as a 3D space, in which the 3D objects represented by such points exist. Similarly, various aspects of a shape can be visualized both on a global (population) or local (object) level. Examples are the deviation from the mean shape or the reconstruction error, both of which can be computed per object or per point on the object.

5.4 Multi-view exploration

We propose a three-view approach to satisfy these requirements (figure 5.1). Overview and global-level visualization are provided by a visualization of the high-dimensional shape space. Local details are provided by a 3D object view, showing a single shape.

Variability, however, is hard to visualize on a local scale. For this purpose we introduce a new type of view: the *shape evolution view*. This view shows a combination of object space and shape space dimensions. Object shapes are reduced to two dimensions by extracting their silhouettes, and the third dimension is used to represent a trajectory through shape space. Essentially this creates a visualization of "2D + variability" space.

In this chapter we use "global" to mean "per object". For information at the shape model or population level, it would be straightforward to aggregate performance measures or other statistics over all individuals (each resulting in a single value) and display these, as is the current approach in the field. For example, model-level compactness can be visualized in a standard compactness plot as used by Davies et al. [DTA+03]. The aim of our visualizations is not to replace such statistical methods. Rather, we provide tools that allow results to be examined at all levels, allowing deeper insight into the distribution-related and shape-related details of such results.

The visualizations used in these three views are discussed in the following sections. Section 5.4.4 discusses linking between the views, which enables interactive exploration of the shape space.

5.4.1 Shape space view

The goal of the shape space view is to give an overview of the high dimensional shape space. Within this view, the user can explore the structure of the population, the statistical distribution modeled by the shape model and global-level information about both.

We visualize shape space using a scatter plot, as shown in figure 5.2. The scatter plot presents a 2D projection of the high-dimensional points making up the population. We base this projection on the technique presented by Blaas et al. [BBP07], in which the plane of the projection can be defined arbitrarily and interactively. To do this, the user manipulates 2D representations of any number of axes or vectors in the shape space. From these, a 2D coordinate frame is created consisting of linear combinations of these vectors.

Specifically, for a point defined by shape vector s, the 2D projection $P(s)$ is given by $P(s) = AFs$, where F is a projection reducing the high-dimensional shape space to the subspace spanned by the current set of axes, and A is the $2 \times N$ matrix with the user-defined 2D vectors for each axis as its columns. Manipulation of an axis corresponds to rotation (and/or scaling) within the reduced subspace. To enhance this perceptually, we change the drawing order of shapes during manipulation to match the order of their projections along the axis being manipulated.

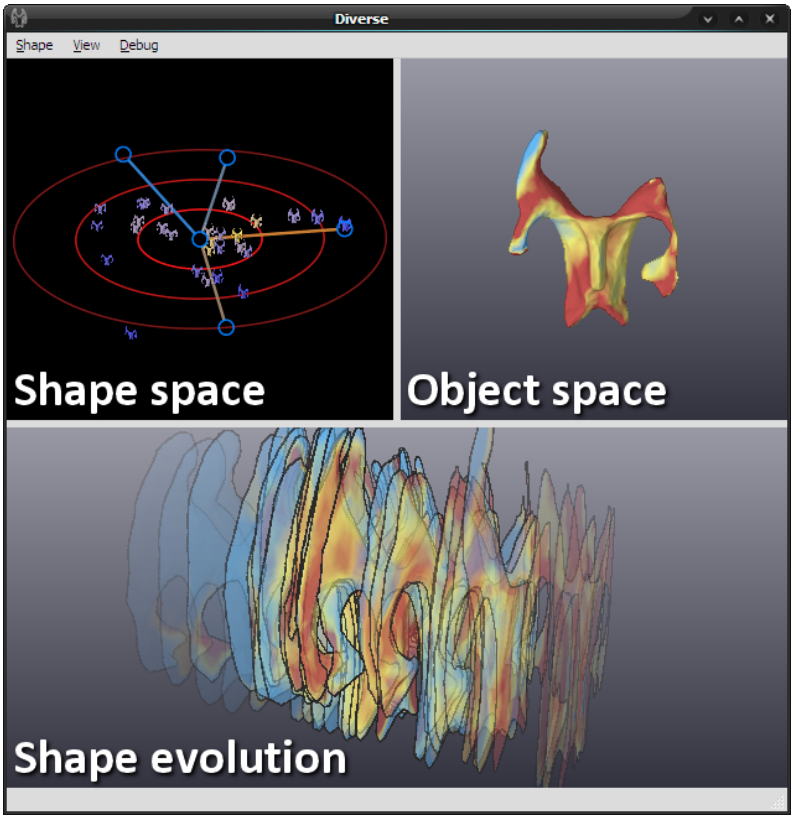


Figure 5.1: Prototype implementation of our multi-view exploration approach, showing the shape space, object space and evolution views. Due to screen space constraints, the latter can be switched between the configurations shown in figure 5.4. Multiple evolution views can be added if preferred.

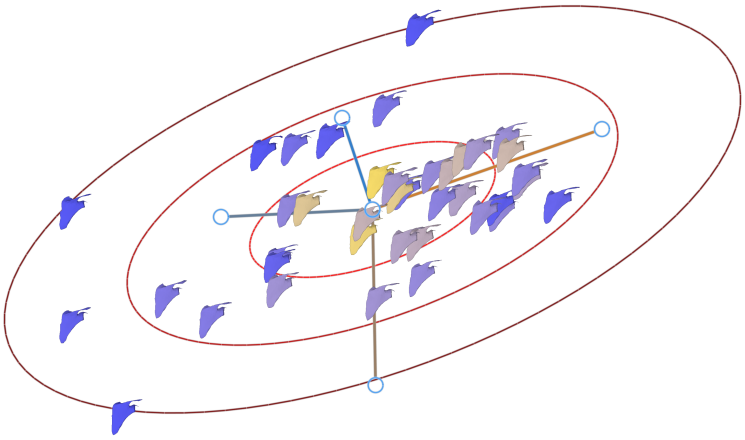


Figure 5.2: The shape space scatter plot view, visualizing a population of segmentations of the human scapula. Colors indicate the likelihood of each shape with respect to the Gaussian model, which itself is visualized by ellipses representing multiples of the standard deviation.

Central to our approach is that the projections of points are updated in real-time during interaction. This way, dependencies between points in shape space and the axis being manipulated can be learned intuitively by observing the speed and direction of their movement relative to the axis. A user therefore often does not need to examine all possible projections; manipulation of each axis in turn suffices to locate features like outliers, and projections that highlight such features can be created incrementally.

To further aid the user in locating useful projections, the axes of the scatter plot can be toggled between dimensions from the original shape space, or from the reduced space created by the shape model. As our shape models use PCA, the resulting reduced set of axes is guaranteed to create a projection showing maximum variability. Alternative techniques for locating good viewpoints, such as the Grand Tour [Asi85] and Projection Pursuit [FT74] methods, could easily be integrated in our approach.

In addition to the points in the population, the projection of the shape model's Gaussian distribution is visualized using a set of elliptical contours representing multiples of the distribution's standard deviation. This projection can be computed by projecting the covariance matrix for the population to the scatter plot frame: $P(C) = AFCF^TA^T$. Principal component analysis is then performed using the projected covariance matrix to extract the axes for the ellipses and the 2D standard deviation of the model.

We enhance the basic scatter plot visualization by rendering each high dimensional shape space point as a small representation of the 3D object represented by that point. These small multiples [Tuf90] can be color coded to visualize additional scalar information, such as the likelihood of the shape with respect to the shape model or the shape model performance measures described in section 5.3.

5.4.2 Object space view

The object space view visualizes the 3D object corresponding to a single, arbitrary point in shape space. The purpose of this view is to give insight into local details of the shape and/or measures (see figure 5.3).

Deformation with respect to the mean, for instance, can be visualized for each point on the object by computing the length of the 3D deformation vector (a subvector of the shape vector) corresponding to that point. Similarly, we can compute the local distance of each point to the corresponding point on any reference shape as the length of the difference between their deformation vectors. A special instance of this is the local reconstruction error, where the reference shape is defined as the shape reconstructed using a limited number of shape model parameters, computed by projecting the shape to a basis of eigenvectors and then back to the original shape space. For visualization purposes, distances are made signed with respect to the angle between the deformation vector and the local surface normal, and visualized using a isoluminant and perceptually linear color map, as suggested by Borland and Taylor [BT07].

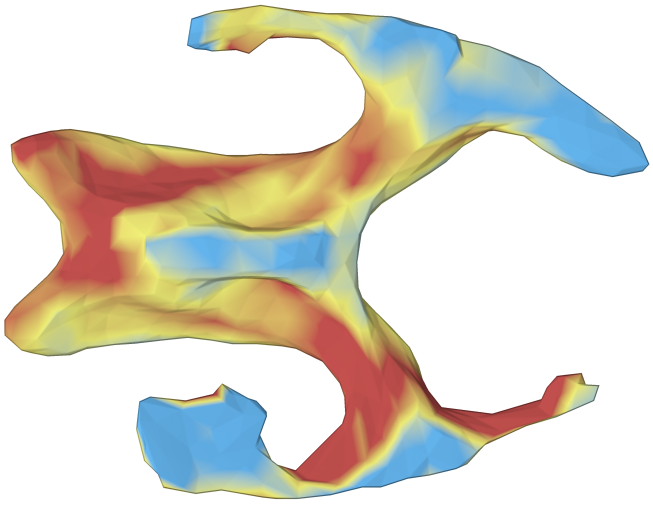


Figure 5.3: The object space 3D view, here showing per-point reconstruction errors for a brain ventricle shape using a color map. Blue indicates areas where the actual shape is larger than the reconstructed surface, yellow indicates similar size and red indicates areas where it is smaller.

In this view, the camera can be manipulated interactively. Alternatively, our system can automatically generate an interesting viewpoint by performing principal component analysis on the 3D deformation vectors. By choosing a viewing direction orthogonal to the plane spanned by the first two principal components, a view can be created that shows maximal deformation in the image plane.

5.4.3 Shape evolution view

The shape space view provides global information, while the object view shows local details for a single shape. Using only these two views, it is hard to gain insight into the variability of such local details in the population. We introduce the *shape evolution view* as a way to solve this limitation.

The evolution view visualizes local shape variability along a single direction or trajectory through shape space. This way, the high-dimensional space is essentially reduced to a single dimension, leaving two dimensions for visualizing object shapes along the trajectory. Our current implementation supports only straight lines for interpolation between shapes, but extension to other curves, for instance user-defined strokes or shape-space geodesics (as used by Kilian et al. [KMP07]), is relatively straightforward.

The goal of the evolution view is to provide a detail level for trends in shape space or comparisons between sets of shapes. To enable these two applications, we define two types of visualization of shape space trajectories:

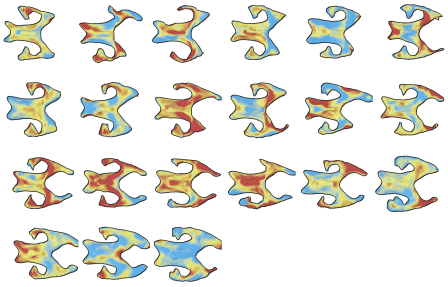
- *Trend trajectories* - trends can be explored by visualizing the population itself, as seen from a user-defined trajectory through shape space. This means we visualize the actual individuals, with positions along the trajectory determined by their projections.

- *Comparison trajectories* - direct comparisons between shapes are possible by generating (synthetic) objects interpolating between their shapes. This is essentially the direct visualization of a line or curve through the high-dimensional shape space connecting the points being compared.

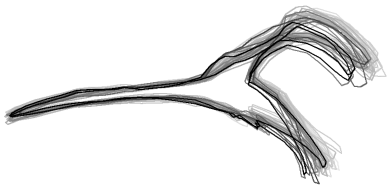
The combined "2D + variability" space of the evolution view can be shown in a number of visualization configurations, shown in figure 5.4. Each of these serves a different purpose in analyzing local variability:

- *Side-by-side* - shapes along the trajectory are visualized using a grid of small multiples, in order of appearance along the trajectory. This configuration can be used to get an overview of the population.

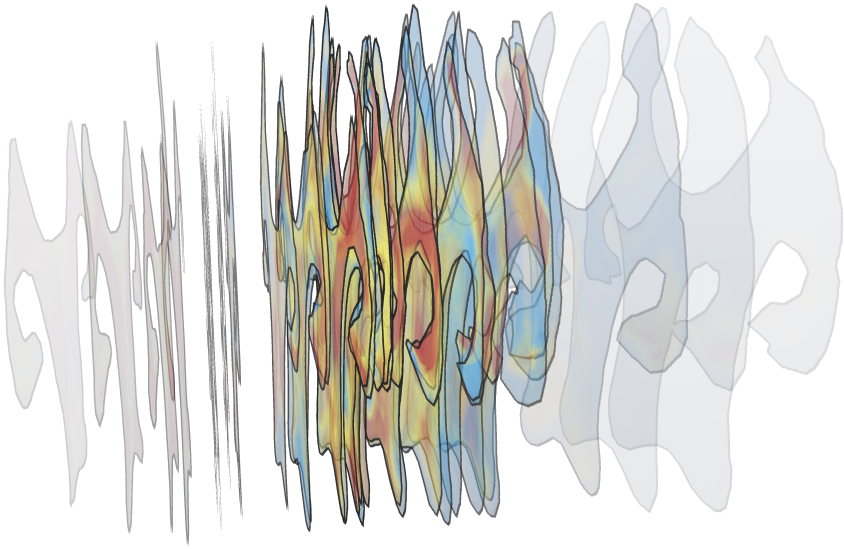
- *Overlaid contours* - by directly overlaying all shapes and only visualizing their projected contours, we create a 2D visualization similar to "onion-skinning" techniques used in animation. Contours are color coded using a perceptually



(a) Side-by-side



(b) Overlaid contours



(c) Shape stack

Figure 5.4: The shape evolution view and its configurations, used to examine variation in the scapula and brain ventricle datasets. (a) Side-by-side display of a subset of the brain ventricle population, ordered by projection along the first principal component. (b) Overlaid contours for a side view of the scapula population, demonstrating that most variation occurs in the acromion and coracoid process. Ordering along the first principal component is visualized using a black-to-white scale. (c) Shape stack visualization of the shapes in (a), showing an increasing trend in the overall size of the shapes along the first eigenvector.

linear black-to-white color map to show their ordering with respect to the trajectory. The resulting visualization allows intuitive comparison of object shapes along the trajectory. Due to the sparseness of contours it is not possible to visualize other measures in this configuration.

- *Shape stack* - the most direct interpretation of 2D + variability is the 3D structure created by stacking all object contours along the third dimension. Offsets between shape "slices" are determined by their positions along the trajectory, thereby visualizing the distribution of shapes. Optionally, interpolation can be used to create a continuous shape, representing a continuous morph between all shapes along the trajectory. This configuration provides a level in-between the other two, in that local details can be compared between shapes without losing information about the relation of those shapes to the population.

5.4.4 Linked interaction

A key part of our multi-view exploration approach is strongly linked interaction between all views (figure 5.1). Such linkage helps to maintain overview and provides continuous feedback about the relations between the different visualizations displayed in each view. Linked interaction is used in four scenarios:

- *Shape selection* - the object view displays only a single shape. By simply hovering the mouse over the shape space view, the object view smoothly deforms between shapes encountered along the path of the cursor, allowing intuitive exploration of the shape variability in the population. This is achieved by using a GPU-based interpolation technique, detailed in section 5.5.2. Alternatively, shapes can be selected in the evolution view in order to examine them in detail. When a shape is selected in either view, the projected position of the shape is highlighted in the shape space scatter plot, and the selected shape is integrated and/or highlighted in the evolution view visualization.

- *Trajectory selection* - in the current implementation, the trend trajectory shown in the evolution view is linked to the x-axis of the scatter plot. It can therefore be defined by manipulating the projection as described in section 5.4.1. Alternatively, a shape comparison trajectory can be defined by selecting two shapes using the selection methods mentioned above. As mentioned in section 5.4.3, sketch-based curve selection can be added in a future extension.

- *Object space camera manipulation* - viewing parameters for the object shape visualizations in all views are linked. The camera for these projections can be manipulated in the object space, updating the other views in real-time. This also affects the viewpoint used for contour extraction in the overlay and shape stack configurations of the evolution view.

- *Dimensionality selection* - the number of axes shown in the shape space view directly determines the number of dimensions used for determining shape likelihood and reconstruction errors. The number of axes can be adjusted interactively by the user, with visualizations in all views updating in real-time.

5.5 Technical details

We implemented a prototype of our multi-view exploration approach (figure 5.1) using C++ and OpenGL. The OpenGL Shading Language was used to implement all GPU-based algorithms and the IT++ math library (`http://itpp.sf.net/`) was used for CPU-side linear algebra operations.

5.5.1 Shape visualization

For the purposes of statistical analysis, populations in shape space are often centered on the mean. Therefore, object shapes in a statistical shape model are defined as the sum of the mean shape and a high-dimensional shape vector. In case dimensional reduction has been applied, a shape vector first has to be transformed back to the original space in order to visualize its corresponding object.

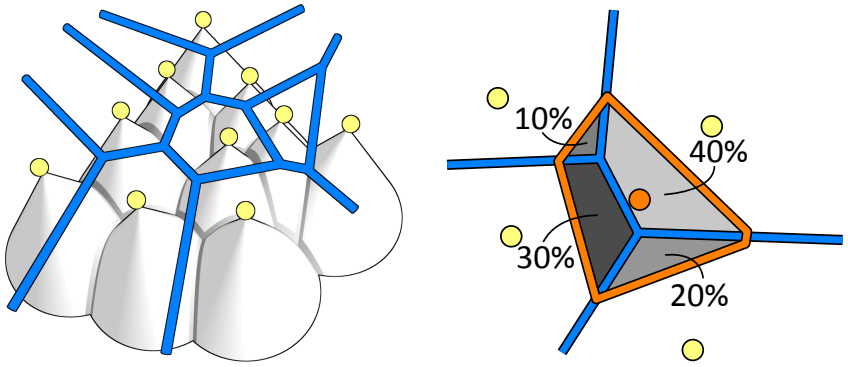
We visualize shapes in object space using a polygonal model of the mean shape, which is deformed on the GPU to match any given shape. Deforming on the GPU allows us to simultaneously compute derived measures to be visualized in object space, such as local reconstruction errors.

The surface normals of the mean shape are no longer valid after deformation. We use a deferred shading pass to compute normals in screen space. When visualizing a large number of shapes, individual objects may be hard to distinguish in the scatter plot and evolution views due to overlap. To alleviate this, the same deferred shading approach used to render the objects can be used to add contours or halos to each of the object miniatures.

5.5.2 Shape interpolation

As described in section 5.4.4, we allow a user to select and morph between shapes in the shape space view using interpolation. The interpolation technique should enable interactive performance with minimal pre-computation, as both querying for shapes and updating the (projected) positions of all original points should be real-time operations. To achieve this, we created a GPU-accelerated implementation of natural neighbor interpolation [Sib81]. Natural neighbor interpolation was chosen because it results in smooth interpolation, and can provide results even outside the convex hull of the projected points.

Our approach is image-based rather than geometry-based, as we do not require the potentially higher accuracy of the latter approach and can therefore avoid its



(a) Rendering the Voronoi tessellation of a set of points (shown in yellow) using cones. When viewed from the top, standard depth-buffering ensures the pixels covered by a cone belong to the Voronoi area for the corresponding point.

(b) Weights for natural neighbor interpolation are computed as the ratio between the area of the Voronoi cell corresponding to a given point if it were inserted into the tessellation (orange), and the overlap between this cell and each of the existing cells in the diagram (gray).

Figure 5.5: Image-based approximation of natural neighbor interpolation

higher computational complexity. Our approach differs from that presented by Fan et al. [FEK+05] in that we make use of hardware occlusion queries to avoid slow reads from graphics memory.

Natural neighbor interpolation is based on the Voronoi tessellation of a given set of points. An image of the Voronoi tessellation can be constructed by rendering each point as a cone seen from the top, centered at the point's projected position. This way, the depth values for the cone's pixels correspond to the distance between that pixel and the original point. By making use of a standard depth buffer, parts of the cone which are closer to a different point will be removed, as their pixels will be covered by the cone for the other point (see figure 5.5(a)). We generate the cone depth values in a full-screen fragment shader pass in order to avoid discretization artifacts that occur when using polygonal approximations.

During rendering of the Voronoi tessellation we use the stencil buffer to record an identifier for each pixel corresponding to the visible cone (i.e. closest point) for that pixel.

In natural neighbor interpolation, we consider insertion of a new point into the Voronoi tessellation. The interpolated value is a weighted summation of the values v_i at the original points p_i. These weights are computed corresponding to the ratio between the Voronoi area $A'(x)$ created by the new point x and the overlap between this area and the areas for the original points in the original tessellation $A(p_i)$ (see figure 5.5(b)):

$$v(x) = \sum_{i=1}^{N} \frac{A'(x) \cap A(p_i)}{A'(x)} v_i$$

We use occlusion queries as a fast way to determine these areas. An occlusion query is a fast hardware-accelerated method to query the number of pixels (potentially) modified by a drawing operation. The total area for the new point can be approximated by simply inserting the corresponding cone into the Voronoi tessellation image. The overlap with existing areas can be computed by repeating the process, but using the stencil buffer to restrict drawing to the area covered by each existing cone. After running all queries, results are collected, weights are computed and the interpolation is performed on the shape vectors in the population.

5.6 Results and validation

In this section, we demonstrate how our approach enables exploration of shape space using two real-world datasets. The first consists of two sets of segmentations of the ventricles in the human brain. One set consists of 28 control individuals, the other of 58 patients with Alzheimer's disease. For each individual, positions are given for 949 points on the ventricle surface. The second dataset consists of 39 segmentations of the human scapula, with 379 points per segmentation surface.

We will use these as example use cases to demonstrate how our approach satisfies each of the requirements defined in section 5.3.

1. Provide insight into the high-dimensional structure of the population in shape space. The interactive scatter plot projection of the shape space view can be used to explore the distribution of the population in the high dimensional shape space. By choosing an appropriate projection, clusters, trends and outliers can be identified. Figure 5.6 shows an example of an outlier, while figure 5.2 shows a skewed distribution of shapes that may not satisfy the Gaussian distribution assumption.

2. Visualize the statistical shape model and its relation to the population. For any population, the shape space view displays the projected Gaussian distribution of the shape model, allowing visual inspection of the model's statistical properties. Shapes in the population with low likelihood can be found by changing the projection, and are highlighted in all projections when coloring shapes by their likelihood (see figure 5.6).

Alternatively, shapes can be colored by *reconstruction error*. As the number of shape model parameters is directly linked to the number of axes in the shape space view, varying this number while visualizing reconstruction errors allows for visual inspection of the model's *compactness*, and helps in determining a good cut-off for the shape model's dimensionality (see figure 5.7).

Specificity is defined as the average distance of random shapes to the nearest shape in the population [DTA+03]. The nearest shape can be determined during

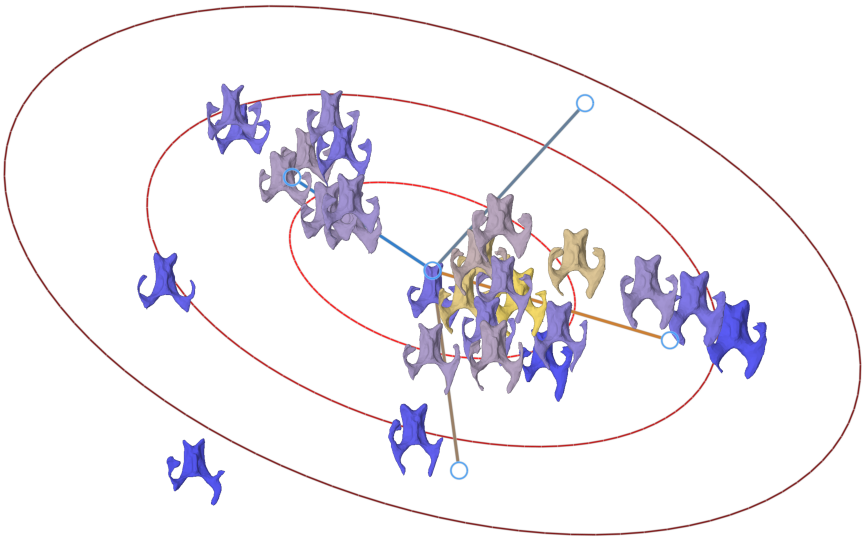


Figure 5.6: Outliers in the population can be highlighted in the shape space view, both by selecting an appropriate projection and by coloring shapes according to their likelihood. High-valued shapes are yellow, low values are blue.

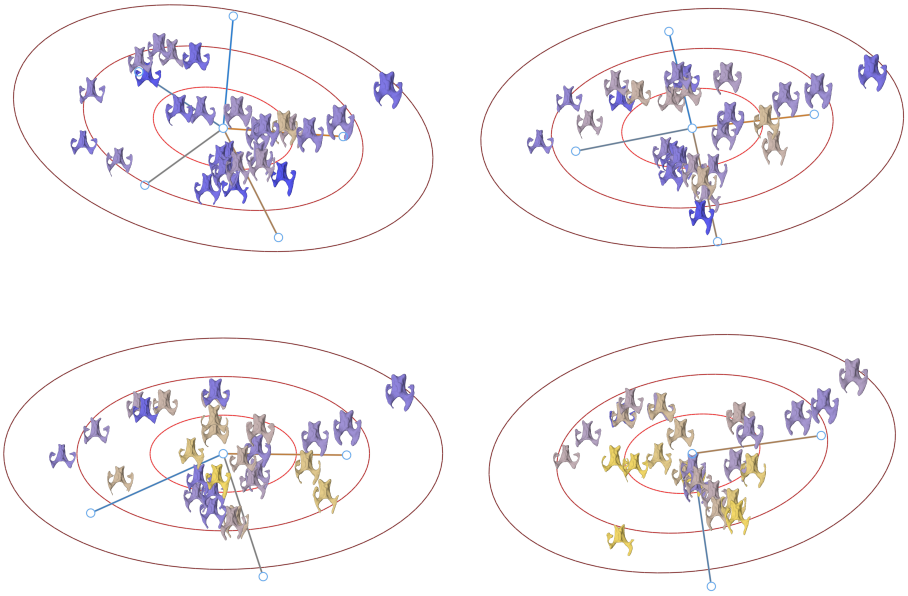


Figure 5.7: Inspecting compactness of the shape model by varying the number of axes. Shapes are colored by their reconstruction error, based on a reconstruction using the axes shown. When reducing the number of parameters of the model from 5 (top left) to 2 (bottom right), the number of shapes with high reconstruction errors (yellow) visibly increases.

interaction in the shape space view, and can be used as the reference shape. This way, the dissimilarity of arbitrary shapes to the population can be visualized on a local level.

For computing the *generalization error*, Ferrarini et al. [FPO+06] use a leave-one-out approach, where the generalization error for any object is essentially the reconstruction error of the object using a shape model based on the population with that object removed. Such models could be created in a pre-processing step, allowing the generalization error to be visualized in the shape space view in a way similar to the reconstruction error, but using a different model for each object in the computation.

3. Present information at both a global and a local level. The strong linking between all views in our exploration concept enables findings in the shape space view to be easily explored at a local level. By simply hovering the cursor over clusters or along trends in the shape space view, the local details of these features can be explored interactively in the object view. If a problematic shape is identified, such as

the outlier in figure 5.6, the local views can be used to determine which parts of the object cause it to be an outlier. Similarly, the reconstruction error can be examined on a per-point basis, as seen in figure 5.3.

The evolution view allows trends to be explored at a local level (see figure 5.4). The three configurations serve different purposes. The side-by-side configuration is suitable for giving a quick overview of the entire population. Overlaid contours give insight into the local variability in the shape of parts of the object. Shape stacks give simultaneous insight into such trends and into the distribution of shapes along a user-defined direction in shape space.

The evolution view can also be used to determine if a certain local error occurs more than once in the population, or if, for instance, high reconstruction errors have various causes when examined locally. In order to facilitate exploration of such trends in local details, we allow the user to define a threshold on the reconstruction error. Parts of the object with values above this threshold will be highlighted in the visualization. While contours remain, the opacity of other parts of the objects is reduced to increase visibility of the areas with large errors.

An example of this can be seen in figure 5.8. As in figure 5.3, red indicates parts of the actual individuals that are larger than the reconstructed shapes while blue indicates smaller parts. Although many errors are unique to specific objects, several shapes show a red region in the area around the glenoid cup as well as red and blue areas around the acromion, indicating that the current shape model has trouble modeling these parts of the scapula.

5.7 Conclusions and future work

We presented a new method for visual shape space exploration and validation, based on multiple strongly-linked views showing both global and local aspects simultaneously. Additionally, we make the following technical contributions:

- The shape evolution view: a new type of view visualizing "2D + variability" space, a combination of shape space and object space. This view allows for direct visual comparison of local details over a single shape-space direction or trajectory.
- A GPU-accelerated implementation of natural neighbor interpolation, allowing real-time continuous exploration of the population in shape space by smoothly interpolating between individuals (section 5.5.2).

The continuity provided by interpolation helps in initial exploration of shape changes and trends, understanding of which is further aided by visualizations in the evolution view. Continuity may also benefit other aspects of the visualizations, for instance, animations may help maintain overview when switching between evolution view configurations or scatter plot projections.

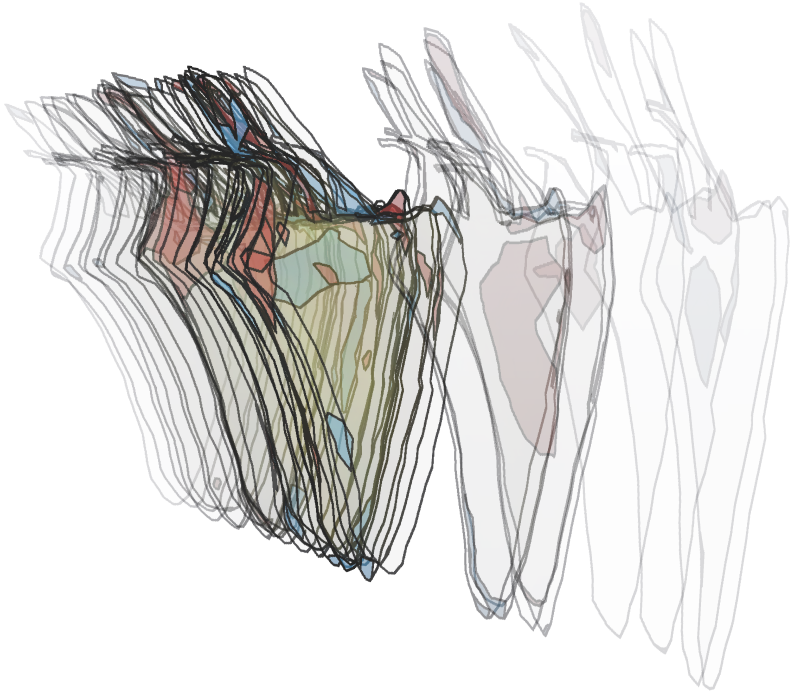


Figure 5.8: The evolution view can be used to explore trends on a local level. Here, reconstruction errors for the scapula dataset are explored on a local level and compared over the population by showing a shape stack highlighting errors above a user-defined threshold.

Although not considered for this work, our shape exploration approach could further benefit from the inclusion of filtering techniques. For example, in addition to highlighting single shapes, it would be helpful to allow subsets of shapes to be extracted and explored. Selections could be made manually, or based on clustering techniques applied to the high-dimensional population.

The complete implementation of our shape space exploration prototype is to be released as open source.

5.7.1 Future extensions

Although the shape modeling approach used in this chapter is the most common in shape modeling literature, alternative techniques have been proposed. Our framework should be extended to allow for the visual comparison of shape models using different parameterizations, landmark sets or statistical distributions. In particular, non-linear shape modeling techniques have been presented in recent literature, allowing for more flexibility in modeling non-Gaussian distributions of shape in a population [CKS01, COS06]. Most of these non-linear techniques consist of first non-linearly mapping shapes to a different space, in which normal PCA or other density estimators can be applied. This means we can either use our current techniques on the space resulting from the mapping. Alternatively, contours of the non-linear distributions can be shown in the original shape space, similar to the figures in the papers by Cremers et al., but using our interactive techniques to select the projection.

Different dimensional reduction techniques such as independent component analysis could also be integrated, to aid in selecting insightful projections of the high-dimensional shape space. Additionally, automated approaches could be integrated, such as the Grand Tour [Asi85] and Projection Pursuit [FT74] algorithms, or even intelligent methods aimed at locating specific patterns.

The current techniques for shape space interpolation remain close to the original points when extrapolating. This could be extended with more accurate extrapolation techniques to allow a user to magnify trends in the population by displaying shapes with "exaggerated" deformations. Boback et al. [BFHU09] enhanced extrapolation in the natural neighbor algorithm by dynamically adding extra "ghost points" to the population. Such a technique would be straightforward to implement on our GPU accelerated implementation, as the basic interpolation scheme remains identical. Alternatively, geodesic-based interpolation and extrapolation techniques could be used, such as those presented by Kilian et al. [KMP07].

Finally, we plan to apply our techniques to interactive shape-model-based segmentation by integrating surface fitting techniques. This way, possible (locally optimal) locations for a new object in shape space can be explored interactively, with both global and local feedback on, for instance, the reconstruction error or likelihood of a given reconstruction of the surface.

Acknowledgments

We are grateful to Dr. L. Ferrarini and Dr. J. Milles of the Department of Radiology, Leiden University Medical Center, for providing the brain ventricle shape models and population data, and to the Department of Orthopaedics, Leiden University Medical Center (Ir. E. van IJsseldijk, Dr. B.L. Kaptein, Dr. E.R. Valstar, Prof. P.M. Rozing) for providing the scapula dataset. This research is supported by the Netherlands Organization for Scientific Research (NWO), project number 643.100.503 "Multi-Field Medical Visualization".

CHAPTER 6

NQVTK

Abstract

This chapter introduces NQVTK, a software framework that enables users to quickly create GPU-based implementations of complicated rendering pipelines for visualization using OpenGL. It integrates with VTK for loading and processing mesh and image data, but provides a separate rendering framework with easy access to cutting-edge GPU functionality without the typical VTK overhead.

The second half of this chapter describes our layered rendering and ray casting rendering pipelines, which have been used as the basis for most of the visualizations in this thesis. The layered ray casting approach is a GPU-accelerated combination of a ray casting implementation and surface depth peeling, which allows for seamless combinations of implicit and explicit surfaces and volumetric data in a single scene, with support for separate deferred shading passes for each depth layer.

6.1 Introduction

VTK [SML04] is an extensive and very flexible library for visualization, but implementing GPU-based (rendering) algorithms was unsupported until recently. Even with current support, there is a high degree of overhead present, which limits our flexibility in getting our data to the GPU and getting our code to run on it. We created NQVTK as an alternative path from VTK to the GPU. The aim is to offer

more flexibility in dealing with the underlying OpenGL API directly, while still being able to benefit from VTK's powerful data processing framework.

There are currently multiple ways of dealing with GPU programming, which can roughly be divided in two classes. First are the graphics-based GPU programming interfaces, such as OpenGL and Direct3D. These offer a set of functionality aimed primarily at creating graphics applications. Alternatively, many GPU platforms also offer one or more GPGPU programming interfaces, such as OpenCL and CUDA. For NQVTK we chose the former approach, as the goal of the library is to create visualizations. The downside is that some effort has to be made to fit GPGPU calculations to the graphics-oriented API. For instance, interpolation between high-dimensional points can be made fast by having the GPU count pixels (chapter 5, section 5.5.2), and a Jacobian determinant volume (chapter 3, section 3.3.1) can be computed by rendering slice-by-slice from one 3D texture into another, thanks to the separability of the Gaussian kernel.

The NQVTK library, along with the implementations of many of the visualizations presented in this thesis, has been released as open source under the permissive MIT license. The code can be downloaded from `http://nqvtk.googlecode.com/`.

In the following sections we take a short tour through the library, starting with a high-level overview of its components: the data framework, the rendering pipelines and the supporting utilities. We then discuss each of these in detail, and finish by giving suggestions for future improvement of the library.

6.1.1 A practical example

Let's start with a short example of the use of NQVTK. In chapter 2, we applied our interactive intersecting surfaces to shape models. These models could be deformed in real-time by changing the contributions of the separate eigenvectors. Additionally, the distances to a second, fixed mesh were visualized interactively on the model surface by sampling a distance field.

The data for these models was pre-processed into a single VTK polydata mesh, representing the mean shape of the population. Using VTK, each vertex of the model was extended with the components of the eigenvectors corresponding to that point, by assigning these to custom named VTK point data arrays. For the fixed mesh, a distance field was precomputed and sampled into a VTK volume dataset.

Using NQVTK, we load these models directly into a set of GPU vertex and index buffers, stored in GPU memory. The additional point data arrays are automatically recognized and loaded as well. Likewise, the distance field volume is loaded into a 3D GPU texture. NQVTK will take care of converting the data to fit the available texture format, and will remember scaling and offset values used so values sampled from the texture can be converted back to the original data values.

Creating the visualization is mostly a matter of writing the shader program. A GPU shader program consists of a vertex shader, responsible for transforming the vertices of the geometry into screen space, and a fragment (or pixel) shader, which

Listing 6.1: Accessing VTK point data in a vertex shader

```
1  // the eigenvectors
2  attribute vec3 eigvecs[NUM_EIGENMODES];
3
4  // user-controlled deformation factors
5  uniform float weights[NUM_EIGENMODES];
6
7  // Shader main function
8  void main() {
9          // original vertex position
10         vec4 vertex = gl_Vertex;
11
12         // Deform the vertex position according to weights and eigenvectors
13         for (int i = 0; i < NUM_EIGENMODES; ++i) {
14                 vertex += vec4(weights[i] * eigvecs[i], 0.0);
15         }
16
17         // Project the result into screen space
18         vec4 pos = gl_ModelViewMatrix * vertex;
19         gl_Position = gl_ProjectionMatrix * pos;
20 }
```

is responsible for computing the color for each pixel covered by the model's faces. In the vertex shader, we can access the eigenvectors simply by referencing the name of the corresponding VTK point data array. NQVTK will ensure the correct state is set up when the program is used together with our mesh.

Interaction with the program is provided by manipulating a set of uniforms, or program-wide variables. The vertex shader, see listing 6.1, multiplies these weights with the eigenvector for each points, and adds the result to the vertex' position. The result is that the object will be interactively deformed along the modes of variation represented by the user-selected eigenvalues. Like attributes, uniforms can simply be referenced by name, both inside the GPU-side program and by CPU-side code using it.

For visualizing the distance to the other mesh, we use NQVTK to link the distance field volume to the mesh. This will automatically make the volume data and related parameters available to any shader program used to render the mesh. We then use this information in the fragment shader (see listing 6.2) to change the visualization based on the actual distance.

6.2 Architecture

The most important aspect of NQVTK is its data framework. This has been designed to closely follow current OpenGL functionality, in order to get data to the GPU with minimal effort and high performance. OpenGL mostly works like a state machine, with state consisting of active textures, shader programs, program parameters and geometry source buffers. Once this state has been set up, rendering large amounts of geometry from the buffers and/or performing large amounts of calculations in shader logic are very fast. This works mainly by minimizing the amount of data needing to be transferred between system and GPU memory.

The NQVTK data framework was designed primarily to aid in managing this state. It provides importers for transferring VTK geometry and image data into suitable OpenGL data structures. It also contains a shader framework for configuring, compiling and setting up GLSL programs to run on the GPU and handling required textures and parameters.

Second, as it was developed primarily as a base for the visualizations presented in this thesis, NQVTK provides a framework for implementing similar rendering approaches. In particular, pipelines are provided for layered rendering, GPU-based ray casting and layered ray casting, each of which is described in more detail in the next section.

Finally, as NQVTK completely replaces VTK's own rendering and windowing systems, we lose the ability to work with VTK's built-in scene graph and interactors. For this reason, NQVTK provides a rudimentary scene management framework, including camera, transforms and interaction support.

6.2.1 Components

NQVTK consists of several base component classes, seen in figure 6.1, most of which can easily be subclassed to suit the requirements of a specific visualization pipeline.

A *renderable* is the base for any object in the visualization. A renderable is responsible for managing and rendering its own geometry. Most renderables inherit from VBOMesh and do this by populating an *attribute set* with a set of helper classes. The attribute set provides a straightforward interface for loading any data associated with the vertices of the object, such as positions, normals or PCA eigenvectors and -values, into OpenGL vertex buffers, and handles setting up the required state in order to render from these.

Per-object parameters for the visualization can be represented using one or more *parameter sets*. A parameter set can configure the shader program for the current pass and can configure textures to be made available under specific names to the program.

Renderables are grouped together in a *scene*, which is shared by one or more *renderers*. The renderers implement the various rendering pipelines, determining the

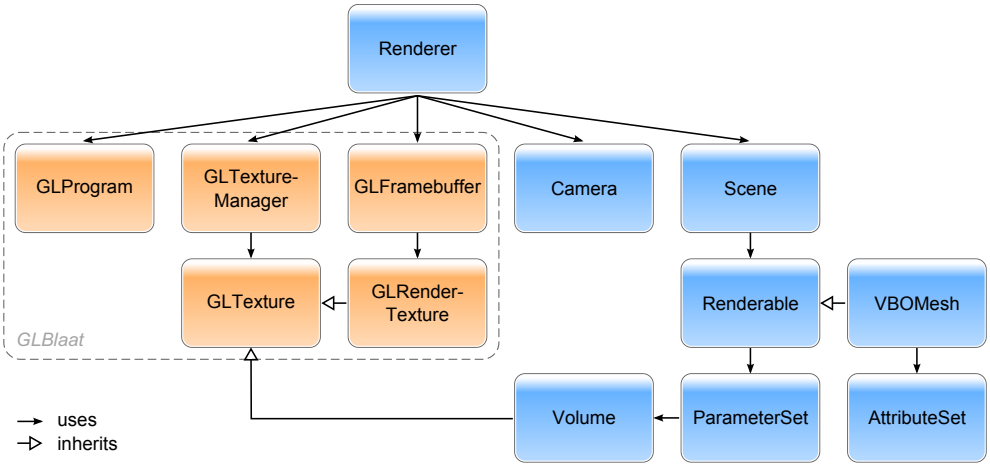


Figure 6.1: NQVTK's main components and relations

shader programs and render targets to use for the various passes required for generating the final image. NQVTK also provides several generic renderer implementations for adding effects such as overlays, stereo rendering and brushing to another renderer.

To accommodate interactive exploration, NQVTK provides a number of camera types for fixed up-direction orbits, arc-ball rotation or orthogonal projection, as well as a simple interactor framework for controlling these using the mouse.

6.3 Rendering

As noted earlier, NQVTK provides rendering pipelines for layered rendering, GPU-based ray casting and our layered ray casting algorithm. These allow for quick implementations of visualizations that can be based on one of these approaches. We will describe each of these pipelines in the following sections.

6.3.1 Basic techniques

The following sections highlight the three techniques used in the layered rendering and layered ray casting pipelines: deferred shading, depth peeling and ray casting.

Deferred shading

Deferred shading [ST90] is the process of separating rasterization of geometry from the computation of lighting and other appearance properties. An initial pass rasterizes geometry, but rather than outputting the final image, a shader program writes all

information required for the lighting computations to an intermediate buffer. A second pass then performs all lighting computations on this buffer, without requiring access to the geometry of the scene.

When lighting computations are computationally expensive, this method generally yields increased performance over a direct approach. This is because scenes tend to have multiple overlapping surfaces. In a direct approach, lighting would be computed for all such surfaces while many will not end up in the final image.

Additionally, by providing an image of the entire visible scene as input, deferred shading allows for several image-based rendering approaches which would not be possible in a direct approach. Examples are screen-space ambient occlusion or global illumination approximations, illustrative rendering effects such as contour detection, or blurring and other types of distortion (e.g., for simulating depth of field or refraction).

Depth peeling

Generally in GPU rasterization, a depth buffer is used to determine which of the various surfaces being rendered lies in front with respect to the camera. Any fragments behind this surface will be discarded. This results in problems when rendering translucent objects, as surfaces will need to be rendered in strict back-to-front order. Sorting surfaces can be computationally expensive, especially when dealing with complex intersections.

Depth peeling [Die96] is a multi-pass algorithm which can render a given scene one layer at a time. The basic idea is to use not one but two depth buffers. The extra buffer is initialized with the results from the previous pass, and used to discard fragments which lie on or in front of the previous surface. The other buffer maintains its function of allowing only fragments in front of those already rendered. The result is that each pass of the algorithm will yield a single layer of the scene. Compositing all layers in fixed order will result in a scene with proper transparency, without requiring object sorting at the geometry level. Moreover, as the combination of multiple layers of translucent surfaces will get increasingly opaque, often only a small number of layers is required to achieve an acceptable result.

In this thesis, in addition to applying depth peeling for correct transparency, we also take advantage of the separation of objects into multiple layers. This separation can be used, for instance, to perform CSG operations on the objects being rendered, or serve as a basis for ray casting between the surfaces. Both applications are discussed in the next sections.

Ray casting

In GPU ray casting [KW03], the first stage of the pipeline renders position information for the front and back faces of the bounding boxes of the volumes being rendered. A second stage then uses a shader program to step through the corresponding volumes

between the positions recorded for these faces, essentially following a ray from the camera through the volume. For each sample along the ray a transfer function is applied, yielding a color and opacity. At the end of the ray these are composited into the final image.

6.3.2 Pipelines

Layered rendering

Layered rendering is discussed in detail in chapter 2. Essentially, it is a combination of depth peeling and deferred shading. The depth peeling algorithm described above is modified to output a geometry buffer, and interleaved with deferred shading passes that generate the image for each layer.

This concept is extended by allowing the shading passes to propagate information between layers. This way we can record, for example, the objects a surface is inside of, which can be used to perform CSG operations. Alternatively, information from the previous layer can be used for thickness-aware translucency or distortion effects. As an example, figure 6.2 shows two versions of the dragon dataset, one smoothed, one not. The surfaces are overlayed, and a layered rendering CSG implementation is used to determine their intersection by tracking front and back surfaces while processing each layer. By comparing the depths of subsequent layers, thickness-aware red fog is added to the parts of both objects outside of the intersection.

NQVTK provides an extensible layered rendering implementation, which only requires two shader programs to be provided. The *scribe* performs depth peeling and is responsible for writing the required geometry information, combining information from the current layer with information propagated from all preceding ones. The *painter* is responsible for converting the information buffer generated by the scribe into the image for the layer. All layers are automatically composited in front-to-back order to form the final image.

Multi-volume ray casting

The ray casting algorithm described in the previous section can easily be extended to support multiple overlapping volumes by applying depth peeling on their bounding geometry. The approach, described in detail in chapter 3, section 3.3.4, then becomes similar to layered rendering, and can be implemented in the layered rendering pipeline. The scribe records position information for each layer of bounding geometry and propagates information on which volumes are active for each pixel. The painter stage then performs ray casting between the previous and current layer for all active volumes.

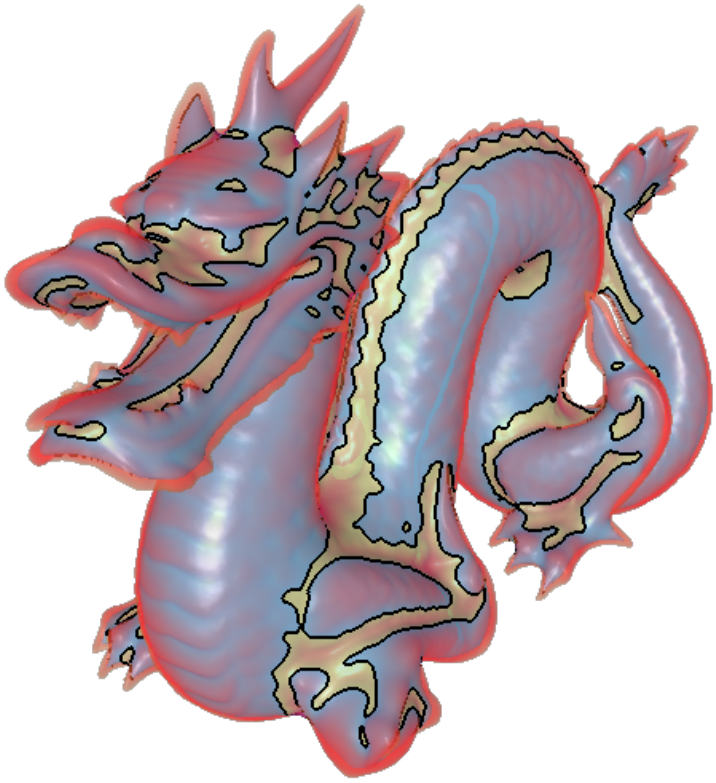


Figure 6.2: A layered rendering example using the dragon dataset, demonstrating CSG and thickness-aware translucency.

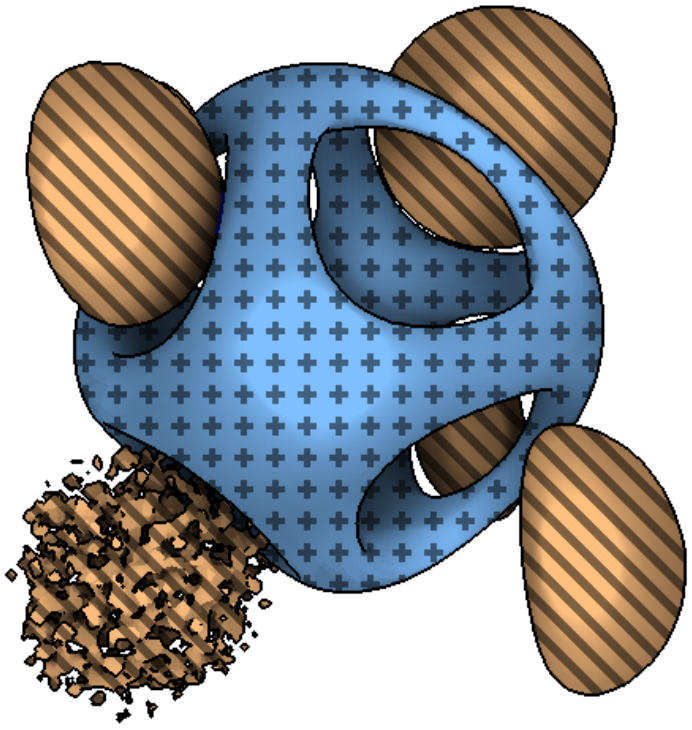


Figure 6.3: Layered ray casting example, showing an implicit surface and several features from a synthetic volume in an illustrative style.

Layered ray casting

In layered ray casting we combine all three techniques in order to apply deferred shading to surfaces extracted on-the-fly from both volume data and actual geometry. A complete description and application of the technique is given in chapter 4 (see section 4.5 for the rendering implementation). In summary, rather than performing ray casting in the painter stage, we insert a separate ray casting stage which again outputs position and geometry information. The depth peeling approach is then modified to read from the position information in this buffer rather than that generated by the scribe stage, resulting in surfaces generated by the ray casting to participate in the depth peeling process. As both scribe and ray caster output the same geometry information, the painter stage can apply deferred shading as though all fragments originate from the same scene. This way, ray casted objects are merged seamlessly with actual geometry, and any layered rendering painter-based effect can be applied equally to both.

Figure 6.3 gives an example layered ray casting image, demonstrating the seamless integration of implicit surfaces (blue) and volume features (orange). It also demonstrates how deferred shading can be used to apply illustrative effects, such as the textures and contours highlighting the different objects.

6.4 Implementation details

At the lowest level, NQVTK uses the GLBlaat library [Bus10]. The GLBlaat library provides a set of C++ classes designed to simplify management of specific OpenGL objects. The following OpenGL functionalities are supported:

- Shader programs - GLBlaat handles compiling, linking and configuring shader code.
- Texturing - 2D, rectangle and 3D textures are supported. A separate *texture manager* helper is provided to simplify setting up the specific textures required by a given shader program.
- Buffer objects - these can be seen as allocated pieces of GPU memory holding object geometry data.
- Framebuffer objects - these provide a convenient and performant way of rendering to off-screen targets, for instance when implementing multi-pass pipelines. A framebuffer can contain several target buffers to which a shader can output color, depth or even geometry information per rendered pixel. Helpfully, these buffers can be textures, which can be used as input for a following pass while remaining in GPU memory.

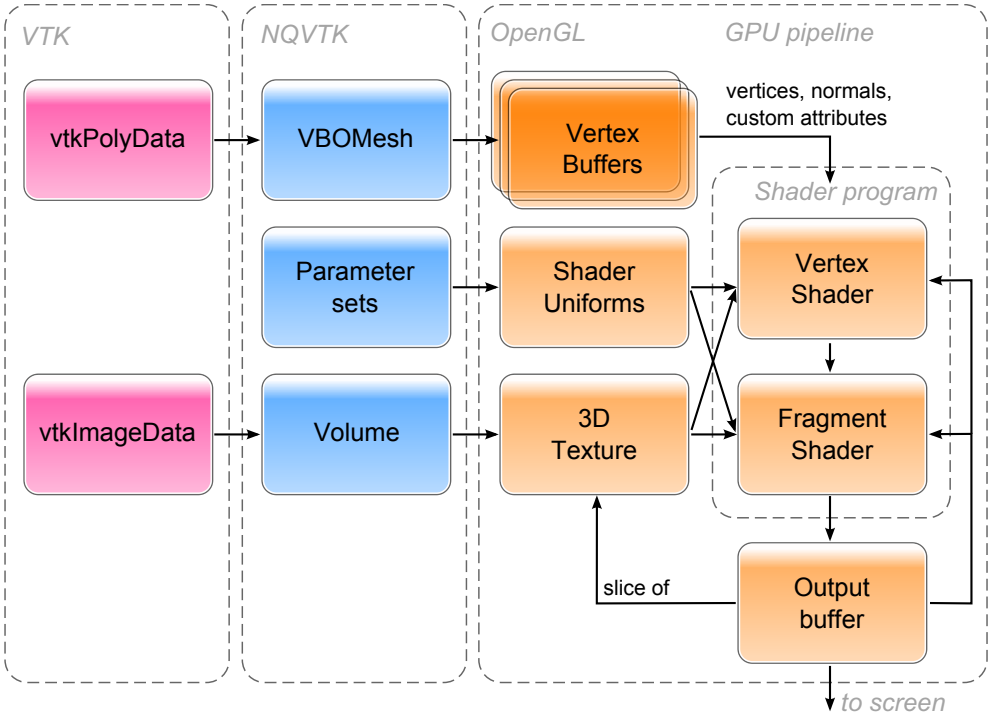


Figure 6.4: Data flow from VTK to the GPU using NQVTK

- Asynchronous queries - OpenGL provides a number of ways to measure the results of rendering. The most useful of these is the occlusion query, which will measure the number of rendered pixels which actually make it to the screen (as opposed to being discarded, e.g., by a failed depth-buffer test).

In addition to GLBlaat, basic 3D vector and matrix classes are provided. Basic arithmetic operators have been implemented in order to make code using these classes more readable.

6.4.1 Data framework

Figure 6.4 illustrates the role of the NQVTK data framework in getting data from VTK to the GPU. The NQVTK VBOMesh renderable is based around one or more vertex buffer objects (VBOs), an OpenGL structure for storing geometry efficiently in GPU memory. VTK PolyData objects are transformed into separate VBOs containing points coordinates, normals and texture coordinates. VTK also provides the option

of assigning arbitrary point data. NQVTK makes each set of such data available to shader programs as a custom attribute, backed by another VBO.

VTK PolyData can consist of points, lines, polygons and triangle strips. NQVTK transforms each of these to a separate index buffer object. For polygons, only triangles are supported. Meshes not satisfying this limitation can easily be converted using the VTK triangle filter.

Similarly, VTK image data can be loaded into volumes implemented as OpenGL 3D floating-point textures. As the range of these is limited to 0 to 1, the data is automatically shifted and scaled from its original values. The inverse transformation is automatically made available to shader programs when a volume is linked to a renderable using a volume parameter set. This way such programs can interpret the volume data in terms of its original values, e.g., for accurate thresholding.

Parameter sets affect the visualization by setting shader program uniforms, and by registering object-specific textures in the GLBlaat texture manager. For attributes, uniforms and textures, names are automatically resolved to binding locations in the linked programs.

A rudimentary framework is also provided for GPU-based filtering of volume data. Volumes are processed four slices at a time due to OpenGL limitations. Filters are provided to compute a Gaussian convolution or the Jacobian determinant of a given volume. Custom filters can be implemented by simply providing a shader program to perform the computation and subclassing the base class in order to implement the required passes over the volume.

6.4.2 Rendering

The various multi-pass rendering pipelines are implemented by rendering all passes except the final pass to an OpenGL framebuffer object (FBO). Each pass receives the results from the previous pass(es) as input using one or more textures, and uses a different shader in order to render the next (partial) result. OpenGL allows up to four output buffers to be attached to a VBO, each of which can have up to four output components (generally referred to as red, green, blue and alpha). This means up to 16 channels, usually each containing 8 bits of information, can be used to output information from a shader.

6.5 Discussion

While the NQVTK framework still has a few rough edges, it has shown to be a huge time saver during development of the visualizations described in this thesis. Combined with the Qt GUI framework, applications can be easily created which present multiple linked views on a dataset. The data framework can be combined with simple VTK pipelines or preprocessing scripts in order to get a given dataset onto the GPU with minimal effort. With the various pipeline frameworks, creating visualizations mostly

comes down to writing the required shader programs and linking up the GUI for tuning their parameters.

For instance, NQVTK enabled rapid development of the Multi-Field Explorer application (see chapter 4). The main visualization was developed on top of the layered ray casting pipeline, reusing code from the intersecting surfaces (chapter 2) and volume deformation (chapter 3) visualizations. Using a specialized NQVTK renderer, specialized views like the slice view and the component feedback widget could easily be implemented on the GPU. Not only do these views share the same data, which only has to be loaded once and is automatically made available to all views, they are also easily linked by manipulating the same set of shared parameters.

6.5.1 Future work

NQVTK was written between 2007 and 2010. With the speed at which GPU technology is evolving, many new options have become available for enhancing the performance and capabilities of our visualizations and similar techniques. It would be interesting to upgrade NQVTK and its pipelines to the latest OpenGL version and benefit from new developments such as order-independent transparency techniques, improvements in state management and compute shaders for GPGPU applications.

The simplicity of NQVTK's flat scene and renderables model makes it easy to get simple scenes up and running, but creates limitations in dealing with more complex scenes. Integration of an (either existing or specialized) scene graph system such as OpenSceneGraph [BO11] may help in these cases. Parameter sets could then be linked to arbitrary nodes in the graph in order to provide the required state for groups of renderables, and such groups could more easily be manipulated as a whole.

Largely because of GLBlaat's OpenGL abstractions, NQVTKs rendering pipelines are fairly straightforward to implement. However, their structure is locked in code. A possible improvement is to make the pipeline configurable by defining a graph of passes and linking up inputs and outputs to other passes, data sources and render target buffers. This could be combined with a task-based rendering pipeline, where individual graph nodes could submit work to be performed on the GPU. Such tasks could then be scheduled and executed in the optimal order, according to their dependencies and by minimizing the required OpenGL state changes. Extensions such as stereo rendering and overlays would then be simple additions to the pipeline graph.

Finally, integration between VTK and NQVTK is currently one-way: data from VTK can be loaded and displayed through NQVTK. This link could be made bi-directional, allowing selections and GPU-processed data from NQVTK to be fed back into a VTK pipeline for further CPU-side processing. This would make NQVTK a good platform for accelerating VTK itself, by implementing not only visualization, but also filters to run on the GPU. When combined with the upgrade to recent GPU programming standards mentioned above, this could even allow GPU-generated geometry to be used by the VTK pipeline.

Listing 6.2: Using a VTK volume in a fragment shader

```
1
2  // the volume and its parameters, automatically populated by NQVTK
3  uniform sampler3D volume;
4  uniform float volumeDataShift;
5  uniform float volumeDataScale;
6  uniform vec3 volumeOrigin;
7  uniform vec3 volumeSize;
8
9  // user-defined threshold
10 uniform float distanceThreshold;
11
12 // Shader main function
13 void main() {
14         [...]
15
16         // Transform vertex position to volume coordinate frame
17         vec3 p = vertex.xyz / vertex.w;
18         p = (p - volumeOrigin) / volumeSize;
19
20         // Sample the texture
21         float dist = texture3D(volume, p).x;
22         dist = abs(dist * volumeDataScale + volumeDataShift);
23
24         // Thresholding based on real distance values
25         if (dist < distanceThreshold) {
26                 col = vec4(1.0);
27         }
28
29         [...]
30 }
```

CHAPTER 7

Discussion

There is an interesting duality in comparison: in order to determine and show what is different (and/or similar) and interesting, one generally has to spend much of time and effort leaving out the unimportant differences[1]. This problem can usually not be solved completely in any generic way. Unfortunately, this means it is often left completely for users to deal with.

In this thesis we have explored several cases of studying variation and variability, with the aim of finding ways to assist a user in exploring these phenomena in their context. The following section summarizes our results with the aim of answering the research questions defined in the initial chapter of the thesis. Next, we generalize our findings and present lessons learned on using IVA approaches for exploring variation and variability. We conclude this chapter with a number of recommendations on following up on our research and techniques, both in terms of a user's analysis process and in terms of continuing the research itself.

7.1 Exploring variation and variability

While this thesis is about visualization of variation and variability, it could never fully cover this huge topic. However, several threads can be identified running through this thesis, each sampling over a dimension of a hypothetical topic "feature space". We

[1] "and/or similarities, correlations, etc." is implied whenever differences are mentioned, unless otherwise specified

will use this topic space to guide the discussion in this chapter, by considering the following dimensions:

- Entity of analysis (scale): objects, groups, populations
- Dimensionality: 2D (surfaces), 3D (volumes), N-D
- Level of comparison: entity to entity, entity to group, entity to population, group to population, population to population
- Level of data abstraction: degree of domain matching and other pre-processing required
- Targets of visualization (feature abstraction): differences, similarities, general characteristics, population structure
- Comparative visualization composition: alignment, integration of focus+context
- Interactivity: basic parameters, interactive exploration, interactive visual analysis

The following table approximately positions the chapters of this thesis in the space defined by these dimensions, using a score of one to three stars.

Dimension	chapter 2	chapter 3	chapter 4	chapter 5
Scale	☆	☆	☆☆	☆☆☆
Dimensionality	☆	☆☆	☆☆☆	☆☆☆
Level of comparison	☆	☆	☆☆	☆☆☆
Data abstraction	☆	☆☆	☆☆☆	☆☆☆
Feature abstraction	☆	☆☆	☆☆☆	☆
Composition	☆	☆☆☆	☆☆	☆☆
Interactivity	☆	☆☆	☆☆☆	☆☆

In the initial chapter of this thesis, we defined the scope of our research by posing three research questions, repeated below. The following sections discuss the chapters of this thesis relevant to each question, in terms of the topic space defined above and the contributions made towards answering each question.

- How can we use an IVA approach for visualizing and exploring object shapes in a direct way? (chapter 2, chapter 5)
- How can we allow a user to interactively manipulate the composition of a visualization in order to clearly visualize variation and variability in their context? (chapter 3, chapter 4)
- How can we assist a user in an interactive application in order to overcome the relevance problem? That is, how can we interactively allow a user to indicate relevance and have the visualization react to this? (chapter 3, chapter 4, chapter 5)

7.1.1 IVA for comparing object shapes

Chapter 2 explored an instance of entity-level comparison of 3D surfaces, as we explored several improvements to the intersecting surfaces visualization. This chapter confirmed the importance of using integrated rather than separate visualizations for comparison. However, our evaluation has also shown some of the difficulties in presenting scenes with high visual complexity in an intuitive way.

The chapter further introduced the layered rendering framework, further detailed in chapter 6, which enables interactive performance for the intersecting surfaces visualization, even as the objects being compared are deformed or moved relative to each other. In the context of shape models, the interactivity this brought to the intersecting surfaces visualization created an intuitive way of exploring the variability of one model relative to a given shape.

In chapter 3 we expanded the dimensionality of the entities under consideration from surfaces to volumes, allowing not only comparison of anatomical surfaces, but also of the effects of such deformation on the surrounding tissue. To reduce visual complexity, the anatomical features serving as context to the visualization were represented using only their contours.

In chapter 5 we continued the shape model theme and extended the scope of comparison to population analysis and entity-population comparison. Here, shapes could be compared interactively by "browsing" through the population using the mouse pointer. This chapter also introduced a novel "shape evolution view", using two spatial dimensions for visualizing object space, with the third used to represent a trajectory through shape space. The resulting set of object silhouettes can be viewed in different configurations, depending on the interests of the user.

7.1.2 Interactive composition

The visualizations in chapter 2 had several parameters which could be manipulated interactively, including camera, surface opacity and the inclusion of various glyphs. The way in which these were combined, however, was by traditional composition into the same 3D scene. Enabling more features at the same time therefore lead to increased visual complexity, and a higher cognitive burden on the user trying to understand the visualizations.

In chapter 3, rather than overlaying the data directly, we directly visualized the deformation field resulting from domain mapping result. This provided information on the differences at a higher semantic level, separately highlighting areas of expansion, contraction and movement. As in the previous chapter the visualizations were rendered at interactive frame rates, enabling fast exploration. Contrary to the previous chapter, we used interactive importance-based composition to combine the features representing the deformation field and the anatomical contours representing their context. Furthermore, the deformation could be applied to the context interactively, showing its effects on the volume.

We generalized the concept of highlighting areas in a volume with specific characteristics in chapter 4. Here, we allowed a user to interactively discover and highlight features of interests in a multi-field dataset by pointing at interesting areas within the data. Using the characteristics of the given point, a similarity field was computed in real-time and visualized to highlight similar areas, effectively allowing interactive entity-level comparison of points within the volume. Multiple features could be specified and manipulated at will, giving the user full control over the resulting visualization. To reduce visual complexity, we used a sparse, illustrative style, using colors and contours to clearly highlight the feature objects selected by the user.

While not strictly composition, chapter 5 used strong linking between its various views to enable the user to understand the combined visualizations. All shape projections share the same camera, including the miniatures in the scatter plot view. Pointing in the scatter plot view shows the corresponding object in the 3D view, and changing the scatter plot projection changes the trajectory shown in the shape evolution view. We have found that linking view in intuitive ways greatly helps in simplifying multiple-views visualizations, by allowing a user to easily discover how views and features shown in them relate to each other. Properly done, linking facilitates mentally mapping results from one view's domain into the others. This way, abstract visualizations can interpreted more easily, as discoveries made in them can be mapped back to the problem domain.

7.1.3 Specification of interest

In addition to interactivity in rendering and the composition of the visualizations, an important part of the IVA approach is the specification of interest. Users need to be able to quickly indicate what is important to them in a dataset, and the visualization needs to respond to that. Interactive composition, is one way to deal with this problem, but more controls can be offered. Interactive specification of interest is especially important when dealing with the relevance problem: many differences may be present, but often only the user knows which ones are relevant to their questions.

In chapter 3, we used importance-based composition as the primary means of specifying interest. Each component of the visualization, deformation features, movement areas and context contours, could be assigned a relative importance and the visualization would immediately reflect this.

Specification of interest was central to the visualizations in chapter 4. We have shown the Euclidean distance in feature space to be an effective measure of similarity, and have enhanced this similarity measure by allowing for optional non-uniform scaling of the feature space in order to create a bias for (or against) certain characteristics. The application effectively uses a multiple-linked-views approach to select points, explore the 3-dimensional similarity field iso-surfaces and highlight and filter the specific characteristics of the selected points.

In chapter 5, the features of interest would be trends observed in the population represented by the shape model. To this end, we used the principal components of the

shape space to assist the user in narrowing down these trends to a more limited number of dimensions. We again left the remaining exploration up to the user, by allowing straightforward manipulation of the shape space projection, detailed examination of individual shapes by morphing based on the cursor position and detailed examination of trends in a specialized view.

7.2 Conclusions

In the previous chapters we have examined several visualizations of variation and variability. This section aims to generalize some of our findings, so they may be applicable to other problems at similar positions in the topic space.

In comparison, linking is key. What is understood to be "the same thing" in the specific problem domain should be identifiable as such. Linking can be accomplished (and maintained) through specific visualizations, through recognizable markers or through multi-view interaction.

In comparative visualization, merged approaches (where representations of the entities under consideration are combined in a single view) are the most powerful. Explicitly showing differences in their natural context enables intuitive exploration of their possible causes and effects. Care must be taken, however, as such approaches are also very sensitive to clutter and other explosions of visual complexity, and much more effort is required to do the combination well.

Illustrative rendering is a useful tool for keeping such visual complexity within bounds. Often, contours are sufficient to provide a good indication where things are in relation to others. Other techniques can also be used, such as curvature-aligned patterns or ambient occlusion to emphasize surface shape, texture patterns to indicate the directionality of changes, or intersection contours to distinguish intersection from occlusion.

A tight user-feedback loop is hugely advantageous when exploring any comparison scenario. Preferably, an application should therefore enable intuitive filtering and decomposition of any differences found, and the visualizations used should provide interactive and immediate response to any changes made in this filtering and to the visualization parameters.

7.3 Following up

The techniques presented in this thesis are in no way the only solution or approach to dealing with variation and variability. We see their primary purpose in the initial exploration of any dataset. Our techniques are aimed to provide insight into data without requiring the user to have much initial knowledge about its specifics. Obviously, after initial exploration is done, the user will want to follow up on any hypotheses found. Likewise, there is much that could still be done in this area of

research. The previous chapters each list smaller-scale ideas related to their specific topics. In this section, we illustrate our vision on further avenues for research and development on a more general level.

7.3.1 On evaluation

A common concern when developing and presenting new visualization techniques is the question when and how to evaluate their effectiveness. Rather than blindly piling big user studies onto every new technique, we have found a more gradual approach to evaluation to be more suitable for techniques and approaches such as those described in this thesis.

This is not to say that evaluation is not important. We recommend always staying in touch with the application domain, even when developing generalized techniques. To this end, one should evaluate both during and after development of general techniques by doing case studies in various application domains. The main goal of these is not to evaluate the specific application to the specific domain, but rather the applicability of the techniques themselves to each class of problems. One therefore has to make sure that feedback gathered during these evaluations applies to the general case. When working with users, this means to be flexible in your implementation, to be able to quickly work around application-specific implementation issues and prevent such issues from getting in the way of the main evaluation.

When adapting an application or generic technique to a specific use case, get help from an expert whenever possible. As no one knows the application domain better than them, expert evaluations (such as performed in chapter 2) also tend to quickly uncover information on what exactly a user is looking for, conventions used in existing approaches for the domain and analysis and follow-up questions relevant to domain data. Such data can help in designing an application as well as in defining future task-specific evaluations of the final implementation. In fact, given the specialist nature of the comparative visualization scenarios explored in this thesis, all users will usually be domain experts. General (novice) user evaluations have therefore been less relevant.

7.3.2 Extension and integration

With regards to our coverage of the variability subject, an important area not touched in this thesis is merged comparative visualization of continuous phenomena. In addition to one-to-one comparison of distinct time steps, it would be interesting to be able to compare data with no natural static representation. Examples include comparing motion, comparing different trajectories through a shape space (chapter 5), or examining the changes in deformation fields over time (chapter 3) in order to estimate changes in the local rates of growth.

Considering the IVA basis for our techniques, more could be done to allow for the interactive specification of interest. We envision a system which combines the

visualizations presented here with interactive, semi-automated decomposition of differences. Such an approach could be effective in solving the relevance problem, as a user can interactively eliminate specific components of the differences found within a comparison. For example, a user could interactively eliminate differences in skeletal positioning by overlaying a skeleton-based deformation model, or the effects of brain shift due to patient orientation by interactively fitting and subtracting a model of this type of deformation.

On a technical level we have aimed for interactivity in all applications presented in this thesis. In order to avoid the technical complexities of dealing with large data sets, we have often restricted our implementations to data sets which fit in (graphics) memory. Real-world data, however, is growing at great pace. A number of good out-of-core processing and rendering techniques exist for dealing with large data sizes, which could be integrated with our techniques to improve their scalability. On a different level, our techniques could be extended to deal with multi-scale or multi-resolution data.

Finally, in terms of integration our approaches could benefit from tight coupling with analytical techniques, statistical modeling and application-specific approaches. A recurring theme is the need for application-specific feedback when generic techniques are used. In an interactive multi-view application, one could add specific views representing data, selections, etc. in the form or forms most natural to the application and problem under consideration. Integration of not only measuring tools, but also statistic analysis and feedback would allow for more informed feature selections. Such information could be presented to the user, or used automatically in guided feature selection. For example, Kehrer et al. [KFH10] used a derived feature space composed of the statistical moments of the dataset under consideration. This was combined with a brushing interface to enable interactive visual analysis of the statistical properties of a given dataset.

Most of the work presented in the thesis has been released in the form of open source software. This includes both the NQVTK framework, detailed in chapter 6, as well as the Shape Space Explorer and Multi-Field Explorer applications. We hope this will enable others to easily build on our techniques and implementations.

Bibliography

[ABH+06] P. Aljabar, K. K. Bhatia, J. V. Hajnal, J. R. Boardman, L. Srinivasan, M. A. Rutherford, L. E. Dyet, A. D. Edwards, and D. Rueckert. Analysis of growth in the developing brain using non-rigid registration. In *IEEE Biomedical Imaging: Nano to Macro, Proceedings*, pages 201–204, 2006.

[AHF+98] J. Ashburner, C. Hutton, R. Frackowiak, I. Johnsrude, C. Price, and K. Friston. Identifying global anatomical differences: Deformation-based morphometry. *Human Brain Mapping*, 6(5-6):348–357, 1998.

[Ale02] M. Alexa. Recent advances in mesh morphing. *Computer graphics forum*, 21(2):173–198, 2002.

[Asi85] D. Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM Journal of Scientific and Statistical Computing*, 6:128–143, 1985.

[BBF+11] S. Busking, C. P. Botha, L. Ferrarini, J. Milles, and F. H. Post. Image-based rendering of intersecting surfaces for dynamic comparative visualization. *The Visual Computer*, 27(5):347–363, 2011.

[BBP07] J. Blaas, C. P. Botha, and F. H. Post. Interactive visualization of multi-field medical data using linked physical and feature-space views. In *Proc. Eurographics / IEEE-VGTC EuroVis*, pages 123–130, 2007.

[BBP09] S. Busking, C. P. Botha, and F. H. Post. Direct visualization of deformation in volumes. In *Eurographics / IEEE-VGTC Symposium on Visualization, Computer Graphics Forum*, volume 28, pages 799–806, 2009.

[BBP10a] S. Busking, C. P. Botha, and F. H. Post. Dynamic multi-view exploration of shape spaces. In *Eurographics / IEEE-VGTC Symposium on Visualization, Computer Graphics Forum*, volume 29, pages 973–982, 2010.

[BBP10b] S. Busking, C. P. Botha, and F. H. Post. Example-based interactive illustration of multi-field datasets. *Computers & Graphics*, 34(6):719–728, 2010.

[BCL+07] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, and C. T. Silva. Multi-fragment effects on the GPU using the k-buffer. In *ACM i3D, Proceedings*, pages 97–104, 2007.

[BFHU09] T. Bobach, G. Farin, D. Hansford, and G. Umlauf. Natural neighbor extrapolation using ghost points. *Computer-Aided Design*, 41(5):350–365, 2009.

[BG05] I. Borg and P. J. F. Groenen. *Modern multidimensional scaling: theory and applications*. Springer, 2nd edition, 2005.

[BH07] A. Bair and D. House. A grid with a view: optimal texturing for perception of layered surface shape. *IEEE Transactions on Visualization and Computer Graphics*, 13:1656–1663, 2007.

[Bla10] J. Blaas. *Visual analysis of multi-field data*. PhD thesis, Delft University of Technology, 2010.

[BO11] D. Burns and R. Osfield. OpenSceneGraph project website. http://www.openscenegraph.com/, 2011.

[BT07] D. Borland and R. M. Taylor II. Rainbow color map (still) considered harmful. *IEEE Computer Graphics and Applications*, (April):14–17, 2007.

[Bus10] S. Busking. GLBlaat - A set of simple C++ wrappers for various OpenGL bits and extensions. https://code.google.com/p/glblaat/, 2010.

[CKS01] D. Cremers, T. Kohlberger, and C. Schnörr. Nonlinear shape statistics via kernel spaces. In B. Radig and S. Florczyk, editors, *Pattern recognition*, volume 2191 of *Lecture Notes in Computer Science*, pages 269–276. Springer Berlin Heidelberg, 2001.

[Cle93] W. S. Cleveland. *Visualizing data*. AT&T Bell Laboratories, 1993.

[CM92] R. Crawfis and N. Max. Direct volume visualization of three-dimensional vector fields. In *Symposium on Volume Visualization, Proceedings*, pages 55–60, 1992.

[CMBC93] R. Crawfis, N. Max, B. Becker, and B. Cabral. Volume rendering of 3D scalar and vector fields at LLNL. In *Supercomputing '93, Proceedings*, pages 570–576, 1993.

[CMS99] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in information visualization: using vision to think*, volume 1st of *The Morgan Kaufmann Series in Interactive Technologies*. Morgan Kaufmann, 1999.

[COS06] D. Cremers, S. J. Osher, and S. Soatto. Kernel density estimation and intrinsic alignment for shape priors in level set segmentation. *International Journal of Computer Vision*, 69(3):335–351, 2006.

[CTCG95] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models – their training and application. *Computer Vision and Image Understanding*, 61:38–59, 1995.

[DGBW09] C. Dick, J. Georgii, R. Burgkart, and R. Westermann. Stress tensor field visualization for implant planning in orthopedics. *IEEE transactions on visualization and computer graphics*, 15(6):1399–406, 2009.

[DGH03] H. Doleisch, M. Gasser, and H. Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 239–248. Eurographics Association, 2003.

[dGS99] V. di Gesú and V. Starovoitov. Distance-based functions for image comparison. *Pattern Recognition Letters*, 20:207–214, 1999.

[Die96] P. Diefenbach. *Pipeline rendering: interaction and realism through hardware-based multi-pass rendering*. PhD thesis, 1996.

[Dol07] H. Doleisch. SimVis: interactive visual analysis of large and time-dependent 3D simulation data. In *2007 Winter Simulation Conference*, pages 712–720. IEEE, December 2007.

[DTA+03] R. H. Davies, C. J. Twining, P. D. Allen, T. F. Cootes, and C. J. Taylor. Shape discrimination in the hippocampus using an MDL model. *Information processing in medical imaging, proceedings*, 18:38–50, July 2003.

[DWS+88] M. Deering, S. Winner, B. Schediwy, C. Duffy, and N. Hunt. The triangle processor and normal vector shader: a VLSI system for high performance graphics. *ACM SIGGRAPH, Proceedings*, 22:21–30, 1988.

[DX08] H. Q. Dinh and L. Xu. Structural, syntactic, and statistical pattern recognition. *Lecture Notes in Computer Science*, 5342:187–196, 2008.

[Eve01] C. Everitt. Interactive order-independent transparency. Technical report, NVIDIA, 2001.

[FAP+07] P. Fillard, V. Arsigny, X. Pennec, K. M. Hayashi, P. M. Thompson, and N. Ayache. Measuring brain variability by extrapolating sparse tensor fields measured on sulcal lines. *NeuroImage*, 34(2):639–50, January 2007.

[FEK+05] Q. Fan, A. Efrat, V. Koltun, S. Krishnan, and S. Venkatasubramanian. Hardware-assisted natural neighbor interpolation. In *Seventh Workshop Algorithm Engineering and Experiments (ALENEX), Proceedings*, 2005.

[FH09] R. Fuchs and H. Hauser. Visualization of multi-variate scientific data. *Computer Graphics Forum*, 28(6):1670–1690, 2009.

[FOP+07] L. Ferrarini, H. Olofsen, W. M. Palm, M. A. van Buchem, J. H. C. Reiber, and F. Admiraal-Behloul. GAMEs: growing and adaptive meshes for fully automatic shape modeling and analysis. *Medical Image Analysis*, 11:302–314, June 2007.

[FPO+06] L. Ferrarini, W. M. Palm, H. Olofsen, M. A. van Buchem, J. H. C. Reiber, and F. Admiraal-Behloul. Shape differences of the brain ventricles in Alzheimer's disease. *Neuroimage*, 32:1060–1069, September 2006.

[Frü93] T. Frühauf. Raycasting vector fields. In *IEEE Visualization, Proceedings*, pages 115–120, 1993.

[FT74] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9), 1974.

[GBP06] G. Guennebaud, L. Barthe, and M. Paulin. Splat/mesh blending, perspective rasterization and transparency for point-based rendering. In *IEEE/Eurographics/ACM Symposium on Point-Based Graphics*, pages 49–58, 2006.

[GGGZ05] T. Gatzke, C. Grimm, M. Garland, and S. Zelinka. Curvature maps for local shape comparison. In *IEEE Shape Modeling and Applications, Proceedings*, pages 244–253, 2005.

[GGSK05] P. Golland, W. E. L. Grimson, M. E. Shenton, and R. Kikinis. Detection and analysis of statistical differences in anatomical shape. *Medical image analysis*, 9(1):69–86, 2005.

[GMTF89] J. Goldfeather, S. Molnar, G. Turk, and H. Fuchs. Near real-time CSG rendering using tree normalization and geometric pruning. *IEEE Computer Graphics and Applications*, 9:20–28, 1989.

[GNB+01] C. Gaser, I. Nenadic, B. R. Buchsbaum, E. A. Hazlett, and M. S. Buchsbaum. Deformation-based morphometry and its relation to conventional volumetry of brain lateral ventricles in MRI. *NeuroImage*, 13(6):1140 – 1145, 2001.

[GPR+04] M. Griebel, T. Preusser, M. Rumpf, M.A. Schweitzer, and A. Telea. Flow field clustering via algebraic multigrid. In *Proceedings, IEEE Visualization*, pages 35–42, 2004.

[GRW+00] D. L. Gresh, B. E. Rogowitz, R. L. Winslow, D. F. Scollan, and C. K. Yung. WEAVE: a system for visually linking 3-D and statistical visualizations applied to cardiac simulation and measurement data. In *Proceedings of IEEE Visualization 2000*, pages 489–492, 2000.

[HA04] A. Helgeland and O. Andreassen. Visualization of vector fields using seed LIC and volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):673–682, 2004.

[Hen98] C. Henze. Feature detection in linked derived spaces. In *Proceedings, IEEE Visualization*, pages 87–94, 1998.

[HFH+04] I. Hotz, L. Feng, H. Hagen, B. Hamann, K. Joy, and B. Jeremic. Physically based methods for tensor field visualization. In *IEEE Visualization, Proceedings*, pages 123–130, 2004.

[HH06] S. Hacker and H. Handels. Representation and visualization of variability in a 3D anatomical atlas using the kidney as an example. In *SPIE Medical Imaging 2006, Visualization, Image-Guided Procedures, and Display*, volume 7, pages 61410B–1 – 61410B–7. SPIE, 2006.

[HM90] R. B. Haber and D. A. McNabb. Visualization idioms: a conceptual model for scientific visualization systems. In G. M. Nielson, B. D. Shriver, and L. J. Rosenblum, editors, *Visualization in scientific computing*, pages 74–93. IEEE Computer Society Press, 1990.

[HM09] T. Heimann and H. P. Meinzer. Statistical shape models for 3D medical image segmentation: a review. *Medical image analysis*, 13(4):543–63, 2009.

[HSK11] M. Hermann, A. C. Schunke, and R. Klein. Semantically steered visual analysis of highly detailed morphometric shape spaces. In *Biological Data Visualization (BioVis), 2011 IEEE Symposium on*, pages 151–158, October 2011.

[HSSK14] M. Hermann, A. C. Schunke, T. Schultz, and R. Klein. A visual analysis approach to study anatomic covariation. In *IEEE PacificVis 2014*, pages 161–168, March 2014.

[HTSW03] H. C. Hege, H. Theisel, H. P. Seidel, and T. Weinkauf. Saddle connectors - an approach to visualizing the topological skeleton of complex 3D vector fields. In *Proceedings, IEEE Visualization*, pages 225–232, 2003.

[IFP97] V. Interrante, H. Fuchs, and S. Pizer. Conveying the 3D shape of smoothly curving transparent surfaces via texture. In *IEEE Transactions on Visualization and Computer Graphics*, pages 98–117, 1997.

[JL91] D. Jespersen and C. Levit. Tapered cylinder dataset. http://www.nas.nasa.gov/publications/datasets.html, 1991.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[JS03] C. R. Johnson and A. R. Sanderson. A next step: visualizing errors and uncertainty. *IEEE Computer Graphics and Applications*, 23(5):6–10, 2003.

[Kan00] E. Kandogan. Star coordinates: a multi-dimensional visualization technique with uniform treatment of dimensions. In *Proceedings of the IEEE Information Visualization Symposium, Late Breaking Hot Topics*, volume 650, pages 9–12, 2000.

[Kan01] E. Kandogan. Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining KDD 01*, 123(3):107–116, 2001.

[Ken84] D. G. Kendall. Shape manifolds, procrustean metrics, and complex projective spaces. *Bulletin of the London Mathematical Society*, 16(2):81–121, March 1984.

[KFH10] J. Kehrer, P. Filzmoser, and H. Hauser. Brushing moments in interactive visual analysis. *Computer Graphics Forum*, 29(3):813–822, 2010.

[Kin04] G. Kindlmann. Superquadric tensor glyphs. In *Proceedings of IEEE TVCG/EG Symposium on Visualization*, pages 147–154, 2004.

[KKH02] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[KKS+07] P. R. Kornaat, M. Kloppenburg, R. Sharma, S. A. Botha-Scheepers, M. P. H. Le Graverand, L. N. J. E. M. Coene, J. L. Bloem, and I. Watt. Bone marrow edema-like lesions change in volume in the majority of patients with osteoarthritis; associations with clinical features. *European Radiology*, 17(12):3073–3078, 2007.

[KMM+01] J. Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, and C. Hansen. Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications*, 21(4):52–61, 2001.

[KMP07] M. Kilian, N. J. Mitra, and H. Pottmann. Geometric modeling in shape space. *ACM Transactions on Graphics (TOG)*, 26(3), 2007.

[KSP07] S. Klein, M. Staring, and J. P. W. Pluim. Evaluation of optimization methods for nonrigid medical image registration using mutual information and B-splines. *IEEE Transactions on Image Processing*, 16(12):2879–2890, 2007.

[KW03] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization, Proceedings*, 2003.

[LHD+03] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf. The state of the art in flow visualization: dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–221, 2003.

[LHGQ06] X. Li, Y. He, X. Gu, and H. Qin. Curves-on-surface: a general shape comparison framework. In *IEEE Shape Modeling and Applications, Proceedings*, pages 38–43, 2006.

[LSL+04] H. Lamecker, M. Seebass, T. Lange, H. C. Hege, and P. Deuflhard. Visualization of the variability of 3D statistical shape models by animation. *Studies in health technology and informatics*, 98:190–196, January 2004.

[LST03] I. S. Lim, S. Sarni, and D. Thalmann. Colored visualization of shape differences between bones. In *IEEE Computer Based Medical Systems, Proceedings*, pages 26–27, 2003.

[Ma07] K. L. Ma. Machine learning to boost the next generation of visualization technology. *IEEE Computer Graphics and Applications*, 27(5):6–9, 2007.

[Mam89] A. Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Computer Graphics and Applications*, 9:43–55, 1989.

[Mau00] S. Mauch. A fast algorithm for computing the closest point and distance transform. Technical report, CalTech, 2000.

[MdF05] P. A. V. Miranda, R. da S. Torres, and A. X. Falcao. TSD: a shape descriptor based on a distribution of tensor scale local orientation. In *SIBGRAPI, Proceedings*, pages 139–146, 2005.

[MIA+03] T. Masuda, S. Imazu, S. Auethavekiat, T. Furuya, K. Kawakami, and K. Ikeuchi. Shape difference visualization for ancient bronze mirrors through 3D range images. *Journal of Visualization and Computer Animation*, 14:183–196, 2003.

[MLD05] A. McKenzie, S. Lombeyda, and M. Desbrun. Vector field analysis and visualization through variational clustering. In *Proceedings, Eurographics / IEEE-VGTC Symposium on Visualization*, volume 2005, pages 29–35, 2005.

[MSC+99] F. M. McQueen, N. Stewart, J. Crabbe, E. Robinson, S. Yeoman, P. L. J. Tan, and L. McLean. Magnetic resonance imaging of the wrist in early rheumatoid arthritis reveals progression of erosions despite clinical improvement. *Ann Rheum Dis*, 58(3):156–163, 1999.

[NKD] M. Nienhaus, F. Kirsch, and booktitle = Computer Graphics, Virtual Reality, Visualization and Interaction in Africa, Proceedings doi = 10.1145/1108590.1108605 pages = 91–98 title = Illustrating design and spatial assembly of interactive CSG year = 2006 Döllner, J.

[ODHW12] S. Oeltze, H. Doleisch, H. Hauser, and G. Weber. Interactive visual analysis of scientific data. Tutorial at the IEEE VisWeek 2012, October 2012.

[Pal99] S. E. Palmer. *Vision science: photons to phenomenology*, volume 4. MIT Press, 1999.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[PHS+08] P. Pieperhoff, L. Hmke, F. Schneider, U. Habel, N. J. Shah, K. Zilles, and K. Amunts. Deformation field morphometry reveals age-related structural differences between the brains of adults up to 51 years. *Journal of Neuroscience*, 28(4):828–842, 2008.

[PNN06] E. Pichon, D. Nain, and M. Niethammer. A laplace equation approach for shape comparison. In *SPIE Medical Imaging, Proceedings*, volume 6141, pages 373–382, 2006.

[PP95] H. G. Pagendarm and F. H. Post. Comparative visualization - approaches and examples. In *Visualization in scientific computing*, pages 95–108. Springer Verlag, Wien, 1995.

[Prc10] V. Prckovska. *High angular resolution diffusion imaging : processing & visualization*. PhD thesis, Eindhoven University of Technology, 2010.

[PVH+03] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: feature extraction and tracking. *Computer Graphics Forum*, 22(4):775–792, 2003.

[RCA+06] D. Rueckert, R. Chandrashekara, P. Aljabar, K. K. Bhatia, J. P. Boardman, L. Srinivasan, M. A. Rutherford, L. E. Dyet, A. D. Edwards, J. V. Hajnal, and R. Mohiaddin. Quantification of growth and motion using non-rigid registration. *Lecture Notes in Computer Science*, 4241:49–60, 2006.

[RGS09] T. Ritschel, T. Grosch, and H. P. Seidel. Approximating dynamic global illumination in image space. In *Proceedings, Symposium on Interactive 3D graphics and games*, pages 75–82, 2009.

[Rhe96] P. Rheingans. Opacity-modulating triangular textures for irregular surfaces. In *IEEE Visualization, Proceedings*, pages 219–225, 1996.

[RLF+04] W. R. Riddle, R. Li, J. M. Fitzpatrick, S. C. DonLevy, B. M. Dawant, and R. R. Price. Characterizing changes in MR images with color-coded Jacobians. *Magnetic Resonance Imaging*, 22(6):769 – 777, 2004.

[Rob07] J. C. Roberts. State of the art: coordinated & multiple views in exploratory visualization. *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization CMV 2007*, 7(Cmv):61–71, 2007.

[RPSH08] T. Ropinski, J. S. Praßni, F. Steinicke, and K. H. Hinrichs. Stroke-based transfer function design. In *IEEE/EG International Symposium on Volume and Point-Based Graphics*, pages 41–48, 2008.

[RSDA02] D. Rey, G. Subsol, H. Delingette, and N. Ayache. Automatic detection and segmentation of evolving processes in 3D medical images: application to multiple sclerosis. *Medical Image Analysis*, 6:163–179, 2002.

[RSH+99] D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes. Nonrigid registration using free-form deformations: application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8):712–721, 1999.

[RVG06] P. Rautek, I. Viola, and M. E. Gröller. Caricaturistic visualization. In *IEEE Visualization, Proceedings*, volume 12, pages 1085–1092, 2006.

[Shn96] B. Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[Sib81] R. Sibson. *A brief description of natural neighbor interpolation*, chapter 2, pages 21–36. John Wiley and Sons, 1981.

[SLM02] A. Stompel, E. B. Lum, and K. L. Ma. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.*, D:394–402, 2002.

[SML04] W. Schroeder, K. Martin, and B. Lorensen. *The visualization toolkit, third edition.* Kitware Inc., 2004.

[SRD+97] G. Subsol, N. Roberts, M. Doran, J. P. Thirion, and G. H. Whitehouse. Automatic analysis of cerebral atrophy. *Magnetic Resonance Imaging*, 15:917–927, 1997.

[ST90] T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. In *ACM SIGGRAPH, Proceedings*, pages 197–206, 1990.

[STR03] H. P. Seidel, H. Theisel, and C. Rössl. Using feature flow fields for topological comparison of vector fields. In *Proceedings, Vision, Modeling and Visualization*, pages 521–528, 2003.

[SW06] H. W. Shen and J. Woodring. Multi-variate, time varying, and comparative visualization with contextual cues. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):909–916, 2006.

[THM+10] P. Tseng, T. Y. Hsu, N. G. Muggleton, O. J. L. Tzeng, D. L. Hung, and C. H. Juan. Posterior parietal cortex mediates encoding and maintenance processes in change blindness. *Neuropsychologia*, 48(4):1063–70, March 2010.

[TMA01] M. Tory, T. Möller, and M. S. Atkins. Visualization of time-varying MRI data for MS lesion analysis. In *SPIE Medical Imaging, Proceedings*, volume 4319, pages 590–598, 2001.

[Tuf90] E. R. Tufte. *Envisioning information.* Graphics Press, 1990.

[TWK02] M. Tittgemeyer, G. Wollny, and F. Kruggel. Visualising deformation fields computed by non-linear image registration. *Computing and Visualization in Science*, 5(1):45–51, 2002.

[Vel01] R. C. Veltkamp. Shape matching: similarity measures and algorithms. In *IEEE Shape Modeling and Applications, Proceedings*, pages 188–197, 2001.

[vHW+06] R. A. van Den Berg, H. C. J. Hoefsloot, J. A. Westerhuis, M. J. van Der Werf, and A. K. Smilde. Centering, scaling, and transformations: improving the biological information content of metabolomics data. *BMC genomics*, 7:142, 2006.

[VKG04] I. Viola, A. Kanitsar, and M. E. Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization 2004*, pages 139–145, 2004.

[VP04] V. Verma and A. Pang. Comparative flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, pages 609–624, 2004.

[vPGL+14] R. van Pelt, R. Gasteiger, K. Lawonn, M. Meuschke, and B. Preim. Comparative blood flow visualization for cerebral aneurysm treatment assessment. *Computer Graphics Forum*, 33(3):131–140, 2014.

[vW91] J. J. van Wijk. Spot noise texture synthesis for data visualization. *SIGGRAPH Computer Graphics*, 25(4):309–318, July 1991.

[vWP94] T. van Walsum and F. H. Post. Selective visualization of vector fields. *Computer Graphics Forum*, 13(3):339–347, 1994.

[WAA+05] D. F. Wiley, N. Amenta, D. A. Alcantara, D. Ghosh, Y. J. Kil, E. Delson, W. Harcourt-Smith, F. J. Rohlf, K. St. John, and B. Hamann. Evolutionary morphing. In *IEEE Visualization, Proceedings*, pages 431–438. IEEE, 2005.

[WB97] P. C. Wong and R. D. Bergeron. 30 years of multidimensional multivariate visualization. In *Scientific visualization, overviews, methodologies, and techniques*, pages 3–33. IEEE Computer Society, 1997.

[WB08] C. F. Westin and D. C. Banks. *Global illumination of white matter fibers from DT-MRI data*, chapter 10, pages 173–184. Mathematics and Visualization. Springer Berlin Heidelberg, 2008.

[WBO97] D. L. Wilson, A. J. Baddeley, and R. A. Owens. A new metric for grey-scale image comparison. *International Journal of Computer Vision*, 24:5–17, 1997.

[Wei06] C. Weigle. *Displays for exploration and comparison of nested or intersecting surfaces*. PhD thesis, University of North Carolina at Chapel Hill, 2006.

[WH91] J. Wejchert and D. Haumann. Animation aerodynamics. In *SIGGRAPH, Proceedings*, pages 19–22, 1991.

[Wie96] T. F. Wiegand. Interactive rendering of CSG models. *Computer Graphics Forum*, 15:249–261, 1996.

[Wil78] L. Williams. Casting curved shadows on curved surfaces. In *Computer graphics and interactive techniques*, pages 270–274, 1978.

[Wit83] A. P. Witkin. Scale-space filtering. In *International Joint Conference of Artificial Intelligence, Proceedings*, pages 1019–1021, 1983.

[WT05] C. Weigle and R. M. Taylor II. Visualizing intersecting surfaces with nested-surface techniques. In *IEEE Visualization, Proceedings*, pages 503–510, 2005.

Summary

Visualization of Variation and Variability

As datasets grow in size and complexity, the importance of comparison as a tool for analysis is growing. We define comparison as the act of analyzing variation or variability based on two or more specific instances of the data.

This thesis explores a number of cases spread across the range of comparisons, from variability within one entity through variability between two or more entities to variability within a population. For each of these we present an exploration tool, combining interaction with high-performance visualization rendering techniques, with the aim of providing more insight into a given dataset.

We explore how different aspects of an application designed for interactive visual analysis can aid the user. This concerns both their initial exploration of a new dataset, as well as their ability to drill down into their discoveries and investigating the underlying details. For instance, multiple linked views can be used to combine highly abstract general-purpose views with highly problem-domain specific views in order to allow a user to translate abstract discoveries into the specific concepts used in their profession. Interactive composition can be applied to quickly focus on areas of interest, suppressing details which may not be relevant at the moment.

Samenvatting

Visualisatie van Variatie en Variabiliteit

Gegevensbronnen groeien voortdurend in omvang en complexiteit. Hierdoor wordt het gebruik van vergelijkingen als gereedschap voor het analyseren van dergelijke gegevens steeds belangrijker. Onder vergelijken verstaan in deze we het analyseren van variatie en variabiliteit, gebaseerd op twee of meer specifieke instanties van de data.

Dit proefschrift beschrijft een aantal voorbeelden verspreid over het gebied van mogelijke vergelijkingen, van variabiliteit binnen een enkele entiteit tot variaties tussen entiteiten tot de onderliggende variabiliteit in een populatie. In elk van deze presenteren we een applicatie voor het verkennen van datasets, gebruik makend van interactiviteit in combinatie met snelle rendering technieken voor visualisatie, met het doel om inzicht te bieden in een gegeven dataset.

We onderzoeken hoe verschillende aspecten van een dergelijke applicatie, ontworpen voor interactieve visuele analyse, de gebruiker kunnen helpen. Dit betreft zowel de initiele verkenning van een nieuwe dataset als de mogelijkheid om dieper te graven in de onderliggende details.Het gebruik van meerdere sterk gekoppelde weergaves, bijvoorbeeld, kan gebruikt worden om abstracte, generieke visualisaties te combineren met domeinspecifieke weergaves. Een effectieve koppeling staat de gebruiker toe om abstracte inzichten eenvoudig te vertalen naar voor hen bekendere concepten. Een ander voorbeeld is interactieve compositie, wat een gebruiker in staat stelt om snel onbelangrijke details te onderdrukken.

Curriculum Vitae

I was born on May 11, 1983 in Vlissingen, the Netherlands. I completed secondary school (VWO) there in 2001 at SSG Scheldemond. After this, I went to study Computer Science & Engineering at the Eindhoven University of Technology, with a combined first year in Mathematics. I received my Master's degree in 2006 cum laude, with a thesis on interactive illustrative medical visualization titled "VolumeFlies - a smart-particle-inspired framework for illustrative volume rendering."

At this time, a collaborative project was starting between the Eindhoven and Delft universities on the visualization of medical multi-field data. This project involved three PhD students: Jorik Blaas, Vesna Prckovska and myself. The research described in this thesis was performed at the Delft University of Technology between 2006 and 2010, under the guidance of Frits Post and Charl Botha.

In 2010 I started work at Liones (now FontoXML) in Rijswijk, the Netherlands, where I am currently lead developer working on user-friendly, web-based editing software for arbitrary, XML-based structured documents.

ISBN 978-94-6186-404-8