Interactive Integrated Exploration and Management of Visualization Parameters

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

Peter Mindek
Matrikelnummer 1129492

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Prof. Dipl.-Ing. Dr.techn. Stefan Bruckner

Diese Dissertation haben begutachtet:

(Prof. Dipl.-Ing. Dr.techn. Stefan Bruckner)

(Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller)

Wien, 16.4.2015

(Peter Mindek)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Interactive Integrated Exploration and Management of Visualization Parameters

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Peter Mindek
Registration Number 1129492

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dipl.-Ing. Dr.techn. Stefan Bruckner

The dissertation has been reviewed by:

(Prof. Dipl.-Ing. Dr.techn. Stefan Bruckner)	(Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller)

Wien, 16.4.2015

(Peter Mindek)

Erklärung zur Verfassung der Arbeit

Peter Mindek
Kohlgasse 24, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasserin)

Abstract

Visualization algorithms are parameterized to offer universality in terms of handling various data types, showing different aspects of the visualized data, or producing results useful for domain experts from different fields. Hence, input parameters are an important aspect of the visualization process. Their exploration and management are tasks which enable the visualization reusability, portability, and interdisciplinary communication.

With increasing availability of visualization systems, which are suitable for a great variety of tasks, their complexity increases as well. This usually involves many input parameters necessary for the meaningful visualization of data. Multiple input parameters form parameter spaces which are too large to be explored by brute-force. Knowing the properties of a parameter space is often beneficial for improving data visualization. Therefore, it is important for domain experts utilizing data visualization to have tools for automatic parameter specification and for aiding the manual parameter setting.

In this thesis, we review existing approaches for parameter-space visualization, exploration, and management. These approaches are used with a great variety of underlying algorithms. We focus on their applicability to visualization algorithms. We propose three methods solving specific problems arising from the fact that the output of a visualization algorithm is an image, which is challenging to process automatically and often needs to be analyzed by a human.

First, we propose a method for the exploration of parameter-spaces of visualization algorithms. The method is used to understand effects of combinations of parameters and parts of the internal structure of the visualization algorithms on the final image result. The exploration is carried out by specifying semantics for localized parts of the visualization images in the form of positive and negative examples influenced by a set of input parameters or parts of the visualization algorithm itself. After specifying the localized semantics, global effects of the specified components of the visualization algorithm can be observed. The method itself is independent from the underlying algorithm.

Subsequently, we present a method for managing image-space selections in visualizations and automatically link them with the context in which they were created. The context is described by the values of the visualization parameters influencing the output image. The method contains a mechanism for linking additional views to the selections, allowing the user an effective management of the visualization parameters whose effects are localized to certain areas of the visualizations. We present various applications for the method, as well as an implementation in the form of a library, which is ready to be used in existing visualization systems.

Our third method is designed to integrate dynamic parameters stored during a multiplayer video game session by the individual participating players. For each player, the changing parameter values of the game describe their view of the gameplay. Integrating these multiple views into a single continuous visual narrative provides means for effective summarization of gameplays, useful for entertainment, or even gameplay analysis purposes by semi-professional or professional players. We demonstrate the utility of our approach on an existing video game by producing a gameplay summary of a multiplayer game session. The proposed method opens possibilities for further research in the areas of storytelling, or at a more abstract level, parameter integration for visual computing algorithms.

Kurzfassung

Um Visualisierungsalgorithmen auf unterschiedliche Datentypen anwenden zu können, um verschiedene Aspekte der darzustellenden Daten in Betracht ziehen zu können, und um, nicht zuletzt, geeignete Ergebnisse für Experten aus unterschiedlichen Bereichen liefern zu können, verwenden gängige Visualisierungsalgorithmen eine Vielzahl von unterschiedlichen Parametern. Diese Parameter stellen somit einen wichtigen Aspekt von Visualisierungsalgorithmen dar. Die Wiederverwendbarkeit und die Portierbarkeit der Visualisierungsalgorithmen, sowie die Kommunikation der Ergebnisse, basiert nicht zuletzt sehr stark auf dem Verständnis der Funktion, und der richtigen Handhabung dieser Parameter.

Mit der stetig wachsenden Verfügbarkeit und Anwendung von Visualisierungssystemen in verschiedenen Bereichen, erhöht sich auch die Komplexität dieser Systeme. Um eine aussagekräftige Visualisierung von Daten zu ermöglichen, wird gewöhnlich eine Vielzahl an Parameterkombinationen benötigt. Diese Parameter sind in Parameterräumen definiert, welche für herkömmliche und naive Untersuchungsmethoden zu komplex und zu groÃŸ sind. Kennt man allerdings die Eigenschaften solcher Parameterräume, kann die Visualisierung der Daten entsprechend angepasst werden. Daher ist es für Experten, die sich für ihre Arbeit Visualisierungssystemen bedienen, oftmals nützlich, Werkzeuge zur Verfügung zu haben, welche automatisch eine geeignete Auswahl an Parametern vorschlagen und eine manuelle Verfeinerung dieser zulassen.

In dieser Arbeit werden verschiedene Ansätze zur Visualisierung, Untersuchung und Handhabung von Parameterräumen vorgestellt. Diese Techniken können auf unterschiedliche Typen von Algorithmen angewandt werden, wobei sich diese Arbeit im Speziellen auf Visualisierungsalgorithmen fokussiert. Da das Resultat solcher Visualisierungsalgorithmen Bilder sind, deren Semantik sich nicht trivial quantifizieren lässt, werden in dieser Arbeit drei Methoden für die Lösung von drei exemplarischen Problemen vorgestellt.

Der erste Teil der Arbeit beschreibt eine Technik, die sich der Untersuchung von Parameterräumen von Visualisierungsalgorithmen widmet. Ziel dieser Technik ist es einerseits, Auswirkungen verschiedener Parameterkombinationen, und, andererseits, Effekte von interne Befehlen der Visualisierungsalgorithmen auf das Ergebnis besser verstehen zu können. Die Untersuchung der Effekte auf das Ergebnis erfolgt durch die Untersuchung lokaler Regionen in den Daten. Für diese Regionen können positive und negative Beispielen definiert werden. Diese Positiv- bzw. Negativbeispiele hängen jeweils von bestimmten Parameterwerten, oder von bestimmten Eigenschaften des Visualisierungsalgorithmus ab. Nachdem einige dieser lokalen Beispiele bestimmt wurden, können die globale Effekte der Parameterwerte bzw. der Teile des Visualisierungsalgorithmus beobachtet werden. Diese Technik ist dabei unabhängig von dem jeweils verwendeten

Visualisierungsalgorithmus.

Im zweiten Teil der Arbeit wird eine Technik zur Handhabung von Selektionen im Bildraum von Visualisierungen, und zur automatischen Verknüpfung dieser mit dem erzeugten Kontext vorgestellt. Der Kontext wird dabei definiert als die Menge aller Werte der verwendeten Parameter des Visualisierungsalgorithmus, welche das Ausgabebild beeinflussen. Die vorgestellte Technik beinhaltet auÃŸerdem Möglichkeiten zur Einbindung von zusätzlichen Anzeigefenstern in die Auswahl, um jene Parameter genauer untersuchen zu können, die nur Auswirkungen auf lokale Regionen in der Visualisierung haben. Die Effektivität der Technik wird anhand verschiedene Anwendungsfälle demonstriert. Weiters wurde im Zuge dieser Arbeit eine Implementierung der Technik als Bibiliothek zur Verfügung gestellt, welche in bestehende Visualisierungssysteme eingebunden werden kann.

Der dritte Teil der Arbeit beschreibt eine Technik zur Integration von dynamischen Parametern, welche während einer Computerspielveranstaltung mit mehreren Spielern individuell für jeden Spieler erfasst werden. Für jeden Spieler beschreiben die veränderten Parameterwerte des Spiels dessen subjektive bzw. persönliche Sicht des Spielverlaufs. Ein Zustand des gesamten Spiels wird dabei durch die Zusammenfassung aller verschiedenen Ansichten in eine einzelne, gemeinsame, visuelle Geschichte beschrieben. Die Anwendung eines solchen Gesamtüberblicks über ein Spiel wird in dieser Arbeit anhand der Zusammenfassung einer Spielrunde eines Computerspiels mit mehreren Spielern beschrieben. Andere Anwendungsfälle für diese Technik finden sich im Unterhaltungsbereich, oder in der Analyse von Spielen für semi-professionelle oder professionelle SpielerInnen. Weiters bietet die Technik Möglichkeiten für die Anwednung im Gebiet der visuellen Geschichtenerzählung, oder, auf einer weitaus abstrakteren Ebene, für die Parameterintegration in Visualisierungsalgorithmen.

Contents

1 Introduction **1**
1.1 Visualization Pipeline . . . 1
1.2 Visualization Parameters and Parameter Spaces . . . 2
1.3 Visualization of Parameter Spaces . . . 7
1.4 State of the Art . . . 12
1.4.1 Parameter-space visualization and analysis . . . 13
1.4.2 Parameter management and provenance . . . 14
1.4.3 Parameter integration and storytelling . . . 15
1.5 Overview of the thesis . . . 17
1.6 Methodology and contributions . . . 20

2 Visual Parameter-Space Exploration **23**
2.1 Introduction . . . 24
2.2 Related Work . . . 25
2.3 Visual Shader-Space Exploration . . . 26
2.4 Feature Vectors . . . 28
2.5 Shader-Space Exploration . . . 29
2.5.1 PEL - Parameter exploration language . . . 31
2.5.2 FVE - Feature-vector editor . . . 32
2.5.3 Visualization of Parameter Spaces . . . 34
2.6 Implementation . . . 36
2.7 Use cases . . . 36
2.7.1 Volume Rendering . . . 36
2.7.2 Image processing . . . 38
2.8 Discussion and Limitations . . . 39
2.9 Conclusion . . . 39

3 Management of Localized Parameters for Data Subsets **41**
3.1 Introduction . . . 42
3.2 Related Work . . . 43
3.3 Overview of Contextual Snapshots . . . 44
3.3.1 Concept of Contextual Snapshots . . . 45
3.3.2 Embedded Visualizations as Linked Views . . . 47
3.4 Contextual-Snapshot Architecture . . . 49

3.4.1 Contextual Snapshots 50
3.4.2 Selections 52
3.4.3 Data transfer 53
3.5 Application Example - Historical Document Analysis 53
3.5.1 Manuscript Visualization 54
3.5.2 Manuscript Visualization Enhancement 54
3.6 Application Example - Heart Visualization 56
3.6.1 Heart Visualization 57
3.6.2 Anatomical Annotations 57
3.6.3 Anchors 58
3.6.4 Selections 58
3.7 Integrating the CSL with Existing Visualization Systems 60
3.8 Discussion 61
3.9 Conclusion 63

4 Integrating Parameter Sets of Multiple Users 65
4.1 Introduction 66
4.2 Related Work 68
4.3 Overview 70
4.3.1 Flock of Cameras 71
4.3.2 Event Graph 71
4.3.3 Camera Operators 73
4.4 ManyCams: Summarizing the Gameplay 75
4.4.1 Story Representation 75
4.4.2 Building the Visual Narrative of the Gameplay 75
4.5 Results 78
4.6 Discussion 80
4.7 Conclusions 80

5 Data-Sensitive Navigation 81
5.1 Introduction 82
5.2 Related Work 83
5.3 Data-Sensitive Navigation 85
5.3.1 Model Overview 85
5.3.2 Data-Sensitive Manipulation 87
5.3.3 Data-Sensitive Guidance 88
5.4 Data-Sensitive Vessel Traversal 89
5.4.1 TreeSlider 91
5.4.2 Data-Sensitive Navigation with the TreeSlider 93
5.4.3 Traversing Vascular Structures 94
5.5 Data-Sensitive 3D Object Rotation 95
5.5.1 Sensitivity of 3D Rotation 96
5.5.2 Data-Sensitive Navigation in 2D 97
5.5.3 Visual Guidance in 3D Object Rotation 98

5.6 Results and Discussion 100
5.7 Conclusions 104

6 Summary **107**

Bibliography **111**

List of Figures

1.1 Generalized visualization pipeline describing an example scenario from a medical domain. 2

1.2 (a) A dataflow network representing a visualization pipeline. The rectangles represent individual modules assembled into the pipeline, while the arrows symbolize the data exchange between the modules. Intermediate visualization products of some of the modules are shown. Note that some of the modules provide their outputs to multiple other modules. (b) The product of the visualization pipeline after its execution. 3

1.3 Parametrized visualization mapping illustrated as a function. Each black dot is a specific set of parameter values (visualization mapping input), which is mapped to a specific visual representation (visualization mapping output). The visualized data remain constant throughout the visualization session. 4

1.4 (a) A simple ramp transfer function based on two numerical parameters (level and window). (b) A two-dimensional transfer function mapping regions of certain ranges of voxel intensities and gradient magnitudes to specific colors. A two-dimensional histogram of the data is shown underneath the areas mapped to the green and the blue color to aid the transfer function design. (c) Using a ramp transfer function, it is usually only possible to highlight structures with value ranges sufficiently different from surrounding voxels. In the depicted MRI data, it is impossible to show the brain using the ramp function, since it has similar intensity values as the skull and skin. (d) Using a 2D transfer function, various internal structures can be visualized. In this case, the brain can be revealed. 5

1.5 (a) A visualization of the parameter space is shown, where two parameters (two isovalues) are represented by the two axes of the plot. The algorithm calculates the similarity of the two respective isosurfaces. The similarity is encoded by the color of the pixels of the plot (black means high similarity, white means low similarity). (b) A volume dataset modelled with the representative isosurfaces, whose isovalues are marked in the plot. (c) Individual representative isosurfaces. Image courtesy of Bruckner and Möller [19]. 8

1.6 (a) Samples of a four-dimensional parameter space visualized through a scatterplot matrix. The scatterplot matrix shows relationships between each pair of the parameters. The individual parameters (A, B, C, D) are color coded. (b) Samples of a four-dimensional parameter space visualized through parallel coordinates. Each polyline represents one point in the parameter space. The parallel coordinates show sampling patterns of the multidimensional parameter space. 9

1.7 (a) The Mandelbrot set as a visualization of the parameter space of the Julia set. Four points in the complex plane are marked with red circles, and the corresponding Julia sets for these points are shown in blue color. (b) Different visualization of the same parameter space. An estimation of the area of the Julia set by counting pixels is used to visualize the parameter space. (c) Small multiples are used to visualize the parameter space of the Julia set. The parameter space represented by the image space is sampled on a regular grid. Each of the small multiples depicting one Julia set is placed at the position of the respective sample. 11

1.8 Three levels of complexity of parameter-space exploration and management. (a) Parameters of a single visualization. (b) Multiple sets of parameters influencing localized parts of the visualization. (c) Integration of multiple sets of parameters of multiple users into a single visualization. 18

2.1 Overview of the visual shader-space exploration (VSSE). 27

2.2 Visualization of a two dimensional case of a membership function for (a) $p = 1$ and (b) $p = 5$. Black color means association with the negative class, white color means association with the positive class. Green circles denote positive feature vectors, red circles denote negative ones. 30

2.3 An example demonstrating the usage of VSSE with a simple shader. In (a) no pixels are classified, opacity is 0.5. In (b) a positive pixel is selected (green circle), opacity is 1 for the whole image. In (c) additionally a negative pixel (red circle) is selected. 31

2.4 Feature vector editor (FVE). 33

2.5 A scatterplot matrix displaying feature vectors. Positive vectors are marked with green circles, negative ones with red circles. (a) shows a magnetic resonance angiography dataset of a human brain, where a pixel displaying a vessel was classified as positive (green circle). In (b), a pixel inside the brain matter is classified as negative (red circle), which results in an enhanced view of the vessels. VSSE confirmed that the explored parameters can effectively classify blood vessels in the brain. . . . 35

2.6 Visualizations of the dual-modality data with positively classified brain and negatively classified occluding tissues for different feature vectors. Feature vectors consist of these attributes: (a) voxCT; (b) voxMRI; (c) voxCT, voxMRI; (d) gmCT; (e) voxDiff; (f) gmCT, voxDiff; (g) gmDiff, voxMax; (h) gmDiff, gmMax; (i) gmMax, voxDiff 37

2.7 Using VSSE for exploring possibilities of variable thresholds of a bilateral filter. In (a) the original image is shown. In (b) and (c) one positive example of an edge (green circle) and one negative example (red circle) was chosen. The thresholds for individual pixels are modified so that the examples are generalized for the whole image. 40

3.1 Interactive anchors (white circles) representing individual contextual snapshots. For better 3D orientation, the anchors are connected with the origin of the coordinate system by a thin line. (a) shows how the thumbnail of an anchor can be displayed. (b) and (c) show the anchors from different camera views. 46

3.2 (a) A selection (marked with the red circle). (b) An integrated view of two variables using the selection after its activation. Histograms are shown for both variables as embedded visualizations. The third embedded visualization is a variable picker. It shows a list of the data variables, where the user can choose which one is displayed. 47

3.3 Overview of the system. The black arrows represent the data flow between the visualization system, the embedded visualizations and the Contextual Snapshot Library (CSL). The gray arrow denotes the transition from the original rendering to the rendering with the enhanced visualization. 48

3.4 Architecture of the visualization system integrating the CSL. The arrows denote the data flow. 49

3.5 Two different selection-mask shaders generate selection masks (black means fully selected, white means not selected at all) for the same selection stroke (in red). (a) and (b) illustrate sequences of creating the stroke, while the selection mask is continuously generated at every step. (c) and (d) illustrate the final selection masks for the given stroke. The gradual change in the level of selection enables smooth brushing. 51

3.6 Two different selection-display shaders. Both shaders are able to show the fuzziness of the selection. 52

3.7 (a) The book reader application displaying the Venetus A manuscript. The icon (violet circle) on the top right corner indicates the availability of UV light data for this particular page. (b) The book reader application enhanced with the CSL to show additional data. An anchor of a contextual snapshot, selections, and embedded visualizations are displayed. (c) The transcript of the displayed pages with the tag cloud in the maximized view. A word selected in the tag cloud is highlighted in yellow. (d) Histogram of a selected part of a natural light (repink) and UV light (dark blue) photograph shown in the maximized view. The histograms are shown in the same view so that they can be easily compared. The histogram is rendered as a JavaScript enabled web page utilizing the D3 library [17]. 55

3.8 (a) A 3D model of a human heart displayed in VolumeShop. (b) The model is displayed with a lower opacity to reveal the internal structures. 57

3.9 (a) The user creates a circular selection and provides a caption for it. (b) If the selection is activated, an embedded visualization of the selected object in the context of the original model is shown with the specified caption at the bottom. (c) Multiple selections in the same context (viewpoint and opacity value) can be created. All of their embedded visualizations are shown at once in the sliding bar. The one in the middle is highlighted, as well as its associated selection (bold yellow circle). (d) The highlighted embedded visualization is shown enlarged (maximized) over the original visualization for a better view. The embedded visualizations can be browsed in the sliding bar or in the maximized view. 59

3.10 Custom usage of graphical elements of the CSL in the heart example: (a) Two groups of otherwise overlying anchors are displayed in a circular layout around the groups' centers. (b) Hovering the mouse pointer over an anchor shows the selections from the respective contextual snapshots in an integrated view. 60

3.11 Different types of selections available in the heart application: circular (1), rectangular (2), free-hand lasso (3) selections. 61

3.12 An example application implementing contextual snapshots using the CSL. (a) A simple rendering of a dynamic object modelled with a distance field is shown. (b) The rendering is enhanced with interactive selections. The rendering style inside the selections is different from the rendering style in the rest of the image. 62

4.1 Overview of the ManyCams, the flock of cameras, and the data on which they operate. Camera paths, camera importance, and the events of the gameplay are extracted from the game and stored in the flock of cameras and the event graph. These structures are then processed by our ManyCams method to produce the video summarizing the gameplay. 70

4.2 An example of an event graph. The circles indicate individual events, the solid arrows are causal links extracted from the game, and the dashed arrows are player-links derived from the flock of cameras. 72

4.3 Some of the pictograms used by the overview operator for a schematic display of the gameplay. 74

4.4 (a) Five cameras in a scene. The gray one is an overview camera. (b) A visibility graph. (c) Switching view from the green camera to the red one would cause a collision with a wall. (d) The view from the green camera can be switched to the view from the red camera through the blue camera, because such a path exists in the visibility graph. (e) The views from any pair of cameras can be switched through the overview camera (as indicated by the gray arrows in (b)). 76

4.5 A summary video of a gameplay of Jake 2 generated by ManyCams. (a) Overview operator showing the movements of two groups of players from the green team, trying to surround players of the red team. (b, c) View-switch operator continuously changing view towards an event which has already occured. To achieve this, time is reversed, which is communicated through a distinct visual style applied to the entire screen. (d) The mark-player operator (red circle in the middle) indicating a player of the opposing team being shot. Description of the event is shown at the top. 2D inset in the left bottom corner shows the view of the dead player. The surrounding is darkened to guide the viewer's attention towards the event. 79

5.1 White arrows represent changes in input space, parameter space, or the output visualization. (a) User input is directly mapped to the input parameters. The changes of the input are not proportional to the changes of the output. (b) Data-sensitive manipulation, where the underlying data or the output are used to dynamically scale the changes in the parameter space. In this way, changes in the output are proportional to the changes in the input space. The mapping is illustrated with the dashed arrows. (c) The mapping information is displayed in the image space to guide users during the data-sensitive navigation (yellow arrows). 86

5.2 Transformation T of an input to get a parameter value. The red dot indicates the input, e.g., a slider position, the blue dot shows the parameter value. The black dots are the sample positions s_i within the parameter space, while $d'(s_i)$ is the normalized importance $d(s_i)$ of the sample s_i. (a) Transformation T if importance values of all samples are uniform (data-sensitive manipulation is disabled). (b) Transformation T if the samples have non-uniform importance values (data-sensitive manipulation is enabled). 87

5.3 Data-sensitive navigation applied to a simple slider for volume slicing of a CTA scan of an aneurysm (marked with the red circle). (a) The red line represents a slider. The visual guidance information shown to enhance the usability of the data-sensitive slider is depicted in blue (global encoding) and green (local encoding) color. (b) A volume rendering of the entire dataset for overview. 90

5.4 Different states of a TreeSlider are shonw in (a) to (d). In (e) to (h), the traversed tree is shown. Red color indicates the active path, the green circle represents a branching node, while the black circle represents the current position set by the TreeSlider. (a) TreeSlider and its components (b) If the handle is moved close to a point where the tree branches, a fork continuously appears. (c) If the handle is within the fork area, it can be moved vertically to choose other branches of the tree for traversal. (d) The traversal continues on the chosen path. 91

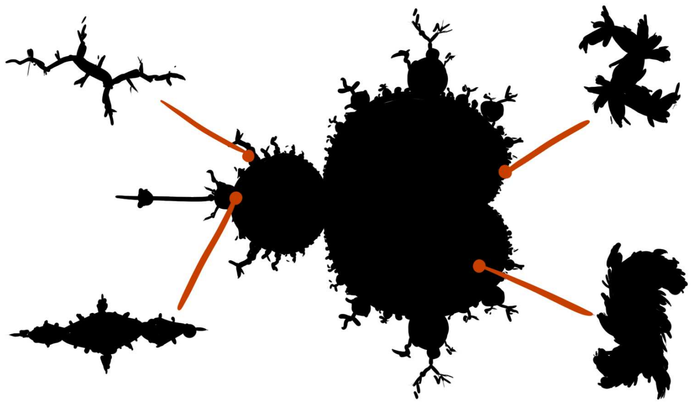
5.5 A directed acyclic graph with two paths (marked blue and red). The relative positions of the fork node b on these two paths are not equal. Therefore, if the user changes between the paths, the position of the fork node b has to be remapped so that the respective branching indicator does not move during the switching of the paths. 92

5.6 TreeSlider with guidance. The red arrows indicate how will the sensitivity of the slider change if the handle is moved either left or right. A longer and more opaque arrow means higher sensitivity, which indicates a lower importance in this direction. 93
5.7 Traversal of a vessel-tree in a CTA scan using the TreeSlider. (a) Axial slices through the selected point (marked with red circles) and a slice perpendicular to the vessel-tree in the current position are shown above the TreeSlider. (b) The vessel-tree is overlaid on top of a 3D visualization of the dataset as an overview. All views in (a) and (b) are linked together. 94
5.8 Reconstructed importance function for each possible viewpoint (a, b) and for each possible user input (c, d). Red color indicates high importance, black color indicates low importance. The blue dots represent the sampling positions. We employ either nearest neighbor (a, c) or weighted average (b, d) interpolation of the samples. . . . 97
5.9 Mouse input of the Arcball rotation. m_c (blue) is the input vector whose length is limited to the interval $[0, 1]$. $\hat{m}_c$ is the normalized vector m_c. p_c is the current mouse position. In this way, the 2D interaction is in each step reduced to a 1D problem as illustrated in Figure 5.2. $\hat{m}_c$ represents the whole slider and p_c represents the current slider position. The red dots show where $z(m_c)$ is sampled in order to produce the importance function for 1D data-sensitive navigation. 99
5.10 Visualization of the importance using the navigation sphere and vignetting. (a) A feature-rich area is shown without vignetting. The front-facing part of the navigation sphere is red, indicating high importance (b) If a featureless area is displayed, vignetting is introduced to indicate the low importance. It is also visible through the black color of the navigation sphere. 100
5.11 Visual guidance in 3D object rotation. The white dotted line indicates the input vector for the Arcball rotation. (a) Initial position. The red color indicates where to move the mouse to rotate the object to a more interesting viewpoint. (b) The mouse was moved towards the red area to reveal interesting features around the neck and jaw. (c) The mouse was moved towards another red area to show the face. Since a large portion of the image comprises the featureless forehead, this red area is less pronounced. (d) The mouse moves towards an area of low interest denoted by the lack of red color. Featureless occipital and parietal bones are shown. The low importance is visible from the dark navigation sphere, as well as darker borders caused by the vignetting. 101
5.12 (a) Blood vessel with an occlusion. Red circles mark where the occlusion begins and where it ends. (b) The vessel-tree is overlaid on the image showing the calculated importance values. Red color indicates high importance, black color indicates low importance. Both the beginning and the ending of the occlusion were considered to be areas of high importance. 103
5.13 TreeSlider positioned to the border of a stent. 105

Preface

THIS thesis summarizes the work carried out during my employment at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology. As a part of the visualization group, I was lucky enough to experience a friendly and motivating environment, which made my time here very enjoyable as well as unforgettable. I would like to sincerely thank Meister Eduard Gröller, the leader of our group, who was always very helpful in every aspect of my employment. My great thanks goes to Stefan Bruckner, my advisor, without whom this thesis would have never been possible. Finally, I would like to thank all my colleagues and collaborators, including Artem Amirkhanov, Douglas Cedrim, Ladislav Čmolík, Martin Haidacher, Alexey Karimov, Mathieu Le Muzic, Gabriel Mistelbauer, Peter Rautek, Johanna Schmidt, Petr Sikachev, Viktor Vad, Andrej Varchola, Ivan Viola, Nicholas Waldin, Manuela Waldner, and all the others visiting our group. I would like to thank Gabriel and Johanna for helping me with the German version of the abstract. The work presented in this thesis is part of the *ViMaL* project, supported by the Austrian Research Fund (FWF), no. P21695, and the Knowledge Assisted Sparse Interaction for Peripheral CT-Angiography (KASI) project, supported by the Austrian Science Fund (FWF) grant no. TRP 67-N23. The work was additionally supported by the Aktion OE/CZ grant number 64p11 and 68p5.

The Venetus A dataset is courtesy of The Homer Multitext Project. The hurricane Isabel dataset is produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR and the U.S. National Science Foundation (NSF). The model of Vienna is courtesy of Michael Wimmer and Peter Wonka. It is available at `http://www.cg.tuwien.ac.at/research/vr/urbanmodels`.



CHAPTER 1

Introduction

VISUALIZATION is a term describing techniques and concepts for creating images communicating visually-encoded information about data. Such a transformation of non-visual data to visual representations is useful in many fields for various purposes, such as data analysis or presentation. Because of the high diversity of tasks and types of visualized data, various branches of visualization emerged. They specialize on specific data types, such as 3D spatial data, which is of interest for scientific visualization, or abstract data studied by information visualization. Some of the branches are concerned with specific classes of tasks, such as visual analytics, a branch which combines visualization with various sophisticated interaction techniques, allowing the user to analyze complex datasets.

1.1 Visualization Pipeline

The common concept for all branches of visualization is a so-called *visualization pipeline* [73]. The visualization pipeline annotates individual steps which are usually necessary to perform in order to produce a visualization of a dataset. The precise structure of the visualization pipeline varies between applications. However, the visualization pipeline could be generalized in a following way: First, raw data are *acquired* and *pre-processed* in order to prepare them for the visualization. This step might include noise reduction, derivation of additional data attributes, and similar operations. For instance, this step might represent scanning of a person in a medical scanner, such as CT or MRI, or performing another type of measurement of the physical world. Afterwards, the data are *filtered*, which means that a specific data subset is selected to be presented to the user, such as only the relevant parts of the medical scan. In this step, focus and context [51] within the dataset can be specified. Subsequently the pre-processed, filtered data is mapped to a visual representation. This step is usually referred to as *visualization mapping*. It defines a way how the data are displayed so that the visualization is comprehensible for the user. The final step, *rendering*, displays the visual representation created in the previous step on the screen. Figure 1.1 illustrates the visualization pipeline in a schematic way.

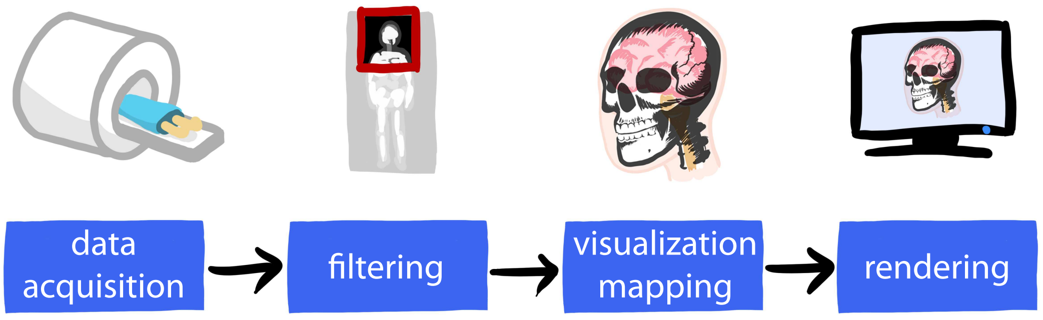


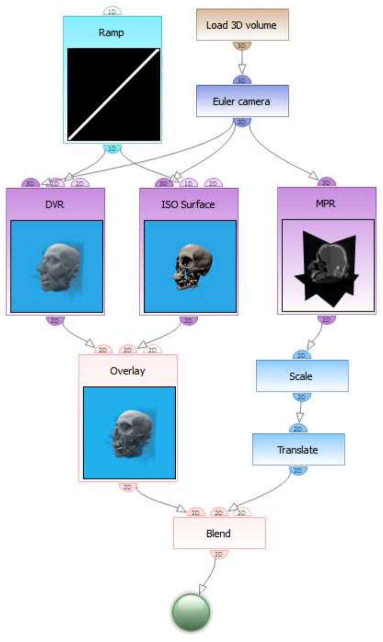
Figure 1.1: Generalized visualization pipeline describing an example scenario from a medical domain.

In real-world scenarios, the visualization pipeline is usually split into many interconnected modules which exchange data in order to produce the final visualization result. Therefore, each step of the abstract visualization pipeline introduced above is in fact represented by multiple independent modules. The execution of the individual modules might be distributed between clients or processors, allowing a parallel execution of those modules which do not depend on each other's intermediate results. Although some of the visualization systems represent the visualization pipeline as a linear sequence of steps, it is common to specify arbitrary data-exchange links between the modules, forming a *dataflow network*. The dataflow network is a directed acyclic graph where the edges represent the exchange of the intermediate visualization products between individual processing modules. Figure 1.2 shows an example of a visualization pipeline represented by a dataflow network.

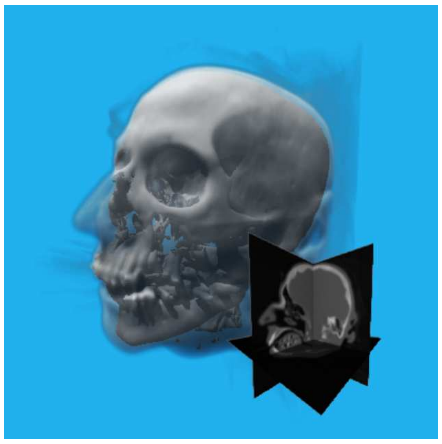
1.2 Visualization Parameters and Parameter Spaces

Visualization methods are tailored for different use cases. From the user's perspective, the degree of complexity of these methods greatly depends on the tasks on which they focus. Specialized, single-purpose visualization algorithms tend to be among the simplest from the user's point of view. However, such methods have very limited abilities in terms of diversity of the input data that they are able to handle, and the variety of output visualizations.

Usually, with increasing generality of the visualization methods, their complexity increases as well. The individual steps of the visualization pipeline need to be parametrized so that the variety of the visualization outputs may be possible. The parametrization allows the user to perform interactive data exploration, since different aspects of the visualization can be modified through the parameters which results in displaying data subsets according to the user's needs. For instance, the filtering step of a volume rendering algorithm can be parametrized so that the user is able to select an arbitrary clipping box. Only the voxels inside the specified clipping box are used in the following steps of the visualization pipeline. This parametrization of the filtering step requires the user to specify the area of interest. However, it allows to examine different parts



(a)



(b)

Figure 1.2: (a) A dataflow network representing a visualization pipeline. The rectangles represent individual modules assembled into the pipeline, while the arrows symbolize the data exchange between the modules. Intermediate visualization products of some of the modules are shown. Note that some of the modules provide their outputs to multiple other modules. (b) The product of the visualization pipeline after its execution.

of the volume by removing surrounding voxels occluding the area of interest, thus constituting a more powerful visualization algorithm.

Although the concepts presented in this thesis could be applied to any step of the visualization pipeline, we are in particular concerned with the visualization mapping. In this step, the visual representations, as to be perceived by humans, are constructed. The visualization mapping can be thought of as a function whose input consists of the examined data and the parameters, while its output is an image visually representing the data. Both the data and the parameters can change during the visualization session, in turn changing the output visualization image. To distinguish between the data exploration and the exploration of the parameter space, let us consider the segment of the visualization session during which the data remain constant. Within this context, the visualization mapping is a function whose domain is a space of possible parameter settings, and its co-domain is a space of possible visualizations of the data. The explored data is in fact a parameter of said function. A schematic representation of the visualization mapping as a function is depicted in Figure 1.3.

The domain of this function if the space of all possible parameter vectors, or the *parameter space*. The co-domain of the function is the space of all possible visualizations, or the *visualization space*. The task of producing a meaningful visualization of a dataset requires that a suitable sample of the parameter space is chosen as an input for the function representing the visualization mapping. The chosen parameter vector is mapped to a point in the visualization space, which is the desired visualization image.

The parameter space is abstract. It does not encode any relevant information by itself. On the other hand, the visualization space consists of concrete samples, each of which could be visually examined by the user of the visualization algorithm, and classified as either informative or not. It is difficult to mentally reconstruct the mapping from the abstract parameter space to the concrete visualization space. Therefore, without any additional information, the search for the parameter-space samples which produce meaningful visualizations is reduced to mere brute-force approaches.

For the simplest visualization algorithms from the user's perspective, the brute-force approach of choosing input parameters is usually sufficient. As an example, let us assume a rendering algorithm displaying a 3D object (a volume or a mesh), where the user is only allowed to change the camera position and orientation in order to see the object from different viewpoints. Given a simple interaction mechanism which would allow the user to select an adequate camera viewpoint, i.e., sample the parameter space, it would be possible to find a desired visualization quite effectively. An alternative approach would be an optimal viewpoint selection algorithm, which searches the parameter space to optimize the visibility of salient features of the displayed object. Both approaches have their applications. For example, the first approach might be useful for medical students who wish to explore an anatomical model by manually rotating it. The second approach might be employed in the automated generation of previews of multiple datasets, where there is no need for a thorough exploration of each dataset from different viewpoints.

In the example outlined above, parameter-space visualization can be utilized to guide the user in refining the selected viewpoint. Viewpoint selection methods often utilize a spherical

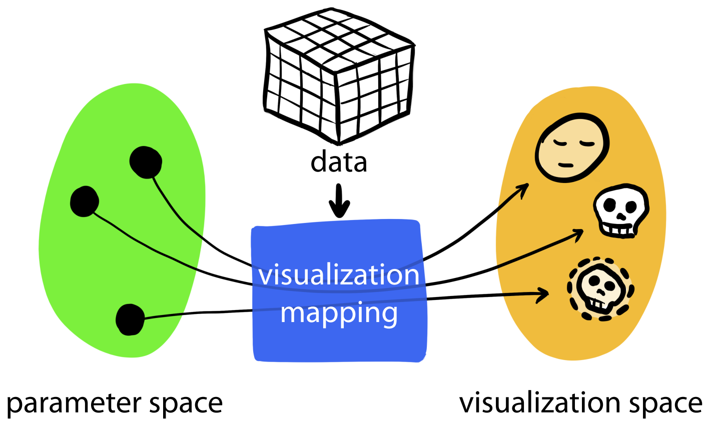
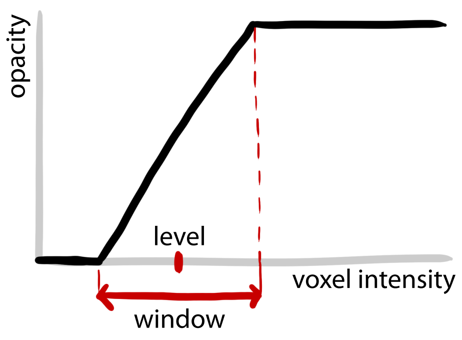
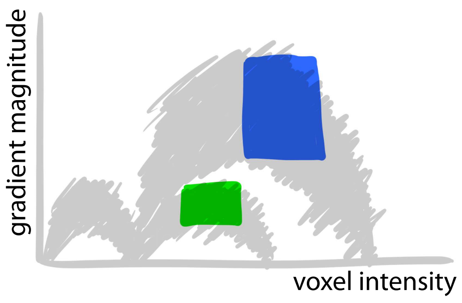


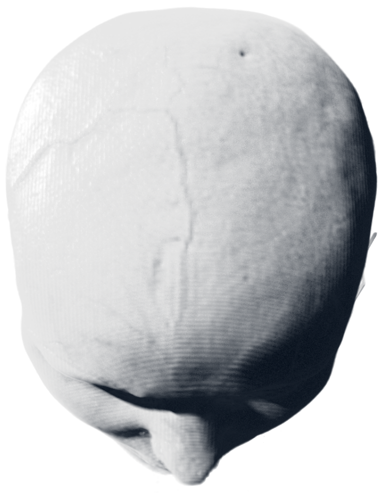
Figure 1.3: Parametrized visualization mapping illustrated as a function. Each black dot is a specific set of parameter values (visualization mapping input), which is mapped to a specific visual representation (visualization mapping output). The visualized data remain constant throughout the visualization session.



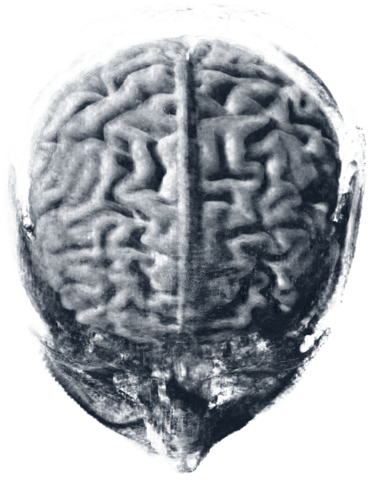
(a)



(b)



(c)



(d)

Figure 1.4: (a) A simple ramp transfer function based on two numerical parameters (level and window). (b) A two-dimensional transfer function mapping regions of certain ranges of voxel intensities and gradient magnitudes to specific colors. A two-dimensional histogram of the data is shown underneath the areas mapped to the green and the blue color to aid the transfer function design. (c) Using a ramp transfer function, it is usually only possible to highlight structures with value ranges sufficiently different from surrounding voxels. In the depicted MRI data, it is impossible to show the brain using the ramp function, since it has similar intensity values as the skull and skin. (d) Using a 2D transfer function, various internal structures can be visualized. In this case, the brain can be revealed.

heat map surrounding the visualized object. Each point on the surface of the sphere represents a viewpoint. The ranking of the viewpoint is color-coded. Such a parameter-space visualization is useful in stability analysis, where the user might rather select a good viewpoint surrounded by a larger area of other good viewpoints than an isolated viewpoint of high ranking.

A more complex scenario arises in the area of volume visualization. In volume visualization, there are usually more input parameters than just the camera viewpoint. Since the actual visualization pipeline can consist of multiple modules, each of them containing a set of its own parameters, all of these parameters need to be considered. Additionally, it is possible that some of the parameters are multidimensional, e.g., transfer functions. In these cases, it is common that additional information about the parameter space is essential for the effective production of visualizations. This information can be communicated through parameter-space visualizations.

Typically, transfer functions are used in volume rendering algorithms [69]. A transfer function maps values of individual data voxels, e.g., measured radiological densities in case of medical CT data, to optical properties, such as colors and opacities. The assigned optical properties are used to display the volume data on the screen. Thus, the transfer function is an input parameter of the visualization mapping.

Transfer functions can be described by a certain number of parameters, which contribute to the parameter space of the visualization mapping. A common aspect of various types of transfer functions is that it typically is not straightforward to predict changes in the rendered images as an effect of transfer-function changes. Therefore, performing a manual, trial and error exploration of the multidimensional parameter-space consisting of the parameters describing the transfer function is a very tedious task. This is especially problematic if the user is not familiar with the explored data.

Since it is not trivial to design a function which effectively captures desired features of the data, transfer functions are in some scenarios reduced to several numerical parameters. In medical visualization, a windowing function is commonly employed when working with volume data [82]. It generates a transfer function based on two numerical parameters, *level* (window center) and *window* (window width). These parameters are used to create a ramp function, which is used as a transfer function. Figure 1.4a illustrates how the windowing function is constructed. The simplicity of the specification of the windowing function, as well as the predictability of its effects, usually allows a brute-force exploration of its two-dimensional parameter space. In the medical domain, it is common that the two parameters describing the windowing function are specified by selecting a point in a 2D rectangular area. However, the windowing function has limited flexibility with respect to the types of features it can depict. It only provides the possibility of modifying brightness and contrast of the displayed data.

On the other hand, certain scenarios require complex transfer functions specified by multiple input parameters in order to efficiently classify the underlying data. An example is a two-dimensional transfer function for volume rendering. Its domain is a two-dimensional space of data attributes, such as voxel value and gradient magnitude. The transfer function maps each point from this domain to a color and an opacity used by the volume rendering algorithm. In this way, it is possible to distinguish between data points which would not be possible to classify with a one-dimensional transfer function, for instance the boundaries and the body of a volumetric object.

Usually, continuous regions of the two-dimensional transfer function's domain are mapped to a certain color and opacity. Baisc geometric shapes, such as triangles or rectangles [62] are commonly used to specify these regions. However, more complex shapes, such as ellipsoidal Gaussian transfer functions proposed by Wang et al. [116], are used as well. Non-binary regions of the transfer function's domain are often utilized to account for uncertainty in the specification of the transfer functions. To guide the user in the process of selecting the desirable regions, a two-dimensional histogram of the data points is often displayed in the transfer function editor. An example of a two-dimensional transfer function with such a histogram is illustrated in Figure 1.4b.

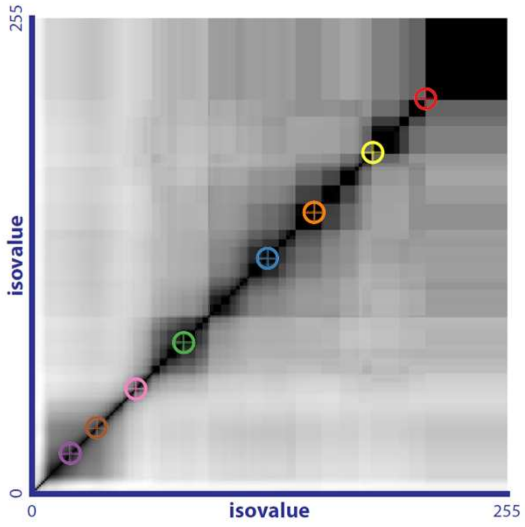
The histogram displayed in the view of the transfer-function editor is a type of parameter-space visualization. It provides information useful for understanding which sections of the parameter space are important for given tasks in the visualization space, e.g., displaying certain structures from volume data.

1.3 Visualization of Parameter Spaces

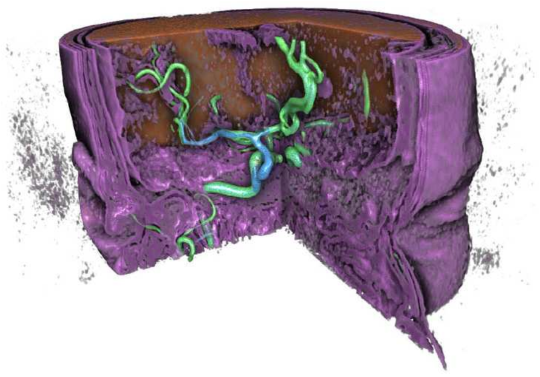
Creating a graphical representation of a parameter space is an essential step of many non-trivial visualization tasks. This meta-visualization allows the user to observe all structures of the parameter space that might be important to understand in order to fulfil a particular task. For instance, by visualizing the parameter space it is possible to perform a stability analysis of parameter settings. Parameter settings residing inside uniform areas within the visual representation of the parameter space are more stable - changing them will not cause tremendous changes to the output of the algorithm. To perform the stability analysis, the parameter-space visualization has to visually encode the changes of the output of the algorithm across the parameter space. Therefore, it is necessary to describe each instance of the algorithm's output, e.g., visualization image, by a single point in the parameter-space visualization.

The parameter-space visualization can be performed with any parametrized algorithm. It is needed for those tasks which require a certain degree of human involvement. If the parameter space cannot be analyzed automatically, the visualization provides means for the user to make the necessary decisions. Oftentimes, the parameter-space visualization is employed in simulations, where only quantitative measures can be derived from the outputs automatically. The visualization of the parameter-space allows the user to manually perform qualitative analyses of the outputs as well. We focus on algorithms whose results are visualizations, or in general, images. The automatic processing of images is very challenging, and often impossible with current approaches. Therefore, effective parameter-space visualization is especially relevant in the context of visualization algorithms.

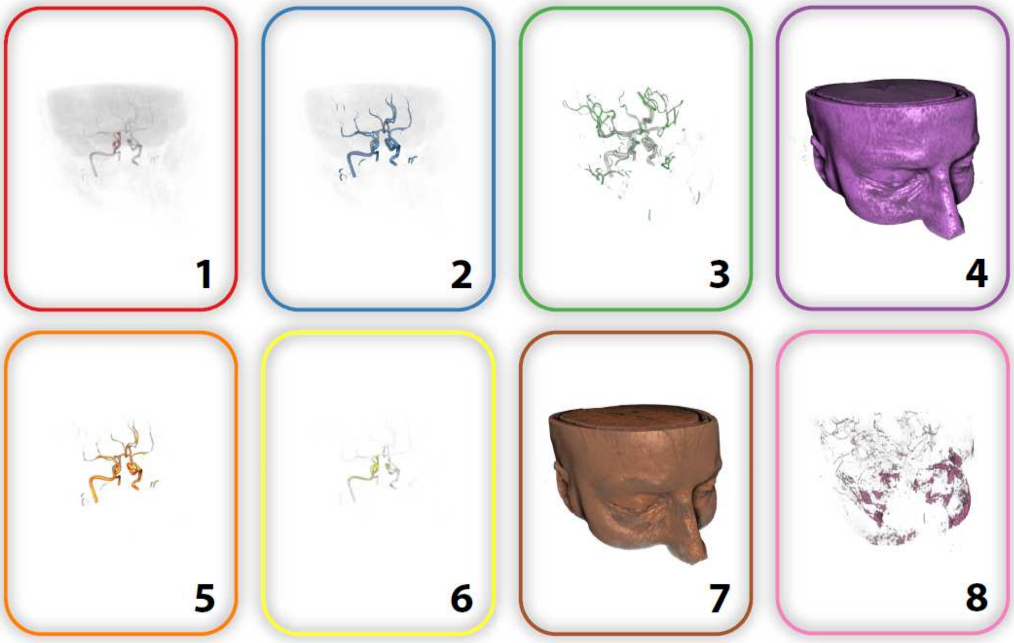
There are two problems which are necessary to address when trying to visualize a parameter space. When the parameter space is multidimensional, it is necessary to reduce its dimensionality in order to display the samples in a 2D image. The second problem is how to visually represent individual samples, e.g., images in case of visualization algorithms. Should the visualization effectively aid the parameter-space analysis, it is necessary to sample the parameter space at an adequate number of samples. Each sample needs to be represented in the parameter-space visualization. In the simplest case, only the presence of the sample needs to be communicated,



(a)

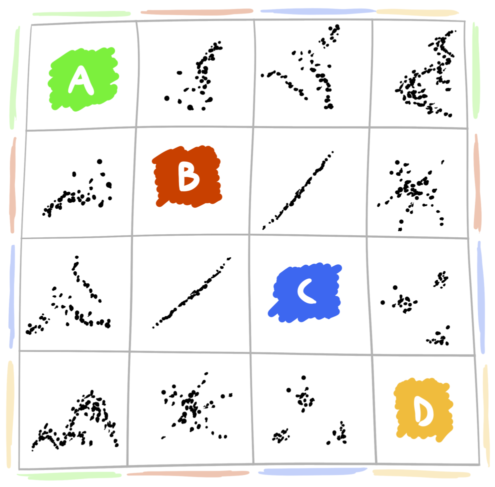


(b)

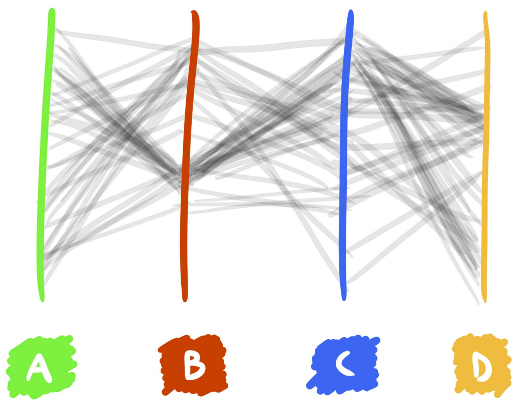


(c)

Figure 1.5: (a) A visualization of the parameter space is shown, where two parameters (two isovalues) are represented by the two axes of the plot. The algorithm calculates the similarity of the two respective isosurfaces. The similarity is encoded by the color of the pixels of the plot (black means high similarity, white means low similarity). (b) A volume dataset modelled with the representative isosurfaces, whose isovalues are marked in the plot. (c) Individual representative isosurfaces. Image courtesy of Bruckner and Möller [19].



(a)



(b)

Figure 1.6: (a) Samples of a four-dimensional parameter space visualized through a scatter-plot matrix. The scatterplot matrix shows relationships between each pair of the parameters. The individual parameters (A, B, C, D) are color coded. (b) Samples of a four-dimensional parameter space visualized through parallel coordinates. Each polyline represents one point in the parameter space. The parallel coordinates show sampling patterns of the multidimensional parameter space.

e.g., when examining the sampling distribution, and very simple visual representations, such as points, are sufficient. In other tasks, each sample might be represented by a thumbnail of the output for the respective parameter settings. Such an approach is employed in *design galleries* [78]. Therefore, the preferred way of visualizing a parameter space depends on its dimensionality as well as on the given task.

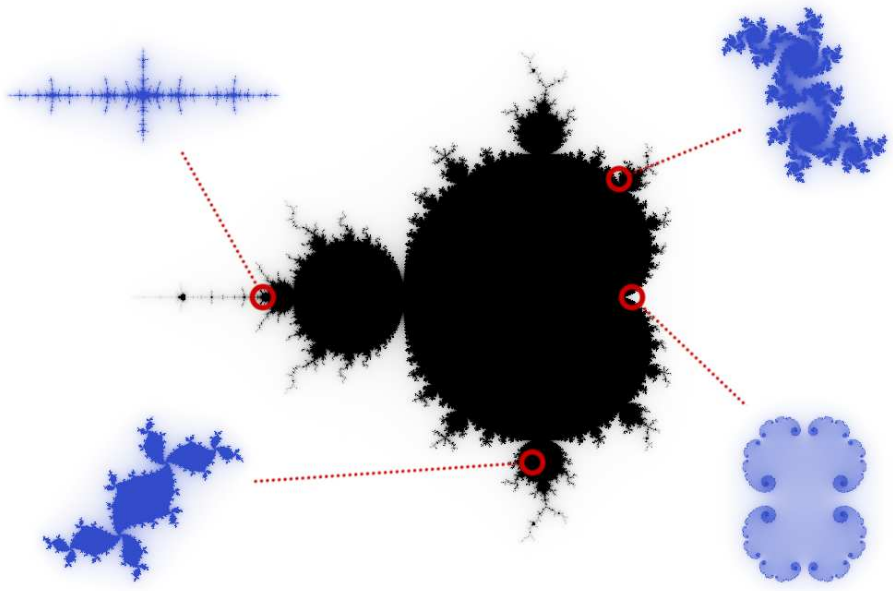
For one-, two-, and potentially three-dimensional parameter spaces, the samples can be represented within histograms or scatterplots. Figure 1.5 shows a visualization of a two-dimensional parameter space of an algorithm evaluating the similarity between different isosurfaces of a volume dataset [19]. In this visualization, each pixel represents one sample of the parameter space and visually encodes the evaluated similarity. When dealing with multi-dimensional parameter spaces, various techniques from the field of information visualization can be employed. Scatterplot matrices can be used to display relationships between all dimension pairs within the parameter space (Figure 1.6a). Another possibility is to use parallel coordinates (Figure 1.6b), where each sample is represented by a single polyline.

The mentioned techniques are able to show patterns in the sampling of the multi-dimensional parameter spaces. However, it is necessary to address the second mentioned problem concerning the representation of the individual samples. If only the quality of the sampling distribution needs to be examined, the samples can be represented in a straightforward way by simple points (in case of scatterplots) or lines (in case of parallel coordinates) with a uniform color. If a certain qualitative measure of the algorithm's output needs to be encoded for every sample, color coding can be used. However, this requires a method which derives a single value for each output of the algorithm (images in case of visualization algorithms). This value is then mapped to a color which is used for rendering the respective sample. This value can be derived computationally from the output of the algorithm. However, this is not always possible. For some, especially visualization algorithms, it is necessary to derive this value by analysing the output image. Usually such an analysis is only a heuristic, making the samples' representations unreliable in certain cases. A third option is to represent each sample by a scaled-down version of the output image. A disadvantage of this approach is that a limited number of samples can be represented in this way due to the rapidly growing visual clutter.

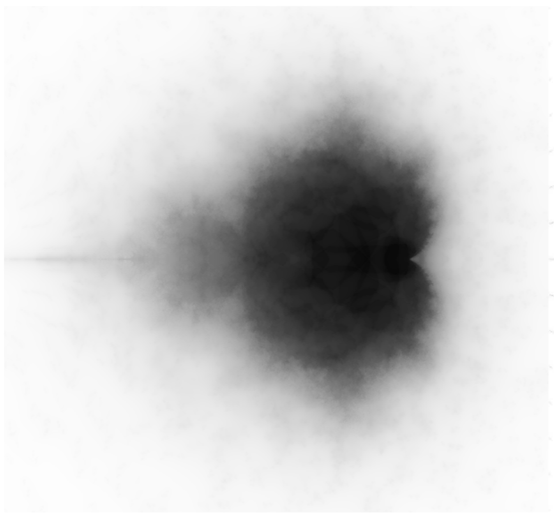
All three methods for representing the samples are demonstrated with a well-know example of parameter-space visualization - the *Mandelbrot set*. The Mandelbrot set is defined in the parameter space of the Julia set, which is a set of complex points not converging to infinity after an iterated transformation specified by a polynomial f, such as $f_c(z) = z^2 - c$ where $z, c \in \mathbb{C}$. Therefore, c is a parameter of the Julia set, and its parameter space is the complex plane. The Mandelbrot set can be visualized by assigning different colors to each point c in the complex plane and show them as pixels in the image space. The color is assigned to the complex point c depending on whether the Julia set parametrized by c is connected or not. Evaluating connectedness provides an abstract characteristic of the Julia set. This characteristic is transformed to a scalar value, which is used to visualize the parameter space of the Julia set. The result is the visualization of the Mandelbrot set (see Figure 1.7a).

This example is given as an illustration of how a complex result of a function needs to be described by a single value so that the visualization of its parameter space is possible. However, it is not always straightforward to determine how this value should be derived. In the case of the Mandelbrot set, its mathematical definition provides one option, as shown in Figure 1.7a. However, it is also possible to determine the representative value in a different way. Figure 1.7b shows a different visualization of the parameter space of the Julia set. Each pixel representing one sample of the parameter space is colored according to the average color of the output image of the Julia set, where the points outside of the Julia set are colored by the number of iterations needed to determine the divergence of the respective point. The average color is calculated from low-resolution images. By increasing the resolution of the images, the parameter-space visualization would approximate the Mandelbrot set as shown in Figure 1.7a. Therefore, this visualization is not in correspondence with the mathematical definition, but conveys different types of information, which might be useful when the resolution of the output images is limited. For instance, this parameter-space visualizations conveys for which points in the complex plane, given this particular way of rendering the Julia set, the output image contains a perceivable shape. In this case, the samples of the parameter space of the Julia set are described by a value derived by analyzing the output images.

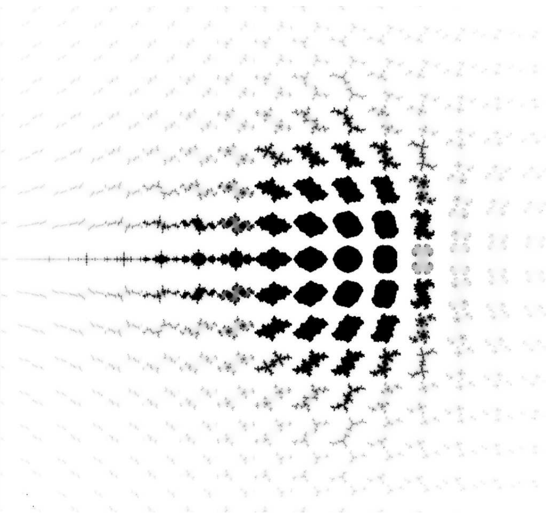
Figure 1.7c again shows the visualization of the same parameter space. Here, small multiples as introduced by Tufte [106] are used to show how the Julia sets look like for different input parameters from the complex plane. Similar to the previous visualization, by increasing the number of the small multiples, the Mandelbrot set would be approximated. However, the small



(a)



(b)



(c)

Figure 1.7: (a) The Mandelbrot set as a visualization of the parameter space of the Julia set. Four points in the complex plane are marked with red circles, and the corresponding Julia sets for these points are shown in blue color. (b) Different visualization of the same parameter space. An estimation of the area of the Julia set by counting pixels is used to visualize the parameter space. (c) Small multiples are used to visualize the parameter space of the Julia set. The parameter space represented by the image space is sampled on a regular grid. Each of the small multiples depicting one Julia set is placed at the position of the respective sample.

multiples are effective in showing how the Julia sets with different shapes are distributed across the complex plane, thus providing yet another type of information about the parameter space. However, this information is only visible it a limited number of samples is taken.

A specific aspect of the small-multiples visualization shown in Figure 1.7c is that the positions of the individual images convey information about the value of the underlying complex parameter c. If it is not possible or desirable to derive spatial information from individual parameter-space samples, the small multiples can be arranged in an abstract way. A common approach for such an arrangement is to evaluate the visual appearance of the rendered images and use it to construct the spatial information, for instance by clustering the images.

The described examples show different ways of representing parameter-space samples. Each of these methods is applicable to different classes of algorithms and they are chosen depending on the tasks which should be aided by the parameter-space visualization. When applied to visualization algorithms, which are the main focus of our research, all three approaches share a common limitation. Each sample represents the entire output image - either obtained through a mathematical model, through image analysis, or by using the scaled-down version of the output image. For certaing tasks, such as visual analytics, this might be a serious drawback. For instance, these approaches are limiting if it is necessary to comprehend connections between localized areas of the visualization images and the parameter space. In this research, we propose parameter-space analysis and visualization methods which overcome this limitation. We design a method for parameter-space exploration, where the parameter space of the visualization algorithm is linked with localized positive and negative examples in image space, so that the global effects of individual parameters on the visualization can be analyzed. A second proposed method allows the user to store image-space selections of local areas within visualizations, and link them with respective parameter values.

1.4 State of the Art

In this thesis, we present novel approaches for the integrated exploration of parameters of visualization algorithms and their management. We approach this area of research from three directions, where we find our techniques to be suitable for many distinct applications. We structure the state-of-the-art report accordingly. In the first part, we concentrate on work dealing with the exploration of parameter spaces, as introduced in Section 1.2. This area of research relates to our proposed way of understanding visualization algorithms by integrating the visualization and the parameter space exploration. In the second part, we review literature related to the management of parameters, especially useful for provenance tracking. This is relevant to our proposed method of storing visualization parameter settings linked with the actual visualizations. The third part of the state-of-the-art report reviews research carried out within the area of visual storytelling. We propose a method for processing parameter spaces of video games to integrate gameplay recordings of multiple players observing the virtual world of the video game from many dynamic points of view. Visual storytelling is one of the dominant application areas of this method.

1.4.1 Parameter-space visualization and analysis

As mentioned earlier, visualization techniques for parameter spaces strongly depend on their dimensionality. By treating each of the input parameters individually, the parameter space is split into one-dimensional subspaces. Each of these subspaces can be visualized separately. Gavrilescu et al. [42] present an improvement to traditional user interface elements. In their work, the authors propose to display information about the effects of parameters in the user interface elements employed for adjustment of these parameters. Here, the parameter is sampled across its domain and a visualization is produced for each of these samples. By comparing the rendered images, it is possible to derive information about the behaviour of the parameter.

Samples of a 2D parameter space can be visualized in two dimensional images. In this case, the parameter-space dimensions are interpreted as spatial dimensions of the image-space. Each sample, i.e., parameter vector, can be assigned a color according to a certain characteristic of the respective output. The samples can be then arranged into a scatterplot, or a regular grid in the image-space. Amirkhanov et al. [3] utilize this approach in their work concerned with specimen placement in industrial X-ray computed tomography scanners. The placement is described by two Euler angles. The parameter space of the placement is visualized by color coding various characteristics of the placements in 2D image space. This approach is useful for a stability analysis of the specimen placements.

In the application area of volume rendering, several approaches to transfer-function specification have been proposed. These methods often utilize various forms of parameter-space visualization and analysis tools, since transfer functions constitute large search spaces where trial and error methods of finding good transfer functions are often ineffective or downright impossible. König and Gröller [65] propose a method where the dimensions of the transfer function are specified separately. The specification is aided by sampling the parameter space along the respective dimension and providing visual feedback on the possible parameter changes.

Marks et al. [78] propose Design Galleries, a method for computer-aided parameter setting. The method visualizes the parameter space by showing perceptually different result images for automatically sampled parameter vectors. The parameter space is sampled in the pre-processing step. Design galleries address the problem of *dispersion*, i.e., sampling of the parameter space so that the visualization space is uniformly covered, and *arrangement*, i.e., placement of the images representing the samples in an intuitive manner.

Jankun-Kelly and Ma [58] propose a spreadsheet-like interface for volume visualization. In the cells of the spreadsheet, results of volume-rendering algorithms with different parameter settings are shown. Rows and columns represent individual parameters, which can be changed in order to explore the data. The authors also provide various dynamic operators, which can be used to manipulate the displayed visualizations. The spreadsheet-like interface gives a 2D view of the multidimensional parameter space sampled according to the values of the rows and the columns. The concept can be adopted for other visualization algorithms besides volume rendering as well.

Bruckner and Möller [20] present a system for aiding parameter setting of simulations for creating visual effects. Parameter spaces of such simulations are usually huge and the mental reconstruction of relationships between parameter vectors and the visual appearance of the simulation outputs is non-trivial for the graphics artists. The system is targeted to graphics artists, who are interested in the appearance of the simulation results rather than the structure of the parameter space. Therefore, the parameter space is sampled and the outputs are clustered in order to allow the user to find sequences of the simulation results according to their needs. This is done in an intuitive manner and replaces the trial and error approach. The fluid simulation algorithm addressed in this work is similar to the visualization algorithms, which are the focus of this thesis. In both cases the output is an image which is difficult to evaluate qualitatively in an automatic way. Therefore, interaction mechanisms for the intuitive exploration of the parameter space are crucial for these algorithms.

Sedlmair et al. [95] propose a conceptual framework for describing parameter-space analysis tasks. Additionally, it provides abstraction of data-flow and navigation strategies independently from the application domain. The framework is based on a large body of visualization literature dealing with the analysis of parameter spaces.

1.4.2 Parameter management and provenance

A visualization session is an iterative process of refining parameters to obtain desired results or to explore a dataset. The intermediate visualization products generated in each step of the session may convey important information. The intermediate products tell a story of how the final visualization was created, but they might also convey essential information taken into account in making decisions during the visualization session. For instance, an intermediate visualization might reveal that desired features are not visible after applying a certain pipeline step, which explains why the following steps of the session were executed. This is especially useful when multiple participants collaborate in visualization and data exploration scenarios. The tracked provenance helps to communicate reasoning and gained knowledge between the involved parties.

In the field of visualization, the concept of *provenance* [99] is often utilized. Provenance is a record of origin or history of an item. In the context of data visualization, it is a record of all the steps necessary for the transformation of the data into the final visualization image. By tracking the provenance information, it is possible to execute visualization pipelines with different input datasets, reuse parts of the pipelines in other visualizations, or compare visualization results produced by different pipeline execution instances. Additionally, the provenance information enhances the possibilities of assigning semantics to the visualization results. Visualizing provenance information serves as a meta-visualization which makes it easier to derive knowledge from the input data. The concept of provenance is especially useful when dealing with large or multivariate datasets.

There are various approaches to track and manage provenance information. A survey of visualization systems with a support for provenance management of computational tasks is given by Freire et al. [40]. The survey specifically deals with two forms of provenance, *prospective* and *retrospective*, as proposed by Clifford at al [27]. Prospective provenance records the specification of the given task. On the other hand, retrospective provenance records an execution of a single instance of the task.

Provenance tracking systems are not limited to visualization algorithms. They usually focus on more general workflows for scientific data processing or simulations. There are standalone provenance frameworks, such as PReServ [47] and Karma [100], which can be included in existing systems. These frameworks use signalling mechanisms to transfer the data of interest and store them in databases. Additionally, such frameworks are required to provide means for efficient querying of the captured provenance information. Moreau et al. [83] present Open Provenance Model (OPM). This model is designed to describe provenance of arbitrary objects, even those not generated by computer systems. The provenance representation is independent from the underlying technology and it is transferable between different systems.

Provenance tracking is employed in many different areas. InProv [16] is a system for visualizing provenance of file systems. It utilizes a radial layout as an improvement over traditional node-link layouts of the provenance data used in older systems, such as Orbiter [76]. The radial layout highlights relationships between items rather than their relative spatial locations as it is the case with the node-link layouts. Chen at al. [24] propose methods for visualizing large-scale provenance datasets generated by observing the data-flow in computer networks. The proposed techniques are used for exploratory visualization, such as understanding a denial of service attack, as well as storytelling. The methods are also applied to a processing pipeline for NASA satellite imagery, which also generates very large amounts of data. This shows that these provenance-visualization techniques, utilizing several graph drawing concepts, are general enough to be used in different scenarios. For both use cases, the authors use the Karma framework for the provenance tracking.

Provenance-visualization concepts derived from the specific requirements of various systems are applicable to provenance-tracking of visualization algorithms as well. As the focus of this thesis is on visualization algorithms, the designs of provenance visualization proposed in these systems are relevant to our work.

When dealing with visualization algorithms, provenance tracking is used to record steps needed to generate specific images from the given data. This is useful in many scenarios, where the process of generating visualizations needs to be reused with different datasets. Patten and Ma [86] propose a method for tracking the provenance of a volume rendered image. In this method, the intermediate results are connected into a so called image graph, where the edges represent parameter changes leading to these results. The graph is progressively built during the visualization session, tracking the provenance of the result image. Additionally, Ma [74] presents various dynamic additions to the image graphs, such as graph editing or properties propagation.

To efficiently track the provenance, storing parameter changes might not be sufficient. The changes to the visualization pipeline itself need to be stored as well. To address this problem, Bavoil et al. [9] propose VisTrails. It is a system for automated provenance tracking in dataflow-based visualizations. It dynamically creates a history of the visualization session, where the changes to the dataflow network are recorded. The history is stored in a so-called *version tree*, which is accessible to the user. This way, the user can revert to any previous step of the data exploration process.

1.4.3 Parameter integration and storytelling

In situations where a complex message needs to be conveyed or data have to be communicated to laypersons, it is sometimes necessary to merge several visualizations to create a desired product.

A simple case is creating an animation. To create an animation presenting data, a visualization mapping whose parameters continuously change over the course of the exploration is taken to create individual frames of the animation. This is a problem of finding a curve or a series of curves in the parameter space of that visualization mapping under certain conditions. Individual frames of the animation are samples taken along these curves.

In more complex cases, data need to be presented through several different visualization mappings in order to convey the intended message. For instance, storytelling approaches for the presentation of large multivariate datasets might integrate multiple heterogeneous visualizations in order to present the data to laypersons. In this case, it is necessary to integrate not only the parameter values, but whole parameter spaces. In other words, integrating different visualizations might require reconstructing a curve through an algorithm space, or $\mathcal{A}$-space, as proposed by Balabanian and Gröller [6].

For a given sequence of points in the parameter space (or the algorithm space), there is an infinite number of interpolating curves passing through every point in the sequence. Therefore, choosing adequate curves requires external knowledge, such as storytelling or cinematographic principles, which are then translated into the terms of the interpolation in the abstract space.

Virtual storytelling, as a means of interdisciplinary communication or data presentation for laypersons, has been studied in the context of data visualization in the past years. Research focused on adapting existing visualization methods to data presentation purposes by applying storytelling principles. Gershon and Page [44] describe how storytelling helps to convey information so that it is more easily understood and remembered by the target audience. Segel and Heer [96] analyze 58 visualizations of abstract data intended to tell a story or support it. These concrete examples are mostly taken from news media. The authors describe seven genres of narrative visualization found in the analyzed examples: *magazine style*, *annotated chart*, *partitioned poster*, *flow chart*, *comic strip*, *slide show*, and *film/video/animation*. Additionally, they identify various narrative strategies used by journalists and describe them in the context of storytelling and information visualization. This analysis of the design space of narrative visualizations outlines possible opportunities for future research in storytelling in the context of information visualization.

Kosara and Mackinlay [66] point out the importance of storytelling approaches for data presentation. They discuss several storytelling scenarios relevant to information visualization and describe how storytelling techniques were utilized to make the data understandable to a wider audience. For instance, the authors point out the importance of animated transitions in creating a coherent story out of visualizations of numerical data. The animated transitions are used to continuously change some of the visualization parameters, creating a succession of steps through the visual story to be told. The authors mainly focus on information visualization, i.e., visualization of abstract data, and relate the research in this direction to the work of journalists, who often need to present numerical data in an intuitive manner. In their paper, the authors also outline research directions for future research in the area of storytelling for data presentation.

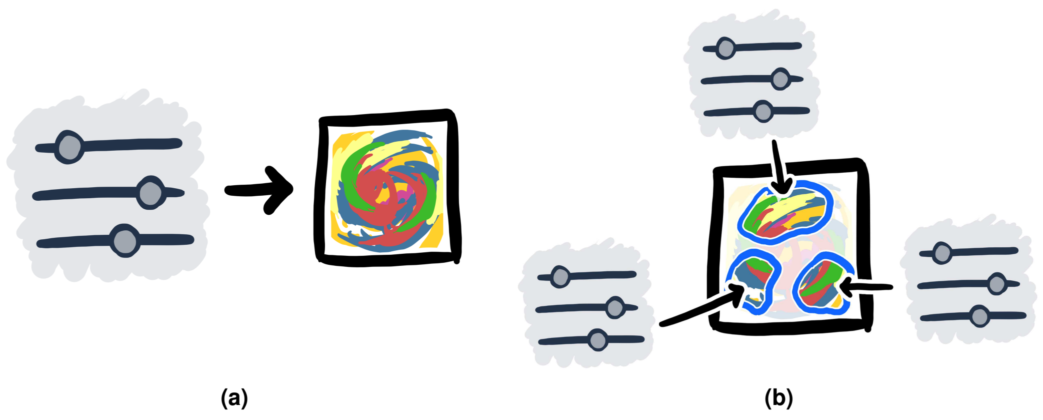
Hullman et al. [57] analyze 42 narrative visualizations made by professional designers to identify how sequences are created to construct an informative story. All the analysed visualizations consist of well-defined steps, such as numbered slides with “Next” buttons, or similar elements. In their analysis, the authors identify different transitions between the individual steps. The analysis suggests that most commonly the transitions are constructed in such a way that the visualization changes as little as possible. Effectively, this means that usually a single parameter or a single dimension of the visual data representation is changed during the transition to move from the given visualization step to another one. For a better comprehension of the narrative by the audience, transitions consisting of changing multiple dimensions are typically split into sequences where only a single dimension is changed. This observation allows the authors to propose an automatic way of identifying possible transitions between a set of visualizations organized into a graph structure. Nodes of the graph represent the visualizations, while edges represent the transitions. An objective cost function is applied to find the most suitable set of transitions between the visualizations to construct a visual narrative. The approach is evaluated with two user studies. In the first study, the effectiveness of visualization-to-visualization transitions is evaluated. The second study examines impacts of transition parallelism, e.g., showing sequences of temporally overlapping data or effects, on the global narrative. The findings presented in their work are interesting for our research on composing summary videos of computer games, where the sequence of the shown events is the main cue for conveying the story. We apply transitions between individual events which occurred during the gameplay to allow viewers to reconstruct the *big picture*, either for entertainment or for analysis of the gameplay.

In the field of scientific visualization, storytelling methods have been applied as well. Ma et al. [75] provide an overview of how storytelling can be incorporated into scientific visualization techniques. Wohlfart and Hauser [118] propose a method concerning volume visualization for data presentation. In their method, story telling and story authoring are two distinct steps. In the story authoring step, the dataset is explored by an expert user. While the data are being explored, the expert user iteratively creates the virtual story. In the story telling step, the findings from the previous step are presented. The user watching the story also has the possibility to intervene with it in an interactive manner and to perform data exploration on her own. The method has been improved by Lie [70] to support more complex visualization methods for story authoring and storytelling.

1.5 Overview of the thesis

In this thesis, parameter-space exploration is examined from several perspectives. Our work is split into four parts. First three parts are concerned with the management and the exploration of parameters at a different scale, or level of complexity: parameters with global influence over the whole image, multiple sets of parameters with localized influence, and the integration of multiple parameter sets linked with multiple users. These levels of complexity are illustrated in figure 1.8. The fourth part describes a model of human-computer interaction for parameter adjustment. This universal model accounts for the underlying data to adjust the interaction so that the changes in the input and the output are made proportional. The projected output changes are visualized to steer the parameter-space exploration.

First, we investigate possible ways of how the parameter-space exploration of a visualization algorithm can be carried out in an intuitive manner. We review existing approaches to parameter-space exploration and analysis, and propose a novel method for analyzing the relationships between the input parameters and the associated effects on the visualization result. Our goal is to design a universal method which could be applied to a wide range of distinct visualization algorithms. The method provides insight into the parameter spaces of the under-



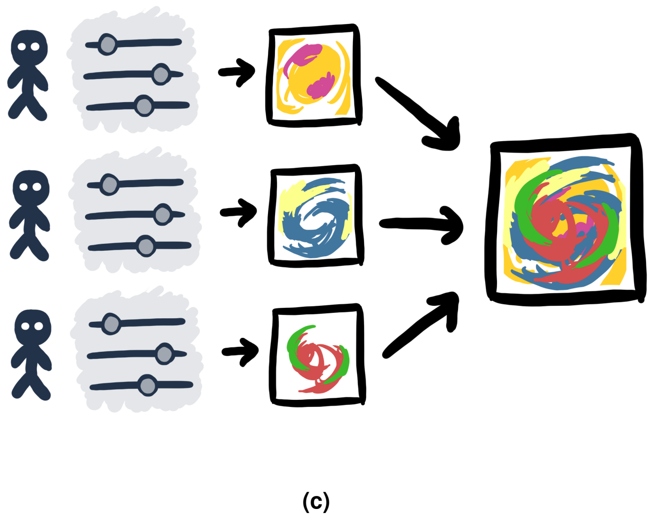


Figure 1.8: Three levels of complexity of parameter-space exploration and management. (a) Parameters of a single visualization. (b) Multiple sets of parameters influencing localized parts of the visualization. (c) Integration of multiple sets of parameters of multiple users into a single visualization.

lying visualization algorithms. This is realized by iteratively exploring the parameter space according to the scientific method: first, a hypothesis is formulated concerning the effects of a certain subset of the parameters on the final visualization. Subsequently, our method allows visualization experts to verify the hypothesis through interaction with the visualization. Within the interaction, several parameter-space samples are classified as positive or negative examples. This classification is then generalized over the whole domain of the parameter subspace defined by the selected parameters and used for an automatic modification of the visualization mapping. This way, the possible effects of the selected parameter combination can be directly observed on the visualization image.

The proposed method allows the user to discover how visualization parameters with global influence on the visualization image can describe certain feature sets present in the data. The parameters of interest are selected through a novel markup language *PEL* (Parameter Exploration Language), which is inserted into the code of the visualization algorithm either manually or using a graphical user interface. The use of a markup language enables the user to explore not only the parameter space, but the entire *shader space* - a combination of the possible visualization mappings (implemented as GPU shaders), their input parameters, and the data. We also provide a tool for visualizing the outputs of the PEL markup, enabling an in-depth analysis of visualization algorithms, with the goal of aiding the optimization for the visualization of specific data aspects.

The second part of the research focuses on the spatially localized influence of parameters. Our interest in this part is the management of multiple sets of parameters with a local influence on the spatial domain within the visualization. We propose a method for augmenting a visualization session, where parameters are iteratively refined in order to explore the data. We enable the user to specify image-space regions where a specific combination of the parameter values causes a certain effect. These regions, or image-space selections, are automatically linked with the current parameter settings and are stored for future use. Through our method, it is possible to browse the stored visualization-system states described by the parameter settings and the associated spatial selections. We provide means for smooth transitions between the states, as well as for linking additional data or dynamic algorithms to the data subsets specified by the image-space selections. These mechanisms are useful for a variety of applications, such as provenance tracking, data annotation and presentation, or management of multiple spatial selections in dynamic environments.

In the third part, we tackled the challenge of integrating dynamic sets of time-dependant parameters from multiple collaborating users. In contrast to the parts described above, in this case we have chosen a concrete application scenario, instead of providing a general solution for a specific class of problems. In this scenario, the users are players engaging in a multiplayer video game. Each player's actions during the gameplay are described by a set of time-varying variables. These are stored during the timespan of the game round, allowing replays of the game, possibly for a post-mortem analysis of the gameplay. However, in the dynamic environment of the video game, with multiple dynamic foci coming from the interactions between players, it is not trivial to mentally reconstruct the gameplay story simply by observing the replays from the perspectives of the individual players. To highlight the causal connections between important events forming a coherent story, we propose a method for the automatic integration of the parameter sets describing the actions of the individual players. By carefully choosing the order in which the events are depicted and integrating game contexts of the individual players described by the stored time-varying parameters, a visual story narrating the gameplay is created. The integration is performed by applying smooth transformations between the contexts, and by merging them in order to reduce redundancy. The generated narrative has various possible uses, for instance as a tool for gameplay analysis for professional gamers, or as a visual summary of the game round.

Allowing the user to explore the parameter space of a video game would not bring significant benefits, since it contains all possible stories within the boundaries of the game rules. The problem of constructing a meaningful visual gameplay narrative is in fact a search for a curve in the parameter-space of the video game. The shape of the curve must correspond with the story composed of the important events that occur during the gameplay. If the interest is to find a good visual representation of the gameplay, such a curve is a good starting point for any further exploration of the parameter space of the game.

The fourth part of the thesis deals with the interaction aspect of parameter settings adjustment. Interaction is an essential part of all three levels of complexity addressed in the first three parts. In this part, we model how the parameter space is connected to input and output spaces, and how it can be utilized to steer the process of visual data-exploration. This is achieved by making the changes in the input and the output proportional, and visualizing by projected output changes in the screen-space of the user interface.

1.6 Methodology and contributions

We classify the problems and the tasks related to the parameters of visualization algorithms into the following categories:

- *parameter-space visualization* as means for better understanding of a function, algorithm, or a model, in general
- *parameter-space analysis* as a step towards solving a specific problem
- *parameter management* for data presentation, provenance tracking, or structuring visualizations of large data
- *parameter integration* for creating data overviews and composite visualizations of multiple datasets
- *human-computer interaction* as means for manual adjustment of parameter values

In this thesis, we address specific problems from each of these categories. Chapter 2 focuses on the visualization and analysis of parameter spaces of visualization algorithms. The goal is to aid the understanding of complex visualization algorithms. Parameter management, as a general approach in the domain of visual representations, is addressed in Chapter 3. A solution to a specific problem of parameter integration in the area of video games is presented in Chapter 4. Finally, in Chapter 5, we present a model for enhancing effectiveness of interaction with visualization systems, where parameter values need to be manually adjusted. Additionally, we present a novel input element for traversing tree structures, which we demonstrate on an example form the application area of vessel visualization.

Exploration of parameter spaces is applicable to a large variety of algorithms from a wide range of different fields. Amongst the types of algorithms requiring this aid are simulations, segmentation and classification algorithms, and other algorithms with multidimensional parameter spaces. In this research, we mainly focus on visualization algorithms. They share the multi-dimensionality of their parameter spaces with the aforementioned types of algorithms. What makes the visualization algorithms unique is that their ultimate product is an image to be shown to and examined by the user. The visualization given in the image is to be understood by the user of the algorithm. Therefore, it is necessary that that the link between the visualization and the parameter space can be mentally reconstructed without significant effort.

In our research, we acknowledge this requirement by designing methods for interaction with the parameter spaces which are closely linked to the visualization results. We utilize the image space, where the visualizations are represented, as a means for interaction with the parameter space. Our methods for exploring global effects of visualization parameters and managing their localized effects, described in Chapters 2 and 3, allow the user to directly interact in image space, while the interactions with the abstract parameter spaces are avoided. This tight integration with the visualizations allows the user to mentally link specific parts of the images with relevant parameters, making them intuitive to use. The interaction methods are independent from the underlying algorithm, which makes them applicable in various distinct scenarios.

The method for integrating parameters of many users described in Chapter 4 finds meaningful curves through parameter space in order to construct a visual narrative. Optimization algorithms for finding suitable curves through a multidimensional space need a fitness function to evaluate the quality of each of the candidate solutions. For constructing such a fitness function with our method, we utilize image space, i.e., views from a video game of the individual players. For constructing an effective narrative, it is important to encode what players saw during the gameplay and use it to construct a story. For instance, evaluating which other players are visible in the field of view of a particular player, i.e., in their image space, gives information about possible interactions between players. This information is necessary to be communicated by the constructed visual story.

For algorithms which require manual adjustment of its parameters, it is essential to design a suitable user interface. The diverse nature of input parameter used in visual-computing algorithms implies usage of different types of input elements. The problem with the input elements is that it is hard to optimize them for all underlying datasets. To address this problem, we describe a model for data-sensitive navigation in Chapter 5. This model describes how the underlying data can be taken into account for enhancing the user interaction with the parameter space, so that the changes in the input are made proportional to the changes of the output. Additionally, the projected output changes are visualized to help users in steering the process of adjusting parameter values.

CHAPTER 2

Visual Parameter-Space Exploration

This chapter is based on the following publication:

P. Mindek, S. Bruckner, P. Rautek, M. E. Gröller. Visual Parameter Exploration in GPU Shader Space. *Journal of WSCG 21, 3 (2013)*, pp. 225—234. ISSN 1213-6972.

There are various methods aiding parameter specification during visual data exploration. A classification of methods for setting transfer functions presented by Pfister at al. [87] can be generalized for all types of visualization parameters. In their work, the authors describe data-centric and image-centric methods of transfer-function design. In the data-centric methods, transfer functions (or in general, the visualization parameters) are specified through interaction with the underlying data model. In such methods, the data model needs to be properly understood. For instance, knowledge of value ranges encoding specific structures in a volume dataset is necessary for designing transfer function which shows these structures. On the other hand, image-centric methods utilize interaction with the rendered image in order to change the visualization-parameter settings. In general, the image-centric methods do not require a detailed understanding of the underlying data model. The user indirectly refines the visualization parameters by specifying desired changes in the result images.

In this chapter, we describe our approach to exploration and visualization of parameter spaces of visualization algorithms. We employ the image-centric approach to manipulate predefined sets of parameters of a visualization algorithm. The image-centric approach allows us to apply our method to arbitrary visualization algorithms, since we can omit any assumptions about the underlying data model. In contrast to traditional methods for parameter-space exploration, our method allows the user to examine how isolated parts of the visualization algorithm and parameters influence different areas of the rendered image, and how the algorithm could be modified to provide a more suitable visualization. Therefore, we enable the exploration of entire visualization algorithms, not only the properties of their parameter spaces.

The wide availability of high-performance GPUs has made the use of shader programs in visualization ubiquitous. Understanding visualization algorithms implemented as shaders is a challenging task. Frequently it is difficult to mentally reconstruct the nature and types of transformations applied to the underlying data during the visualization process. We propose a method for the visual analysis of GPU shaders, which allows the user to explore and investigate algorithms, parameters, and their effects, in a flexible manner. We introduce a method for extracting feature vectors composed of several attributes of the shader, as well as a direct manipulation interface for assigning semantics to them. The user interactively classifies pixels of images which are rendered with the investigated shader. The two resulting classes, a positive class and a negative one, are employed to steer the visualization. Based on this information, we can extract a wide variety of additional attributes and visualize their relation to this classification, e.g., in which areas do they belong to the positive or the negative class. Our system is designed for the interactive exploration of shader space and we demonstrate its utility for several different applications.

2.1 Introduction

In data visualization, GPU shader programs are often used to process large amounts of data and to create suitable visual representations. In this case, shaders usually implement algorithms which provide a mapping between the data and the intended visual representation. The way how the data is displayed depends on the shader program and its input parameters. The vector of values of the input parameters is a point in the corresponding parameter space. An interpretation of the resulting image requires knowledge of the relationships between the parameter space of the underlying algorithm and the visualization. These relationships might not be trivial to grasp given only the source code of the shader implementing the algorithm and the resulting image.

Data-visualization algorithms implemented as GPU shaders can be investigated from various perspectives. Rendered images, i.e., image space, are presented to the user. The mapping process from data space to image space is specified in a shader program. We refer to the combination of possible shader programs and their parameters as shader space. While the image space is usually interesting for the domain expert for whom the visualization mapping was originally created, the shader space is interesting for the visualization expert. In situations where the shader program needs to be modified, it is important for the visualization expert to understand how the algorithm affects the resulting visualization. Therefore, there is a necessity for tools allowing for exploration of the shader space.

We propose a method that allows users to explore GPU shader programs and their parameter spaces by assigning semantic classifications to parts of the rendered images. This user-assigned information is subsequently used to modify the visualization mapping by generalizing it to the whole rendered image. In this way, the influence of the examined parameters or data attributes on the display of the features of interests becomes apparent.

The shader exploration is facilitated by a domain specific language (DSL) that extends the shader language under consideration. The DSL is used to annotate the shader program without changing its functionality. The annotations are inserted as comments that are parsed and interpreted by our system. This strategy is similar to well known documentation tools like Doxygen and JavaDoc. The benefits are:

- the annotations are kept close to the original code (resulting in easier maintainability)
- the annotations are transparent to the host language

We refer to the insertion of comments that include commands from the DSL as shader augmentation. The augmented shader can either be run unmodified to fulfil its original purpose, or be transformed for the shader-space exploration. This provides functionality for the exploration of desired parameters and data attributes. As the additional functionality is part of the augmented shader, it is executed on the GPU, which benefits the performance of the system.

The main contributions included in this chapter are:

- a method for translating semantic classifications from image space to shader space and using them to explore the effects of parameters on the rendered images
- a parameter-exploration language (PEL), which is used to extract values of relevant parameters and data attributes from the shader program for further parameter-space exploration
- a graphical user interface for the automatic and transparent generation of PEL code.

While the PEL can be employed by visualization experts, the graphical user interface allows the user to explore the shader-space even without knowledge of the shading language, which might be a benefit for domain experts.

2.2 Related Work

Our method is designed for exploring parameter spaces of various visualization algorithms. The method is particularly applicable to volume rendering. These algorithms are usually complex and it is difficult to predict the effects of parameters and data attributes. Therefore, in this work, we focus mainly on volume rendering as a possible application area for our method.

In volume rendering, the mapping between data and rendered images is usually adjusted using transfer functions. A transfer function maps one or more data attributes to optical properties. Therefore, it enables users to visually examine these attributes of the data. Wu and Qu [121] present a framework for interactive transfer function design based on genetic algorithms and image similarity, where the user edits direct volume rendered images. Another approach is presented by de Moura Pinto and Freitas [31]. It is based on dimensional reduction of the voxel attributes using self-organizing maps. Correa and Ma [29] propose visibility-driven transfer functions for enhancing the visibility of important features in volume data. Tzeng and Ma [109] present an interface for data classification in a cluster space of different materials present in the data.

Multidimensional transfer functions can be employed for the examination of effects of particular data attributes on the visualization. We approach the problem from a different angle by allowing the user to redefine how any of the data attributes and parameters of the visualization mapping influence the final image. This possibility makes our method capable of complex explorations of the shader space, which is difficult to carry out using only editing operations.

Tzeng et al. [108] propose a volume data classification approach based on machine learning. This approach employs a sketch-based interface to select a primary classification, which is then used as a training set for a machine learning algorithm. Our system uses a similar strategy, however we focus on parameter exploration of arbitrary visualization algorithms, not the design of transfer functions. Similar work is also presented by Guo et al. [48]. Their WYSIWYG (What You See Is What You Get) approach allows the user to define one dimensional transfer functions by direct interaction with the rendered images. Yuan et al. [125] present a method for sketch-based volume segmentation. Rautek et al. [91, 92] propose semantically driven constructions of visualization mappings.

One of the main intended use cases of our method is to help analyze and understand visualization shader programs, possibly for debugging purposes. Various systems for analyzing and debugging visualization software have been presented [35, 55, 103]. Crossno and Angel [30] present a case study on debugging of visualization software. Meyer-Spradow et al. [81] propose a framework for rapid prototyping of visualization algorithms by connecting modules in a data-flow network. The framework also provides debugging abilities by displaying the outputs of the individual modules. Our method enables the user to visually identify effects of particular shader parameters and data attributes on the rendered images. Rather than displaying values of individual variables, it provides visual feedback on how the values correspond to the user-defined classification of the rendered image. These findings can be compared to expected behaviours for analysis of the visualization algorithms. Our method can be used as a complementary tool to other shader debugging solutions.

Jankun-Kelly et al. [59] propose a model for the visualization-exploration process. McCormick et al. [79] present Scout - a visualization system utilizing the GPU for exploration and visualization by applying queries directly to the data. Gerl et al. [43] describe a method where properties of visualization mappings are rendered, and brushing is used on these rendered images to explicitly specify semantics for volume visualization. Our method uses similar principles. However, our focus is not on exploring the data, but rather on exploring the visualization algorithms. We apply our method to investigate the effects of parameters of particular visualization algorithms on the resulting image. Knowledge gained from this process can be used in the exploration of other datasets using the visualization algorithm. McDonnel and Elmqvist [80] present the concept of using the GPU for information visualization. They propose an interface for creating visualizations on the GPU without shader-programming knowledge. We utilize a similar interface, but it is intended for exploration and analysis of existing visualization algorithms.

2.3 Visual Shader-Space Exploration

The usual way to explore visualization algorithms is through examination of the results, i.e., rendered images. However, the effects of various isolated parts of the visualization algorithm are difficult to comprehend by looking only at output images, because parts of the algorithm might interoperate in a complex, unpredictable manner. Even a deep knowledge of the algorithm might not be sufficient to fully predict its behaviour on a given dataset.

On the other hand, given a known dataset is being visualized, it is usually possible to identify areas of the rendered image where certain features of interest are visible, and areas where they are occluded or not displayed correctly. These areas can serve as positive or negative examples of

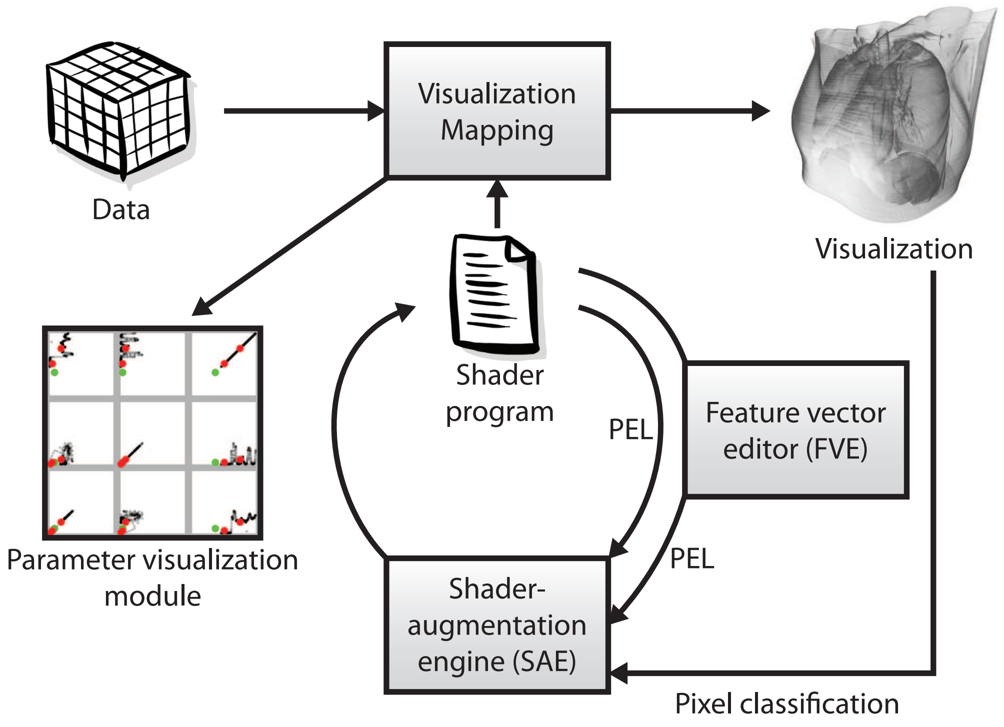

Visualization Mapping
Data
Visualization
Shader program
PEL
Feature vector editor (FVE)
PEL
Parameter visualization module
Shader-augmentation engine (SAE)
Pixel classification

Figure 2.1: Overview of the visual shader-space exploration (VSSE).

the behaviour of the visualization algorithm. In our work, we utilize such an area identification as interaction interface for exploring the behaviour of visualization algorithms and specific parts thereof.

The purpose of the method is to help analyze visualization mappings provided by GPU shader programs and to find ways how the mappings could be enhanced. We call our method visual shader-space exploration (VSSE). VSSE aids the effort to find variables of the shader, or their combinations, which have similar values for the rendered pixels classified as positive or negative, but different values for pixels belonging to different classes. Such a knowledge is important for designing visualization mappings, and optimizing existing ones, for visualization of specific features of various datasets.

We refer to the combinations of shader variables as feature vectors. The components of the feature vectors consist of shader parameters, data attributes, or derived variables used in the shader. The user explores the shaders by selecting which variables will be used for the feature vectors. Subsequently, VSSE tries to generalize the classification, set by the positive and negative examples identified by the user, to the whole image using the specified feature vectors.

Our method allows users to select parameters that are to be examined. Subsequently, the user identifies several pixels in the image space of the visualization, i.e., rendered image, where features of interest are visible, and several pixels where they are invisible. We refer to the pixels displaying the features of interest as the positive class, while the other ones constitute the negative class. VSSE uses values of the examined parameters and data attributes for the pixels identified by the user as belonging to either the positive or the negative class to generalize the classification to all pixels of the rendered image. The generalized classification is encoded in a value generated for each pixel of the visualization. This value can be employed in the shader to show the users the effects of the examined parameters on the visualization of the features of interest.

Figure 2.1 shows an overview of VSSE. An image is created by transforming the data through the visualization mapping. The visualization mapping is implemented by a shader program. Subsequently, the visualization mapping can be modified by the user. In order to modify the visualization mapping, the parameters of the original shader which are to be examined are selected. This can be done either using the Parameter Exploration Language (PEL) which is embedded into the original shader code, or using the Feature Vector Editor (FVE). The FVE also generates PEL code, so the PEL and the FVE can be used simultaneously. The PEL code in the original shader is then parsed with the Shader Augmentation Engine (SAE) and an augmented version of the shader is generated. This augmented version of the shader provides the original visualization as well as the possibility to classify pixels into positive and the negative classes. This classification is in turn used to modify the visualization mapping so that the user can examine the effects of the selected parameters. Additionally, values of the examined parameters can be visualized in a scatterplot matrix using the parameter-visualization module.

2.4 Feature Vectors

For every classified pixel a feature vector based on the values of the examined parameters and attributes is extracted. The classification of the pixel is then assigned to the extracted feature vector. VSSE provides means to calculate a fuzzy classification, or a membership degree of a feature vector to either the positive or the negative class in any state of the execution of the shader. The membership degree is in the interval $[0, 1]$, where 0 means association with the negative class and 1 means association with the positive class. The value of the membership degree can be freely used in the shader to modify the visualization mapping so that the classification of the rendered pixels is displayed. We also refer to the membership degree as *confidence*.

A feature vector can be extracted in every stage of the shader program. During the execution of the shader, the feature vector may be extracted multiple times for a single pixel. Since the color of the pixel depends on all of the extracted feature vectors, they have to be composited into one feature vector representing the pixel. We refer to it as pixel-feature vector. The pixel-feature vector is assigned the classification (positive or negative) that was specified for the pixel by the user.

The user can classify multiple pixels in one session. Therefore, the shader has access to multiple feature vectors associated with either class. At any point of the execution of the shader, this information can be used to calculate the confidence that a most recently extracted feature vector belongs to the positive or the negative class. The confidence is determined by a membership function.

The membership function is used for an automatic classification of pixels that have not been classified by the user as either positive or negative. The function should be smooth and it should not be sensitive to slight changes in its inputs, since these are provided only with a certain accuracy through interaction in image space. For this purpose, Euclidean distances of the classified feature vectors to the pixel-feature vector can be employed.

The membership function is defined for pixel-feature vectors $\vec{x}_i$ as $f(\vec{x}_i) = c_i$ where c_i is the user defined classification for the feature vector of pixel i. The classification is given as follows:

$$c_i = \begin{cases} 1 & \text{if positive} \\ 0 & \text{if negative} \end{cases} \tag{2.1}$$

For feature vectors not classified by the user ($\vec{x} \neq \vec{x}_i$), the membership function is defined as follows:

$$f(\vec{x}) = \frac{\sum_{i=1}^{n}(c_i \frac{1}{\|\vec{x}-\vec{x}_i\|^p})}{\sum_{i=1}^{n}(\frac{1}{\|\vec{x}-\vec{x}_i\|^p})} \tag{2.2}$$

where $\vec{x}_{1..n}$ are the user-classified feature vectors. p is an additional parameter that adjusts softness of the membership function. This fine-tuning is useful when the VSSE is applied to different types of data. Figure 2.2 gives an example of a two-dimensional feature-vector membership-function with $p = 1$ and $p = 5$.

If there are no user-classified feature vectors yet, the confidence for any feature vector cannot be evaluated. In this case, the membership function is defined as $f(\vec{x}) = 0.5$.

Different weighting functions can be used for the classification as well. Other options include employment of algorithms such as k-means clustering. The function given in Equation 2.2 was chosen for its smoothness and relatively simple evaluation.

2.5 Shader-Space Exploration

Generally, parameter-space exploration methods are used to understand how parameters of a given visualization algorithm influence the output images. The visualization algorithm applies a parametrized transformation to create a visual representation out of the underlying data. The methods for parameter-space exploration convey the relationships between the changes of the parameter values and the changes of the output images.

Our method improves the traditional way of exploring the parameter space. It allows users to observe effects of a selected subset of the parameters, or arbitrary variables internally used by the shader program, on the rendered image. In this way, the working of the visualization algorithm can be analyzed more thoroughly than by only examining its parameter space. Therefore, we talk about *shader space exploration*.

In VSSE, the shader-space exploration is carried out by visualizing how parts of the examined shader influence displayed data features. Every pixel of the final image is calculated by a single instance of the visualization shader. The instance is described by a feature vector. The feature vector consists of values extracted during execution of the particular shader instance. Any expression of the shading language can be used for these values. In this way it is possible to extract values of variables or functions at particular positions within the shader program, even in loops or conditional statements. It is also possible to extract multiple feature vectors for a single instance. In that case, every feature vector describes a particular state of the shader program. For instance, in volume rendering, the color of every pixel is determined by compositing several data samples along a ray. For each data sample, a feature vector describing the current sample can be extracted.

Figure 2.3 demonstrates our method on a simple shader displaying a texture. In this case, the three shader variables are chosen as dimensions of feature vectors describing pixels of the image: texel luminance, and S and T coordinates within the texture. The original shader displaying the texture is modified so that the confidence value calculated by VSSE is used as the opacity, making negative pixels transparent, while positive pixels are opaque. Before any positive or negative examples are set, the confidence for all pixels in the image is 0.5 (neither positive nor negative), therefore the texture in Figure 2.3a appears semi-transparently. After a single positive example is set in the upper right corner of the image (Figure 2.3b), the confidence for all pixels in the image is 1, since there are no negative examples yet. Afterwards, a negative example is set in the upper left corner (Figure 2.3c). Thanks to the confidence value used as the pixel opacity, it can now be observed how the image can be split into parts when pixels are described by their luminance and coordinates.

To set-up this experiment, PEL commands can be inserted into the source code of the shader. Without using SAE, the shader working is not modified, since the PEL commands are also

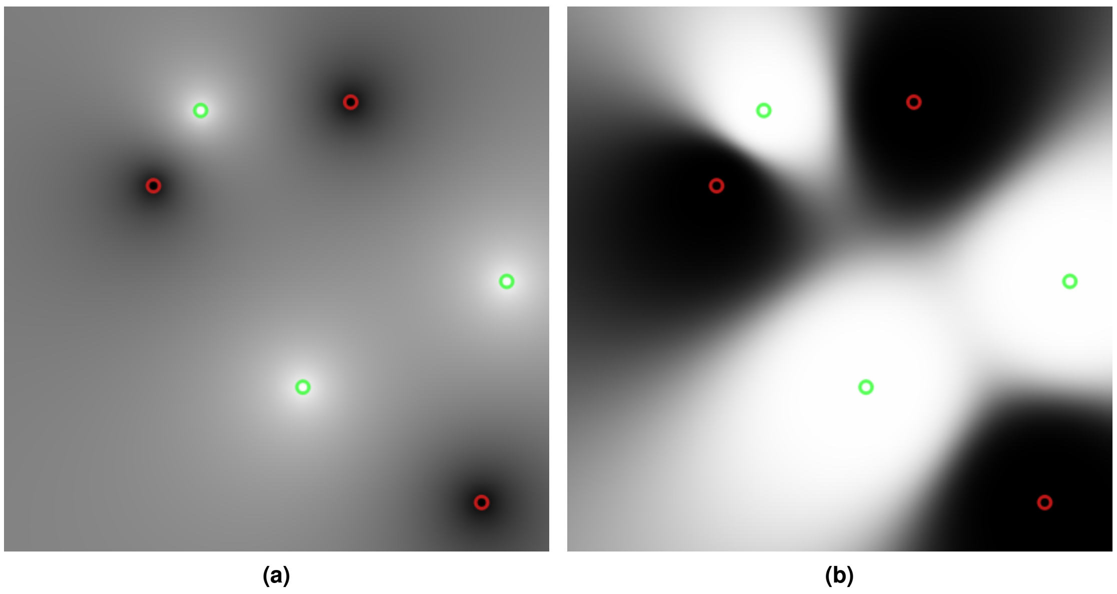
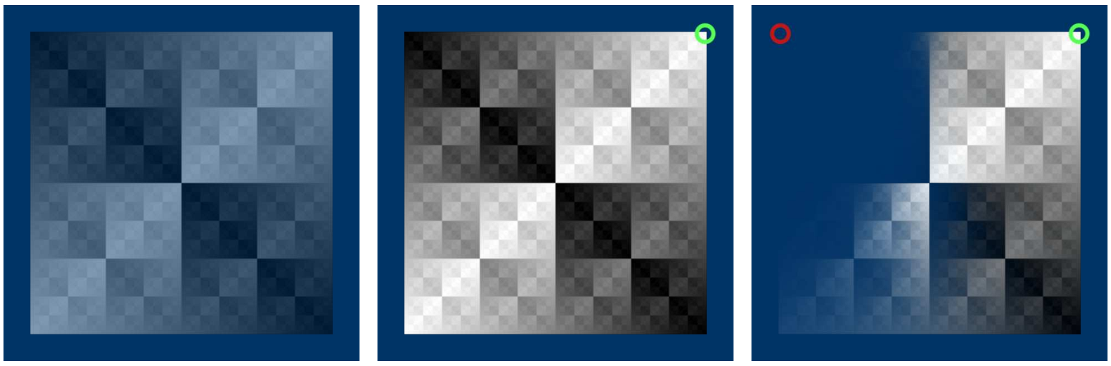


Figure 2.2: Visualization of a two dimensional case of a membership function for (a) $p = 1$ and (b) $p = 5$. Black color means association with the negative class, white color means association with the positive class. Green circles denote positive feature vectors, red circles denote negative ones.



(a) (b) (c)

Figure 2.3: An example demonstrating the usage of VSSE with a simple shader. In (a) no pixels are classified, opacity is 0.5. In (b) a positive pixel is selected (green circle), opacity is 1 for the whole image. In (c) additionally a negative pixel (red circle) is selected.

comments of the shading language. If the shader is parsed with SAE, its shader space can be explored with VSSE. Alternatively, the FVE user interface can be used to transparently generate appropriate PEL commands without the need to manually modify the shader code.

2.5.1 PEL - Parameter exploration language

We propose a parameter exploration language (PEL), which is a language for shader annotation used for parameter exploration. The purpose of the PEL is to mark which parameters or data attributes used in the shader program should be examined and how should they be visualized (by modifying the original visualization mapping). The advantage of using PEL over simply rewriting the shader program in the desired way is that with the PEL, it is possible to store multiple values of the examined parameters and data attributes during the shader execution. These values can be stored, classified as positive or negative examples, and used in subsequent executions of the shader program. By using the PEL, this functionality is added to the shader program automatically and there is no need for explicitly creating it.

The PEL can be embedded into the shader source code as shading language comments. Both line and block comments are supported. Comments starting with a dot are intepreted as PEL annotations. They are transformed to regular shader code by the shader-augmentation engine. This way, expressions of the PEL and the shading language can be easily combined.

A feature vector can be extracted using the following annotation:

```
//. vector p[0]; ... p[n]; weight
```

This annotation is replaced by code that evaluates the expressions *p[0]*, ... *p[n]* and *weight* and extracts them as the corresponding feature vector.

The keyword *confidence* can be used in PEL annotations to specify how to visualize effects of the examined parameters in a flexible manner. This keyword is replaced with a function that evaluates the membership degree for the most recently extracted feature vector. The value represents likeliness of the current shader-program state to belong to the positive or the negative class. This value can be arbitrarily displayed by the shader program.

The following listing shows a part of the fragment shader source for the example in Figure 2.3:

```
vec4 color = texture2D(tex, co.st);
//.vector color.r; co.s; co.t; 1.0
fragColor = vec4(vec3(color.r),
            1.0 /*. - confidence */);
```

In the listing the texel luminance (*color.r*) and coordinates (*co.s, co.t*) define a feature vector. In this example, one feature vector is extracted for every rendered pixel. Some of the pixels are user-classified as negative or as positive. For every other pixel, the confidence of the respective feature vector is calculated and used as opacity (*confidence* keyword) for the pixel.

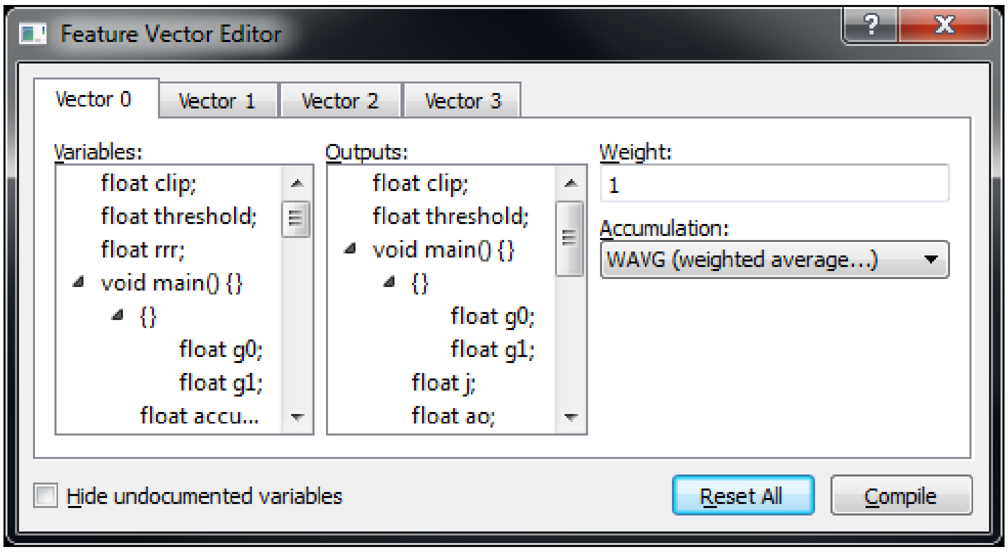
To create pixel-feature vectors from feature vectors extracted during the execution of the shader for a particular pixel, various user-selectable composition algorithms, or accumulation types, can be applied. These are defined using the *accumulation* keyword of the PEL. The default accumulation type is weighted average using weights assigned to individual extracted feature vectors.

The composition of feature vectors is helpful for applications where the color of the pixel is influenced by multiple data samples, for example volume rendering. The accumulation type should reflect the strategy used for calculating the color of the pixel. Assigning a classification to the pixel-feature vector ensures that the positive or negative example is suitably placed in the feature-vector space.

The augmented version of the shader can be switched to one of two modes. The first mode is the rendering mode without processing of user input. In this mode, feature vectors are extracted, and the membership function is evaluated for the most recently extracted feature vector. The extracted feature vectors are not composited into pixel-feature vectors and no classification is assigned to them. The second mode is the classification mode. In this mode, one pixel and its user-defined classification has to be selected. For the pixel that is being classified, extracted feature vectors are composited into a classified feature vector at the end of the execution of the shader. The purpose of this design is to enable the host application to use the same shader for rendering as well as user-input processing (classification of feature vectors).

2.5.2 FVE - Feature-vector editor

The PEL offers a high degree of flexibility when designing a parameter-exploration scenario for a particular shader program. However, it requires users to actually understand the shader source code in order to create meaningful PEL annotations. To address this issue, we propose a Feature-vector editor (FVE). It is a graphical user interface which allows users to create and extract feature vectors from the shader without having to understand it. The FVE interface is shown in Figure 2.4.



Variables:

```
float clip;
float threshold;
float rrr;
void main() {}
    {}
        float g0;
        float g1;
    float accu...
```

Outputs:

```
float clip;
float threshold;
void main() {}
    {}
        float g0;
        float g1;
    float j;
    float ao;
```

Weight: 1

Accumulation: WAVG (weighted average...)

Figure 2.4: Feature vector editor (FVE).

The FVE parses the shader source code and extracts all floating point variables. The extracted variables are then presented to the users so that they can pick any of them to form a feature vector. In the shader code, the formed feature vector is extracted after the last of the selected variables is defined. This means that the variables used for the feature vectors should be initialized with a meaningful value. This is one of the limitations of the FVE. It is not as flexible as using the PEL directly, where the user can extract feature vectors from any place in the shader code. Another limitation is that the user can only use variables for the feature vectors, while in PEL any language expression is possible.

The FVE displays the names of the variables as extracted from the shader source code. These names may be confusing for the user and documentation is needed for the effective usage of the FVE. Therefore, we have included two special annotations, *name* and *desc*, in the PEL. These annotations are provided for programmers to document individual variables directly in the shader source code so that it is convenient to explore the shader program using the FVE. These annotations can change a displayed name of a particular variable and add a description to it. An example is shown in the following listing:

```
//. name Brightness
//. desc Brightness of the pixel
float br;
```

The two PEL annotations in the first two lines cause FVE to display *Brightness* instead of *float br;* in the list of variables. Additionally, the FVE displays a pop-up text with the given description on hovering the mouse over the variable.

This mechanism creates a level of abstraction between the shader-program source and the parameter exploration. It is the responsibility of the programmer of the shader to provide names and descriptions of variables so that a domain expert can understand them. By using expressive names and descriptions, the domain expert can explore the shader space without shading-language knowledge, or without a deep understanding of the particular shader program.

We designed our method in such a way that it can benefit domain experts and visualization experts. Visualization experts can use the PEL or the FVE for analysing their shaders by examining the effects of selected parameters. The FVE can also be employed for fast prototyping of new algorithms or visualization mappings derived from the original shader without recompiling the host application.

The visualization expert can prepare the shader for the exploration carried out by the domain expert by adding expressive domain-specific variable names and descriptions using PEL annotations without actually renaming the variables. The domain expert can then use the FVE for exploring the visualization mapping by changing relationships between its parameters and data attributes. The shader program is transparent to the domain expert because the FVE shows the domain-specific names and descriptions of individual variables available for the exploration.

The flexibility of the FVE is not as high as the one provided by the PEL directly. In case the FVE is not sufficient in a certain scenario, it is possible to use the PEL annotations together with the FVE. Naturally, in this case knowledge of the shader program is needed.

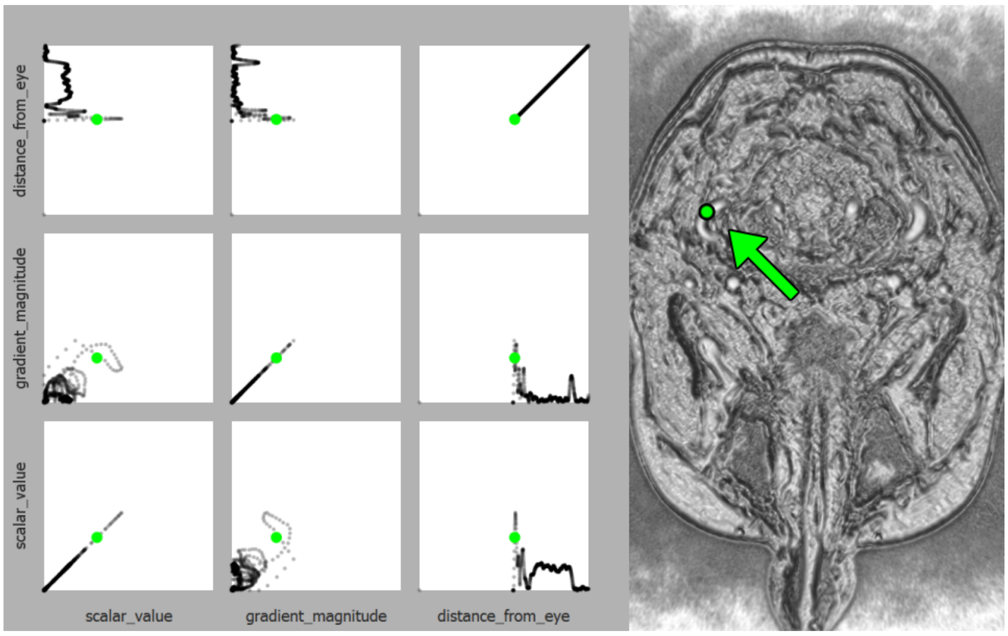
2.5.3 Visualization of Parameter Spaces

During classification, many potentially multi-dimensional feature vectors are stored in the GPU memory. In addition to visualizing their effects on the rendered image, it is also useful to examine their relationships. For instance, if VSSE is used for identifying suitable parameters for a multi-dimensional transfer function, the parameters of the examined shader program are sequentially chosen for the feature vectors. If a good set of parameters is identified, it is necessary to check whether some of the parameters are correlated. If a pair of the parameters correlates, one of them can be omitted from the designed transfer function.

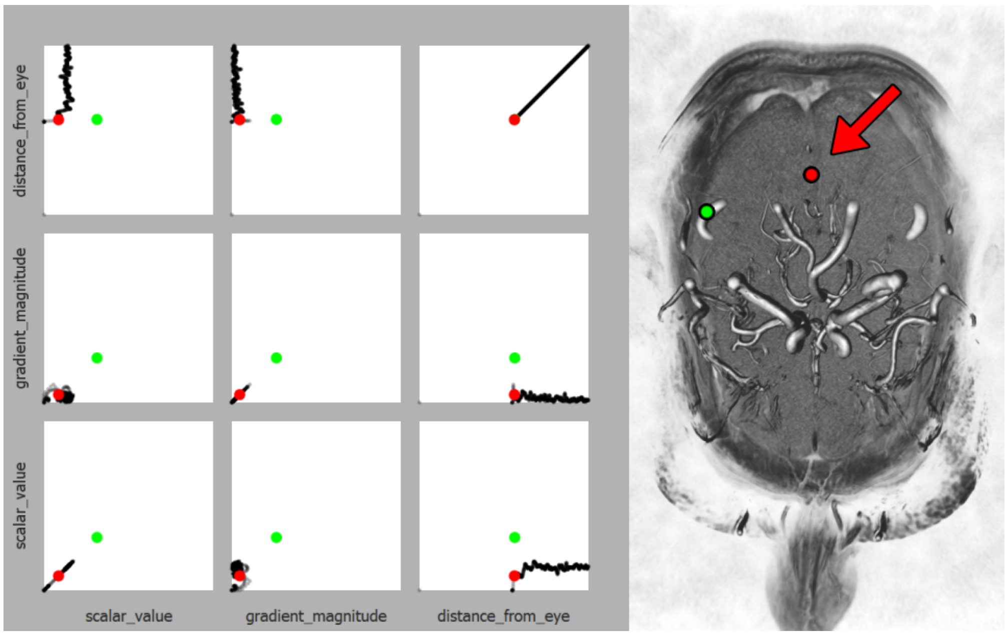
We introduce a visualization module which is able to depict all positively and negatively classified feature vectors, as well as extracted feature vectors for the currently selected pixels as a context. The visualization module uses a scatterplot matrix, which provides an overview of the bilateral relationships between elements of the vectors.

The visualization module is shown in Figure 2.5. In this case, feature vectors are composed of the following values: scalar data value (*scalar_value*), gradient magnitude (*gradient_magnitude*), distance from the virtual camera (*distance_from_eye*). The goal is to determine whether these three data attributes can be used for effective classification of blood vessels in an MRA scan of a brain. A part of a displayed blood vessel is classified as a positive example for being a vessel. Part of brain matter is classified as a negative example for being a vessel. The opacity of all rendered voxels now depends on how close their attributes are to the specified positive and negative examples.

The visualization module shows the relation of each pair of variables of the feature vector in a scatterplot matrix. For every user-defined classification, multiple values of the variables are extracted and composited into a feature vector. The visualization module can be used to determine the best composition strategy for calculating the feature vectors. In the example of Figure 2.5, the scatterplots with the distance from the virtual camera (*distance_from_eye*) on one of the axes



(a)



(b)

Figure 2.5: A scatterplot matrix displaying feature vectors. Positive vectors are marked with green circles, negative ones with red circles. (a) shows a magnetic resonance angiography dataset of a human brain, where a pixel displaying a vessel was classified as positive (green circle). In (b), a pixel inside the brain matter is classified as negative (red circle), which results in an enhanced view of the vessels. VSSE confirmed that the explored parameters can effectively classify blood vessels in the brain.

show peaks on the positively classified pixel, while there are no peaks in the negatively classified one. As the axis of the scatterplots is the distance to the virtual camera, the scatterplots actually show ray profiles. The ray profiles of both the positive and the negative example begin with values close to zero. This means the feature vectors should be composited using a weighted average, otherwise the positive and the negative examples could not be distinguished.

2.6 Implementation

The system is implemented in C++ using the Qt library. It provides an application programming interface for an easy integration into existing host visualization systems.

The shader augmentation engine is able to enhance shaders written in the GLSL language by replacing PEL annotations with GLSL code. For storing and loading feature vectors in OpenGL textures, the inserted GLSL code uses the EXT_shader_image_load_store extension.

If there is a request from the host application to classify a pixel, the host application has to provide the augmented shader with the screen space coordinates and the desired class (positive or negative) of the selected pixel. The augmented shader is then automatically switched to the classification mode. The pixel has to be rendered in this mode, so that its pixel-feature vector is calculated, classified, and stored. Other pixels can be rendered in the classification mode as well, but the code for feature vector extraction would be ignored for any pixel with different screen space coordinates from the pixel that was selected, e.g., by clicking on the rendered image.

The parameter-visualization module is implemented using GLSL shaders to access extracted feature vectors and render them in a scatterplot matrix. A geometry shader is used to create the visualization from the data stored in the GPU memory to minimize data transfer between GPU and CPU.

2.7 Use cases

In the following section we present two use cases of VSSE. We apply our method to two distinct shader programs to demonstrate its versatility.

2.7.1 Volume Rendering

To demonstrate the shader-space exploration abilities of our method, we employ it for the investigation of a direct volume rendering algorithm. For this use case, we take dual-modality data, i.e., a co-registered CT and MRI scan of a human head. The MRI data is displayed on the screen, while the CT data is used only to extract additional data attributes. We use VSSE to find out which of the data attributes can be employed to design a transfer function for the effective classification of the brain in this type of data.

We identified several data attributes to be taken into consideration. The data attributes are: scalar value from the MRI data (*voxMRI*), scalar value from the CT data (*voxCT*), gradient magnitude from the MRI data (*gmMRI*), gradient magnitude from the the CT data (*gmCT*), difference of the two scalar values (*voxDiff*), difference of the two gradient magnitudes (*gmDiff*), maximum of the two scalar values (*voxMax*), maximum of the two gradient magnitudes (*gmMax*).

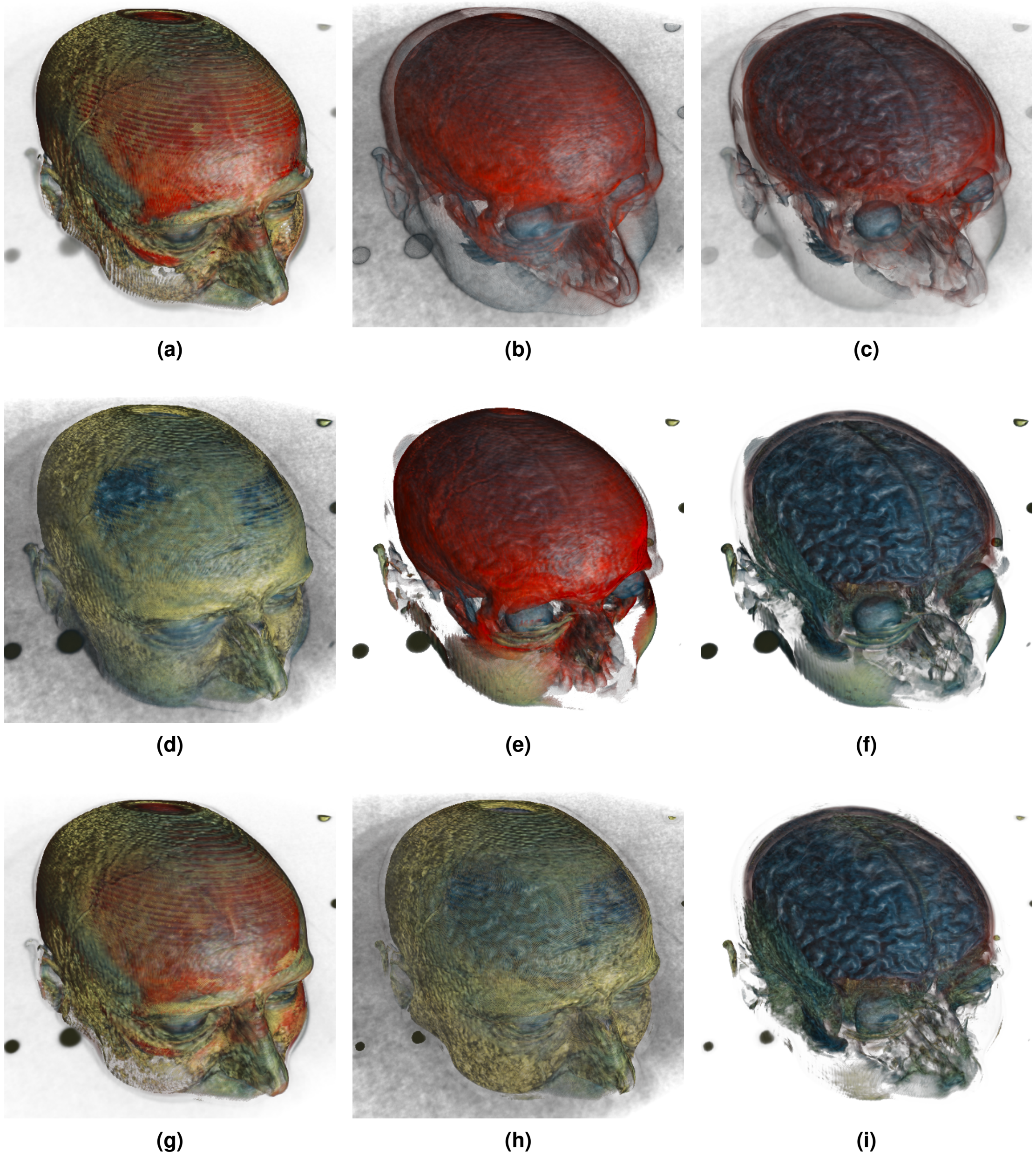


Figure 2.6: Visualizations of the dual-modality data with positively classified brain and negatively classified occluding tissues for different feature vectors. Feature vectors consist of these attributes: (a) voxCT; (b) voxMRI; (c) voxCT, voxMRI; (d) gmCT; (e) voxDiff; (f) gmCT, voxDiff; (g) gmDiff, voxMax; (h) gmDiff, gmMax; (i) gmMax, voxDiff

Using VSSE, we extract feature vectors for every processed voxel. The feature vectors are formed from various combinations of these data attributes. The method for the composition of the feature vectors is weighted average. The weight for every feature vector is the contribution of the respective voxel to the final pixel color. Therefore, feature vectors composited using this strategy accurately describe the pixels. Finally, we use the confidence value to modulate the opacity of every processed data sample. This means that data samples with feature vectors from the negative class do not contribute to the pixels of the rendered image, thus they do not occlude areas of interest.

Using the FVE we select a set of variables representing the data attributes to form a feature vector. We try various different feature-vector setups in order to find most suitable ones. Using a clipping plane, we reveal brain in the visualization of the MRI data. We positively classify one pixel inside the brain. Additionally, three pixels in the skull and other occluding tissues are classified negatively. We use the same user-specified classification for every feature-vector setup. Since we classified a pixel displaying the brain as positive, and pixels displaying occluding tissues as negative, the resulting visualization should reveal the brain. If this is not the case, we can conclude that the current set of data attributes is not suitable for designing a transfer function for the desired visualization mapping. By trying different sets of data attributes, we identify those suitable for classifying brain. Figure 2.6 shows the results. The best choices are in Figures 2.6c, 2.6f, and 2.6i, because the brain is revealed as expected.

2.7.2 Image processing

In this example, we apply our method to an image-processing shader implementing a bilateral filter. The bilateral filter is used for smoothing images while preserving edges. It replaces every pixel of the input image with a weighted average of surrounding pixels. The weights are determined by a two-dimensional Gaussian function and a color difference between the original and the surrounding pixel. If the difference is higher than a threshold, the weight of the pixel is zero. The result is a smoothed image with preserved edges. The application of the bilateral filter is for instance noise reduction.

The bilateral filter has two parameters: blurring radius and threshold. A higher blurring radius will result in the removal of more high-frequency noise, while the threshold determines how strongly the edges should be preserved.

We use our method to explore the effects of using a different threshold for every pixel of the rendered image. Since the threshold controls how edges are preserved, we apply the Sobel operator to calculate the gradient magnitude for every pixel and use it as a feature vector. The gradient magnitude is high for edges and low for homogeneous areas of the image. In this setup, the user can select examples of edges (positive examples) and homogeneous areas (negative examples) by clicking on them. These examples are subsequently generalized for calculating the threshold for each pixel.

Figure 2.7 shows two different selections of edge examples as well as the original image. In Figure 2.7b only strong edges are visible. In Figure 2.7c, a more subtle edge was selected as a positive example. The thresholds were modified so that more edges are visible.

2.8 Discussion and Limitations

Our proposed method is intended for analyzing existing visualization algorithms in order to gain a better insight into their inner working. As we show in section 2.7.1, our method enables the rapid exploration of shader space of a visualization algorithm resulting in the identification of data-attribute combinations interesting for a specific field. Achieving this goal by simple shader programming would require a cumbersome manual evaluation of the effectiveness of transfer functions for each visualization mapping. In section 2.7.2 we show that our method can be employed for a different type of shader analysis as well.

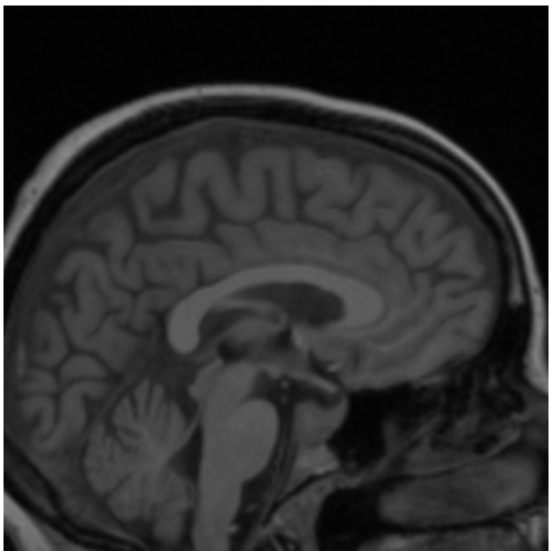
There are several limitations in our method that should be addressed in future work. Currently, the shader-augmentation engine is able to parse only GLSL shaders. However, the presented concepts do not depend on using shaders. There is no principal obstacle to provide implementations for other languages realizing the visual mapping.

We have evaluated the performance of the system on a volume-rendering shader displaying various datasets. For a dataset of 424x279x190 voxels and a window size of 800x600 pixels, the shader was running at around 120 fps. After augmenting the shader with the PEL annotations and classifying five pixels using three-dimensional feature vectors, the framerate dropped to approximately 20 fps. This performance drop is caused by additional GPU memory accesses for extracting values for the feature vectors and reading them back to the CPU for the calculation of the classification of every processed voxel. The timings were obtained with a GeForce GTX 480 graphics card.

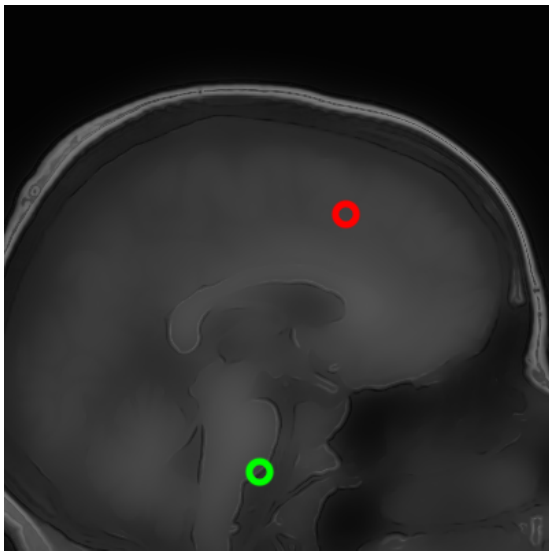
2.9 Conclusion

We have proposed visual shader-space exploration, a method for the visual analysis of GPU shader programs. The method enables users to explore the effects of various parameters of visualization algorithms. It provides a visual feedback based on a user-specified semantic classification of parts of the rendered images. We have implemented the method as a set of C++ classes so that it is independent of the underlying visualization system.

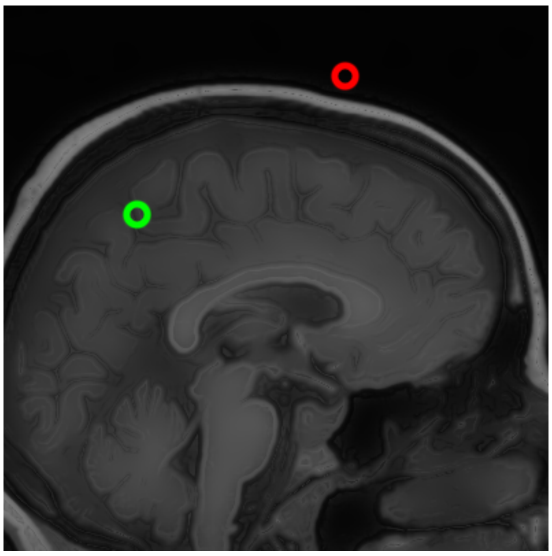
The implementation could be further enhanced by adding support for different language back-ends (such as OpenCL, CUDA, etc.). Various code optimizations would improve the performance on large datasets, e.g., using caching for the membership functions.



(a)



(b)



(c)

Figure 2.7: Using VSSE for exploring possibilities of variable thresholds of a bilateral filter. In (a) the original image is shown. In (b) and (c) one positive example of an edge (green circle) and one negative example (red circle) was chosen. The thresholds for individual pixels are modified so that the examples are generalized for the whole image.

CHAPTER 3

Management of Localized Parameters for Data Subsets

This chapter is based on the following publications:

P. Mindek, M. E. Gröller, and S. Bruckner. Managing Spatial Selections with Contextual Snapshots. *Computer Graphics Forum 33, 8 (2014)*, pp. 132—144. DOI: 10.1111/cgf.12406.

P. Mindek, S. Bruckner, M. E. Gröller. Contextual Snapshots: Enriched Visualization with Interactive Spatial Annotations. In *Spring Conference on Computer Graphics*, SCCG'13, pp. 49—56, New York, NY, USA, 2013. ACM. SCCG Best Paper Award.

To display data in such a way that the desired information is conveyed, it is necessary to adapt the visualization algorithm for the specific instance of the data under examination. In other words, the interactive process of visual data exploration requires the selection of appropriate parameter settings of the visualization system. However, it is rarely the case that the dataset is in its entirety uniform with respect to the appropriate parameter values. Multiple aspects of the dataset are sequentially discovered in interactive sessions, where the parameters are iteratively refined between the individual findings. These findings are commonly interrelated as parts of the big picture constructed during the data-exploration process.

For the purpose of the interdisciplinary communication or data presentation, it is essential to store the whole data-exploration process. Various provenance-tracking systems, such as VisTrails [9], have been designed for this purpose. In these systems, the histories of the visualization-parameter changes are recorded so that the individual findings discovered during the visualization session can be reconstructed. However, in order to explore the connections between the individual findings, it is necessary to display them in the same views. This is possible by bounding the influence of the parameter settings to specific spatial locations only. To enable such a process, we propose a system for the management of spatial selections in visualization images.

Spatial selections are a ubiquitous concept in visualization. By localizing particular features, they can be analyzed and compared in different views. However, the semantics of such selections often depend on specific parameter settings and it can be difficult to reconstruct them without additional information. In this chapter, we present the concept of contextual snapshots as an effective means for managing spatial selections in visualized data. The selections are automatically associated with the context in which they have been created. Contextual snapshots can also be used as the basis for interactive integrated and linked views, which enable in-place investigation and comparison of multiple visual representations of the data. Our approach is implemented as a flexible toolkit with well-defined interfaces for integration into existing systems. We demonstrate the power and generality of our techniques by applying them to several distinct scenarios such as the visualization of simulation data, the analysis of historical documents, and the display of anatomical data.

3.1 Introduction

Visual analysis of large datasets often requires displaying of various subsets of the examined data using different visualization techniques. For this purpose, linked views and integrated views are commonly employed. The investigation of multivariate or otherwise complex data may require a specification of spatial regions which are to be examined individually using integrated or linked views. Creating spatial selections, or brushing, is a widely used method for specifying such regions. Brushing techniques are also often employed to specify a *degree of interest* (DOI) function for focus+context visualization as described by Furnas [41]. Smooth brushing concerns the specification of a non-binary DOI function, which defines a continuous transition between focus and context data.

Another aspect of visualizing complex datasets is a frequent need for creating annotations in the rendered images. The annotations assign semantics to parts of the image. They are useful for providing additional insight into the data, or for keeping provenance information. The annotations are related to the spatial selections, since they refer to particular spatially bounded image regions. The annotations are also bound to the current parameter settings of the visualization. For instance, if a structure is annotated in a volume dataset, the annotation loses its meaning when the transfer function, the viewing angle, or other parameters change in such a way that the structure in question is no longer visible. In such cases, it is necessary to keep track of the visualization settings together with the annotations.

We propose a method for managing arbitrary selections in the image space of the visualizations. We define several terms for the purpose of describing the proposed method. A *selection* is a non-binary DOI function in image space. A *visualization snapshot* is a set of parameter values describing the state of the visualization system at a particular point in time. Finally, we introduce the concept of *contextual snapshots*. A contextual snapshot is an entity which holds multiple selections together with a visualization snapshot. The visualization snapshot provides context for the associated selections. By keeping the selections in contextual snapshots, it is possible to recover the states of the visualization system in which the selections have been created. The contextual snapshots allow us to work with multiple selections created in different states of the visualization system.

To demonstrate the proposed concept, we implemented the contextual snapshots in three scenarios, one of which is a volume visualization application. The application displays a multivariate 4D hurricane dataset. The user can create selections in image space by using a lasso metaphor. The selected data can be further analyzed using linked views. We use this example to introduce the parts the contextual snapshots consist of, and explain how they work together.

In applications such as 4D data visualization, it is sometimes necessary to select and annotate the visualized data multiple times while the visualization parameters are changing. Contextual snapshots provide a basis for keeping track of these interactions. In current systems, selections made in image space have to be processed before the image changes. Otherwise the selections will become invalid with the new parameter settings, which we refer to as context. Contextual snapshots record the selection together with the context, so that it can be processed even after the context has changed.

3.2 Related Work

We propose a method for managing spatial selections in image space. There are various scenarios where multiple selections are made in order to achieve a certain goal. The goal might be to select subsets or features of a dataset. Furnas [41] present DOI functions for the specification of focus data. Doleisch and Hauser [34] use a DOI function obtained by smooth brushing to modify the visualization mapping in 3D flow visualization. Doleisch et al. [33] present a framework for the specification of data features visualized in several linked views. Ulinski et al. [110] propose two-handed methods for creating selections in volume rendering. Unger et al. [111] use smooth brushing in the visualization of statistical characteristics for subsets of large datasets. Various methods for increasing the usefulness of 3D scatterplots incorporating brushing have been developed [67, 88]. Streit et al. [102] propose a model-driven design process for exploring multiple linked datasets. Yu et al. [124] discuss methods for selecting data in large 3D point clouds by screen-space interaction. In visualization applications which employ brushing or similar techniques, the user interaction is typically limited to the common context. Contextual snapshots remove this limitation by providing means for keeping the context for each individual interaction instance.

Gerl et al. [43] incorporate brushing on renderings of data attributes for the specification of semantics in volume visualization. Guo et al. [48] introduce a sketch-based interface for direct volume rendering which replaces the traditional way of transfer function design. Wei et al. [117] propose a sketch-based interface for an interactive 3D vector field exploration. The concept of contextual snapshots is designed in such a way that the spatial selections could be employed to handle the just mentioned types of user interaction. Contextual snapshots increase the scalability of such interaction methods. They allow the system to manage multiple interaction instances simultaneously, while each instance can be meaningful in a different context.

In addition to the concept of contextual snapshots, we propose a method for combining them with various views of the visualized data. The integration and linking of multiple views has been extensively explored [5, 105]. Bier et al. [12] propose a see-through interface as a natural way of displaying additional data. Balabanian et al. [7] introduce a framework for the specification of visualization parameters for time-varying data. Rungta et al. [93] present ManyVis - a framework for easy integration of existing applications to create custom visualization tools. Santos et al. [94] propose VisMashup, a framework for simplifying the creation of custom visualization applications. The authors of VisMashup combine various visualization pipelines to create a new visualization application. In our work, we aim at extending existing visualization pipelines with interaction possibilities.

Contextual snapshots can also be used for preserving user-created provenance information for a visualization. Bavoil et al. [9] propose VisTrails. It is a system for creating and maintaining visualization pipelines with the possibility to execute them and to record their provenance information. Our method differs form VisTrails in that the user can create spatial selections of the explored data in a specific context and annotate them to store the visualization provenance information. This provides a strong link between the provenance information and the underlying data. Heer et al. [53] present a design space analysis of history-keeping systems. Kreuseler et al. [68] propose an approach to include a history mechanism into a visual data-mining framework. Groth and Streefkerk [46] present a method for capturing the history of the knowledge-discovery process using a visualization system with the ability to create annotations for provenance information. Ellkvist et al. [36] discuss an architecture for provenance interoperability between multiple sources. In contrast to these systems, contextual snapshots provide means to insert annotations into particular spatial data at a particular stage of the visualization session. As the annotations are automatically linked with the current context, they store provenance information besides their actual content.

3.3 Overview of Contextual Snapshots

Many visualization systems use brushing, selections, and linked views to provide means for the exploration of complex datasets. There are various tools for specifying the selections and they usually serve only one specific purpose. The idea of contextual snapshots is to harness a single mechanism of 2D spatial selections for different tasks, such as data selection and manipulation, data annotation, or specification of DOI functions. In contextual snapshots, this is achieved by the following concepts: multiple selections can be stored and each of them can be created in a different context, i.e., parameter settings of the visualization system; an algorithm of transforming the user input to the DOI functions of the selections is interchangeable; each selection can be linked with a number of additional views which we refer to as *embedded visualizations*. It is possible to display, i.e., embed, them directly in the visualization image. Embedded visualizations are interactive and they can display arbitrary graphical user interface elements or visualize data specified by a corresponding selection.

In the example application, the selections can specify a spatial region in image space of a volume visualization. An embedded view which displays the histogram of the volume data in the specified region is linked with each selection. Contextual snapshots calculate a histogram of the selected voxels and provide it to the embedded visualization. How these data are used depends on the implementation of individual embedded visualizations. Another embedded visualization is a simple text field. It does not display the selected data region, but it allows users to type in arbitrary text-based annotations. Such an embedded visualization is linked with every selection, thus providing a possibility to annotate selected data subsets.

3.3.1 Concept of Contextual Snapshots

A visualization system may apply various parameters to modify the visual mapping. With respect to contextual snapshots, the values of a chosen subset of said parameters, the *visualization snapshot*, define a state of the visualization system. For any state of the system, a user can create several selections in the rendered images. A contextual snapshot stores a visualization snapshot together with all selections created when the state of the visualization system corresponded to this visualization snapshot.

In the hurricane-visualization application, we use the position and the orientation of the virtual 3D camera as the parameters stored within the visualization snapshot. Therefore, each image-space selection is bound to one 3D camera view. The selection is valid only if the volume is rendered using this camera view. Other parameters, such as current timestep, are not stored in the visualization snapshots. Changing these parameters does not make the selections invalid. This is application-specific and in the given case, it allows users to explore how the hurricane data change over time in specified spatial regions. If contextual snapshots are integrated with a visualization system, the system integrator has to choose the set of parameters which appropriately define the context for the selections.

In this work, a selection is a function $f : \mathbb{R}^2 \rightarrow [0, 1]$ which specifies a DOI for each pixel. The contextual snapshots do not assume any particular definition of this function. Therefore, arbitrary types of image-space selections can be used. This is demonstrated in sections 3.5 and 3.6.

A contextual snapshot c_i is defined as:

$$c_i = (v_i, s_i, t_i) \tag{3.1}$$

$v_i = \{(p_0, x_0), ..., (p_{n-1}, x_{n-1})\}$ is a visualization snapshot containing n parameters of the visualization system, describing its state at one instant. The visualization snapshot consists of n pairs (p_j, x_j), where p_j is a parameter name and x_j is its value. $p_0, ..., p_{n-1}$ are the same for all contextual snapshots, while the parameter values $x_0, ..., x_{n-1}$ are specific for each visualization snapshot. $s_i = \{f_0, ..., f_{m_i}\}$ is the set of all selections created at the state, or context, described by v_i. $f_0, ..., f_{m_i}$ are the user-defined, real-valued selections as described above. Finally, t_i is a thumbnail image of the visualization in the state described by v_i.

The idea of contextual snapshots is based on the fact that the semantics of the user-made selections depend on what is currently displayed. If the user selects a particular feature in the image, the selection is meaningful only until the way how the feature is displayed changes. Therefore, we extract a visualization snapshot every time a new selection is created. The visualization snapshot is linked to the selection to create a contextual snapshot. The contextual snapshot then provides a reproducible spatial selection related to what was displayed when the selection was made. It stores the appropriate visualization context in the form of the values of the visualization-system parameters. All selections with the same visualization snapshot are stored together within one contextual snapshot.

The strength of contextual snapshots is that they can maintain multiple selections created in different states of the visualization system. The information stored within a contextual snapshot can be used to restore the given state, so that the selections can be displayed and actively

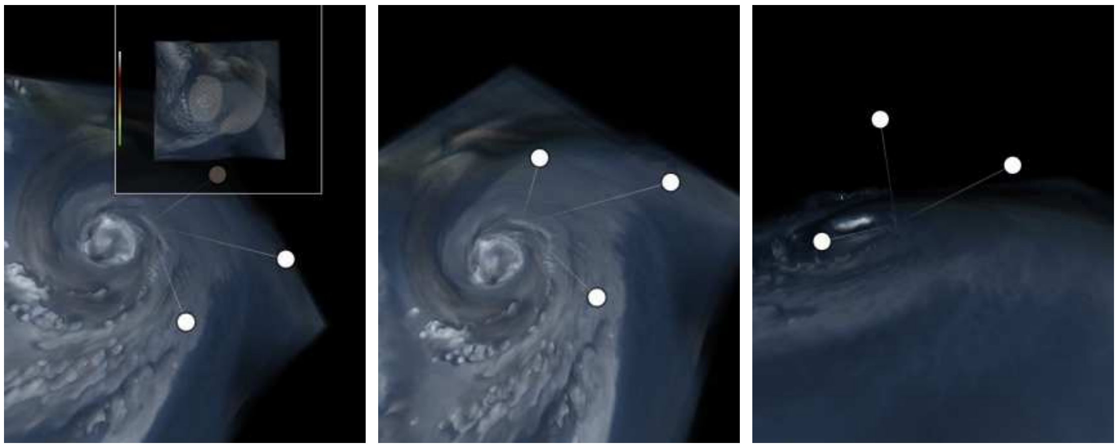
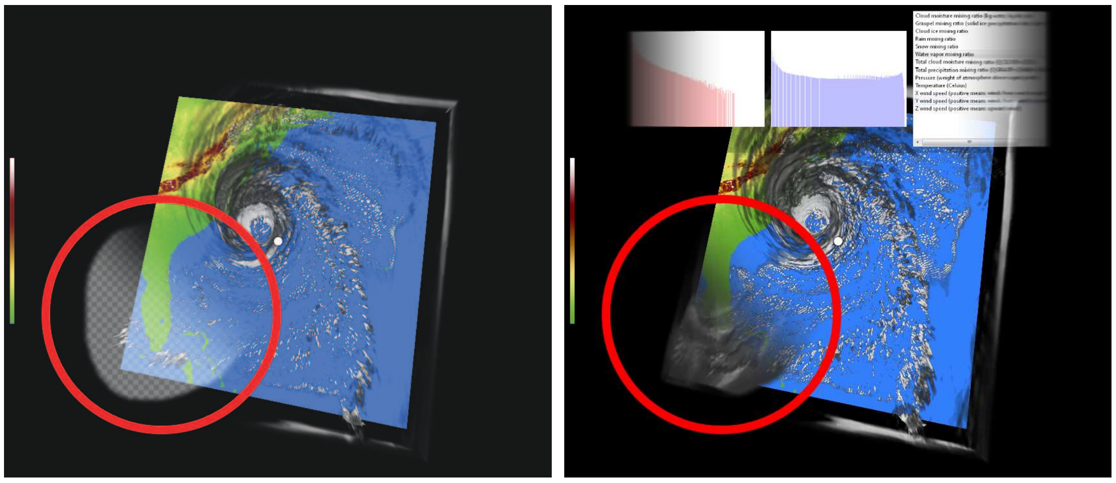

(a) (b) (c)

Figure 3.1: Interactive anchors (white circles) representing individual contextual snapshots. For better 3D orientation, the anchors are connected with the origin of the coordinate system by a thin line. (a) shows how the thumbnail of an anchor can be displayed. (b) and (c) show the anchors from different camera views.

used. By restoring the state of the visualization system according to the individual contextual snapshots, it is possible to browse all selections created within a visualization session.

Contextual snapshots are represented by icons which we refer to as anchors. The anchors are embedded in the original visualization as interactive graphical elements. They constitute abstract previews of the corresponding contextual snapshots. For instance, in the hurricane-visualization application, the anchors are positioned in 3D space to represent the camera positions when the respective contextual snapshots have been recorded. An anchor can also display a thumbnail of how the visualization looked like when the respective contextual snapshots have been created. The anchors are interactive and they are used to restore the visualization-system state to the respective contextual snapshot. They serve as a user interface for browsing through the contextual snapshots created during the visualization session. Figure 3.1 shows the graphical representation of anchors.

The visualization-system parameters used as context for the selections vary in different applications. The graphical representation of anchors can be customized to convey the information stored in the contextual snapshot, as demonstrated in Section 3.6. The default implementation uses the viewing-transformation matrix of the camera to position the anchors in 3D space. In this case, the position of the anchor conveys the position of the camera at the time when the contextual snapshot has been recorded. This is only suitable for visualizations using a 3D camera. For other use cases, we allow programmers to set an arbitrary screen-space position for each anchor. This way, the anchors can be positioned with respect to a feature of the visualization to potentially reveal the semantics or content of the contextual snapshot. This is demonstrated in the use case described in Section 3.5.



(a) (b)

Figure 3.2: (a) A selection (marked with the red circle). (b) An integrated view of two variables using the selection after its activation. Histograms are shown for both variables as embedded visualizations. The third embedded visualization is a variable picker. It shows a list of the data variables, where the user can choose which one is displayed.

3.3.2 Embedded Visualizations as Linked Views

To broaden the possibilities of using context-aware selections, we provide a method for linking interactive embedded visualizations for each selection. The additional visualizations can show different aspects of the selected data, or they can display comparisons of various selected areas. To demonstrate different ways how the selections can be used, we implemented the following embedded visualizations for the hurricane-visualization application: a histogram of selected data values, a text-based annotation widget, and a variable picker.

The embedded visualization displaying the histogram of selected data values can be linked to multiple selections at once. It shows histograms for individual selections in overlays so that they can be easily compared. A separate text-based annotation widget is linked to each selection. It provides the user with the possibility to type in a short description of the selected data subset. The variable picker is a GUI element which displays a list of all variables present in the dataset. The chosen variable is displayed in those parts of the image, where the selection has been created. Figure 3.2 shows how the picked variable is integrated with the rest of the visualization by a smooth transition. The smooth transition is due to the non-binary DOI function of the selection.

In our method, each image-space selection can be linked with multiple embedded visualizations. Each embedded visualization has access to the data subsets specified by the selections to which they are linked. A single embedded visualization can be used to display and compare aspects of different subsets of the explored data by simply linking it with multiple spatial selections.

For the purpose of displaying the embedded visualizations, we propose a mechanism for

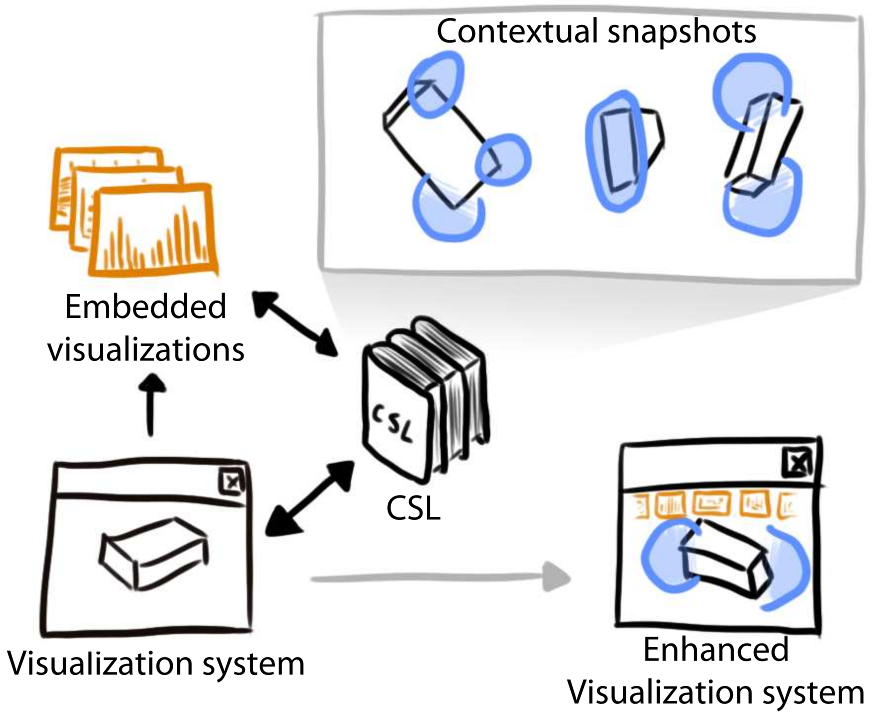


Figure 3.3: Overview of the system. The black arrows represent the data flow between the visualization system, the embedded visualizations and the Contextual Snapshot Library (CSL). The gray arrow denotes the transition from the original rendering to the rendering with the enhanced visualization.

activating individual selections. One or multiple selections can be activated by the user at once. In this case, only those embedded visualizations are displayed which are linked with every activated selection. The rationale for this mechanism is that the embedded visualizations can show different aspects of the data specified by multiple selections at once. This way, the selections and their embedded visualizations can be used to compare several data subsets.

A sketch-based interface is used for activating selections. The user activates selections by painting a stroke in image space. The selections which are crossed by the stroke are activated and subsequently their linked embedded visualizations are displayed. The embedded visualizations are grouped together in a sliding bar, which is displayed either at the top, the bottom, the left, or the right side of the visualization image. The position of the sliding bar is determined by the direction at the end of the stroke which was used to activate the selections. The interaction method of using a stroke was chosen so that an arbitrary subset of the selections can be activated, which might be difficult with various standard selection mechanisms.

The sliding bar is capable of showing several embedded visualizations at once. In case there are more embedded visualizations for the activated selections than actually fit on the screen, the sliding bar enables scrolling of its content. The scrolling is executed by an animated transition, so that the users have a visual feedback on the direction of the scrolling. The sliding bar fades

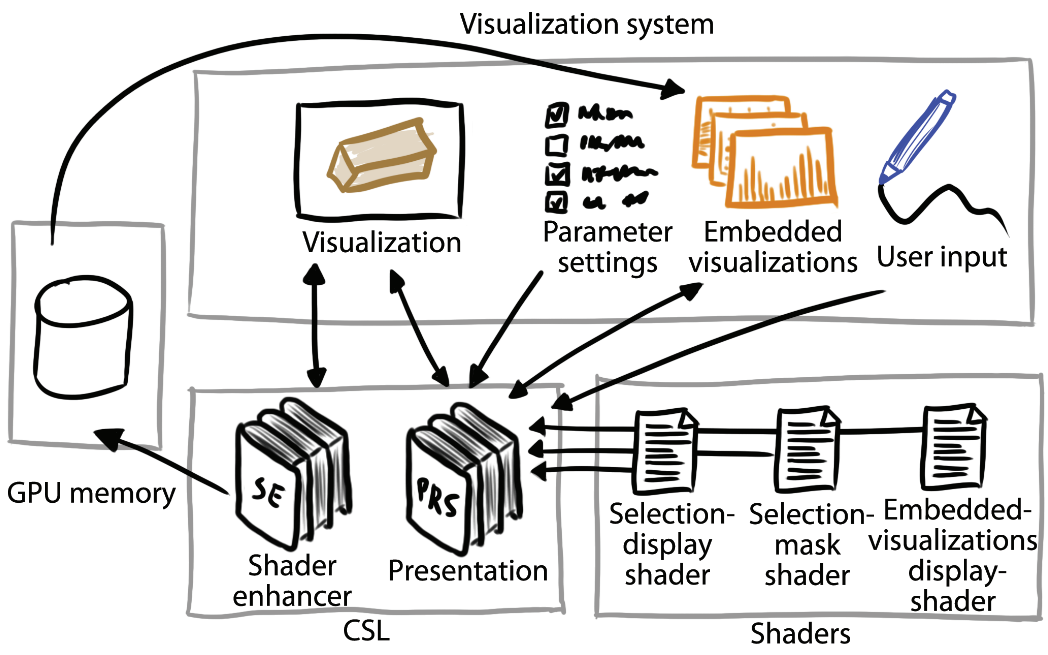


Figure 3.4: Architecture of the visualization system integrating the CSL. The arrows denote the data flow.

to the original visualization on both sides for better integration. A gradual blurring filter was used on both sides of the sliding bar, so that the attention of the users is guided to the embedded visualizations currently shown in the middle of the bar.

Displaying several embedded visualizations at once provides users with an overview of the additional data depicted for the activated selections. However, to facilitate interaction it might sometimes be necessary to enlarge individual embedded visualizations. For this purpose, the embedded visualization displayed in the middle of the sliding bar can be switched to a so called maximized view. If the view is maximized, the embedded visualization is displayed on the whole screen rather than just in the sliding bar.

3.4 Contextual-Snapshot Architecture

Contextual snapshots are meant to be used in existing visualization systems. We have implemented contextual snapshots as a library which can be integrated with an underlying visualization system on the source-code level. We call it Contextual Snapshot Library (CSL). The CSL is responsible for rendering the anchors, the selections, and the embedded visualizations into the original visualization image. Additionally, it provides an interface for the data transfer between the selections and the embedded visualizations. It also handles user input so that the anchors and embedded visualizations are interactive. Contextual snapshots integrate the visualization and the graphical elements for interactive data exploration and annotation. This approach is particularly well-suited for the rapidly growing area of mobile devices such as tablets where the display also serves as the input device.

Figure 3.3 shows the data flow between the visualization system, the embedded visualizations, and the CSL. The visualization system gathers user input and transmits it to the CSL. The CSL stores contextual snapshots generated from the user input. Additionally, it renders the selections, the anchors, and the embedded visualizations into the original visualization image. The result is an enhanced visualization system.

The CSL renders all of the graphical elements (anchors, selections, embedded visualizations, as illustrated in Figures 3.1 and 3.2) into the original visualization and provides the result as a texture. The underlying visualization system can be modified to display this texture so that the graphical elements of the contextual snapshots are visible.

Figure 3.4 shows the overall architecture of an existing visualization system using the CSL. The library itself is split into two parts. The part responsible for managing the contextual snapshots, interaction, and rendering of the graphical elements, is called Presentation (PRS). To exploit the capabilities of modern GPUs, the PRS uses several shaders to render all the graphical elements. The selection-mask shader transforms user input into the DOI function of the selection. The selection-display shader renders the selection on the screen. The embedded-visualizations display-shader renders the sliding bar. Each of these shaders can be exchanged to modify how selections are treated.

The functionality of the PRS can be extended by the Shader Enhancer (SE). The SE is an auxiliary tool for the data transfer between individual modules of the visualization system. It stores data specific to individual selections of the contextual snapshots in the GPU memory so that it can be used in different visualization pipelines of the system. The SE simplifies the utilization of the selections in the visualization system by providing access to all data subsets specified through the corresponding DOI functions. The motivation of storing the data in the GPU memory is that GPU implementations of visualization algorithms can access the data without having to transfer them to CPU memory.

The CSL is implemented in C++, using the Qt library. It uses OpenGL for rendering of the graphical elements and for the data exchange with the visualization system via textures. The specifications of contextual snapshots can be stored on the hard drive in XML format. The selection masks and the thumbnails are stored as PNG files. The CSL also provides functionality to load this information and recreate all the contextual snapshots for the currently visualized data.

3.4.1 Contextual Snapshots

If the parameters of the visualization system are set so that their values match the values stored in visualization snapshot v_i (as described in Section 3.3.1), the contextual snapshot c_i is activated. The default implementation requires that each pair of corresponding values are equal in order to activate the respective contextual snapshot. However, it is possible to provide custom matching functions for each parameter data-type or even for individual parameters. The custom implementation of the matching functions can treat parameter values, whose difference lies in a certain range, as a match. This way, the contextual snapshot c_i is activated even if the parameter values set in the visualization system are not exactly equal to those stored in v_i, but the values of the corresponding parameters are similar enough.

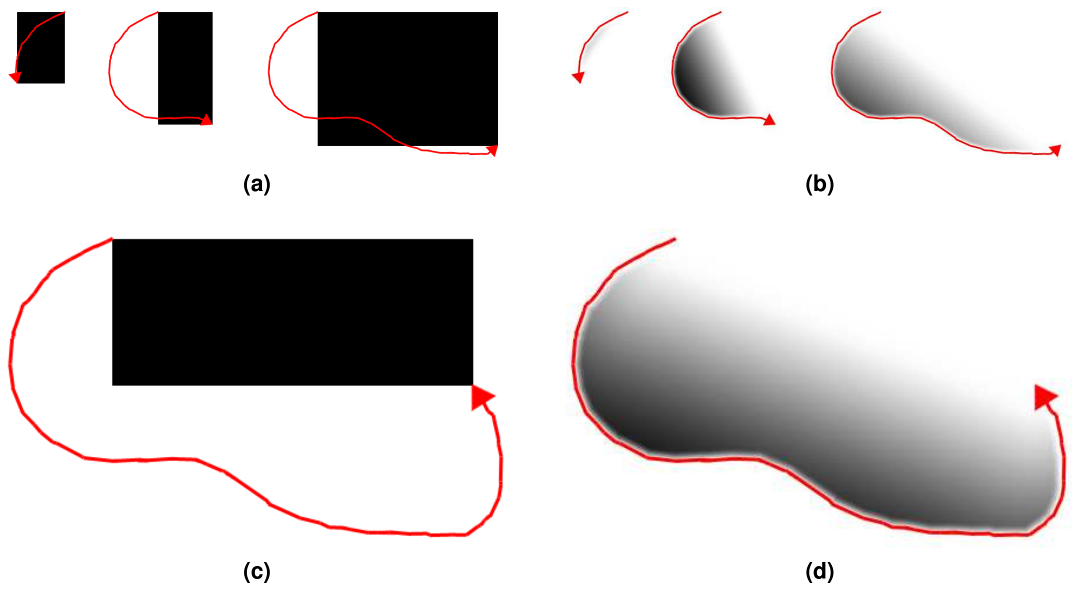


Figure 3.5: Two different selection-mask shaders generate selection masks (black means fully selected, white means not selected at all) for the same selection stroke (in red). (a) and (b) illustrate sequences of creating the stroke, while the selection mask is continuously generated at every step. (c) and (d) illustrate the final selection masks for the given stroke. The gradual change in the level of selection enables smooth brushing.

Another possibility to activate contextual snapshots is by using anchors. The visualization system can request the CSL to select an anchor which is displayed at a certain position in image space, e.g., on mouse click. The contextual snapshot represented by the selected anchor becomes active. The state of the visualization system is changed so that it corresponds to the active contextual snapshot. All selections belonging to this contextual snapshot can now be displayed.

The activation of the contextual snapshot is accompanied by an animated transition from the values of the parameters it stores to the current values in the present state of the visualization system. For individual parameters, different transition functions can be used in order to achieve an appropriate interpolation for a specific data type, e.g., *Slerp* [97] for rotation matrices. This enables smooth transitions between system states while switching between them. In the hurricane-visualization application, the state of the visualization system is defined by the camera position and orientation. Therefore, the activation of a contextual snapshot causes the visualization system to smoothly change the 3D camera to the view with which the contextual snapshot was recorded.

Internally, parameter values are represented by Qt's QVariant class, so that any parameter type can be used. However, transition and matching functions have to be implemented for every type. The CSL provides implementation of these functions for several common types, e.g., integer and real numbers, transformation matrices, 2D and 3D vectors. These can be modified to meet specific requirements from individual applications. New transition and matching functions

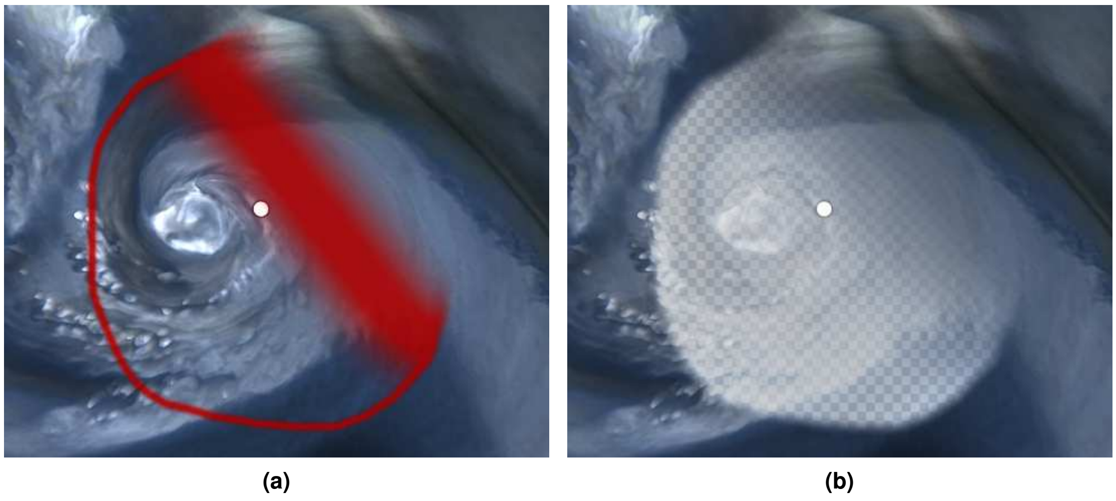


Figure 3.6: Two different selection-display shaders. Both shaders are able to show the fuzziness of the selection.

can be added by the system integrator to extend the CSL with new parameter types.

3.4.2 Selections

Selections are created by calling functions of the CSL's API. If the contextual snapshot c_i is active, the newly created selection is automatically assigned to it, i.e., it is added to s_i (Definition 3.1). Otherwise a new contextual snapshot is created from the current values of the visualization-system parameters and the selection is assigned to it. This mechanism enables multiple selections created in the same context to be assigned to a single contextual snapshot.

The process of creating a selection consists of three steps: recording of the user input, transforming the input to a DOI function in image space, displaying the DOI function on screen to represent the selection. In our approach, we made a clear separation between these steps. Each of them is implemented as a stand alone shader program with clearly defined input and output. Because of this separation, it is possible to customize the process of creating selections for various applications. Examples are selections using the lasso metaphor or rectangular selections. In both cases, the only component of the CSL that is exchanged is the shader realizing the transformation of the user input to the DOI function.

If a user interacts with the visualization system to create an image-space selection, a series of points in the image space is recorded. We refer to this series as a selection stroke. The selection stroke can be created by mouse, graphics tablet, or a similar input device. The CSL transforms the selection stroke to a grey-scale mask representing the non-binary DOI function. Pixel luminosity encodes the degree of interest in the respective point. This selection mask is generated by a selection-mask shader. The input of the shader program is the selection stroke encoded in a one-dimensional texture. The output of the shader program is the selection mask.

Depending on the shader program used to generate the selection masks, the selections may enable the visualization system to realize smooth brushing. We provide several different shader programs for generating the selection masks. Figure 3.5 shows how the selection masks generated by two different shaders look like for the same selection stroke. The shader illustrated in Figure 3.5c creates a simple rectangle based on the first and the last point of the stroke. This is a common way to create a rectangular selection. The shader shown in Figure 3.5d uses the so called lasso metaphor. The stroke defines a closed polygon whose interior is filled. We enhance the lasso metaphor to account for selection uncertainty by introducing a smoothing of the edges. The smoothing between the first and the last point of the stroke is stronger if these two points are farther away. The other edges are smoothed by a constant factor. This enables users to control the amount of smoothing as well as its spatial location. The motivation for such an approach is that if a user does not fully enclose the selected region, the selection status of the area between the beginning and the end of the stroke remains uncertain.

The selections are displayed on the screen using the selection-display shader. The way how the selections are displayed is important for specific applications. Figure 3.6 shows two different choices how the selections can be displayed. In Figure 3.6a, an example of outline drawing is shown. The red color marks borders between selected and unselected areas. The thickness of the border denotes the uncertainty of the selection in that particular area. In Figure 3.6b, the selection is displayed using an overlay texture. This method shows the selected area in a clear way, but it also partially occludes displayed data underneath.

3.4.3 Data transfer

The selection masks are stored in the GPU memory as a texture array so that the visualization system can access them at all times and employ them in the visual mapping. We used this ability in the hurricane-visualization application to create a smooth transition between the two visualized variables.

The SE can be used to further increase possibilities of the selections. It allows the visualization algorithm to extract processed data samples to the GPU memory. The activated selections specify the degree of interest, which is used to automatically weight every extracted data sample. The extracted data can then be accessed in the embedded visualizations.

Currently, we provide an implementation of the SE for GLSL shaders. The SE inserts GLSL code for the extraction of data samples and their weighting by the selections' DOI functions in the visualization shader before it is compiled. The inserted code uses atomic operations and the GL_EXT_shader_image_load_store extension to output desired data. After the extraction, the data are available to the embedded visualizations as a texture stored in the GPU memory. It is possible to extend the SE for other languages as well.

3.5 Application Example - Historical Document Analysis

In addition to the example application introduced in Section 3.1, we present two more use cases of contextual snapshots. In the first one, we take a simple book reader application. A page spread consisting of two pages of a manuscript is displayed. A bar showing the current page within the manuscript is located below the pages. For a better user experience, a simple page turning animation is realized whenever the current page changes.

We combine our method with the described book reader application in order to add functionality enabling the users to employ it as an advanced manuscript analysis tool. We used the CSL to implement spatial selections on the displayed pages. The CSL automatically binds every new selection to the current page, so many selections on different pages can be created.

In this example, we demonstrate how the selections can be incorporated into an existing system, how anchors can be positioned on the screen to serve as bookmarks for individual selections, and how the data specified by the selections can be displayed in the embedded visualizations. This example also demonstrates the possibility to compare data from multiple selections in the same embedded visualization. The embedded visualization can further contain various interactive elements, such as JavaScript-enabled web pages.

3.5.1 Manuscript Visualization

The dataset taken in this example consists of 723 high resolution photographs of pages of the Venetus A, a tenth century (AD) manuscript of the Iliad catalogued as Marcianus Graecus Z. 454, now 822. In addition to natural light photographs, some of the pages were recorded using UV photography as well. The UV light photographs were taken in order to reveal some details of the manuscript which were hardly visible with natural light. Together with the photographs, the transcript of the Iliad in ancient Greek was available as well.

For demonstration purposes, we manually preprocessed the acquired data. Some of the UV photographs were registered with the natural light photographs so that they could be easily used in the application. Additionally, appropriate passages of the transcript were matched with some of the photographed pages.

The described book reader application is only capable of displaying the natural light photographs. The book reader shows an icon for those pages where the UV data is available. The goal in this example is to use the CSL to display parts of the UV photographs on selected regions of interest. The regions of interest are rectangular areas specified for the pages with UV data available.

3.5.2 Manuscript Visualization Enhancement

Figure 3.7a shows the book reading application without the enhancements. Figure 3.7b depicts the application enhanced by using the CSL. The contextual snapshots were used to manage the selections. The selections are employed to show parts of the UV photograph of the page. The transcript of the page as well as color histograms of the selected parts of the natural light and UV photographs are displayed on the left side of the image. These additional visualizations are linked with the activated selections. The activated selections are concurrently used to display parts of the registered UV photograph of the selected page. Any of the additional visualizations can be maximized, as shown in figure 3.7c.

The only parameter with an impact on the manuscript visualization is the index of the current page. This parameter determines which pages of the manuscript are displayed. As the turning of

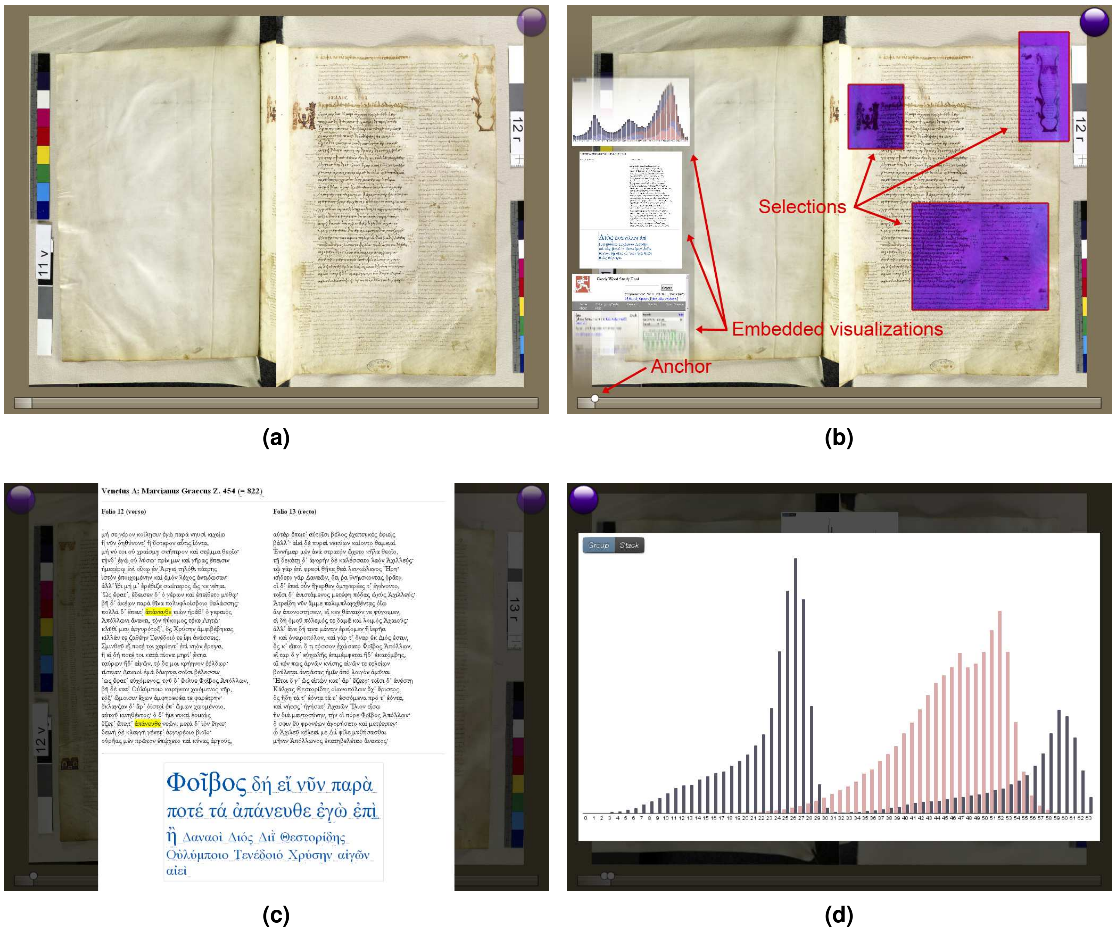


Figure 3.7: (a) The book reader application displaying the Venetus A manuscript. The icon (violet circle) on the top right corner indicates the availability of UV light data for this particular page. (b) The book reader application enhanced with the CSL to show additional data. An anchor of a contextual snapshot, selections, and embedded visualizations are displayed. (c) The transcript of the displayed pages with the tag cloud in the maximized view. A word selected in the tag cloud is highlighted in yellow. (d) Histogram of a selected part of a natural light (repink) and UV light (dark blue) photograph shown in the maximized view. The histograms are shown in the same view so that they can be easily compared. The histogram is rendered as a JavaScript enabled web page utilizing the D3 library [17].

the pages is animated, we allow the current-page index to be a real number. The fractional part of the index is used for the animation of the page turning.

Parts of the displayed pages can be selected. The selection is meaningful only for the page in which it was created. The current-page index constitutes the visualization snapshot for this application, because it alone fully describes the context for the selections. The current-page index is visualized by the book reader as a slider displayed below the pages. The anchor of each contextual snapshot is placed on the slider according to the current-page index. The anchor therefore visually represents the position of the displayed page spread and can be used as a bookmark.

The rectangular selections support the visual analysis of interesting parts of the photographs. By clicking the UV icon, displaying of the UV light photographs in the selections can be enabled or disabled. As the UV and natural light photographs are co-registered, the selections create a comprehensible integrated view.

We have implemented three web views which give web pages as embedded visualizations. With the web pages, we demonstrate that the embedded visualizations managed by the CSL are interactive and that they can contain arbitrary content. The first web view contains color histograms from the selections. As multiple selections can be activated, we employed the JavaScript library D3 [17] capable of displaying multiple histograms at once. For each selection, a color histogram is displayed. For the pages where the UV data are available, the histograms from the UV photographs are displayed as well. As all of the histograms are given in one view, they can be easily compared.

The second web view shows the Greek transcript of the displayed pages. Contextual information of the selections, i.e., the current-page index, is used to load the appropriate pages from the transcript. A tag cloud of the most frequent words generated by the JavaScript is displayed below the text.

The third web view contains a web page of the Perseus Word Study Tool [77]. This web application provides an English translation of a specified Greek word, as well as further information. We have connected this view with the Greek transcript of the displayed pages. The user can double-click on any word in the transcript to automatically display its definition with the Perseus Word Study Tool.

The application of the CSL in the book reader example demonstrates various ways how the contextual snapshots integrate different views of the visualized data. This use case contains several types of annotations which can be helpful in analysing the historical manuscript. It shows that the integration of vastly differing visualization techniques including online content and GPU-based rendering is easily possible with our approach.

3.6 Application Example - Heart Visualization

In order to illustrate the possibilities of the CSL and the contextual snapshots in general, we apply it to extend an existing visualization system. We choose *VolumeShop* [18], a powerful visualization system with a modular architecture. We extend a VolumeShop plug-in for rendering 3D meshes with contextual snapshots to create an application where it is possible to easily add annotations for individual parts of the geometric model.

In this example, we demonstrate how user input can be transformed to different types of selections and how anchors can be customized to provide abstract previews of stored contextual snapshots. Additionally, we show how the stored selections can be used in the visualization mapping. Finally, we demonstrate how the embedded visualizations can be employed as informative annotations of the selected data.

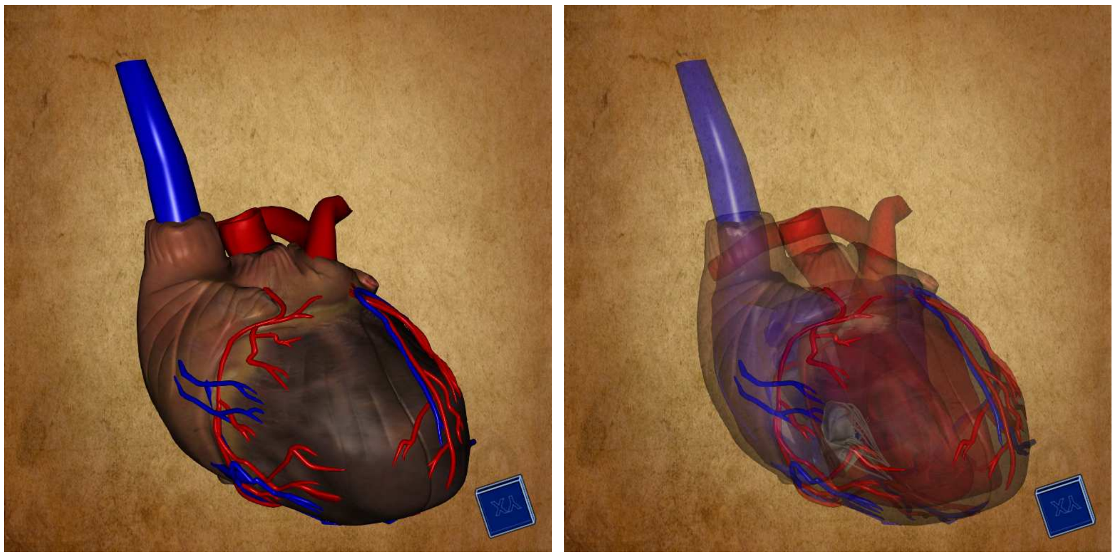

(a) (b)

Figure 3.8: (a) A 3D model of a human heart displayed in VolumeShop. (b) The model is displayed with a lower opacity to reveal the internal structures.

3.6.1 Heart Visualization

In our example, a VolumeShop application displays a 3D model of a human heart. The model is composed of 32 parts representing individual anatomical structures. It is possible to freely rotate, zoom, and pan the model to look at it from different viewpoints. The rendering algorithm also allows us to specify the opacity of the displayed model parts to reveal occluded structures. Figure 3.8 shows how the heart model is displayed in VolumeShop.

3.6.2 Anatomical Annotations

In this example, our goal is to allow domain experts to annotate individual parts of the displayed 3D model. The application can be presented to non-expert users who can interactively explore the model and learn the names of the individual constituting parts.

The CSL extends the interaction possibilities of the VolumeShop plug-in. The basis for the annotations are spatial selections created in the image-space of the 3D model visualization. The context of the selections consists of the camera viewpoint and the opacity value used for the rendering. If either of these parameters changes, the selections might not encompass the desired structures any more, and therefore they disappear. Each selection is assigned to a contextual snapshot, which stores the respective camera viewpoint and the opacity value.

For each created selection, a list of all objects visible within the selection is composed. The users indicate which of these objects did they intend to highlight with the selection. They can also specify a caption which will be paired with this selection. Afterwards, a rendering is created where all structures of the 3D model are semi-transparent. It is overlaid with the user-selected objects rendered in full opacity. The specified caption is displayed underneath. This rendering is then depicted in an embedded visualization assigned to the selection. This way it is possible to annotate individual objects or groups of objects, which are then given in the context of the original 3D model. This process is illustrated in Figure 3.9

We implemented the object selection through the stencil buffer and occlusion queries. After creating a selection, each object is displayed separately with the selection rendered in the stencil buffer. Using occlusion queries, we determine whether any of the pixels were rendered to the framebuffer. If so, the given object intersects the selection and it is listed as one of the selected items. The user can decide which of these objects were meant to be selected, since the selection might also intersect objects which are not of interest. This demonstrates how individual selections can be used within the host system to implement new functionality.

3.6.3 Anchors

The anchors are placed in the 3D space so that they convey the viewpoints associated with the respective contextual snapshots. The context also contains an opacity value. In case only the opacity value changes but not the viewpoint, several anchors representing different contextual snapshots can be placed at the same 3D position.

To alleviate the occlusion problem, we offer the possibility to customize how the anchors are rendered. If two or more anchors are placed at 3D positions which are projected to similar image-space positions, they are grouped together and are displayed on a circle around the center of the group, as shown in Figure 3.10a. This way, occlusion of anchors is avoided and it is possible to interact with all of them. For detecting anchors on similar positions, we use the DBSCAN clustering algorithm [39].

Additionally, we enhanced the customized rendering of the anchors by conveying the opacity value stored within the respective contextual snapshots. The difference between the stored opacity value and the opacity value of the current system state is mapped to the anchor's size and color. Black means the anchor was created with the same opacity value as given in the current system's state. White means the current opacity value is largely different from the one stored within the contextual snapshot. The darker colors of the anchors are also emphasized by larger sizes of the anchors.

3.6.4 Selections

A default behaviour of the CSL is to show a thumbnail of the visualization if the user hovers the mouse pointer over an anchor. This functionality enables users to quickly browse through recorded contextual snapshots and choose relevant ones for the data exploration. We extended this functionality to ease the exploration of the contextual snapshots. If the current viewport of the visualization system is very close to one stored within a contextual snapshot, hovering the mouse pointer over its anchor will not simply show the thumbnail. The selections of the contextual snapshot are rendered, and the selected areas are overlaid with the rendered image of the model using the opacity value stored in the contextual snapshot. This creates a meaningful integrated view of the current rendering and the rendering of the contextual snapshot, because both viewpoints are very similar or equal. This functionality is illustrated in Figure 3.10b.

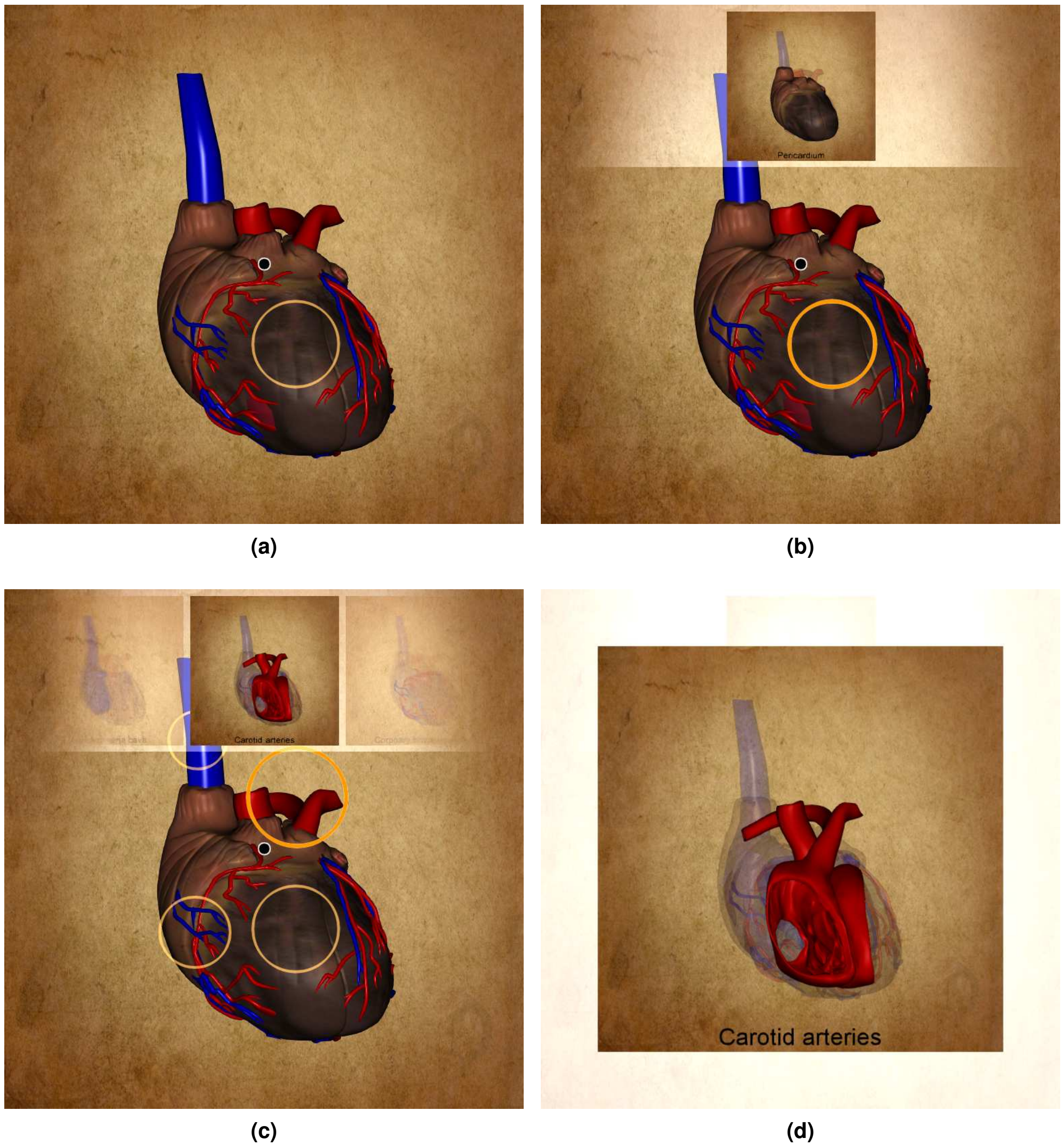


Figure 3.9: (a) The user creates a circular selection and provides a caption for it. (b) If the selection is activated, an embedded visualization of the selected object in the context of the original model is shown with the specified caption at the bottom. (c) Multiple selections in the same context (viewpoint and opacity value) can be created. All of their embedded visualizations are shown at once in the sliding bar. The one in the middle is highlighted, as well as its associated selection (bold yellow circle). (d) The highlighted embedded visualization is shown enlarged (maximized) over the original visualization for a better view. The embedded visualizations can be browsed in the sliding bar or in the maximized view.

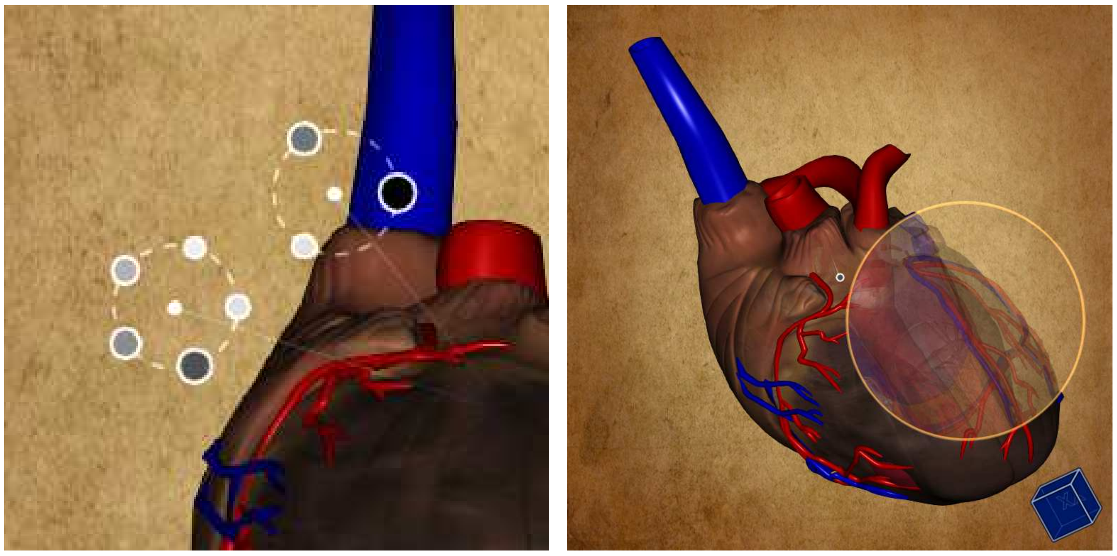

(a) (b)

Figure 3.10: Custom usage of graphical elements of the CSL in the heart example: (a) Two groups of otherwise overlying anchors are displayed in a circular layout around the groups' centers. (b) Hovering the mouse pointer over an anchor shows the selections from the respective contextual snapshots in an integrated view.

As the application in this example deals with annotating anatomical strucures, the circular selection might not always be suitable. Therefore, we implemented three different types of spatial selections: circular, rectangular, and free-hand lasso ones. The user can choose the type of selection before its creation. This functionality also demonstrates the extensibility of the CSL. Different types of selections can be specified using simple external shader programs, which can be easily interchanged at run-time. All three types of selections are illustrated in Figure 3.11.

3.7 Integrating the CSL with Existing Visualization Systems

Our implementation of the CSL uses the Qt library. It is necessary to link Qt together with the CSL to the host system. In our examples, the Qt library was already part of the host visualization systems, therefore there were no additional dependencies.

The CSL provides functions for creating and accessing the selections, activating them and adding embedded visualizations. However, there is no functionality for handling user input. It is necessary to implement handling of keyboard and mouse events, and call relevant functions from the CSL in the respective event handlers.

It is necessary to modify the rendering pipeline of the visualization system in such a way that the output is rendered to a texture, which is provided to the CSL as input. The CSL then renders all visual elements (anchors, selections, embedded visualizations) with the input texture as background into an output texture. The output texture is sent back to the visualization system.

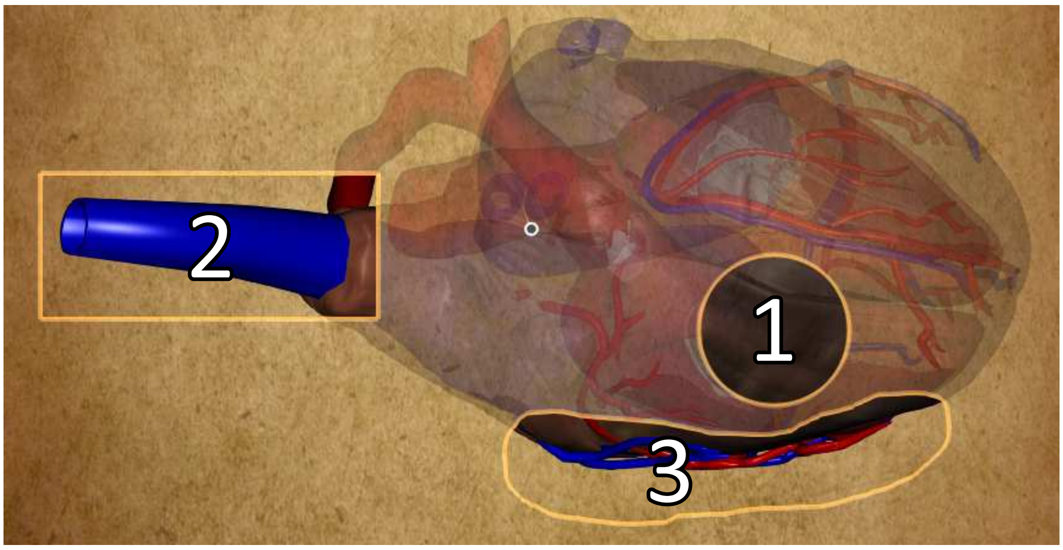


Figure 3.11: Different types of selections available in the heart application: circular (1), rectangular (2), free-hand lasso (3) selections.

The rendering pipeline of the visualization system should be extended by displaying the output texture of the CSL on the screen instead of its original output. This is usually fairly easy to achieve by taking a framebuffer object as a rendering target. Afterwards, a quad is rendered which covers the whole screen and which is textured with the output texture of the CSL.

In the heart-visualization example, the rendering of the anchors is customized. This is done by subclassing a CSL class responsible for the rendering of the visual elements, and overriding the function for rendering of the anchors. The subclass also implements handlers which react to events from the CSL, such as requests to update in case visual elements of the CSL need to be repainted. If the host visualization system uses Qt, an alternative way for this would be to use Qt's signal/slot mechanism.

3.8 Discussion

The goal of our work is to introduce a general concept for handling spatial selections created in changing contexts during a visualization session. Instead of realizing a new standalone system, we implemented this concept as a flexible toolkit, i.e., the CSL. The presented examples demonstrate that the CSL is ready to be integrated with different existing visualization systems. Section 3.7 describes the implementation efforts needed for the integration of the CSL in the provided examples.

In sections 3.5 and 3.6 we give examples how the concept of contextual snapshots can be employed. State-of-the-art visualization systems usually treat selections in such a way that it is necessary to use several linked views to work with multiple selections simultaneously. To employ selections as interactive annotations, each selection would have to be assigned a separate view, possibly in a separate window. Contextual snapshots allow us to realize multiple selections

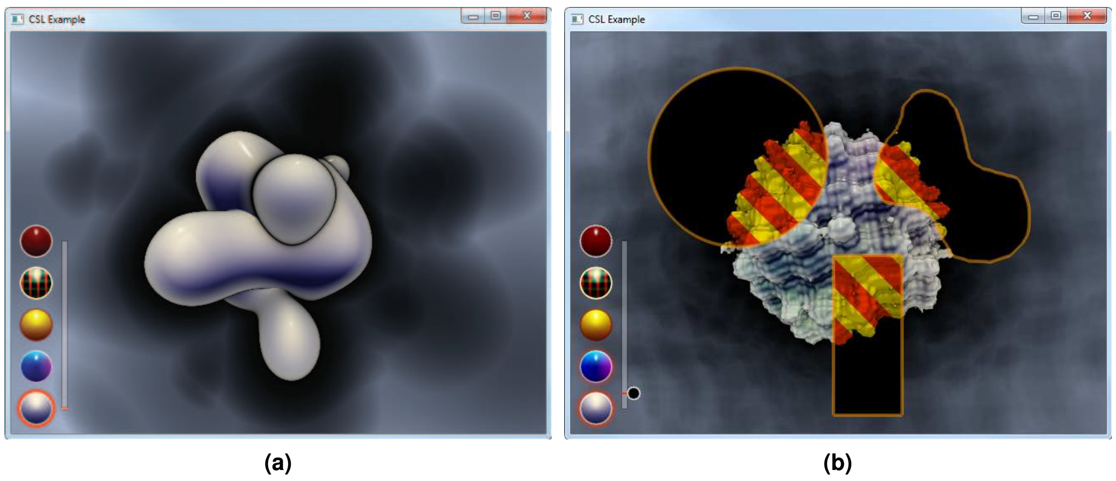


Figure 3.12: An example application implementing contextual snapshots using the CSL. (a) A simple rendering of a dynamic object modelled with a distance field is shown. (b) The rendering is enhanced with interactive selections. The rendering style inside the selections is different from the rendering style in the rest of the image.

in the same view while the changes of the visualization are automatically tracked. Our method does not provide a guidance for finding appropriate views or means for selecting the data automatically. It extends the common possibilities of data selections to act as data annotations, to convey and to communicate users' findings. In this way, the contextual snapshots support users in the data exploration process.

By employing the CSL to render the selections, the anchors, and the embedded visualizations, the performance of the rendering dropped from 60 FPS to 30 FPS in the historical document-analysis example. The performance drop of the whole system mainly depends on the temporal requirements of the embedded visualizations. This aspect can be improved in the future by parallelizing the rendering of individual embedded visualizations. In the heart-visualization example, there is no significant performance drop after integrating the CSL, since the embedded visualizations only show static images.

We encourage the usage of the CSL, since contextual snapshots can be beneficial for a wide variety of applications. Therefore, we made the CSL available with full source code on our web page [1]. On this web page, there is a detailed tutorial explaining with a simple application case how to ingrate the CSL into existing visualization systems. The application used in the tutorial is shown in Figure 3.12a and renders an animation with a simple graphics effect. The user can choose the rendering style from a palette of styles. The tutorial explains how to implement a functionality which allows users to select regions in the image space of the animation. In these regions, a different rendering style is used. The selected rendering style serves as a context for the selections, so for each rendering style, different regions can be selected. The application enhanced with the CSL is shown in Figure 3.12b.

3.9 Conclusion

In this chapter, we proposed a method for managing image-space selections which can be used for various tasks, such as highlighting of interesting regions in visualizations, displaying additional views for selected data, or comparing different spatial regions. We demonstrated the utility of the method by applying it to three distinct use cases, namely analysis of a historical manuscript, analysis of multivariate weather simulation data, and annotation of the geometrical model of a human heart.

Our method is meant to be applied to visualization systems where the state changes over the duration of a visualization session. Most of the interactive systems fulfil this characteristic. In our method, the user-made selections in image space are linked with all necessary contextual information so that they remain meaningful during the whole visualization session.

CHAPTER 4

Integrating Parameter Sets of Multiple Users

This chapter is based on the following publication:

P. Mindek, L. Čmolík, I. Viola, M. E. Gröller, S. Bruckner. Automatized Summarization of Multiplayer Games. In *Spring Conference on Computer Graphics*, SCCG'15. To appear.

It is common that multiple users participate in cooperative actions within a single framework, performing various tasks. There are systems providing means for multi-user collaboration to create visualizations or other products in a distributed manner. The state of such a system is described by a parameter set. Whenever multiple users have the possibility to modify the state of the system, it is necessary to integrate parameter sets associated with the individual users in order to produce the desired product. In systems where the product is an output of the system in a single specific state, the integration can be performed in real-time. In other words, every change of the state of the system is immediately propagated to all participating users.

A wide-spread example of a system whose state can be simultaneously accessed and modified by multiple users is a multiplayer video game. In such games, the state is also updated in real-time, so that the actions of the individual players are immediately visible to all participants. If we think of a multiplayer game as a data-processing system, its product is the game outcome, i.e., ranking of the players, or a classification of the players into *winners* and *losers* in case of a team gameplay.

However, in multiplayer video games, there are other products besides the outcome that can be generated. For instance, various statistics of the gameplay can be of interest. Another example is a summary of the gameplay in the form of a visual story, showing the actions of the players and events of the gameplay, and putting them into context. This product cannot be created by simply taking the game state at discrete time intervals, since such a state typically contains multiple views of the participating players. To actually show the gameplay, it is necessary to record the individual states of all players, where each state is a point in the multidimensional parameter space of the game, including the view of the respective player. Subsequently, to show temporal evolution, these player states need to be integrated into a curve within the parameter space of the game. By sampling the parameter space along this curve, a video dramatization of the gameplay can be reconstructed. However, the integration of the player states into such a curve is not straightforward, since there are many ways how the states of the players can be connected, and only some of them meaningfully tell the story of what happened during the game.

We present a novel method for creating automatized gameplay dramatization of multiplayer video games. The dramatization serves as a visual form of guidance through dynamic 3D scenes with multiple foci, typical for such games. Our goal is to convey interesting aspects of the gameplay by animated sequences creating a summary of events which occurred during the game. Our technique is based on processing many cameras, which we refer to as a flock of cameras, and events captured during the gameplay, which we organize into a so-called event graph. Each camera has a lifespan with a certain time interval and its parameters such as position or look-up vector are changing over time. Additionally, during its lifespan each camera is assigned an importance function, which is dependent on the significance of the structures that are being captured by the camera. The images captured by the cameras are composed into a single continuous video using a set of operators based on cinematographic effects. The sequence of operators is selected by traversing the event graph and looking for specific patterns corresponding to the respective operators. In this way, a large number of cameras can be processed to generate an informative visual story presenting the gameplay. Our compositing approach supports insets of camera views to account for several important cameras simultaneously. Additionally, we create seamless transitions between individual selected camera views in order to preserve temporal continuity, which helps the user to follow the virtual story of the gameplay.

4.1 Introduction

Multiplayer video games often contain complex scenes with multiple dynamic foci. The scenes and the game events are viewed by the individual players from different positions. The view of each player is relevant to their part of the gameplay. If an observer watches the players' views, or some additional overview provided by cameras observing the scene, it is not trivial to see the big picture of what has happened during the game. It is likely that some important events will be missed, because they occurred while the observer watched a different view at that time, since the foci are possibly overlapping in time but not in space. However, it is possible to arrange the individual camera views into a story summarizing the gameplay and conveying the big picture.

There are various situations where composing a summary of the gameplay might be beneficial. State-of-the-art game consoles include functionality for the recording of the gameplay and sharing it on social media. There are online services, such as twitch.tv [107], where users can stream gameplays of various video games. There are also tournaments in multiplayer video games, where the summaries of individual matches are created to be shown to a wider audience. The summaries are also used by the participating teams to analyze the gameplay in order to design strategies, detect mistakes made by individual players, and improve their overall performance. There is also a practice of using video games for creating cinematic productions by navigating the game characters according to a pre-defined screenplay, called *Machinima* [11]. These developments demonstrate that there is a demand for tools aiding in the creation of gameplay summaries or visual narratives.

To create a visual story of a gameplay, various cinematographic effects can be used. However, some specifics regarding multiplayer games have to be taken into account in order to produce a meaningful story. The very fast pace of multiplayer games, as well as many concurrent events require that hard cuts are avoided whenever possible. Instead, it is essential to provide linkage cues in the portrayal of the individual events, such as continuous camera transitions when portraying a sequence of events occurring at different locations.

The aim of this work is to design a method for creating animated summaries of gameplays captured by many dynamic cameras, as in multiplayer video games. The summaries have to portray many dynamic events, possibly occurring at the same time, to facilitate easy understanding of the game action. During the gameplay, the cameras and important game events are recorded. Afterwards, a summary is created which shows the views from the cameras in such a way that the displayed events are meaningfully linked together and they compose a coherent story narrating the gameplay.

By integrating events associated with individual players, and by smoothly transitioning between them rather than splitting the gameplay into separate segments, our method provides an output suitable for analyzing the gameplay itself. Aspects such as interaction between players or whole teams, and causal dependencies of individual events are hard to convey by conventional methods, such as sequentially showing the views of all players.

To efficiently portray the linkage of the events, we record their causality, as defined by the game rules and implied by the gameplay. Using this information, we arrange the events into a structure called *event graph*. Additionally, each event is assigned a camera which captures it. This is possible since the events are linked to the players and/or their locations, which are in turn linked to the individual cameras. The event graph together with the linkage between the events and the corresponding cameras contain sufficient information to reconstruct the story which visually narrates the gameplay.

In addition to describing the gameplay by the event graph, we propose two concepts: *flock of cameras* and *ManyCams*. The flock of cameras is a set of cameras whose parameters change dynamically over time. Each camera has a specified lifespan. During its lifespan, a continuous importance function is assigned to the camera. It describes the importance of the corresponding captured events in time. The flock of cameras is a structure which holds all cameras relevant to the gameplay.

ManyCams is a method for selecting the most interesting views captured by the flock of cameras. By merging these views, and inserting smooth transitions between them, a story is created which visually narrates the gameplay. The generated video is a linearization of the gameplay, which has a parallel structure with multiple events occurring at different spatial locations, possibly overlapping in time. ManyCams uses the event graph and the flock of cameras, which are constructed during the gameplay, to create a video summary for a post-mortem visual analysis of the course of the game.

4.2 Related Work

There are various techniques for integrating visual data into a condensed form which can be presented as a story. Correa and Ma [28] present a method for creating a compact dynamic narrative from videos. These narratives remove redundancy by using the common background for different stages of motion of various objects in the video. They allow the viewer to quickly inspect the content of the video. Barnes et al. [8] present a method for creating continuous images summarizing video clips. The method dynamically changes the level of temporal detail by continuous zooming. The generated image shows key features, which would otherwise have to be inspected by watching the entire video. In contrast to these techniques, which summarize linear videos into more condensed forms, our proposed technique summarizes complex parallel structures of multiplayer gameplays by linearizing them into a single summary video.

Various approaches for creating visual stories and presenting data have been developed. Segel and Heer [96] review different types of narrative visualizations for presenting data based on approaches used in news media. Gershon and Page [44] describe how storytelling helps to convey information so that it is more easily understood and remembered by the target audience, while Ma et al. [75] provide an overview of how storytelling can be incorporated into scientific visualization techniques. Wohlfart and Hauser [118] propose a method for storytelling in volume visualization. In their method, storytelling and story authoring are two distinct steps. In the story authoring step, the dataset is explored by an expert user. While the data are being explored, the expert user iteratively creates the virtual story. In the storytelling step, the findings from the previous step are presented. The user watching the story also has the possibility to interrupt it and perform data exploration on his own. Yu et al. [123] propose a system which extracts events from time-varying datasets and organizes them into an event graph. By traversing the event graph, an animation with a narrative structure is created which shows the events in a meaningful order. We also employ the concept of event graphs to organize the events extracted from the video game. However, we use it to merge views captured by the flock of cameras into a continuous animated sequence. This allows us to illustrate dynamic scenes of the video games with multiple temporally-overlapping foci. Viola et al. [114] propose a technique for focusing on selected features of a volume dataset. A viewpoint showing the selected feature in the most informative way is determined. An animation switches between viewpoints for different features. This technique is complementary to our method, since we only focus on compiling the views into a coherent summary, not on selecting adequate viewpoints for individual cameras to capture the scene or the dataset in a certain way.

Creating summaries of computer games has been examined as well. Halper and Masuch [49] propose a method for extracting game events by analyzing computer game variables to evaluate how interesting individual time-steps of the gameplay are. The authors present several methods for merging these events into scenes, which then compose the story of the gameplay. Cheong at al. [25] present *ViGLS* - a system for visualizing gameplay summaries from game logs. The game logs are analyzed and sequences of summary actions are extracted from them. These are then sent to a game engine, which replays the actions to generate a visual summary. The goal of our work is to extend these approaches by providing visual links between individual events so that multiple concurrent foci occurring in the dynamic game scene can be captured by the composed summary. Our method linearizes the gameplay, so that continuous dramatizations with a parallel structure can be created. Therefore, our method is able to create summaries of complex gameplays, such as those of multiplayer games, where events are observed by multiple cameras, and where it is important to illustrate causality between the events.

An important aspect of creating visual narratives and informative overviews of 3D scenes is the virtual camera setup. An extensive overview of camera control in computer graphics is given by Christie et al. [26]. Löffelmann and Gröller [72] propose extended cameras. By using extended cameras, it is possible to render images with non-realistic perspectives. This enables various effects, such as the seamless combination of several different views of a virtual object in the same image. Agrawala et al. [2] use multiple projections for different objects in the scene to create various artistic effects. Hsu et al. [56] present a framework for using several arbitrary cameras to render a single scene at different levels of detail. The cameras are combined to create a single multiscale rendering of the dataset. For this purpose, a mask is specified to define which part of the image should be generated by which camera. Seamless transitions between these image parts are then created.

He et al. [52] present *Virtual Cinematographer*, a system for the automatic setup of cameras to capture events in dynamic 3D environments according to cinematographic principles. A similar system, utilizing an agent-based camera, is proposed by Hornung et al. [54]. Oskam et al. [85] propose a method for planning collision-free camera-paths between two points in a 3D scene, so that a selected focus point is visible during the camera transition. Such camera-shot planning-systems can be also used with our method, since we do not assume any particular camera-placement method for capturing events of the game.

Virtual cameras have been employed for different types of tasks as well. For instance, Qureshi and Terzopoulos [90] propose multiple virtual cameras to simulate real-world problems, such as person-tracking in video-surveillance feeds.

In this work, our goal is to display 3D scenes in such a way that multiple dynamic foci are captured in a clear and semantically meaningful way. For this purpose, we utilize a flock of cameras, which is composed of multiple camera paths. The paths of the cameras can be either directly extracted from the video game, such as view paths of individual players, or created by game-level designers. Each camera path has a specific lifespan and it is assigned a continuous importance function over the period of its existence. The focus of this chapter is not on creating these camera paths, but instead lies in visualizing image outputs of the cameras according to their importance. The intent is that the captured events are meaningfully linked together and the gameplay can be easily analyzed. The goals are: to display outputs of the cameras with high importance, to make the changes between individual cameras smooth and continuous so that users can easily maintain an overview, and to simplify the flock of cameras in a smart way so that the method can be effectively used for summarizing and analyzing gameplays. Our proposed method extends existing storytelling approaches by creating animations where multiple foci, or game events, possibly overlapping in time or space, are conveyed by the generated visual story.

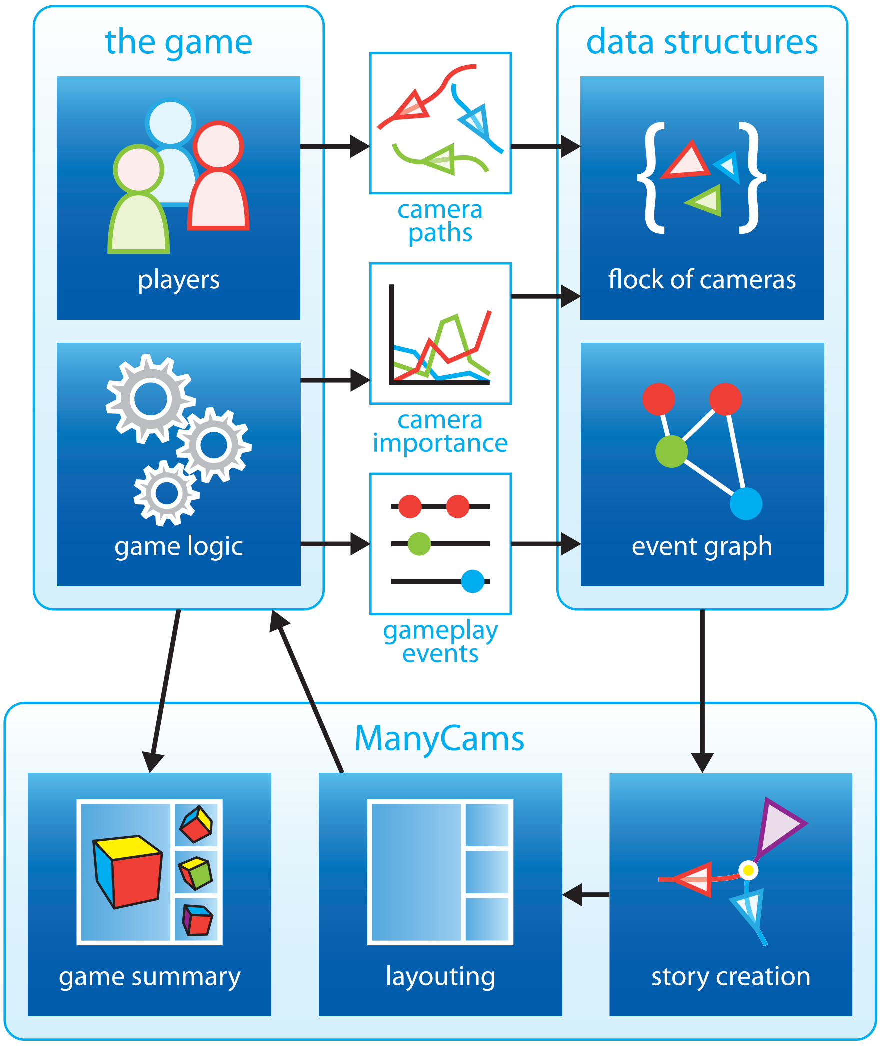


Figure 4.1: Overview of the ManyCams, the flock of cameras, and the data on which they operate. Camera paths, camera importance, and the events of the gameplay are extracted from the game and stored in the flock of cameras and the event graph. These structures are then processed by our ManyCams method to produce the video summarizing the gameplay.

4.3 Overview

We solve the problem of composing gameplay summary-videos from available gaming data. Figure 4.1 shows an overview of our method. The black arrows illustrate the data flow. First, camera paths are extracted from the game, e.g., views of individual players as they move through the game world. Additionally, the importance of the individual views in time as well as interesting game events are extracted during the gameplay. Our method does not assume particular event types or functions which evaluate the importance of each view. This information has to be provided by the game, since it depends on the game logic. Our method is only concerned with building the video summary of the gameplay described by the extracted data. If the game logic does not inherently contain any metric which could be used to evaluate the importance of the individual cameras, an importance of individual event types could be specified manually instead.

As a tangible example, let us consider a deathmatch game (where the goal is to kill all opponents) between four players A, B, C, and D. During the game, all but one players are sequentially killed. The game ends as soon as there is just one last player alive. Suppose the gameplay progresses as follows: Players A and B join the game and meet at a certain place. Soon after A spots C and kills him. Player C has joined the game shortly before that. After C is killed, he drops his weapon. Meanwhile, D also joined the game. He finds the weapon of player C. He picks it up and uses it to kill A. Shortly after that he also kills player B. At this point the game ends, because D is the last player alive.

The goal of our method is to generate a video, which visually tells this story by showing sequences of what individual players saw during the gameplay. We achieve this by extracting the data about the players and the game events, e.g., a player kills another one, from the gameplay. The extracted data are organized in data structures which are further described in Sections 4.3.1 and 4.3.2. These structures, i.e., *flock of cameras* and *event graph*, are processed by *ManyCams* after the gameplay finishes. The ManyCams method integrates the cameras into several continuous views which visually express important events as well as their causal contiguity, forming a visual story. Afterwards, the layouting algorithm juxtaposes the generated views in space or time in the screen space, so that the game summary can be generated. This information is sent back to the game engine, which then renders the game world from the perspective of the views generated by the ManyCams method to produce the final gameplay summary-video.

4.3.1 Flock of Cameras

We assume that the events which occur during the gameplay are captured by the cameras present in the 3D scene, such as views of individual players, or cameras surveying certain locations. While the players control the cameras capturing their views by navigating their avatars, the surveying cameras can be either positioned manually by the level designers or their parameters can be determined dynamically by one of many camera-path planning-methods [50, 85, 122].

A flock of cameras is a structure which maintains all the necessary information about the cameras in the 3D scene. A flock of cameras $F = \{c_0(t), ..., c_n(t)\}$ is a finite set of camera functions $c_i(t)$. For every time step t, $c_i(t)$ specifies the camera parameters $p_i(t)$ as a transformation matrix, an image of a camera view $v_i(t)$ rendered using $p_i(t)$, and a camera importance $i_i(t)$ as a real number in the interval $[0, 1]$, where 0 signifies the lowest importance.

4.3.2 Event Graph

The events occuring in the game are not separate entities, but rather interconnected elements of a narrative portraying the action of the gameplay. In our approach, an effective visual summary of the gameplay should convey the causality of the individual events. This way, the summary tells a coherent story, which can be followed in order to understand what happened during the

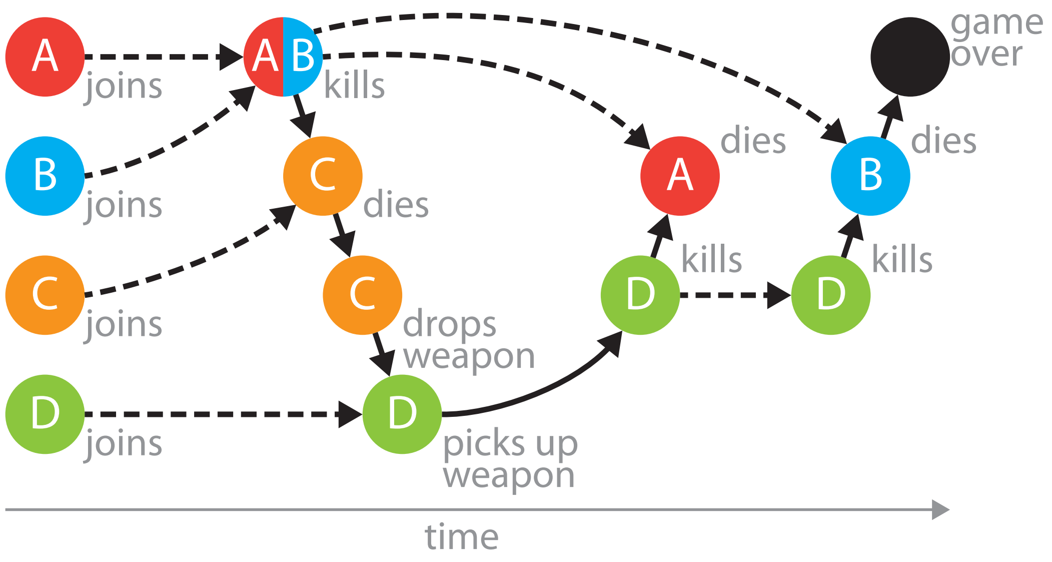


Figure 4.2: An example of an event graph. The circles indicate individual events, the solid arrows are causal links extracted from the game, and the dashed arrows are player-links derived from the flock of cameras.

gameplay. Understanding the story of the gameplay allows us not only to analyze the behaviour of the individual players, but also enables us to see the consequences of their actions.

To compose a meaningful narrative of the gameplay, we extract events and their causality from the game while it is played and store this information in an *event graph*. The types of events and their connections depend on the actual game. For instance, in a first person shooter, there are events such as *player A shoots* or *player B dies*. If player *A* killed player *B*, the event *player A shoots* caused the event *player B dies*. Another example is that player *A* throws a grenade and kills several other players. An event can cause zero, one, or several other events, and it can be caused by zero, one, or more events. We connect causally dependent events in the event graph by *causal links*.

Another aspect of gameplay dramatization in multiplayer games is that there are possibly multiple players participating in individual events. As they move through the game, individual players or groups of players participate in different events. Events with the same participants are semantically related within the story. To capture this concept in the event graph, we connect such events by *player-links*.

To build the player-links, we use the information stored in the flock of cameras. We treat each camera in the flock as a player. Therefore, all entities which are able to observe the action of the game, such as spectators or surveillance cameras, are treated as players. Instead of participating players, each event is associated with a set of cameras which observe the event. The player-links are then constructed based on the events' associations with the cameras from the flock. Details are provided in Section 4.4.1. Besides actual players, this abstraction also accounts for non-playing participants, such as spectators in first person shooters.

The causal links and the player-links connect the events according to the interactions of the players during the gameplay. Therefore, the event graph abstracts the story of the gameplay. Figure 4.2 shows the event graph of the previously described example gameplay. The circles represent the events of the game, while the letters indicate which players participated in individual events. The causal links (solid arrows) indicate how individual events are related according to the game logic. The player-links (dashed arrows) indicate how players moved between events. Both types of links encode the progression of the story which we want to tell.

The event graph is a directed graph G:

$$G = (E, K) \tag{4.1}$$

$$E = \{e_0, ..., e_n | e_i = \{a_i, t_i, d_i, P_i\}\} \tag{4.2}$$

$$K = \{k_0, ..., k_l, r_0, ..., r_m | k_i = (e_x, e_y) \in E^2, r_j = (e_z, e_w) \in E^2\} \tag{4.3}$$

E is the set of all events e_i which occurred during a gameplay. Each event consists of the event type a_i, the time of occurrence t_i, the duration of the event d_i, and a set of cameras P_i which are associated with the event.

K is the set of the edges representing links between the events. It consists of causal links $k_0, ..., k_l$ extracted from the game, and player-links $r_0, ..., r_m$ derived from the flock of cameras. Each link is an oriented edge between two events in the event graph.

4.3.3 Camera Operators

In order to produce a summary of the gameplay telling a story of what happened in the game, the events from the event graph need to be displayed using the available cameras. A naïve approach would be to sequentially show views from all the cameras stored in the flock. In this way, every event is included in the story, since everything that the game participants saw during the gameplay is shown. However, such a summary does not communicate any causality between the events, other than what each of the players did separately. In this case, the story of the gameplay is not effectively conveyed. An additional drawback is that possibly redundant and unimportant information is present.

Another approach, often employed in video surveillance, is to show all the views simultaneously. Although all the events captured by the cameras would be shown on screen, the viewer would potentially have to focus on multiple events at the same time. This compromises the utility of the approach for scenarios such as multiplayer video games, where multiple dynamic foci possibly occur simultaneously.

In our work we aim to overcome the drawbacks of the mentioned methods. We compose the visual story by following the links between events in the event graph, so that the coherence of the action's portrayal is maintained. Additionally, the redundancy in the views of the available cameras is reduced before the views are employed to display individual events. This way, the generated video summary tells a story which helps to make sense of the events in the game.

The method presented in this work operates on virtual cameras displaying 3D scenes. In contrast to real-world scenarios where multiple video cameras are employed, such as sports

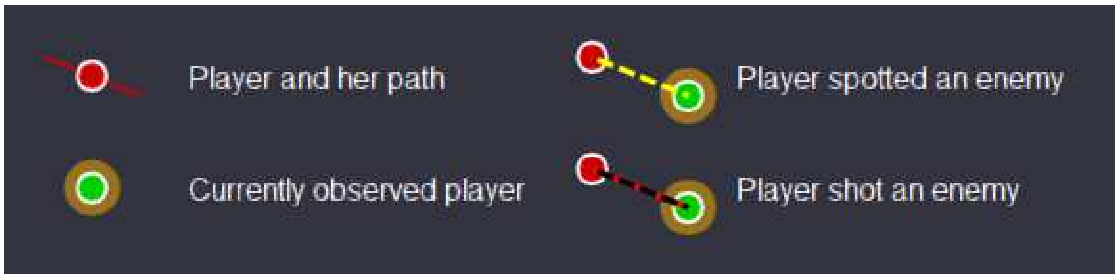


Figure 4.3: Some of the pictograms used by the overview operator for a schematic display of the gameplay.

events or video surveillance, processing virtual cameras enables the modification of the camera parameters and rendering of the scene from new viewpoints. We utilize this possibility by applying various operators on the flock of cameras. They reduce redundant information and provide visual links between related events.

The *Overview* is an operator which shows the generated visual narrative from a bird's-eye view provided by a dedicated overview camera placed above the scene. The players, their movements, and actions are depicted with animated pictograms (some of them are shown in Figure 4.3). This operator is inspired by schematic depictions used in sports to analyze the strategies of a team.

The *Time-lapse* operator smoothly changes the speed of the generated visual narrative. It is used to seamlessly pass through uninteresting parts of the story. It avoids creating a hard cut, which might be confusing in a summary of a fast-paced video game.

The *Mark-player* operator is used to point out individual events shown by the generated visual narrative. It stops the playback when an important event occurred, and highlights the entity which caused the event. In our use case, we apply this operator to mark players important for the story. The operator is based on the practice often employed in live-tv broadcasting of sports events, when the broadcast video is briefly stopped so that an important object or person can be marked to guide the viewers' attention.

The *Camera-merge/split* operators are applied to multiple cameras showing the same event from similar viewpoints. If several players move together as a group in the same direction, the camera-merge operator is applied. When they no longer move together, the camera-split operator is used. When the cameras are merged, their views can be used in the summary video interchangably. The merged cameras are seen as a single entity within the story.

The *View-switch* operator is applied if events occurring at different times or places need to be shown in a sequence. Instead of jumping from one event to the other one, the operator interpolates between the camera parameters of the views capturing both events. Simultaneously, time is interpolated from the end time of the first event to the beginning time of the second event. If the second event occurred before the first one, the time flows backwards during the interpolation, giving the view-switch a dramatic effect. The view-switch communicates spatial (by interpolating camera parameters) and temporal (by interpolating time) relationships between the two events shown in a sequence. Similar techniques are often used in cinematography, where multiple scenes are displayed in one continuous shot to communicate the continuity of the portrayed events.

2D inset-show/hide operators show and hide 2D insets of a camera view which is currently not the main focus, but is relevant to the current situation. An example scenario is when several players observe the same situation from substantially different viewpoints which cannot be merged. Since these players observe the same focus, it is possible to display their views in parallel without confusing the viewer of the summary.

4.4 ManyCams: Summarizing the Gameplay

ManyCams is a method for traversing the event graph and matching patterns with operators appropriate for the given situation, which are then applied to the flock of cameras to produce a summarizing visual narrative of the gameplay. To achieve this goal, it is necessary to create a virtual representation of the story, and to provide an algorithm for creating a visual narrative from this representation.

4.4.1 Story Representation

The story is represented by the event graph. The data extracted from the game contain the nodes of the graph (events), and causal links between them. In order to represent the story of the gameplay by the event graph, it has to be further pre-processed.

First, the events are associated with the cameras from the flock. Depending on the game, this information might be available from the extracted gaming data. Alternatively, we provide an implementation of a density-based clustering algorithm DBSCAN [39], which can be used to associate multiple cameras with similar views with a single event. However, a heuristic for estimating the similarity between camera views has to be provided as well. For first-person shooter games, we use the sum of spatial-position differences and viewing-angle differences as the heuristic. This can be evaluated quickly, while it is robust enough to efficiently cluster views of players in first-person shooter games.

The duration of the summary video can be specified. Only the n most important events, which would fit into the specified duration, are included in the summary video. Since the events are associated with the cameras from the flock, the time-varying importance of the cameras can be used to automatically estimate the importance of the events. If there is no camera importance information available in the extracted gaming data, the importance of individual event types can be specified manually.

Finally, player-links are added to the event graph. A player link is created between each pair of chronologically successive events (e_i, e_j) with sets of associated cameras P_i, P_j, such that $P_i \cap P_j \neq \emptyset$. If there are multiple events fulfilling this condition for the event e_i, only the one with the closest timestamp is chosen to create the player link.

4.4.2 Building the Visual Narrative of the Gameplay

The summary video is constructed by appliyng operators on the flock of cameras. In the story of the gameplay scenario described at the beginning of Section 5.3.1, players A and B meet and

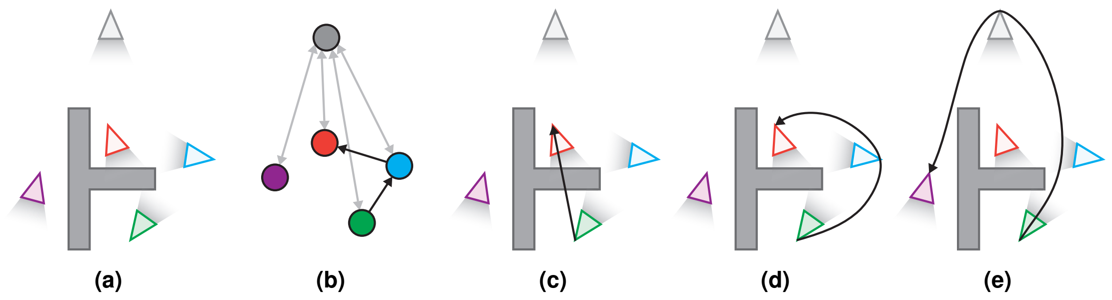


Figure 4.4: (a) Five cameras in a scene. The gray one is an overview camera. (b) A visibility graph. (c) Switching view from the green camera to the red one would cause a collision with a wall. (d) The view from the green camera can be switched to the view from the red camera through the blue camera, because such a path exists in the visibility graph. (e) The views from any pair of cameras can be switched through the overview camera (as indicated by the gray arrows in (b)).

for a period of time, they observe similar parts of the scene. Here, the camera merge operator can be applied. At times, the described narrative is non-linear in the sense that the events are not described in their chronological order. In the visual summary, this could be achieved by applying the view-switch operator. The story itself follows the links in the event graph. By traversing the event graph and applying the camera operators, it is possible to generate a visual summary narrating the story as described in the beginning of Section 5.3.1.

To create the summary video of the gameplay, the layouting algorithm arranges the camera views of the events, while it applies the described camera operators. It begins by showing the view of the chronologically first event. The algorithm continues by traversing the event graph in a depth-first order, following the causal links and player-links. In this way, the story progresses by placing semantically relevant events, as determined by the links successively connecting them. The links to the events with higher importance are followed first. When the next event is shown, the camera merge operator is applied to all cameras associated with it, and the camera split operator is applied when the event finishes. This ensures that the events with multiple participating players are not shown multiple times. The depth-first search of the event graph ensures that showing related events in sequence is preferred to applying the view-switch operator between unrelated events. This keeps the coherency of the parallel narrative of the visual story.

4.4.2.1 View-Switch Operator

In our storytelling approach employing multiple cameras capturing the events of the gameplay, it is often necessary to switch between views. Our method tries to sequence the events so that the ones which are temporally adjacent in the summary video are related, as indicated by the edges in the event graph. Since the event graph is not necessarily traceable, i.e., containing a Hamiltonian path, this might not be possible for the whole story. Therefore, we provide the view-switch operator which allows us to seamlessly switch between views of potentially unrelated events.

During the rendering of each frame, we build a *visibility graph*. It is a directed graph, where the nodes represent individual cameras from the flock, and the directed edges from a node indicate which cameras are seen by the camera represented by this node. Figure 4.4a illustrates a set of cameras and Figure 4.4b shows their corresponding visibility graph. The overview camera, which sees all the players, is included in the calculation of the visibility graph. It ensures that a path exists between any two cameras in the flock. For the overview camera, the rendering has to be adjusted, e.g., by making ceilings transparent, so that all the players are visible.

We use the visibility graph for seamless view switching. If the view from a camera should be switched to the view of another camera, we first find the shortest path between nodes in the visibility graph the Dijkstra algorithm [32]. Subsequently, we create a curve spatially connecting cameras of all the views on the path. This curve can be used to smoothly interpolate between the desired views. This approach ensures that the generated curve does not intersect any obstacles in the 3D scene, since the cameras on the path are sequentially visible from each other. Figure 4.4c illustrates an undesired situation if two cameras would be interpolated directly. This would result in collisions with the scene geometry. Our approach employing the visibility graph is illustrated in Figure 4.4d.

The view-switch operator seamlessly interpolates between views of two events. Since the time of the occurrence of these events might not correspond with the length of the view-switch, the current timestep of the gameplay is interpolated together with the camera parameters. This means the time might be dilated during the view-switch. It might even flow backwards, in case the events occurred in reversed order compared to how they are shown in the summary video. The effect of reversing the time while moving the camera to a new location is used in cinematography if it is important to correctly portray the order of the events.

4.4.2.2 Overview Operator

The player-links in the event graph join events where the same players are participating. Since the story is constructed by following these links, it can be split into continuous blocks of events involving individual players. Before each block, we apply the overview operator, which shows a schematic depiction of movements and actions of the involved players from the perspective of the overview camera. Each event is illustrated by a simple animation, while the movement of the players between temporally adjacent events is shown by low-pass filtered paths of their positions recorded during the game.

After the whole event sequence is shown in the overview, we apply the view-switch operator to rewind the time and show the same sequence of events from the viewpoint of the player associated with the events in this story block. If there are several players associated with all the events in the block, any one of them is chosen, since their views are sufficiently similar.

This combination of operators gives the audience an initial overview, followed by a detailed depiction of the events, as seen by the players. The view-switch operator provides a spatial and temporal reference to the other parts of the story, since the camera always moves continuously in space and time.

4.4.2.3 2D Inset-Show/Hide Operator

2D inset operators are used in situations where it is beneficial to show multiple camera views juxtaposed in image space. If a currently shown event caused an event associated with another player, the view of the other player is shown in the 2D inset. For instance, when player A kills player B, we show the view of the player A on the entire screen, while the view of player B is shown in the 2D inset. The same situation is shown from two different perspectives.

4.4.2.4 Camera Operators for Enhancing First-Person Views

If the view of a player moving between related events is shown, the time-lapse operator continuously increases the playback speed, as long as no important events are in the depicted field of view. As the player is getting closer to a certain event, the playback is continuously slowed down. This form of temporal fish-eye lens allows us to connect related, but rather distant events, by a continuous view of the player. In case either the view-switch or time-lapse operator changes the playback speed, a distinct visual style is applied to illustrate the modified time flow (Figures 4.5b, 4.5c). In this visual style, the image is desaturated and it is added a blue tint. Additionally, the borders of the image are deformed to appear wavy.

It is important that the changes in the playback speed are smooth so that they are not distracting the viewers. For this reason, we resample the individual frames in the time domain. Slowing the playback down is achieved by interpolating between successive frames. Making the playback faster is done by averaging several frames. A drawback of both approaches is the introduction of a certain blur. This is visible in Figure 4.5c, where multiple frames are blended together. The playback speed is here increased because the sequence was already shown, and it is displayed again in reversed order only to provide temporal context.

If another player interacts with the player whose view is currently shown, causing an event (such as shooting), the *mark-player* operator is applied. The time stops for a brief moment, and the position of the interacting player is highlighted with a growing circle. The circle is color-coded according to the type of caused event, e.g., shooting or spotting. At the same time, the surroundings of the marked player are darkened, to further guide the attention of the viewers, as depicted in Figure 4.5d.

4.5 Results

In order to evaluate our method, we apply it to a real computer game. We use the multiplayer, first-person shooter game *Jake2* [21], a Java port of *Quake II* by *id Software*, to acquire the gaming data consisting of movements and actions of the players, their views, and game events. These data are processed by our method, which calculates additional views used for smooth transitions between the shown events, and composes the summary video. It also inserts a stopwatch into the generated video which helps to indicate the flow of time. Text captions describing the currently shown events from the event graph are automatically inserted as well. Still images from a generated summary video are depicted in Figure 4.5. The full video is included as supplementary material.

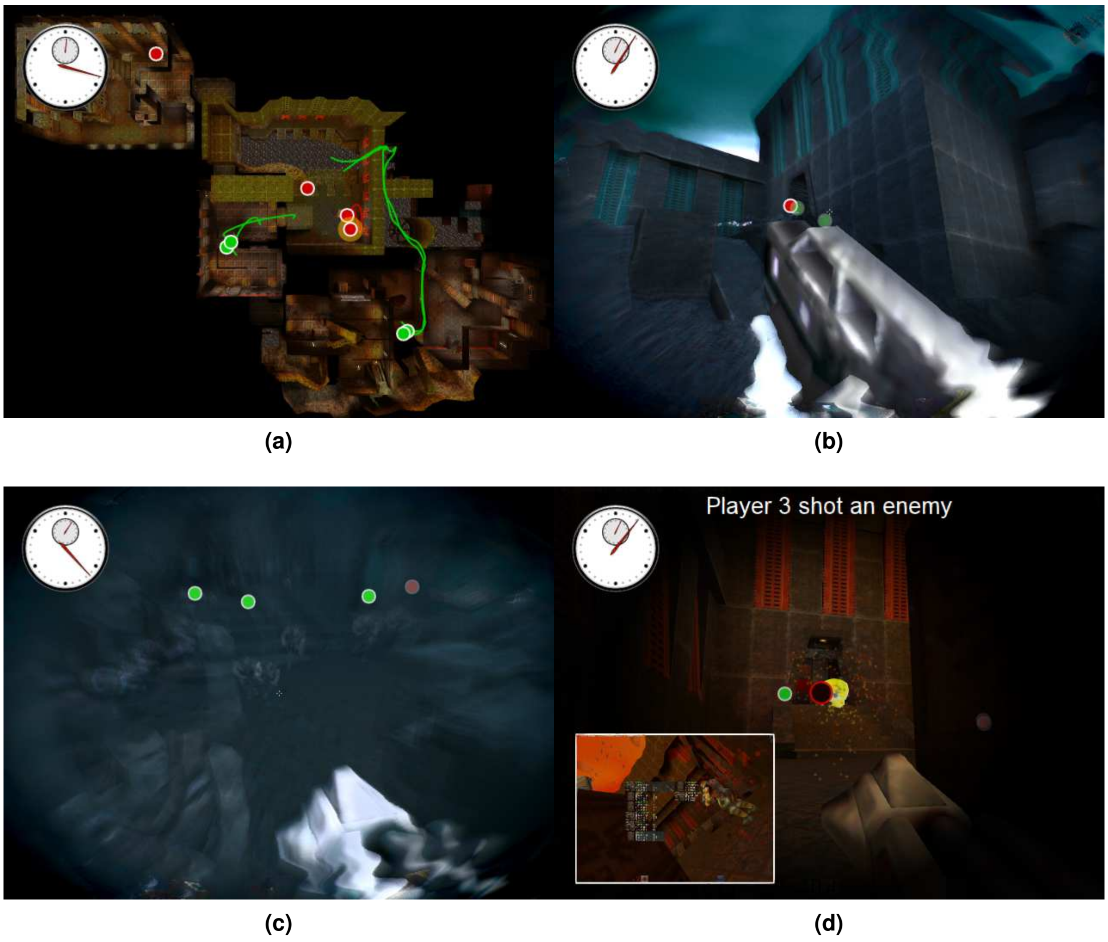


Figure 4.5: A summary video of a gameplay of Jake 2 generated by ManyCams. (a) Overview operator showing the movements of two groups of players from the green team, trying to surround players of the red team. (b, c) View-switch operator continuously changing view towards an event which has already occured. To achieve this, time is reversed, which is communicated through a distinct visual style applied to the entire screen. (d) The mark-player operator (red circle in the middle) indicating a player of the opposing team being shot. Description of the event is shown at the top. 2D inset in the left bottom corner shows the view of the dead player. The surrounding is darkened to guide the viewer's attention towards the event.

Our method is mainly targeted for games played in teams, which often utilize various strategies. Our method is particularly useful for reviewing the execution of such strategies, either for training or entertainment purposes. We recorded several *team deathmatch* games, where the goal is to eliminate all players of the opposing team. There were two teams, each of four players. The players were given strategies, which they followed during the game. The summary video of the game generated by our method clearly shows these strategies and tasks carried out by individual players. We showed the summary video to experienced video-game players. They confirmed that the video sufficiently illustrates the performed strategies and tasks of individual players, even though at some instances it was not immediately clear to them why the events were shown in a particular order.

4.6 Discussion

Our gameplay-summarizing method captures multiple temporarily and spatially overlapping events in a visual story with a parallel structure, which narrates the course of the gameplay. We organize multiple cameras capturing the gameplay into a flock of cameras, and store game events in an event graph. The redundancy in the views shown by the cameras of the flock is reduced by applying several operators. The story is constructed by displaying the camera views capturing the most important events. Since the action in multiplayer games is often complex, smooth transitions are placed between the displayed events so that their spatial relations and temporal succession is communicated. Our method allows users to analyze the behaviour and interactions of participating teams or individual players during the gameplay.

We plan to extend this work in the future by displaying the event graph and allowing interactions with it to steer the narrative. In this way, it would be possible to show story paths which were not selected automatically for the summary. Additional operators could be explored to achieve various cinematographic effects for different games. Various templates for different types of narratives, altering the way how the event graph is traversed, could be explored as well.

4.7 Conclusions

In this chapter, we described a system for generating summary videos of multiplayer gameplays. These summaries are suited for entertainment, or for analyzing and exercising team strategies. The challenge of this approach is to deal with multiple players, who interact with the system and create multiple, possibly concurrent events at different places. Our method smoothly integrates the recorded gaming data of all players, utilizing overviews and time dilation effects, so that the gameplay is illustrated as a continuous and expressive visual story.

CHAPTER 5

Data-Sensitive Navigation

This chapter is based on the following technical report:

P. Mindek, G. Mistelbauer, M. E. Gröller, and S. Bruckner. Data-Sensitive Navigation in Scientific Visualization. *Technical Report TR-186-2-15-01. Institute of Computer Graphics and Algorithms, Vienna University of Technology. April 2015.*

So far, we presented methods for the exploration and visualization of parameter spaces, the management of parameter values specified in changing contexts, and the integration of multiple sets of parameter values created by different users into a single summary visualization. An important aspect of all these methods is human-computer interaction. Efficient means for interaction with user interfaces is crucial for the usability of any algorithm where parameter settings need to be manually adjusted in order to fulfil certain tasks.

An example of such a task is visual data-exploration. Different types of data require special tools to allow domain experts to identify and evaluate features of interest. These tools contain various graphical user interface elements which enable the domain experts to manipulate and steer the visualization of the explored dataset. In application domains such as vascular visualization, the heterogeneous nature of the explored data poses a challenge to the commonly used interaction elements. Their precision might not be sufficient for targeting fine features of interest in medical scans, while in larger homogeneous regions of the dataset their sensitivity might be too low. In other words, the user input is not proportional to the visual change of the generated output. In this chapter, we propose a model for enabling data-sensitive navigation for commonly employed user-interface elements. This model is applied to normalize the user input according to the visual change, and to visually communicate this normalization. In this way, the exploration of heterogeneous data using common interaction elements can be performed in an efficient way without unnecessary overhead. We apply our model to the field of medical visualization and present guided navigation tools for traversing vascular structures and for camera manipulation of 3D volumes. The presented examples demonstrate that the model scales to interaction elements where multiple parameters are set simultaneously.

5.1 Introduction

Interactive visual systems typically feature a wide variety of user interface elements to control visualization parameters. Especially in the case of large, heterogeneous, or complex data, it is challenging to design an interface that offers efficient visual tools for data analysis. Such tools consist of means for the traversal and filtering of data in order to quickly spot and identify regions of interest. Simultaneously, the interface has to allow the users to make fine parameter adjustments, so that features of interest and other subtle elements of the dataset can be explored in detail. This often requires two separate sets of tools which respectively provide coarse and fine parameter adjustments.

User interfaces for the visual exploration and analysis of data are often designed ad-hoc to suit the needs of the specific application. The interaction elements are calibrated to the expected data, while parameter presets are often included to aid the interaction in case the types of input data vary. While this approach is feasible in applications where a certain data type can be expected, it might be ineffective in more general scenarios. The possibility of making assumptions about the processed data is limited in such scenarios, as is the applicability of specific parameter presets. Here, the heterogeneous nature of the input data has to be addressed with more general input mechanisms. Since compromises have to be made in the precision of input elements in favor of accommodating the variability of the underlying data, e.g., increasing the ranges of the input elements, the performance of such input elements is often suboptimal.

Specifically, given the non-linear nature of the visualization mapping functions used to display the data, the changes in the user input do not always proportionally reflect the visual changes in the output image which can make the exploration process counter-intuitive. This is, for instance, often an issue in general-purpose volume rendering applications, where it is not always possible to prepare adequate presets. In such applications, transfer functions have to be adjusted manually which can be challenging, particularly for users without a strong technical background. Small changes in certain parts of the transfer function may cause significant changes in the visualization, while modifying the rest of the transfer function does not have a strong effect on the output image. In general, it is hard to predict how the changes of the transfer function will influence the output image.

Similar problems exist in colorimetry. Most of the color spaces are perceptually non-uniform, such as the RGB, HLS, or *CIE xyz* color model. Changes of color will be perceived differently by the human observer depending on their specific location in the color space. For this reason, color spaces such as *CIELUV* were devised which are perceptually uniform. Here, a unit color change leads to a unit perceptual change independent of where in the color space this change occurs. We address a similar issue, but in the area of human-computer interaction with interactive visual systems. Our goal is to make changes in the input parameters more predictable to the perceived changes in the output image and to actively support users in the anticipation of the expected effects of the interaction.

For this purpose, we introduce our model of *data-sensitive navigation* consisting of two elements:

- *Data-sensitive manipulation* describes how the behavior of input elements is modified so that the changes of the visual output are made proportional to the changes of the user input.

- *Data-sensitive guidance* is a way of displaying the information used for normalizing the output changes to help users steer the interaction towards specific goals.

Based on our concept we develop *TreeSlider*, a novel data-sensitive navigation tool for the investigation of tree structures in spatial data. Specifically, we address the field of angiography where vessel-trees need to be traversed in order to identify potential pathologies such as stenoses or occlusions. Radiologists typically have to inspect hundreds to thousands of slices, and important pathologies can be easily missed. By adapting the interaction mechanism to the underlying data and explicitly encoding information about potentially important regions in the data, we show how our approach can assist users in this tedious and time-consuming task. Furthermore, we apply our approach to the more general scenario of camera manipulation where we show how viewpoint relevance measures can be employed to provide additional visual guidance in the interaction with 3D visualizations.

The chapter is structured as follows. In Section 5.2 we review related work. Our model of data-sensitive navigation is introduced in Section 5.3. In Section 5.4, we present TreeSlider, our novel data-sensitive interaction tool for vessel-trees. We then show how our approach can be applied to camera manipulation in Section 5.5. Further results are presented and discussed in Section 5.6. The chapter is concluded in Section 5.7.

5.2 Related Work

While the motivation of our work are specific tasks from the field of medical visualization, our model is generally applicable to various computer graphics areas where interaction is employed as well. Therefore, we subsequently review existing literature from both areas, namely interaction in computer graphics as well as medical visualization.

Non-linear Interaction. Lindow et al. [71] present a method for transforming input parameters of visualizations to obtain a linear relationship between the input and the output. Gavrilescu et al. [42] customize common interaction elements by visualizing the amount of change caused by interaction with these elements. In this chapter, we propose a general model which unifies these two concepts considering the user input as a distinct space. We apply our model to more complex input elements, such as the novel TreeSlider, in order to demonstrate its utility in a broad application spectrum, especially when existing methods are not usable. Van Wijk and Nuij [112] describe a metric for efficiently navigating between two regions in large 2D environments. Their non-linear approach simultaneously performs zooming and panning and and ensures a constant change of velocity. We also employ a non-linear technique to gradually change the speed of scrolling between important regions of the underlying data. Blanch et al. [13] present the *semantic pointing* interaction metaphor. Objects within a user interface have two different size properties: One in motor space (importance for interaction) and the other one in visual space (the displayed visual size). An example would be the adaptation of the distortion of a fish-eye lens with respect to the distance moved in screen space. Elmqvist and Fekete [37] describe several mapping functions for picking objects in a 3D scene by adjusting the ratio between motor and visual space. In a region without target objects, the cursor moves fast and is enlarged, whereas when approaching an object, the cursor progressively becomes smaller and slower. Chapius et al. [23] present an interaction technique that couples the size of the cursor to its speed while minimizing visual distraction. Approaching an object (the closest one), the area of the cursor decreases until it behaves like a common point-cursor when hitting the object. Elmqvist et al. [38] propose a space deformation technique for exploring large geometric spaces such as maps or networks. They fold the space similar to an accordion in order to reduce the consumed screen size while simultaneously preserving the overall context. Ji and Shen [60] describe an approach that selects the optimal view of a static scene by analyzing the opacity, color and curvature of images from various viewing angles. By employing an optimization to maximize the perceived information, they achieve smooth and constant visual changes of the optimal viewpoint in dynamic scenes of time-varying data. Kohlmann et al. [63] present an interaction metaphor for linking 2D slice views with 3D volume rendering. Depending on a minimal set of input parameters such as patient orientation, local object shape and viewpoint history, an optimal viewpoint of a user-picked slice position is determined. In their follow-up work [64], they additionally adjust the transfer function within the 3D view in order to reveal the structure of interest without obstruction. In this chapter, we propose a general model for data-sensitive navigation towards interesting spots in 2D as well as 3D.

Angiography. The analysis of blood vessels is a crucial and fundamental procedure in medicine for investigating embolisms, calcifications, stenoses, or occlusions within the lung, the brain or the peripheral arteries, such as the legs. All these examples share a tree of the arterial system as basis for their analysis. Starting with the most basic but wide-spread procedure for analyzing vascular pathologies, all axial or transverse slices of an acquired medical volume dataset have to be viewed and precisely inspected. Since this demands a lot of manual work from physicians, Kanitsar et al. [61] discuss Curved Planar Reformation (CPR) that creates a cut along a blood vessel in order to inspect its interior (or lumen) in an obstruction-free manner. This approach significantly reduces the number of images to investigate. Auzinger et al. [4] propose a technique to view the lumen of blood vessels from unrestricted viewing angles. Portugaller et al. [89] investigate the importance of viewing slice-images together with supporting techniques such as Maximum Intensity Projection (MIP). They conclude that transverse slices still play an essential role for detecting arterial lumen narrowings, although techniques such as CPR reduce the number of image to analyze. Motivated by this fact, we enhance the well-established and accepted role of transverse slice images by a data-sensitive notion. The layout of vascular structures is important when browsing through the vessels. In order to reflect the spatial arrangement of the investigated blood vessels, Borkin et al. [15] developed a horizontal arrangement for analyzing the endothelial shear stress of heart arteries. Although this layout represents the vascular system well, its spatial orientation would be problematic if investigating arteries in general, such as the ones of the human lower extremities, since these are mostly vertically oriented. We propose an extended interface element that remedies this problem while utilizing a well-known interaction paradigm, the slider. Inspired by the simplicity of a slider and the necessity to browse through a vessel-tree, we combine both aspects into one interface element, called *TreeSlider*. Oeltze and

Preim [84] describe an organically looking visualization for vascular structures based on convolution surfaces. The vessel-tree is convolved with a Gaussian filter and artifacts at branching points, such as bulging or blending, are handled by reducing the kernel size. Wu et al. [120] propose an adaptive refinement for surface-based vascular visualization to have more polygons in regions with high curvature. A comparison of different surface modeling approaches for visualizing blood vessels is given by Wu et al. [119]. Based on the focus and context principle for vascular structures proposed by Straka et al. [101] we blend the vessel-tree over a MIP of the underlying dataset.

5.3 Data-Sensitive Navigation

To address various problems arising in the interaction with visualization systems through diverse input elements, we present a model of data-sensitive navigation. This model can be applied as a basis for enhancing interaction in various vessel visualization scenarios. In its generality, the model could be employed in other application areas as well.

5.3.1 Model Overview

In visual computing, interaction consists of a feedback loop between the user input and the visual output. The user sets the parameters, he observes the generated output image, and refines the parameter settings accordingly until the output is satisfactory. To model this process, we split the data-sensitive navigation into two parts - *manipulation* and *guidance*. Data-sensitive manipulation transforms the user input with respect to the underlying data, so that the changes in the input are made proportional to the changes of the output. Data-sensitive guidance augments the visual output with information derived from the remapping used in the manipulation stage. The guidance informs the user about the effects of potential future changes of the input elements. Depending on the interaction method, the guidance information can overlay either the input elements themselves or the output image. The location of the overlay is chosen so that the user is not distracted during the interaction, yet the guidance information is properly communicated.

Figure 5.1 gives an overview of our proposed model. In Figure 5.1a, traditional interaction is illustrated. Changes of the user input are proportional to the changes of the parameter values, which often do not have direct semantic relevance to the user. Changes in the output image, providing the only visual feedback for the user to steer the input, are typically not proportional to the input changes. The worst-case scenario is that certain desired outputs are impossible to achieve with the given input elements.

Our model addresses this problem in two different ways. Figure 5.1b shows *data-sensitive manipulation*. The goal of this concept is to non-linearly modify the sensitivity of arbitrary input elements to make the changes of the input proportional to visual changes of the output images. The sensitivity of an input element is the amount of parameter-value change caused by a unit interaction, i.e., how much does the underlying parameter value change when a slider handle moves a unit distance. This non-linear sensitivity modification ensures that the screen-space of the input elements is properly utilized for efficiently controlling the changes of the output. For

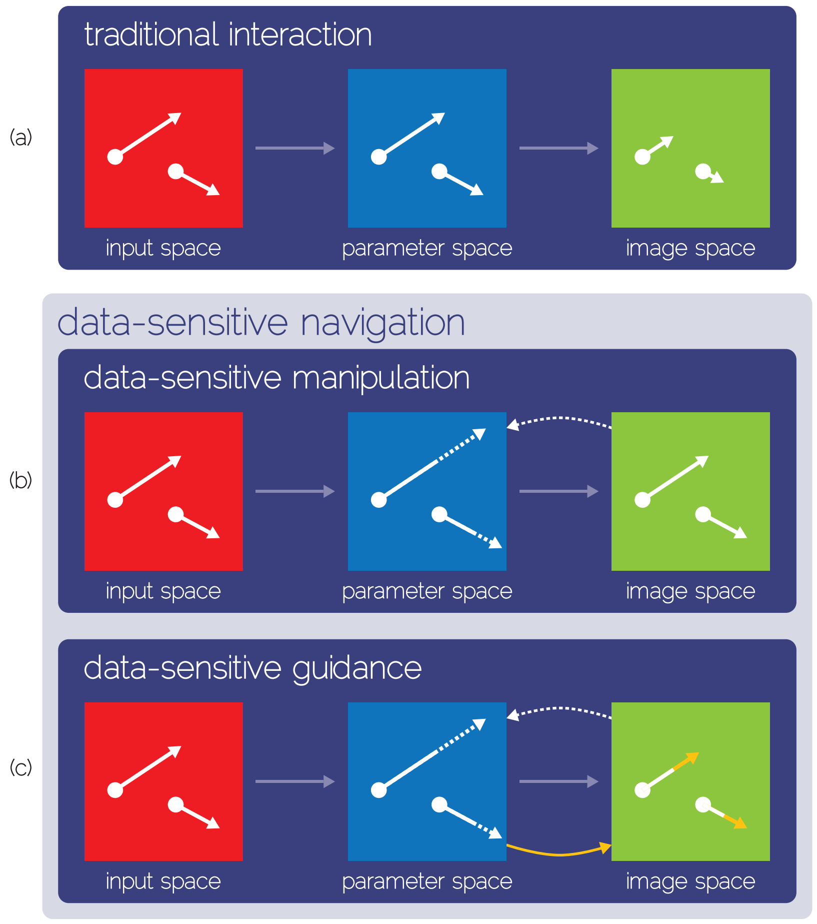
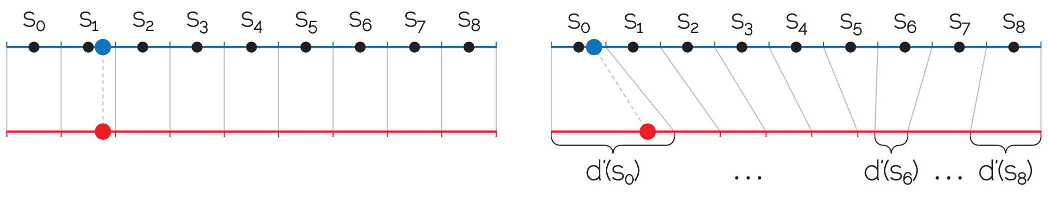


Figure 5.1: White arrows represent changes in input space, parameter space, or the output visualization. (a) User input is directly mapped to the input parameters. The changes of the input are not proportional to the changes of the output. (b) Data-sensitive manipulation, where the underlying data or the output are used to dynamically scale the changes in the parameter space. In this way, changes in the output are proportional to the changes in the input space. The mapping is illustrated with the dashed arrows. (c) The mapping information is displayed in the image space to guide users during the data-sensitive navigation (yellow arrows).

instance, a slider could be non-linearly rescaled so that larger portions of the slider track map to the areas of high interest, thus making the sensitivity of the slider lower around these areas.

On the other hand, data-sensitive guidance (Figure 5.1c) displays gathered information about interesting aspects of the underlying data in a way that helps users to find them more quickly. An example is a slider augmented with non-uniform ticks, indicating non-linear changes in the output with respect to linear slider changes. This form of visualization reduces the amount of trial-and-error visual exploration by steering users in the specification of parameter settings to obtain desired results. Depending on the type of the input element, this information can be



(a) (b)

Figure 5.2: Transformation T of an input to get a parameter value. The red dot indicates the input, e.g., a slider position, the blue dot shows the parameter value. The black dots are the sample positions s_i within the parameter space, while $d'(s_i)$ is the normalized importance $d(s_i)$ of the sample s_i. (a) Transformation T if importance values of all samples are uniform (data-sensitive manipulation is disabled). (b) Transformation T if the samples have non-uniform importance values (data-sensitive manipulation is enabled).

shown in the output image, or it can be used to augment the input element itself to make it easier to use. This is illustrated by yellow arrows which represent the displayed guidance information.

5.3.2 Data-Sensitive Manipulation

Our goal is that our proposed model can be applied to various input elements which are used to control parameters of different types. Therefore, we use a formalism which describes the problem of non-proportionality between input and output in a general way.

A common characteristic of input elements is that they are used to gather user input I, which is transformed into parameter settings P. These parameter settings are used to generate an output image O. Typically, the changes in the input are not proportional to the changes in the output. This is due to the fact that there is usually a non-linear relationship between P and O, while I is linearly mapped to P ($\Delta I \propto \Delta P \not\propto \Delta O$).

To rectify this situation, it is necessary to apply a transformation T which introduces a non-linear relationship between I and P:

$$P = T(I) \mid \Delta I \propto \Delta O \tag{5.1}$$

During data-sensitive navigation, parameter values are determined by the transformation $T(I)$ instead of the traditional linear mapping of I to P. This makes interaction with the input elements proportional to the visual changes of the output images.

In most cases, the transformation T cannot be defined analytically, as this would require a complex mathematical model of the visualization mapping as well as of the underlying dataset. Such a model is typically not available. Therefore, we assume that it is possible to define a function $d(P)$ which estimates the *importance* of the respective output image O generated from the given parameter settings P. The transformation T is then constructed from d as described below. The function d is obtained by sampling the parameter space and applying an adequate interpolation. Such an approach was employed by Lindow at al. [71].

Figure 5.2 illustrates how the transformation T is constructed from the sampled importance function d. First, n samples of the function d are taken by analyzing the corresponding output images. These sampled importance values are then normalized so that their sum is equal to the range of the given input element. The normalized sample values are denoted as $d'(s_i)$. The input domain is split into regions corresponding to individual samples. The size of each region is proportional to the normalized importance $d'(s_i)$ of the respective sample s_i. If all outputs would have equal importance, the input domain would be split into regions of equal size and the input would be mapped to the parameter values in a traditional, linear way (Figure 5.2a). If each output would be assigned a different importance, parameter-space regions with higher importance would be assigned larger regions of the input domain, e.g., ranges in a slider (Figure 5.2b). Essentially, the sensitivity of the input element is modified in such a way that its resolution is increased in those areas which account for more visual changes in the output image.

5.3.3 Data-Sensitive Guidance

As previously explained, data-sensitive manipulation changes the way how the user input maps to the displayed output in traditional interactive environments. This mapping, achieved through the transformation T, allocates more screen-space of the interaction elements to those parts of the parameter space which account for greater visual changes of the output images. However, this might not always be apparent by observing the output image alone. Even though the resolution of the input element is increased in potential regions of high interest, the non-linear mapping between the input and the output through the abstract parameter space might cause confusion for the user. Therefore, in addition to to data-sensitive manipulation, we employ the concept of data-sensitive guidance. Data-sensitive guidance is a way of displaying the importance function in image space to steer the user while interacting with the system. In this context, the image space is either the output visualization, any linked view, or the portion of the screen where the graphical representation of the input element itself is placed. Therefore, it is possible to show the guidance information wherever it is most suitable for the given application and interaction type. This is illustrated in Figure 5.1c as the yellow arrow between the parameter space and the image space.

Since there are many different ways how the importance can be shown, we categorize them into two groups:

- *Local encoding*: The importance value is shown for the current parameter settings or its close neighborhood. This type of encoding is useful when a continuous path within the parameter space needs to be explored by the user. The local encoding of the importance guides the user where to go next from the current parameter settings, or how fast can they proceed. For instance, when traversing blood vessels, the user moves through the vessel continuously. It is unlikely that they will jump from the current position within the vessel to other than neighbouring positions. In this case, local encoding is sufficient to provide the guidance in the traversal process.
- *Global encoding*: The importance is visualized for the whole input domain or the parameter space. This type of encoding of the guidance information is useful if the parameter

space is not necessarily explored in a continuous manner. In this case, it is not clear how the user should proceed in the exploration from the current parameter settings. Therefore, the importance is shown for all available choices to help the user decide to how to change the input. An example is selecting adequate viewpoints in the visualization of 3D objects. It is not essential that the viewpoints are continuously explored, but good ones need to be found. In this case, the global encoding of the importance is appropriate.

Displaying importance instead of (or in addition to) the non-linear mapping of the input to the parameter values is an effective way of guiding the user towards areas of potentially high interest within the explored data. In the following sections we present several ways how this type of visualization can be utilized to steer the navigation in interactive visualization systems.

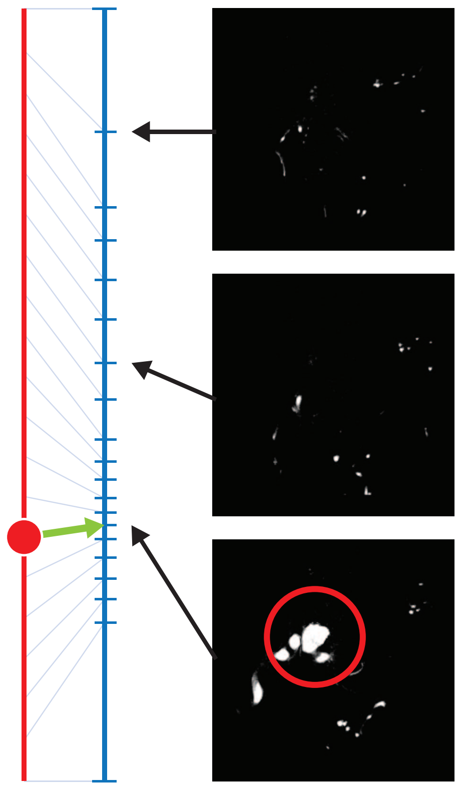
Figure 5.3 illustrates data-sensitive navigation on a simple vertical slider used for traversing a stack of slices of a volume dataset. The slider (shown in red color) can be moved to show a particular slice within the volume. The sensitivity of the slider is modified using data-sensitive manipulation, so that it provides more resolution around the areas of interest (in this case, an aneurysm). A simple pixel difference between consecutive slices is used as the importance function, while the transformation T is constructed as illustrated in Figure 5.2. In this way, the slices where a lot of visual changes occur are considered as areas of high interest. Because of the uneven shape of the aneurysm, this metric assigns high importance to the areas around it.

The importance function used to construct the transformation T from the input to the parameter value is visualized next to the slider in blue color. This visual guidance in the form of ticks indicates how the screen-space of the slider is allocated to individual parts of the slice stack. Areas with higher importance values are indicated by a higher density of the ticks. This is an example of global guidance encoding. It gives the user the possibility to quickly locate areas of potential interest, as well as means for estimating their extent. The green arrow points from the the slider handle to the current position within the stack. It links the current slider position with the visualized importance function. If it points to a group of ticks close together, it indicates the high importance of the current slice. The green arrow is an example of local guidance encoding.

5.4 Data-Sensitive Vessel Traversal

The goal of this chapter is to describe how our proposed model of data-sensitive navigation can be employed to address interaction issues in medical visualization. Specifically, we want to enhance the usability of various input elements so that the interaction with them is adjusted according to the underlying data. In the simple example of single-value slider presented so far, only the generated images, i.e., volume slices, were considered. However, in medical visualization, this might not always be a sufficient way to model areas of interest within the underlying data. In some cases, high-level structure, such as branchings of blood vessels, have to be considered as well to assign the importance to individual areas. To demonstrate that it is possible to apply our model in such cases as well, we apply data-sensitive navigation to the traversal of blood vessels.

The visual examination of vascular structures is an important topic in medical visualization [4, 61, 84]. Vascular pathologies such as stenoses, i.e., an abnormal narrowing of blood vessels,


(a)

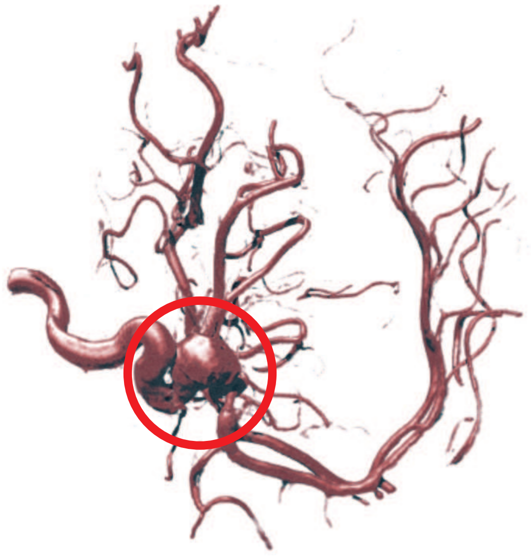

(b)

Figure 5.3: Data-sensitive navigation applied to a simple slider for volume slicing of a CTA scan of an aneurysm (marked with the red circle). (a) The red line represents a slider. The visual guidance information shown to enhance the usability of the data-sensitive slider is depicted in blue (global encoding) and green (local encoding) color. (b) A volume rendering of the entire dataset for overview.

or vascular calcifications, i.e., a deposition of material inside a blood vessel potentially leading to a blockage of the blood flow, are often small in relation to the acquired data. However, their identification is of critical importance. Other regions of interest include stents, bypasses, or vascular occlusions, i.e., blockage of the blood flow through a vessel.

Since vascular structures usually consist of many branches, examining all of them simultaneously or sequentially is a cumbersome task. The vascular structures are modelled by a tree referred to as *vessel-tree*.

To free physicians from the laborious process of inspecting the entire vessel-tree by repeatedly choosing a path and traversing it using a simple slider, we introduce a novel interaction element, called *TreeSlider*. Using TreeSlider, it is possible to traverse the entire tree without the necessity of traversing the parts of the tree common for more paths multiple times. Addition-

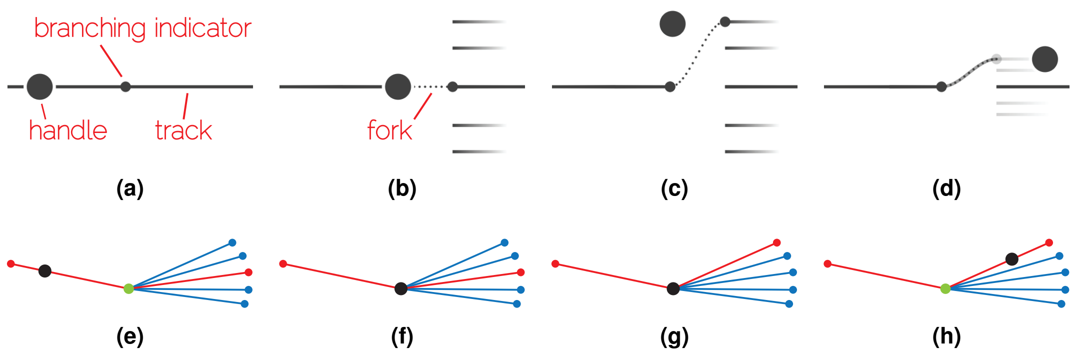


Figure 5.4: Different states of a TreeSlider are shonw in (a) to (d). In (e) to (h), the traversed tree is shown. Red color indicates the active path, the green circle represents a branching node, while the black circle represents the current position set by the TreeSlider. (a) TreeSlider and its components (b) If the handle is moved close to a point where the tree branches, a fork continuously appears. (c) If the handle is within the fork area, it can be moved vertically to choose other branches of the tree for traversal. (d) The traversal continues on the chosen path.

ally, it allows the user to easily navigate through the vessel-tree using data-sensitive navigation considering not only the volume data, but the vessel-tree as well. Since the TreeSlider is used to traverse trees, this example shows how data-sensitive navigation can be applied in a more complex scenario than a simple single-value slider.

5.4.1 TreeSlider

We present our novel interaction element for the traversal of tree structures. The TreeSlider has to comply with the following requirements: it has to be scalable with respect to the size of the traversed tree, it has to allow users to visit every part of the tree, and it has to support data-sensitive navigation. Scalability means that the image-space dedicated to the TreeSlider has to be as small as possible, while allowing the users the traversal of small as well as large trees. This is a challenging task since the trees may contain a large number of branches.

To satisfy these requirements, we utilize the design of a classic slider, which consists of a single line called *track*, and a *handle*. The handle can be dragged along the track. The position of the handle represents the slider value, which is mapped to a position on a specific path within the tree, i.e., a path between the root node and a leaf node. Since only one path is traversed at a time, the screen-space of the slider does not have to account for all the tree branches at the same time, thus maintaining scalability. The components of the TreeSlider are shown in Figure 5.4a.

The second requirement dictates that the user has to be able to visit every part of the tree using the TreeSlider. Therefore, the path mapped to the slider track can be changed through an element called *fork*. Forks are elements displaying all branches of the tree originating in a specific node. Forks are placed on the slider track at positions corresponding to these nodes where the tree branches (nodes with more than one child). Through these nodes, which we refer

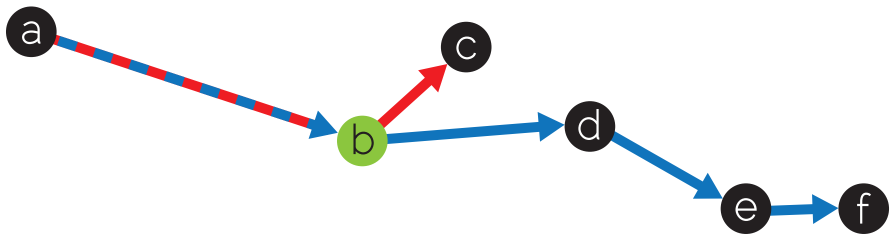


Figure 5.5: A directed acyclic graph with two paths (marked blue and red). The relative positions of the fork node *b* on these two paths are not equal. Therefore, if the user changes between the paths, the position of the fork node *b* has to be remapped so that the respective branching indicator does not move during the switching of the paths.

to as *fork nodes*, the currently traversed path can be changed. By default, the forks are invisible and their positions on the track are illustrated by dots called *branching indicators*. Other nodes, with no more than one child node, are not displayed on the TreeSlider track, since only one path goes through each of these nodes.

If the handle is near a branching indicator, the fork appears. It consists of several parallel lines, each representing one alternative path where the user can go from the respective branching node. At this point, the handle can be moved vertically to switch between possible paths. This process is illustrated in Figures 5.4b and 5.4c. As the user moves the handle further along the track, the fork continuously collapses to bring the handle back to the track (Figure 5.4d). Now the track maps to a different path within the tree. Figures 5.4e to 5.4h show how the traversed path is changed using the fork.

The problem with switching the paths in fork nodes is that the relative positions of the fork node within different paths might be different. The problem is illustrated in Figure 5.5. Suppose the current path is the path marked in blue. If the handle reaches the fork node *b*, it is possible to switch the current path to the one marked in red. However, the position of the fork node *b* is much further within the red path than it is within the blue path.

This would result in discontinuities in the interaction with the TreeSlider. Therefore, when switching the paths through a fork node, its position within the new path is linearly remapped so that it matches the position within the previous path. This is achieved by scaling the portion of the TreeSlider track representing path before the fork node and the portion after it. In this way, the respective branching indicator does not move during the path switching. As the user moves the handle away from the fork after switching the paths, the scaling factors of both parts of the TreeSlider track are continuously interpolated to the original values. In this way, the remapping is continuously removed as the user is moving the handle. This allows users to seamlessly switch between different paths within the tree.

This mechanism might cause discontinuities in the interaction if two forks nodes are very close. In this case, it might be impossible to smoothly remove the remapping before the second fork is reached. Therefore, for multiple fork nodes, which are closer than a specified minimum distance, we only use a single fork. This fork contains all the branches of these fork nodes. This

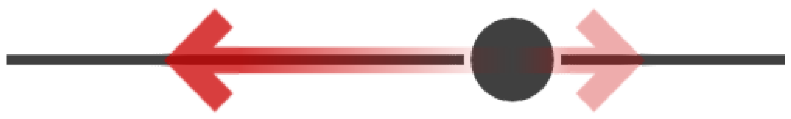


Figure 5.6: TreeSlider with guidance. The red arrows indicate how will the sensitivity of the slider change if the handle is moved either left or right. A longer and more opaque arrow means higher sensitivity, which indicates a lower importance in this direction.

is useful in scenarios such as blood-vessel traversal, as the vessel-trees rarely contain nodes with more than two branches. However, several branching nodes are commonly very close together. Using a single fork for these fork nodes is therefore an efficient option.

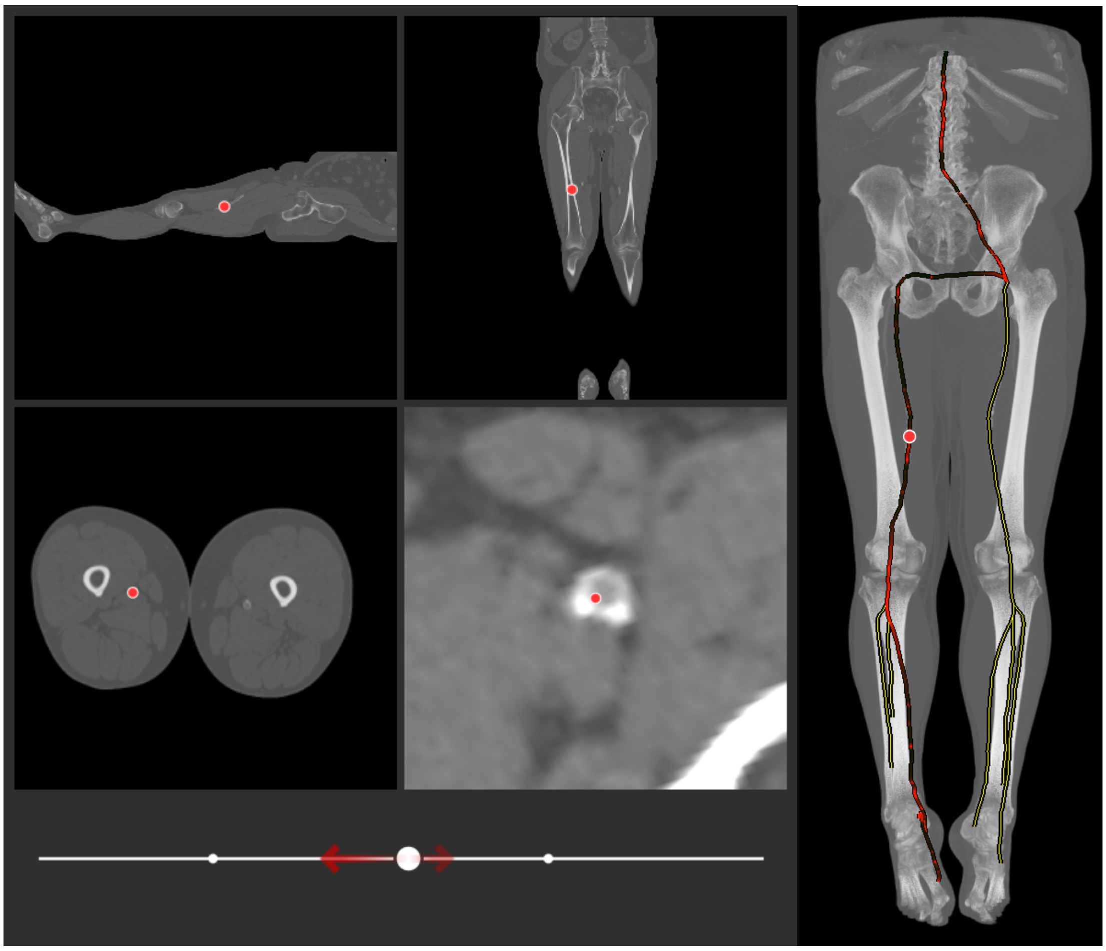
5.4.2 Data-Sensitive Navigation with the TreeSlider

The traversal of trees with the TreeSlider is a complex interaction. There is no straightforward way of applying the transformation T to the input to obtain parameter values P, since the P values also depend on what path within the tree is currently traversed. Therefore, data-sensitive navigation cannot be directly applied to the TreeSlider as is the case with the simple slider. The input of the TreeSlider has to be modified so that the transformation T can be applied to it, and the changes of the output can be made proportional to the changes of the input.

To achieve this, we split the traversed tree into segments separated by the fork nodes. Between each pair of fork nodes, there is exactly one path, since the tree branches only in the fork nodes. Therefore, each segment can be linearized and the slider position within the segment can be transformed by T. For this reason, it is required that the importance function is defined along all paths of the vessel-tree, so that the transformation T can be constructed separately for each segment. Each segment of the tree is treated as a simple data-sensitive slider.

The importance along the paths within the tree is also used for data-sensitive guidance. As the TreeSlider is used to continuously traverse tree structures, a local encoding of the guidance is adequate. We display the guidance as two arrows pointing forward and backward, as shown in Figure 5.6. These arrows indicate the sensitivity of the slider should the handle move in the respective direction. A longer arrow, indicating higher sensitivity, means that there are less interesting areas in that direction. On the other hand, a shorter arrow means that the parameter value will change slowly if the slider moves in this direction, since there are areas of high importance. This information allows the user to predict the behavior of the slider when moved in a certain way.

To calculate the guidance information, we sample the importance function along the current path within a predefined distance before and after the current position. A weighted average is calculated, where the weight is the distance from the current position. The averaged importance values before and after the current position are mapped to the lengths of the respective arrows. In this way, the information about the neighborhood of the current point is communicated.



(a) **(b)**

Figure 5.7: Traversal of a vessel-tree in a CTA scan using the TreeSlider. (a) Axial slices through the selected point (marked with red circles) and a slice perpendicular to the vessel-tree in the current position are shown above the TreeSlider. (b) The vessel-tree is overlaid on top of a 3D visualization of the dataset as an overview. All views in (a) and (b) are linked together.

5.4.3 Traversing Vascular Structures

Our motivation for designing the TreeSlider was to be able to use data-sensitive navigation for the traversal of vascular structures within volume data modelled by a vessel-tree. The TreeSlider allows users to move a cursor along paths within the vessel-tree. A rectangular slice of the underlying data perpendicular to the vessel-tree at the position of the cursor is taken and presented to the user. The perpendicular slices are calculated using *Rotation Minimizing Frames* [115]. In this way, the blood vessels can be traversed.

To make the interaction data-sensitive, it is necessary to define the importance function

$d(P)$, which estimates the relevance of individual slices along the vessel-tree. To find a suitable importance function, one has to consider how doctors are looking at stacks of volume slices. While moving along tubular structures, such as blood vessels, the doctors look for any structural changes along the way. Such changes indicate areas of interest, since long uniform segments usually do not provide any interesting information. The changes might be subtle, but important to identify.

To model this approach, we define the importance function $d(P)$ for a given parameter setting P (position within the vessel-tree) as the difference between consecutive slices along the vessel-tree at the position P. More specifically, only a small rectangular area around the blood vessel is taken into account. To calculate the differences, we use a simple *mean squares* metric, which takes the sum of squared differences in intensities of corresponding pixels. This importance function $d(P)$ is used to construct the transformation T so that the data-sensitive manipulation is enabled. The resolution of the TreeSlider is effectively increased in those areas where a lot of changes in the slices around the blood vessels are occurring.

Additionally, we also want to take the structure of the vessel-tree into account when modifying the resolution or sensitivity of the TreeSlider. The most important structural features of the vessel-tree are branchings, i.e., the fork nodes. Therefore we increase the importance values for all points along the vessel-tree which are close to a fork node. In this way, the resolution of the TreeSlider is also increased near branchings of the vessel-tree. This shows that complex data models can be used to construct the transformation T of the data-sensitive navigation.

Figure 5.7 shows our interface for vessel traversal using the TreeSlider. In Figure 5.7a, the TreeSlider with the locally encoded guidance (red arrows) is shown. Above the slider, four slices of the volume data are shown. These are three axial slices passing through the cursor position, and a slice perpendicular to the path in the vessel-tree. These views are linked with a 3D overview visualization (Figure 5.7b), where the entire vessel-tree overlays a Maximum Intensity Projection (MIP) of the dataset. In each of these linked views, the cursor position is depicted as a red dot. The 3D overview also contains globally encoded guidance information, where high importance along a blood vessel is shown in red, while low importance is indicated in black. In this way, the users immediately see areas of high importance.

5.5 Data-Sensitive 3D Object Rotation

The aim of our model is to be a basis for interaction enhancement of various input elements used in visualization systems. So far, we only considered elements which set a single parameter at a time. However, our model makes no such requirements, as P can represent a set of parameter settings rather than a single parameter. Indeed, in medical visualization, input elements which specify two or more parameters at a time are commonly employed. Examples include *windowing*, where contrast (*window*) and brightness (*level*) of an image are set simultaneously by dragging the mouse in a 2D plane. Another example is an unrestricted 3D object rotation, where the movement of the mouse maps to an orientation of the 3D object.

In medical visualization, 3D volume rendering can be used to provide an overview of the analyzed data. In our vessel-traversal example, we use 3D visualization to show the entire vessel-tree and the current cursor position in the context of the underlying data displayed using volume rendering. Even though this visualization serves merely as overview, it is important that a desired viewpoint can be selected easily. To aid users in the manual selection of the viewpoint in 3D visualization, we apply the data-sensitive navigation to an Arcball rotation widget [98]. Since the input of the Arcball widget consists of two parameters (2D mouse position) which are specified simultaneously, this application demonstrates the scalability of our model to interactions with higher dimensional inputs.

5.5.1 Sensitivity of 3D Rotation

The goal of applying data-sensitive navigation to the Arcball widget is to modify the sensitivity of the rotation according to the displayed 3D data. In areas of high importance, the sensitivity should be *low*. The users should be able to make fine adjustments in object orientation when these areas are shown. On the other hand, if the importance is low, the sensitivity should be *high*. This is because if an unimportant view of the 3D object is displayed, the users will most probably want to quickly rotate towards more important views, and fine adjustments would not be performed here.

When using Arcball rotation, the camera moves on a surface of a sphere bounding the displayed 3D object. In order to use data-sensitive navigation with the Arcball widget, it is necessary to provide a measure of importance defined on the surface of the bounding sphere. Therefore, the importance function $d(v)$ has to be defined for an arbitrary viewpoint v. The value of $d(v)$ is used as the sensitivity measure for the viewpoint v, where high importance maps to low sensitivity and vice-versa.

Since we do not know in advance how the displayed object will be rotated, the importance has to be pre-calculated for every possible orientation. This is achieved by sampling the importance function at several viewpoints uniformly distributed on the surface of the bounding sphere and reconstructing it through interpolation. We implement two interpolation strategies - *nearest neighbor* and *weighted average*. The nearest neighbor algorithm is fast, and in case of a large number of sampling points, various acceleration data structures can be employed [10, 22, 45]. However, the weighted average creates a smooth reconstruction of the importance function, which might result in a smoother interaction. Both strategies are illustrated in Figure 5.8.

The purpose of the Arcball rotation widget is to allow users to select an appropriate orientation of the displayed 3D object. Here, the appropriateness is judged based on what is visible from a particular viewpoint. This should be considered when specifying an appropriate importance function $d(v)$. For each viewpoint v, it should evaluate how important is the image rendered from v. The advantage of this approach is that it can be used with any rendering pipeline, e.g., volume rendering or mesh rendering, since it operates on the rendered images rather than on the underlying data.

Unlike in the previous scenario, where slices of the volume data were examined one by one, in the 3D rotation, importance of the viewpoint does not always depend on the neighbouring viewpoints. Indeed, in the rotation of 3D objects, the change of the displayed features does not necessarily indicate high importance. Therefore, in this application, we base the importance on the single image generated from the given viewpoint. For this, various image-based metrics for estimation of viewpoint quality could be used.

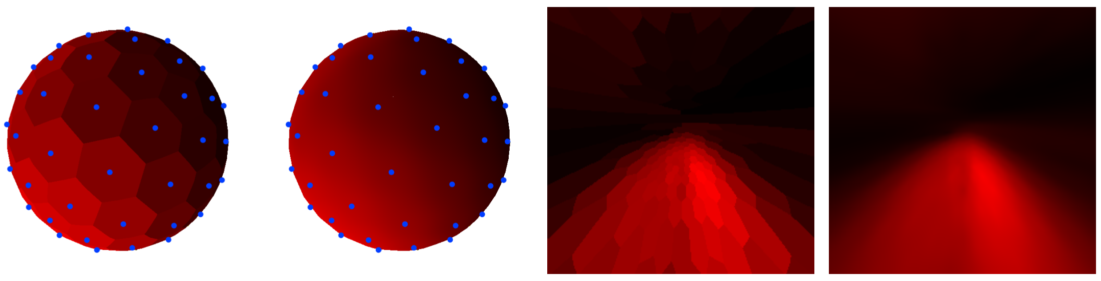


Figure 5.8: Reconstructed importance function for each possible viewpoint (a, b) and for each possible user input (c, d). Red color indicates high importance, black color indicates low importance. The blue dots represent the sampling positions. We employ either nearest neighbor (a, c) or weighted average (b, d) interpolation of the samples.

Several approaches for viewpoint quality estimation have been proposed in computer graphics [113] and visualization, both in a generic [14] and feature-driven manner [104]. To demonstrate our concepts, we use a simple image-based metric. It uses the Sobel operator to detect edges in the generated image, and sum the pixel luminosities. The assumption of this approach is that more distinct features will result in more edges, which are highlighted by the Sobel operator. Therefore, higher importance is assigned to those viewpoints, which show more features of the 3D object. On the other hand, flat and uniform areas are assigned lower importances. The metric for the evaluation of the importance could be easily replaced to suit different applications, e.g., estimating the visibility of objects of interest, such as in Viola et al. [114]. This is due to the fact that the metric is independent from the rest of the pipeline.

5.5.2 Data-Sensitive Navigation in 2D

The Arcball widget takes a 2D input and converts it into a corresponding rotation of a 3D object. In this section, we explore how data-sensitive navigation can be applied in this 2D scenario.

The Arcball widget recognizes three interaction events: *mouse-press*, *mouse-move*, and *mouse-release*. On mouse-press, the initial rotation r_0 represented as a single quaternion is stored. The initial mouse position p_0 is stored as well. On mouse-move, the current mouse position p_c is recorded, and it is used together with the initial mouse position p_0 to calculate an axis and an angle of a rotation of the displayed 3D object. The axis and the angle are stored as quaternion Δr. The current object rotation r_c is then calculated as $r_c = r_0 \Delta r$. On mouse-release, the rotation is stopped.

In this approach, the rotation of the object is specified by the input vector $\vec{m_c} = p_c - p_0$. If $|\vec{m_c}| = 0$, the rotation remains unchanged ($r_c = r_0$). The p_c is restricted so that it can move within a unit circle around p_0. Therefore, $0 \leq |\vec{m_c}| \geq 1$.

To enable data-sensitive navigation in this setting, we need to know the desired sensitivity for each m_c. The desired sensitivity can be obtained by sampling $d(v_c)$, where v_c is the viewpoint obtained from the rotation specified by m_c. We perform this sampling at a pre-defined resolution on the mouse-press event and store it in a 2D *importance map* z:

$$v_c = t(m_c) \tag{5.2}$$

$$z(m_c) = d(v_c) \tag{5.3}$$

where $t(m_c)$ is a function which calculates the viewpoint v_c after applying the Arcball rotation specified by the input vector m_c to the initial rotation r_0. An example of a complete 2D map z is shown in Figures 5.8c and 5.8d.

During the Arcball interaction, the 3D object is rotated by r_c, which is specified by the user through m_c. To continuously rotate the object from the initial rotation r_0 to r_c, one could continuously change the input vector from m_0 to m_c. This linear change of a 2D input vector maps to a linear change of the current viewpoint. However, the viewpoint change should be slower in areas with features of high importance. We achieve this by applying data-sensitive manipulation to the Arcball rotation.

Since the parameter space of the Arcball rotation is two dimensional, it is necessary to extend data-sensitive manipulation to two dimensions as well. However, simply making the transformation T would lead to discontinuities in the transformed input. Therefore, we reduce the transformation of the Arcball input to a 1D problem. In this way, we can apply the same method for constructing the transformation T as we used with the 1D slider.

Figure 5.9 shows how the 2D Arcball interaction can be reduced to a 1D remapping problem. $\hat{m}_c$ is the normalization of the input vector m_c. It represents the maximum length of the input vector in this direction. $\hat{m}_c$ can be viewed as a 1D slider, where p_c is the current slider position. By sampling the map $z(m_c)$, which is transformed so that p_0 is in its middle, the importance along the slider is obtained. Now it is possible to transform the point p_c through data-sensitive manipulation as if it was a handle of a 1D data-sensitive slider. In this way, the speed of the Arcball rotation is adjusted so that it is proportional to the perceived change of the rendered image by taking the underlying data into account.

5.5.3 Visual Guidance in 3D Object Rotation

We utilize the ability to access the importance values in the visualization mapping to display several forms of visual guidance based on data-sensitive navigation. The visual guidance is displayed in the output image, since this is where the interaction takes place in this scenario. As an example, we apply the guidance to a simple CT scan of a human head, where there is a clear distinction between feature-rich and featureless areas (face and neck versus occipital and parietal bones). In this way, the principles of the data-sensitive navigation are demonstrated in a tangible way.

First, we take the importance of the current viewpoint and use it to modulate a vignetting effect introduced to the output visualization. The vignetting effect gets gradually stronger with decreasing importance. In feature-rich areas with high importance, the vignetting is not visible. As the user rotates the 3D object towards areas with less features, the vignetting effect makes

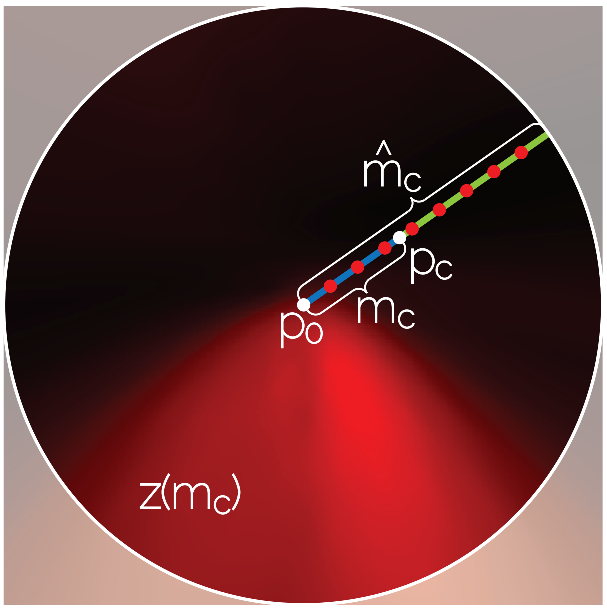


Figure 5.9: Mouse input of the Arcball rotation. m_c (blue) is the input vector whose length is limited to the interval $[0, 1]$. $\hat{m}_c$ is the normalized vector m_c. p_c is the current mouse position. In this way, the 2D interaction is in each step reduced to a 1D problem as illustrated in Figure 5.2. $\hat{m}_c$ represents the whole slider and p_c represents the current slider position. The red dots show where $z(m_c)$ is sampled in order to produce the importance function for 1D data-sensitive navigation.

the borders of the output image increasingly darker. This is an example of a local encoding of the importance.

To encode the importance globally, we display a sphere illustrating the importance around the 3D object. We refer to it as *navigation sphere*. We display the navigation sphere to give the users an idea about the sensitivity of surrounding viewpoints. The navigation sphere rotates in the same way as the examined 3D object. The navigation sphere, as well as the vignetting effect are illustrated in Figure 5.10.

The third form of the visual guidance, displayed during the mouse-move, shows the user where to move the mouse so that areas of higher interest are shown. This visualization overlays the output image with the importance map z generated when the mouse-move started. Additive blending is used so that the visualization is not fully occluded. This creates a red color overlay in those parts of the image space, where the mouse could be dragged to rotate the object towards areas of higher interest. Such a visualization provides suggestions as to how to interact with the Arcball rotation widget. The overlay is shown only while the object is being rotated. This effect is illustrated in Figure 5.11. The red color of the overlay distinguishes it form the achromatic vignetting effect.

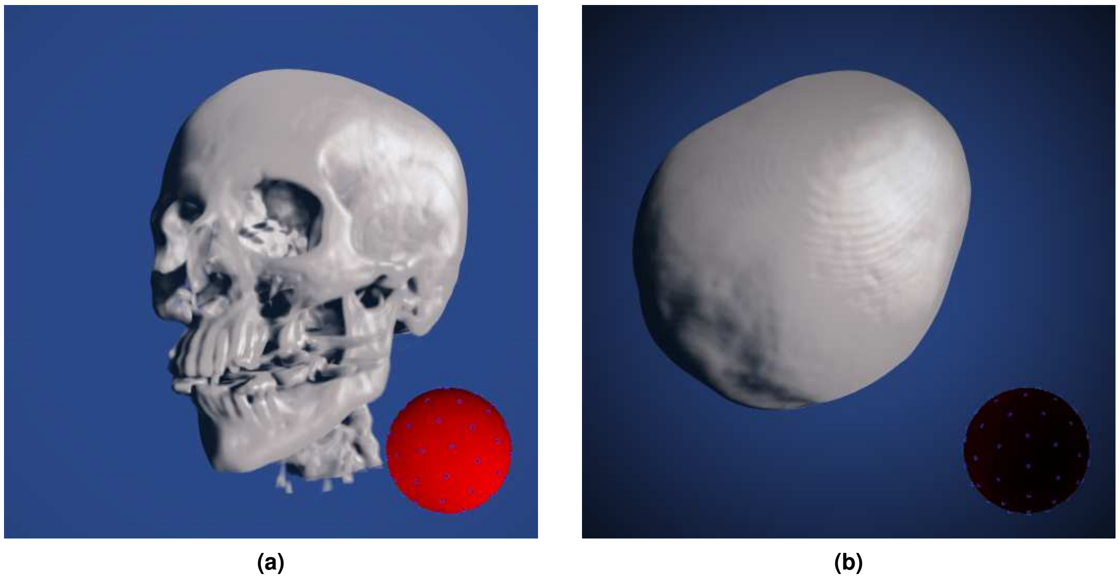


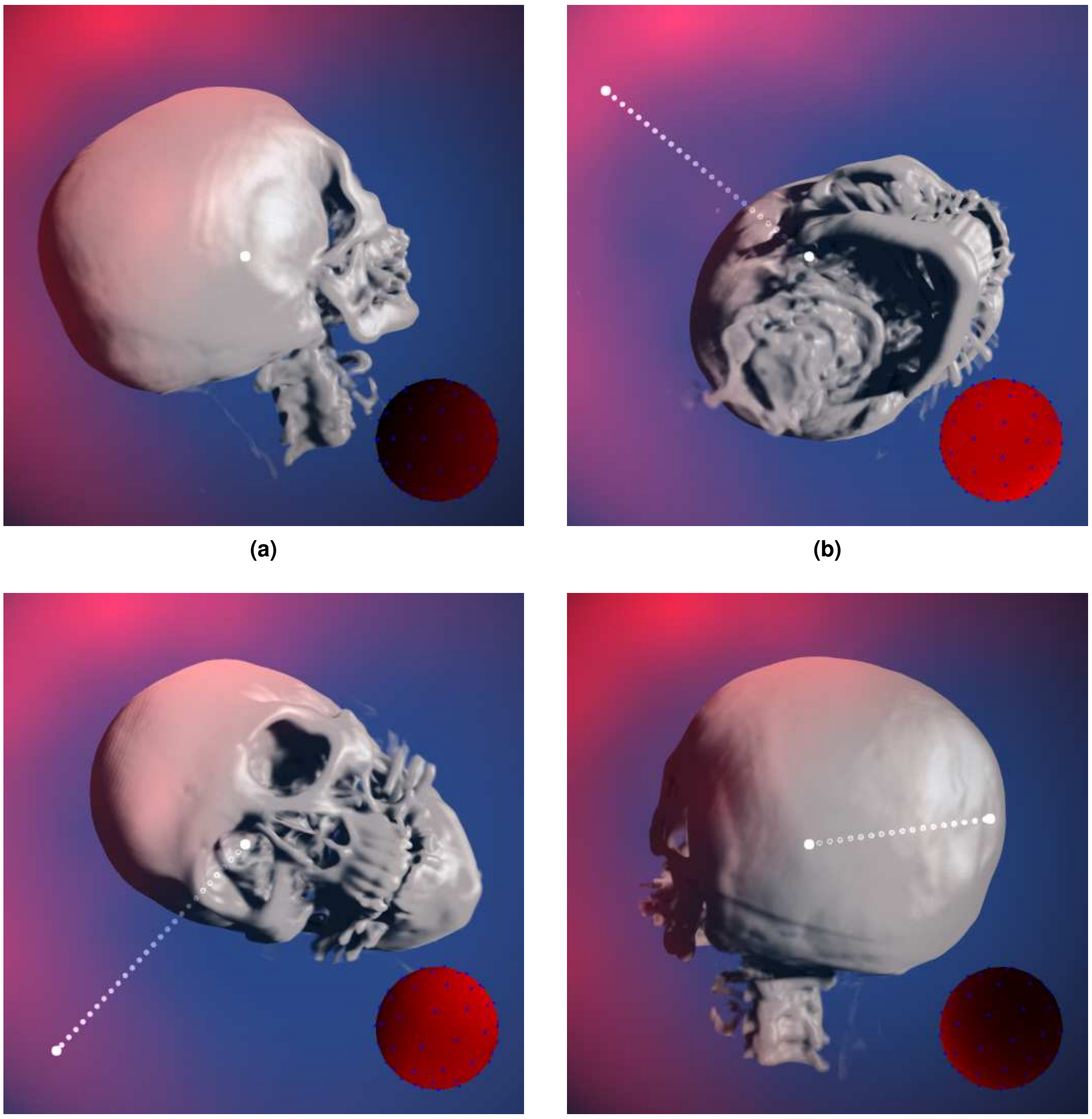
Figure 5.10: Visualization of the importance using the navigation sphere and vignetting. (a) A feature-rich area is shown without vignetting. The front-facing part of the navigation sphere is red, indicating high importance (b) If a featureless area is displayed, vignetting is introduced to indicate the low importance. It is also visible through the black color of the navigation sphere.

5.6 Results and Discussion

The aim of this chapter is to enhance input elements used in medical visualization. Since the elements commonly employed in practice are very diverse, we propose a model of data-sensitive navigation, which is independent from the underlying input elements. The model describes how the underlying data can be utilized to make the changes in the input proportional to the changes of the output, and how these changes can be visually encoded. Additionally, we propose TreeSlider – a novel interaction element, which we employ for the traversal of vascular structures. The data-sensitive navigation model is applied to the TreeSlider as well.

Our model is also independent from the input device. For instance, instead of using mouse movement in two axes to simultaneously set two parameters, a graphics tablet with a tilt-sensitive stylus can be employed. The tilt of the stylus can be taken as the input instead of changing the mouse position. In the 3D object rotation scenario, more tilting will cause the 3D object to rotate faster, while moving the stylus perpendicularly to the tablet will stop the rotation. Since the rotation is controlled by the tilt, position (and possibly pressure) of the stylus can be used for the modification of other parameters.

We apply our model to three distinct input elements in order to demonstrate the applicability of the concepts to different scenarios as well. We had concrete tasks in mind when designing the model, such as volume-slices traversal, or manual viewpoint selection of overview 3D visualizations. However, the model itself is general and abstracts from any particular type of


(a)

(b)

(c)

(d)

Figure 5.11: Visual guidance in 3D object rotation. The white dotted line indicates the input vector for the Arcball rotation. (a) Initial position. The red color indicates where to move the mouse to rotate the object to a more interesting viewpoint. (b) The mouse was moved towards the red area to reveal interesting features around the neck and jaw. (c) The mouse was moved towards another red area to show the face. Since a large portion of the image comprises the featureless forehead, this red area is less pronounced. (d) The mouse moves towards an area of low interest denoted by the lack of red color. Featureless occipital and parietal bones are shown. The low importance is visible from the dark navigation sphere, as well as darker borders caused by the vignetting.

interaction.

We used the TreeSlider with data-sensitive navigation for the traversal of vessel-trees in Computed-Tomography Angiography (CTA) datasets. We examined areas of interest in these datasets, to see how well are they captured by our concepts. Figure 5.12a shows a CTA scan of a leg, where the blood vessel is occluded (the blood-flow is completely blocked). The global encoding of the data-sensitive guidance (Figure 5.12b) shows that on both ends of the occlusion there are areas of high interest (illustrated in red). When traversing this blood vessel, the TreeSlider allocates more screen-space, or portion of its track, to these areas. In this way, they can be more precisely explored with low sensitivity of the TreeSlider. The occlusion itself has low importance (indicated in black), since there is no blood vessel to show. Hence, only a small portion of the track is allocated to this area, making the sensitivity of the TreeSlider high within the occlusion.

In the other datasets, we explored stents, stenoses, and branchings of the blood vessels. All of these features are assigned higher importances, and so their exploration is supported by higher precision of the TreeSlider in these areas. Figure 5.13 shows a TreeSlider moving the cursor along the femoral artery with a stent. As is visible from the visualization, the stent is assigned a higher importance than the rest of the artery. The slider is positioned on the boundary between areas of low and high importance. This is visually encoded on the slider by the guidance arrows (red). The longer arrow, which is pointing to the left side, indicates that the cursor will move faster if the handle is moved to the left. On the other hand, the shorter arrow points to the right side and indicates that the cursor will move slowly in this direction. Hence, these areas of high importance (in this case, the stent) can be explored with a high precision.

Such arrows can be used with 1D input elements. The situation differs in higher-dimensional input elements, such as the presented Arcball widget. Unlike the slider, here the input can change in any direction that is not known beforehand. Therefore, such a guidance arrow cannot be displayed. Instead, a scalar field overlay is used to indicate the effects of potential input changes.

One of the contributions presented in this chapter is the TreeSlider. To gain a better impression of how people interact with the TreeSlider, we performed an experiment with nine participants, all familiar with medical volume data. The test subjects were presented the interface shown in Figure 5.7. They could interact with the TreeSlider, while observing the cursor movement along the vessel-tree in the 3D visualization. After a short explanation of the functionality of the TreeSlider, we asked them to perform the following simple task: A blue dot is displayed at a random position on the vessel-tree; use the TreeSlider to move the cursor along the vessel-tree to the position of the blue dot. After the position of the dot is reached, the dot moves to a new random position and the task is repeated. We asked the subjects to always move the cursor to the position of the dot in the most efficient way, i.e., using the shortest possible path.

For each run, we recorded the starting cursor position, the position of the dot, and the path along which the cursor was moved. Afterwards, we used Dijkstra algorithm to find the shortest possible path between the starting position and the position of the dot. For each run, we calculate the optimality of the recorded cursor-path, which is defined as the percentage of the cursor-path that overlaps with the shortest possible path calculated by the Dijkstra algorithm. An optimality

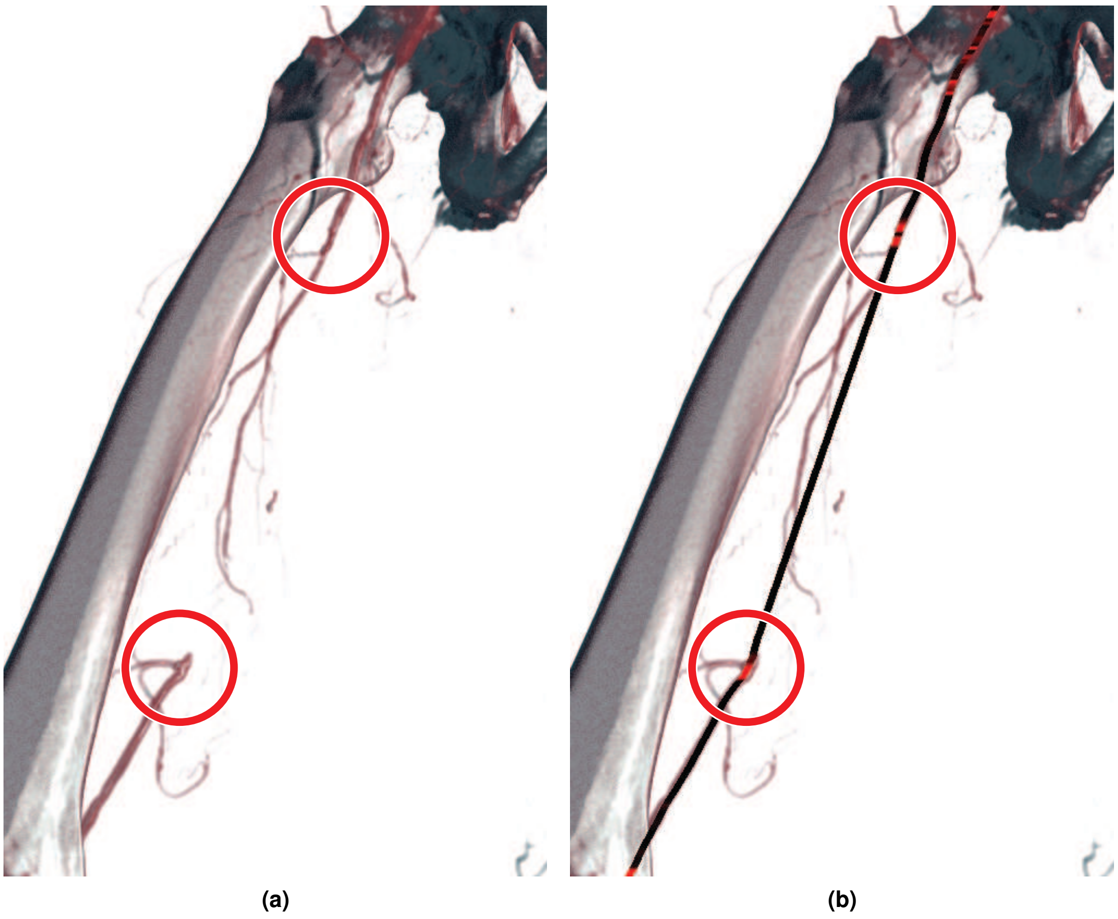


Figure 5.12: (a) Blood vessel with an occlusion. Red circles mark where the occlusion begins and where it ends. (b) The vessel-tree is overlaid on the image showing the calculated importance values. Red color indicates high importance, black color indicates low importance. Both the beginning and the ending of the occlusion were considered to be areas of high importance.

of 100% means that the participant moved the cursor from the initial position to the position of the dot along the shortest possible path. On a single-value slider, where there is only a single possible path between two values, it can be assumed that 100% optimality would always be achieved. In this experiment, we were interested to see how much lower would the optimality be for the TreeSlider.

We gave this task to nine subjects, who performed 232 iterations in total. The average optimality was 95%, which means the average cursor-path was 5% longer than necessary. This overhead was mainly due to movements of the cursor around forks in certain cases. Most of the runs scored 100%. These cases were those where the forks either did not have to be used, or where they could be used without any overhead, such as moving the cursor forward and seamlessly switching to a different branch of the vessel-tree.

Additionally, we gathered user feedback from the test subjects. In general, they liked the concept of the TreeSlider. Some of them reported that it was enjoyable to interact with it. Some concerns were raised regarding the mapping between a fork of the TreeSlider and a branching in the traversed tree. From the visual representation of the fork, it might not be immediately clear which branch maps to which line within the fork and, hence, it has to be determined by trial and error. This is not a significant issue in traversing vessel-trees, since the forks typically only consist of two to three branches. However, it is a point for future work to make this mapping clear from the visual representation, so that the TreeSlider can be efficiently used to traverse trees with nodes of higher degrees.

We observed that a very short accommodation period is needed before using the TreeSlider in an efficient way. The subjects were able to start performing the given task without any initial training. They did not report any confusion caused by the movements of the branching indicators after switching paths within the vessel-tree, as these are smooth and do not cause discontinuities in the movement of the cursor.

As we learned from interviews with the subjects, some of them realized that most of the time, it is not necessary to look at the TreeSlider during the interaction. This is due to the fact that it uses relative changes of the mouse input instead of absolute positions. Indeed, the TreeSlider only acts as a supporting tool and it does not imply significant changes to the workflow of domain experts. The locally encoded guidance which augments the TreeSlider is mostly useful when users wish to continue the interaction after it was interrupted.

The main limitation of the TreeSlider as a general-purpose input element is its inability to handle tree nodes of high degrees. Without making the screen-space of the TreeSlider wider, the too many branches originating in a single fork node might have to be displayed too close together in the fork. This might prevent users from being able to browse through individual branches by vertical mouse movement. This limitation could be overcome, for instance, by using the mouse wheel to switch between branches instead of the vertical movement. When used with a touch-screen, multi-touch gestures could be employed as input for switching the branches. However, this limitation does not apply in various scenarios where degrees of the nodes are relatively low, such as in the traversal of large vessels in the peripheral vascular system. As we have shown, the TreeSlider can be efficiently used to browse vessel trees, which is an important task required in various medical-visualization scenarios.

Besides trees, the TreeSlider can be used to browse arbitrary directed acyclic graphs. In our implementation, the forks can work simultaneously in both directions. Additionally, there is no assumption made that there is only a single root node in the browsed graph. Therefore, all parts of a directed acyclic graph can be reached with the TreeSlider. We tested this functionality on several synthetic directed acyclic graphs.

5.7 Conclusions

In this chapter, we address the problem of interaction for the visualization of medical data. To tackle the challenge of an efficient and intuitive interaction, we propose a general model which can be used to improve navigation in visual-computing applications. The model consists of

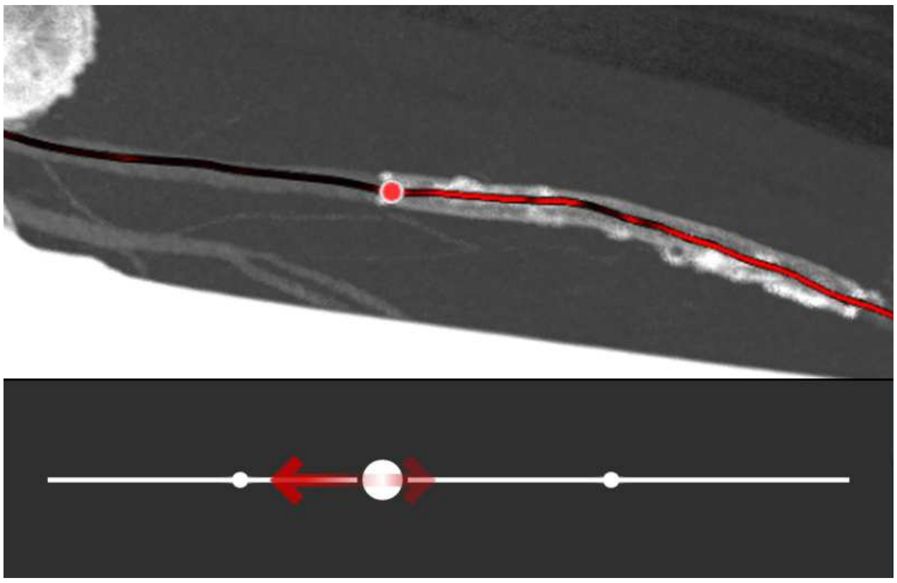


Figure 5.13: TreeSlider positioned to the border of a stent.

data-sensitive manipulation, a way of making changes in the user input and output proportional, and data-sensitive guidance, which displays the remapping information from the manipulation change to steer the user interaction.

We demonstrate with several examples that the proposed model can be applied to 1D and 2D input elements such as sliders, or 3D object rotation widgets. Additionally, we propose a TreeSlider, which is able to traverse trees, or directed acyclic graphs in general. By splitting the tree into segments and treating each of them separately, we are able to apply data-sensitive navigation to the TreeSlider. The resolution of the TreeSlider is dynamically increased in areas of high interest, allowing the users to perform a more precise interaction in these areas. Data-sensitive guidance is displayed on the TreeSlider to steer the users in the examination of the underlying data.

Our proposed model has potential to be used in various application scenarios. The advantage over previous work is that it can be applied to interaction elements where more than one parameter are set at the same time, such as the Arcball rotation widget presented in Section 5.5. Another example is windowing, a commonly used tool in medical visualization. Here, the movement of the mouse is used to change the contrast and brightness of the displayed image. Each of the movement axes maps to one parameter. The same approach as described in Section 5.5 could be applied here as well.

CHAPTER 6 Summary

In this thesis, we present novel methods for the integrated exploration of parameter spaces and the management of complex parameter settings. In addition, we propose a model for data-sensitive interaction for efficient parameter adjustment and a novel input element for interaction with tree structures. Our methods are primarily targeted towards visual-computing algorithms, but the employed concepts are also potentially useful outside of this application area. We successfully applied the proposed methods to various algorithms with high-dimensional parameter spaces, such as multi-modal volume visualization.

Parameter Exploration Language. We propose a markup language and an underlying method for gaining insight into inner the working of visual-computing algorithms, such as volume rendering or image processing. We achieve this by exploring so-called shader spaces of these algorithms. A shader space is a combination of an algorithm's parameter space and all possible visualization mappings implemented as a GPU shader program. This approach allows us to analyse existing algorithms and their possible changes, which might reveal hidden aspects of the visualization mapping. For instance, the exploration of the shader space can lead to an understanding of how individual parts of a complex visualization mapping contribute to the final image. In general, existing approaches to parameter-space analysis and visualization assume that the analyzed algorithm is static. Our markup language allows developers to specify a dynamic behaviour based on a semantic classification of the algorithm's output. This way it is possible with our method to see whether the quality of the output can be increased by modifying parameter settings, as well as the algorithm itself.

Contextual Snapshots. The second method presented in this thesis allows visualization experts to enhance their systems with a functionality to manage multiple parameter settings selected by the users during visualization sessions. Each parameter setting is linked with one or multiple parts of the visualization image, or image-space selections. The selections can be of arbitrary shapes. They are defined by custom degree-of-interest functions, possibly allowing smooth brushing. The parameter settings provide context for each selection. This is due to the fact that the selection might not be relevant after the parameters, and subsequently the visualization image, change. Each selection can be linked with additional views, displaying or processing the selected data subsets. In this way, the visualization systems can be easily enhanced with a brushing and linking functionality, contextually-bound data annotations, or arbitrary image-space selections used for various purposes. All of this is achieved by a common framework implemented as the Contextual Snapshot Library.

Multiplayer Games Summarization. In the third part of the thesis, we present a method for the automated production of summary videos of multiplayer video games. This is in fact the task of a seamless integration of curves in the parameter space of a video game generated by different users who participate in the game. Such an integration is subject to certain constraints. Namely, the result of such an integration must group related game events together and place smooth transitions between them, so that a visual story is formed. To comply with this constraint, we apply various cinematographic effects to integrate the views of the players participating in the game into a seamless summary video. The story represented by the summary is extracted by analyzing the occurred events and their causal dependencies, and arranging them into an oriented graph. The cinematographic effects are implemented as camera operators applied to individual views as the graph is traversed. In this way, the visual story is created. The structure of the story is constructed by the graph-traversing algorithm designed for this specific purpose.

Data-Sensitive Navigation. In the last part of the thesis, we present a universal model for describing how user interactions can be made proportional to changes of the output, and how the expected output changes can be visualized to steer the parameter adjustment. The model, which we refer to as data-sensitive navigation, can be applied to a large variety of input elements. We demonstrated this by applying it to tree different input elements - a single-value slider, an Arcball rotation widget, and a novel input element for traversing trees, which we call TreeSlider. We used the TreeSlider and data-sensitive navigation to aid tasks in medical visualization. However, both the model and the TreeSlider are described in such a way that they can be utilized in other application areas as well. Medical visualization only serves as an example scenario to demonstrate all the necessary concepts. An interesting direction for future research is to integrate the TreeSlider with data-sensitive navigation in the multiplayer-games summarization-method described in Chapter 4. Even though the approach is currently fully automatic, an option of browsing through the generated story might be a valuable addition. Movie players usually employ such a browsing mechanism as a slider representing the length of the movie. Since we generate stories with an acyclic-directed-graph structure, TreeSlider could be employed to traverse the story. Data-sensitive navigation would ensure that the most relevant parts of the story would be assigned larger portions of the length of the TreeSlider than uninteresting parts.

We have implemented our methods as reusable systems, so that they can be applied in different real-world scenarios. Parameter Exploration Language and Contextual Snapshots are implemented in stand-alone, reusable C++ libraries, ready to be included in existing software systems. The implementation of ManyCams requires the game logs in a specific format, and a functionality on the side of the game allowing to render gameplay scenes at specific times from arbitrary viewpoints. Other than this additional functionality that has to be implemented in the game, the ManyCams is a stand-alone application independent from the actual game engine. The TreeSlider is implemented as a stand-alone class and it is ready to be used in user interfaces, potentially utilizing data-sensitive navigation.

In this thesis, we address several problems of parameter-space analysis, exploration, and management, which are not solved by existing approaches. We propose various applications for each of the methods, and evaluate them in different scenarios. The proposed concepts and methods could lead to further research. Some of the possible research directions emerging from the research presented in this thesis are described in the respective chapters.

Bibliography

[1] Contextual snapshot library. `http://cg.tuwien.ac.at/downloads/csl/`. Accessed: April 2014.

[2] Maneesh Agrawala, Denis Zorin, and Tamara Munzner. Artistic multiprojection rendering. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 125–136, London, UK, 2000. Springer-Verlag.

[3] Artem Amirkhanov, Christoph Heinzl, Michael Reiter, and Eduard Gröller. Visual optimality and stability analysis of 3dct scan positions. *IEEE Transactions on Visualization and Computer Graphics*, pages 1477–1487, October 2010.

[4] Thomas Auzinger, Gabriel Mistelbauer, Ivan Baclija, Rüdiger Schernthaner, Arnold Köchl, Michael Wimmer, M. Eduard Gröller, and Stefan Bruckner. Vessel visualization using curved surface reformation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2858–2867, 2013.

[5] Jean-Paul Balabanian. *Multi-Aspect Visualization: Going from Linked Views to Integrated Views*. PhD thesis, Dept. of Informatics, Univ. of Bergen, Norway, October 2010.

[6] Jean-Paul Balabanian and M. Eduard Gröller. A. In *Scientific Visualization (Proc. Dagstuhl Seminar)*, Dagstuhl Seminar Proceedings, 2010.

[7] Jean-Paul Balabanian, Ivan Viola, Torsten Möller, and Eduard Gröller. Temporal styles for time-varying volume data. In Stephan Gumhold, Jana Kosecka, and Oliver Staadt, editors, *Proceedings of 3DPVT'08 - the Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, pages 81–89, June 2008.

[8] Connelly Barnes, Dan B. Goldman, Eli Shechtman, and Adam Finkelstein. Video tapestries with continuous temporal zoom. *ACM Trans. Graph.*, 29(4):89:1–89:9, July 2010.

[9] Louis Bavoil, Steven P. Callahan, Carlos Eduardo Scheidegger, Huy T. Vo, Patricia Crossno, Cláudio T. Silva, and Juliana Freire. Vistrails: Enabling interactive multiple-view visualizations. In Cláudio T. Silva, Eduard Gröller, and Holly Rushmeier, editors, *IEEE Visualization*, pages 135–142. IEEE Computer Society, October 2005.

[10] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[11] Leo Berkeley. Situating machinima in the new mediascape. *Australian Journal of Emerging Technologies and Society*, 4(2):65–80, 2006.

[12] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 73–80, New York, NY, USA, 1993. ACM.

[13] Renaud Blanch, Yves Guiard, and Michel Beaudouin-Lafon. Semantic pointing: Improving target acquisition with control-display ratio adaptation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 519–526, 2004.

[14] U.D. Bordoloi and Han-Wei Shen. View selection for volume rendering. In *Proceedings of IEEE Visualization*, pages 487–494, 2005.

[15] Michelle A. Borkin, Krzysztof Z. Gajos, Amanda Peters, Dimitrios Mitsouras, Simone Melchionna, Frank J. Rybicki, Charles L. Feldman, and Hanspeter Pfister. Evaluation of artery visualizations for heart disease diagnosis. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2479–2488, 2011.

[16] Michelle A. Borkin, Chelsea S. Yeh, Madelaine Boyd, Peter Macko, Krzysztof Z. Gajos, Margo Seltzer, and Hanspeter Pfister. Evaluation of filesystem provenance visualization tools. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2476–2485, 2013.

[17] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[18] Stefan Bruckner and Eduard Gröller. Volumeshop: An interactive system for direct volume illustration. In Cláudio T. Silva, Eduard Gröller, and Holly Rushmeier, editors, *IEEE Visualization*, pages 671–678. IEEE Computer Society, October 2005.

[19] Stefan Bruckner and Torsten Möller. Isosurface similarity maps. *Computer Graphics Forum*, 29(3):773–782, June 2010.

[20] Stefan Bruckner and Torsten Möller. Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1467–1475, October 2010.

[21] Bytonic Software. Jake2. `http://bytonic.de/html/jake2.html/`, 2006. Accessed: 2015-3-17.

[22] Lawrence Cayton. A nearest neighbor data structure for graphics hardware. In Rajesh Bordawekar and Christian A. Lang, editors, *Proceedings of the First International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, pages 9–14, 2010.

[23] Olivier Chapuis, Jean-Baptiste Labrune, and Emmanuel Pietriga. Dynaspot: Speed-dependent area cursor. In *CHI '09: SIGCHI conference on Human Factors in computing systems*, CHI '09, pages 1391–1400, 2009.

[24] Peng Chen, Beth Plale, You-Wei Cheah, Devarshi Ghoshal, Scott Jensen, and Yuan Luo. Visualization of network data provenance. *20th Annual International Conference on High Performance Computing*, pages 1–9, December 2012.

[25] Yun-Gyung Cheong, Arnav Jhala, Byung-Chull Bae, and R. Michael Young. Automatically generating summary visualizations from game logs. In Christian Darken and Michael Mateas, editors, *AIIDE*, pages 167–172. The AAAI Press, 2008.

[26] Marc Christie, Patrick Olivier, and Jean-Marie Normand. Camera control in computer graphics. *Computer Graphics Forum*, 27(8):2197–2218, September 2008.

[27] Ben Clifford, Ian Foster, Jens-S. Vöckler, Michael Wilde, and Yong Zhao. Tracking provenance in a virtual data grid. *Concurr. Comput.: Pract. Exper.*, 20(5):565–575, April 2008.

[28] Carlos D. Correa and Kwan-Liu Ma. Dynamic video narratives. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(3):88:1–88:9, 2010.

[29] Carlos D. Correa and Kwan-Liu Ma. Visibility histograms and visibility-driven transfer functions. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):192–204, February 2011.

[30] P. Crossno and E. Angel. Visual debugging of visualization software: a case study for particle systems. In *Visualization '99. Proceedings*, pages 417–554, October 1999.

[31] Francisco de Moura Pinto and Carla M. D. S. Freitas. Design of multi-dimensional transfer functions using dimensional reduction. In *Eurographics - IEEE VGTC Symposium on Visualization*, pages 131–138, 2007.

[32] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[33] Helmut Doleisch, Martin Gasser, and Helwig Hauser. Interactive feature specification for focus+context visualization of complex simulation data. In *Proceedings of the symposium on Data visualisation 2003*, VISSYM '03, pages 239–248, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[34] Helmut Doleisch and Helwig Hauser. Smooth brushing for focus+context visualization of simulation data in 3d. In *Journal of WSCG*, volume 10, pages 147–154, 2012.

[35] Nathaniel Duca, Krzysztof Niski, Jonathan Bilodeau, Matthew Bolitho, Yuan Chen, and Jonathan Cohen. A relational debugging engine for the graphics pipeline. *ACM Transactions on Graphics*, 24(3):453–463, 2005.

[36] Tommy Ellkvist, David Koop, Juliana Freire, Cláudio Silva, and Lena Strömbäck. Using mediation to achieve provenance interoperability. In *Proceedings of the IEEE International Workshop on Scientific Workflows, 2009*, pages 291–298, Washington, DC, USA, 2009. IEEE Computer Society.

[37] Niklas Elmqvist and Jean-Daniel Fekete. Semantic pointing for object picking in complex 3d environments. In *Proceedings of Graphics Interface*, pages 243–250, 2008.

[38] Niklas Elmqvist, Yann Riche, Nathalie Henry-Riche, and Jean-Daniel Fekete. Mélange: Space folding for visual exploration. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):468–483, 2010.

[39] Martin Ester, Hans P. Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996. AAAI Press.

[40] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[41] George W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '86, pages 16–23, New York, NY, USA, 1986. ACM.

[42] Marius Gavrilescu, Muhammad Muddassir Malik, and Eduard Gröller. Custom interface elements for improved parameter control in volume rendering. In *14th Int. Conf. on System Theory and Control 2010*, pages 219–224, October 2010.

[43] Moritz Gerl, Peter Rautek, Tobias Isenberg, and Eduard Gröller. Semantics by analogy for illustrative volume visualization. *Computers & Graphics*, 36(3):201–213, 2012.

[44] Nahum Gershon and Ward Page. What storytelling can do for information visualization. *Commun. ACM*, 44(8):31–37, August 2001.

[45] M. Greenspan, G. Godin, and J. Talbot. Acceleration of binning nearest neighbour methods. In *Proceedings of the Vision Interface Conference*, pages 337–344, 2000.

[46] Dennis P. Groth and Kristy Streefkerk. Provenance and annotation for visual exploration systems. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1500–1510, November 2006.

[47] Paul Groth, Simon Miles, and Luc Moreau. Preserv: Provenance recording for services. In *UK e-Science All Hands Meeting 2005*. EPSRC, 2005. Event Dates: September 2005.

[48] Hanqi Guo, Ningyu Mao, and Xiaoru Yuan. Wysiwyg (what you see is what you get) volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17:2106–2114, 2011.

[49] Nick Halper and Maic Masuch. Action summary for computer games – extracting and capturing action for spectator modes and summaries. In Loo Wai Sing, Wan Hak Man, and Wong Wai, editors, *Proceedings of 2nd International Conference on Application and Development of Computer Games*, pages 124–132, Hong Kong, China, January 2003. Division of Computer Studies of the City University of Hong Kong.

[50] Nicolas Halper, Ralf Helbing, and Thomas Strothotte. A camera engine for computer games: Managing the trade-off between constraint satisfaction and frame coherence. *Computer Graphics Forum*, 20(3):174–183, 2001.

[51] Helwig Hauser. Generalizing focus+context visualization. In Georges-Pierre Bonneau, Thomas Ertl, and Gregory M. Nielson, editors, *Scientific Visualization: The Visual Extraction of Knowledge from Data*, Mathematics and Visualization, pages 305–327. Springer Berlin Heidelberg, 2006.

[52] Li-Wei He, Michael F. Cohen, and David H. Salesin. The virtual cinematographer: a paradigm for automatic real-time camera control and directing. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 217–224, New York, NY, USA, 1996. ACM.

[53] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1189–1196, November 2008.

[54] Alexander Hornung, Gerhard Lakemeyer, and Georg Trogemann. An autonomous real-time camera agent for interactive narratives and games. In Thomas Rist, Ruth S. Aylett, Daniel Ballin, and Jeff Rickel, editors, *Intelligent Virtual Agents*, volume 2792 of *Lecture Notes in Computer Science*, pages 236–243. Springer Berlin Heidelberg, 2003.

[55] Qiming Hou, Kun Zhou, and Baining Guo. Debugging gpu stream programs through automatic dataflow recording and visualization. *ACM Trans. Graph.*, 28(5):153:1–153:11, December 2009.

[56] Wei-Hsien Hsu, Kwan-Liu Ma, and Carlos Correa. A rendering framework for multi-scale views of 3d models. *ACM Transactions on Graphics*, 30(6):131:1–131:10, December 2011.

[57] Jessica Hullman, Steven Drucker, Nathalie Henry Riche, Bongshin Lee, Danyel Fisher, and Eytan Adar. A deeper understanding of sequence in narrative visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2406–2415, 2013.

[58] T.J. Jankun-Kelly and Kwan-Liu Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, 2001.

[59] T.J. Jankun-Kelly, Kwan Liu Ma, and Michael Gertz. A model for the visualization exploration process. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 323–330. IEEE Computer Society, 2002.

[60] Guangfeng Ji and Han-Wei Shen. Dynamic view selection for time-varying volumes. *IEEE Transaction on Visualization and Computer Graphics*, 12(5):1109–1116, 2006.

[61] Armin Kanitsar, Dominik Fleischmann, Rainer Wegenkittl, Petr Felkel, and M. Eduard Gröller. CPR – Curved Planar Reformation. In *Proceedings of IEEE Visualization*, pages 37–44, 2002.

[62] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of the Conference on Visualization '01*, VIS '01, pages 255–262, Washington, DC, USA, 2001. IEEE Computer Society.

[63] Peter Kohlmann, Stefan Bruckner, Armin Kanitsar, and M. Eduard Gröller. Livesync: Deformed viewing spheres for knowledge-based navigation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1544–1551, 2007.

[64] Peter Kohlmann, Stefan Bruckner, Armin Kanitsar, and M. Eduard Gröller. Livesync++: Enhancements of an interaction metaphor. In *Proceedings of Graphics Interface*, pages 81–88, 2008.

[65] Andreas König and Eduard Gröller. Mastering transfer function specification by using volumepro technology. In *Proceedings of the 17th Spring Conference on Computer Graphics (SCCG'01)*, pages 279–286, 2001.

[66] Robert Kosara and Jock Mackinlay. Storytelling: The next step for visualization. *Computer*, 46(5):44–50, 2013.

[67] Robert Kosara, Gerald N. Sahling, and Helwig Hauser. Linking scientific and information visualization with interactive 3d scatterplots. In *Proceedings of the 12th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 133–140, 2004.

[68] Matthias Kreuseler, Thomas Nocke, and Heidrun Schumann. A history mechanism for visual data mining. In *Proceedings of the IEEE Symposium on Information Visualization*, InfoVis '04, pages 49–56, Washington, DC, USA, 2004. IEEE Computer Society.

[69] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[70] Andreas Lie. Advanced storytelling for volume visualization. Msc thesis, Visualization Group, Department of Informatics, University of Bergen, November 2009.

[71] Norbert Lindow, Daniel Baum, and Hans-Christian Hege. Perceptually linear parameter variations. *Computer Graphics Forum*, 31(2pt4):535–544, 2012.

[72] Helwig Löffelmann and Eduard Gröller. Ray tracing with extended cameras. *Journal of Visualization and Computer Animation*, 7(4):211–227, 1996.

[73] Bruce Lucas, Gregory D. Abram, Nancy S. Collins, David A. Epstein, Donna L. Gresh, and Kevin P. McAuliffe. An architecture for a scientific visualization system. In *Proceedings of the 3rd Conference on Visualization '92*, VIS '92, pages 107–114, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[74] Kwan-Liu Ma. Image graphs–a novel approach to visual data exploration. In *Proceedings of the Conference on Visualization '99: Celebrating Ten Years*, pages 81–88. IEEE Computer Society, October 1999.

[75] Kwan-Liu Ma, Isaac Liao, Jennifer Frazier, Helwig Hauser, and Helen-Nicole Kostis. Scientific storytelling using visualization. *IEEE Computer Graphics and Applications*, 32(1):12 –19, January 2012.

[76] Peter Macko and Margo Seltzer. Provenance map orbiter: Interactive exploration of large provenance graphs. In *Proceedings of the 3rd Workshop on the Theory and Practice of Provenance (TaPP), USENIX Association*, 2011.

[77] Anne Mahoney. Studying the word study tool. *New England Classical Journal*, 28(3):181–183, 2001.

[78] Joe Marks, Brad Andalman, Paul A. Beardsley, William Freeman, Sarah Gibson, Jessica Hodgins, Tom Kang, Brian Mirtich, Hanspeter Pfister, Wheeler Ruml, Kathy Ryall, Joshua Seims, and Stuart Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 389–400, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[79] Patrick S. McCormick, Jeff Inman, James P. Ahrens, Charles Hansen, and Greg Roth. Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 171–178. IEEE Computer Society, 2004.

[80] Bryan McDonnel and Niklas Elmqvist. Towards utilizing gpus in information visualization: A model and implementation of image-space operations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1105–1112, November 2009.

[81] Jennis Meyer-Spradow, Timo Ropinski, Jörg Mensmann, and Klaus Hinrichs. Interactive design and debugging of gpu-based volume visualizations. In *Computer Graphics Theory and Applications*, pages 239–245, 2010.

[82] Gabriel Mistelbauer. *Smart Interactive Vessel Visualization in Radiology*. Phd thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, November 2013.

[83] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Beth Plale, Yogesh Simmhan, Eric Stephan, and Jan Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743 – 756, 2011.

[84] S. Oeltze and B. Preim. Visualization of vasculature with convolution surfaces: method, validation and evaluation. *IEEE Transactions on Medical Imaging*, 24(4):540–548, 2005.

[85] Thomas Oskam, Robert W. Sumner, Nils Thuerey, and Markus Gross. Visibility transition planning for dynamic camera control. In *Proceedings of the Third international conference on Motion in games*, MIG'10, pages 325–325, Berlin, Heidelberg, 2010. Springer-Verlag.

[86] James Patten and Kwan-Liu Ma. A graph based interface for representing volume visualization results. In Wayne A. Davis, Kellogg S. Booth, and Alain Fournier, editors, *Graphics Interface*, pages 117–124. Canadian Human-Computer Communications Society, 1998.

[87] Hanspeter Pfister, Bill Lorensen, Chandrajit Bajaj, Gordon Kindlmann, Will Schroeder, Lisa Sobierajski Avila, Ken Martin, Raghu Machiraju, and Jinho Lee. The transfer function bake-off. *IEEE Computer Graphics and Applications*, 21(3):16–22, May 2001.

[88] Harald Piringer, Robert Kosara, and Helwig Hauser. Interactive focus+context visualization with linked 2d/3d scatterplots. In *Proceedings of the Second International Conference on Coordinated & Multiple Views in Exploratory Visualization*, CMV '04, pages 49–60, Washington, DC, USA, 2004. IEEE Computer Society.

[89] Horst R. Portugaller, Helmut Schoellnast, Klaus A. Hausegger, Kurt Tiesenhausen, Wilfried Amann, and Andrea Berghold. Multislice spiral CT angiography in peripheral arterial occlusive disease: a valuable tool in detecting significant arterial lumen narrowing? *European Radiology*, 14(9):1681–1687, 2004.

[90] Faisal Qureshi and Demetri Terzopoulos. Virtual vision and smart camera networks. *Working Notes of the International Workshop on Distributed Smart Cameras (DSC 2006)*, pages 62–66, 2006.

[91] Peter Rautek, Stefan Bruckner, and Eduard Gröller. Semantic layers for illustrative volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1336–1343, November 2007.

[92] Peter Rautek, Stefan Bruckner, and Eduard Gröller. Interaction-dependent semantics for illustrative volume rendering. *Computer Graphics Forum*, 27(3):847–854, May 2008.

[93] Atul Rungta, Brian Summa, Dogan Demir, Peer-Timo Bremer, and Valerio Pascucci. Manyvis: Multiple applications in an integrated visualization environment. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2878–2885, 2013.

[94] Emanuele Santos, Lauro Lins, James Ahrens, Juliana Freire, and Claudio Silva. Vismashup: Streamlining the creation of custom visualization applications. *IEEE Transactions on Visualization and Computer Graphics*, 15:1539–1546, 2009.

[95] Michael Sedlmair, Christoph Heinzl, Stefan Bruckner, Harald Piringer, and Torsten Möller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2161—2170, December 2014.

[96] Edward Segel and Jeffrey Heer. Narrative visualization: Telling stories with data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1139–1148, November 2010.

[97] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 245–254, New York, NY, USA, 1985. ACM.

[98] Ken Shoemake. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics Interface*, pages 151–156, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[99] Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[100] Yogesh Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru. Performance evaluation of the karma provenance framework for scientific workflows. In Luc Moreau and Ian Foster, editors, *International Provenance and Annotation Workshop (IPAW'06)*, volume 4145, pages 222–236, Chicago, IL USA, June 2006. Springer-Verlag.

[101] Matus Straka, Arnold Köchl, Michal Cervenansky, Milos Sramek, Dominik Fleischmann, Alexandra La Cruz, and Eduard Gröller. The VesselGlyph: Focus & Context Visualization in CT-Angiography. In *Proceedings of IEEE Visualization*, pages 385–392, 2004.

[102] Marc Streit, Hans-Jorg Schulz, Alexander Lex, Dieter Schmalstieg, and Heidrun Schumann. Model-driven design for the visual analysis of heterogeneous data. *IEEE Transactions on Visualization and Computer Graphics*, 18:998–1010, 2012.

[103] Magnus Strengert, Thomas Klein, and Thomas Ertl. A hardware-aware debugger for the opengl shading language. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, GH '07, pages 81–88. Eurographics Association, 2007.

[104] S. Takahashi, I. Fujishiro, Y. Takeshima, and T. Nishita. A feature-driven approach to locating optimal viewpoints for volume visualization. In *Proceedings of IEEE Visualization*, pages 495–502, 2005.

[105] Melanie Tory. *Combining two-dimensional and three-dimensional views for visualization of spatial data.* PhD thesis, Simon Fraser University, Burnaby, BC, Canada, 2004.

[106] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, USA, 1986.

[107] Twitch Interactive, Inc. Twitch. `http://www.twitch.tv/`, 2011. Accessed: 2014-2-21.

[108] Fan-Yin Tzeng, Eric Lum, and Kwan-Liu Ma. An intelligent system approach to higher-dimensional classification of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 11(3):273–284, 2005.

[109] Fan-Yin Tzeng and Kwan-Liu Ma. A cluster-space visual interface for arbitrary dimensional classification of volume data. In *Eurographics - IEEE TCVG Symposium on Visualization, 2004*, pages 17–24, 2004.

[110] Amy Catherine Ulinski, Catherine A. Zanbaka, Zachary Wartell, Paula Goolkasian, and Larry F. Hodges. Two handed selection techniques for volumetric data. In *IEEE Symposium on 3D User Interfaces*, Charlotte, North Carolina, USA, 2007.

[111] Andrea Unger, Philipp Muigg, Helmut Doleisch, and Heidrun Schumann. Visualizing statistical properties of smoothly brushed data subsets. In *Proceedings of the 12th IEEE International Conference on Information Visualisation*, INFOVIS'08, pages 233–239, London, UK, 2008. IEEE Computer Society.

[112] Jarke J. van Wijk and Wim A. A. Nuij. Smooth and efficient zooming and panning. In *Proceedings of the 9th IEEE Conference on Information Visualization*, INFOVIS'03, pages 15–22, Washington, DC, USA, 2003. IEEE Computer Society.

[113] Pere-Pau Vázquez, Miquel Feixas, Mateu Sbert, and Wolfgang Heidrich. Viewpoint selection using viewpoint entropy. In *Proceedings of the Vision Modeling and Visualization Conference*, VMV '01, pages 273–280. Aka GmbH, 2001.

[114] Ivan Viola, Miquel Feixas, Mateu Sbert, and Eduard Gröller. Importance-driven focus of attention. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):933–940, October 2006.

[115] Wenping Wang, Bert Jüttler, Dayue Zheng, and Yang Liu. Computation of rotation minimizing frames. *ACM Trans. Graph.*, 27(1):2:1–2:18, 2008.

[116] Yunhai Wang, Wei Chen, Guihua Shan, Tingxin Dong, and Xuebin Chi. Volume exploration using ellipsoidal gaussian transfer functions. In *Pacific Visualization Symposium (PacificVis)*, pages 25–32. IEEE Computer Society, March 2010.

[117] Jishang Wei, Chaoli Wang, Hongfeng Yu, and Kwan-Liu Ma. A sketch-based interface for classifying and visualizing vector fields. In *Pacific Visualization Symposium (PacificVis)*, pages 129–136. IEEE Computer Society, 2010.

[118] Michael Wohlfart and Helwig Hauser. Story telling for presentation in volume visualization. In *Proceedings of the 9th Joint Eurographics / IEEE VGTC conference on Visualization*, EUROVIS'07, pages 91–98, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[119] Jianhuang Wu, Qingmao Hu, and Xin Ma. Comparative study of surface modeling methods for vascular structures. In *Computerized Medical Imaging and Graphics*, pages 4–14, 2013.

[120] Jianhuang Wu, Renhui Ma, Xin Ma, Fucang Jia, and Qingmao Hu. Curvature-dependent surface visualization of vascular structures. In *Computerized Medical Imaging and Graphics*, pages 651–658, 2010.

[121] Yingcai Wu and Huamin Qu. Interactive transfer function design based on editing direct volume rendered images. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1027 –1040, September 2007.

[122] I-Cheng Yeh, Chao-Hung Lin, Hung-Jen Chien, and Tong-Yee Lee. Efficient camera path planning algorithm for human motion overview. *Computer Animation and Virtual Worlds*, 22(2-3):239–250, April 2011.

[123] Li Yu, Aidong Lu, William Ribarsky, and Wei Chen. Automatic animation for time-varying data visualization. *Computer Graphics Forum*, 29(7):2271–2280, 2010.

[124] Lingyun Yu, Konstantinos Efstathiou, Petra Isenberg, and Tobias Isenberg. Efficient Structure-Aware Selection Techniques for 3D Point Cloud Visualizations with 2DOF Input. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2245–2254, 2012.

[125] Xiaoru Yuan, Nan Zhang, Minh X. Nguyen, and Baoquan Chen. Volume cutout. *The Visual Computer (Special Issue of Pacific Graphics 2005)*, 21(8–10):745–754, 2005.

Curriculum Vitae

NAME:	Peter Mindek
PLACE AND DATE OF BIRTH:	Slovakia, 9 May 1988
ADDRESS:	Priekopnícka 36, 82106 Bratislava, Slovakia
PHONE:	+421 907 065 304
E-MAIL:	mindek@cg.tuwien.ac.at
CITIZENSHIP:	Slovak
LANGUAGES:	English, Slovak

Education

2011 - 2015	Doctoral programme in Engineering Sciences - Computer Sciences Institute of Computer Graphics and Algorithms Vienna University of Technology, Austria *Advisor:* Prof. Dr. Stefan Bruckner
2009 - 2011	Master's degree – Information systems Faculty of Informatics and Information Technologies Slovak University of Technology in Bratislava, Slovakia
2006 - 2009	Bachelor's degree – Informatics Faculty of Informatics and Information Technologies Slovak University of Technology in Bratislava, Slovakia

Publications

P. Mindek, L. Čmolík, I. Viola, M. E. Gröller, S. Bruckner. Automatized Summarization of Multiplayer Games. In *Spring Conference on Computer Graphics*, SCCG'15. To appear.

P. Mindek, M. E. Gröller, and S. Bruckner. Managing Spatial Selections with Contextual Snapshots. *Computer Graphics Forum 33, 8 (2014)*, pp. 132—144. DOI: 10.1111/cgf.12406.

P. Mindek, S. Bruckner, M. E. Gröller. Contextual Snapshots: Enriched Visualization with Interactive Spatial Annotations. In *Spring Conference on Computer Graphics*, SCCG'13, pp. 49—56, New York, NY, USA, 2013. ACM. SCCG Best Paper Award.

P. Mindek, S. Bruckner, P. Rautek, M. E. Gröller. Visual Parameter Exploration in GPU Shader Space. *Journal of WSCG 21, 3 (2013)*, pp. 225—234. ISSN 1213-6972.

A. Karimov, G. Mistelbauer, J. Schmidt, P. Mindek, E. Schmidt, T. Sharipov, S. Bruckner, M. E. Gröller. ViviSection: Skeleton-based Volume Editing. *Computer Graphics Forum 32, 3 (2013)*, pp. 461—470.

P. Mindek, P. Kapec. Graph visualization using the metaphor of biological neural nets. In *Proceedings of the 27th Spring Conference on Computer Graphics (SCCG'11), Stephen Spencer (Ed.), 2011.* ACM, New York, NY, USA, pp. 141—148.

P. Mindek. Visual Artefacts Removal in Volumetric Visualization. *Information Sciences and Technologies Bulletin of the ACM Slovakia 3, 2 (2011)*, pp. 96—99. ISSN 1338-1237.

P. Mindek. Maximum Intensity Projection Weighted by Statistical Cues. In *Proceedings of CESCG 2011: The 15th Central European Seminar on Computer Graphics. Viničné, Slovakia, May 2-4, 2011.* Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2011, pp. 149—155. ISBN 978-3-9502533-3-7.

P. Mindek. New Method for Checking Plagiarism. In BIELIKOVÁ, M. *Student Research Conference 2010. Vol.1. 6th Student Research Conference in Informatics and Information Technologies Bratislava, April 21, 2010 : Proceedings in Informatics and Information Technologies.* STU in Bratislava FIIT, 2010, pp. 91—97. ISBN 978-80-227-3266-6.

P. Mindek, M. Freml, M. Noskovič, M. Mego, M. Sabo, D. Chalupa. Copypaste: Plagiarism Detection Framework. In BIELIKOVÁ, M. *Student Research Conference 2010. Vol. 2. 6th Student Research Conference in Informatics and Information Technologies Bratislava, April 21, 2010 : Proceedings in Informatics and Information Technologies.* STU in Bratislava FIIT, 2010, pp. 549—550. ISBN 978-80-227-3267-3.

P. Mindek. Room Information System. In BIELIKOVÁ, M. *Student Research Conference 2009. 5th Student Research Conference in Informatics and Information Technologies Bratislava, April 29, 2009 : Proceedings in Informatics and Information Technologies.* STU in Bratislava FIIT, 2009, pp. 469—470. ISBN 978-80-227-3052-5.

Master thesis: Data Visualization Using Metaphors Inspired by Visualization of Biological Processes. *Advisor:* Dr. Peter Kapec

Bachelor thesis: Interactive Browsing of Multimedia Content in 3D Space. *Advisor:* Dr. Peter Kapec

Awards

The Best Paper Award, *Spring Conference on Computer Graphics 2013*

ACM Slovakia Chapter Prize, *Student Research Conference in Informatics and Information Technologies 2011*

ACM Slovakia Chapter Prize, *Student Research Conference in Informatics and Information Technologies 2010*

Reviewing history

IEEE VIS (2015, 2014), EuroVis (2015, 2014, 2012), Pacific Graphics (2014), PacificVis (2013, 2012), CGI (2015, 2013), VCBM (2014), 3DVis@IEEEVIS (2014), SouthCHI (2013), GRAPP (2013), CESCG (2013, 2012), ICCAIE (2011)

Talks

Gameplay Storytelling in Multiplayer Games. *November 7, 2014. PIXELvienna 9 - Connect To Science, Vienna, Austria*

ViviSection: Skeleton-based Volume Editing. *May 28, 2014. Czech Technical University in Prague, Czech Republic*

Managing Spatial Selections with Contextual Snapshots. *February 12, 2014. King Abdullah University of Science and Technology, Saudi Arabia*

Visual Parameter Exploration in GPU Shader Space. *Conference talk. June 25, 2013. WSCG 2013, Pilsen, Czech Republic*

Contextual Snapshots: Enriched Visualization with Interactive Spatial Annotations. *May 29, 2013. Czech Technical University in Prague, Czech Republic*

Contextual Snapshots: Enriched Visualization with Interactive Spatial Annotations. *Conference talk. May 2, 2013. Spring Conference on Computer Graphics, 2013, Smolenice, Slovakia*

VisGroup: Visualization Research Projects. *May 9, 2012. Slovak University of Technology in Bratislava, Faculty of Informatics and Information Technologies, Slovakia*

Employment history

OCTOBER 2011 - PRESENT	Research assistant, Vienna University of Technology, Austria Research and students supervision at the Institute of Computer Graphics and Algorithms.
AUGUST 2011 – SEPTEMBER 2011	PHP and Java developer, Net & Web Services, s.r.o., Slovakia Design and implementation of information systems.
MARCH 2010 – JULY 2011	Web developer, WEBYNAMIERU, s.r.o., Slovakia Development of web-based applications and information systems, webdesign.