Focus and Context for Volume Visualization

by

Marcelo Cohen

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy.

The University of Leeds
School of Computing

July 2006

Abstract

Scientific investigation and simulation have been producing increasingly large datasets. Usually that kind of data is visualized employing some scientific visualization technique. Another related field, information visualization, deals with non-scientific large scale data, using different approaches to achieve an effective understanding. In this thesis, we aim to apply information visualization concepts to a scientific visualization approach: volume rendering.

We demonstrate this idea through a practical application: the effective visualization of cerebral aneurysms. This is a very important issue in the field of neurosurgery, usually carried out through the visualization systems offered by the imaging equipment being used. However, the use of those systems in the operating theatre is not commonly possible, as the system requires the equipment to be present and the user interface is frequently too complex. In addition to that, usually it is difficult to display both the aneurysm with sufficient detail, and the vessel network of the brain at the same time.

Hence in this thesis we introduce a framework for volume visualization, where the idea of spatial distortion is combined with other novel effects to achieve an effective visual result - this builds on the information visualization concept of focus and context. The framework was implemented through direct programming of the graphics processor and integrated into a volume rendering system. The application was evaluated by a number of medical professionals related to the field of neurosurgery, and we present an analysis of the results.

Acknowledgements

First I would like to acknowledge the ever present support, dedication, vision and invaluable insight offered by Professor Ken Brodlie, my supervisor - thank you, Ken! I thank Mr. Nick Phillips for providing us with a real problem to tackle and for his availability and support in giving constant feedback for this work during the course of the research.

I thank the Brazilian Ministry of Education, and especially CAPES agency for the financial support in the form of the scholarship for this course.

I also acknowledge the invaluable help of Selan dos Santos, especially for his effort in assisting us to settle in Leeds, even before we arrived here and for the rich discussions and suggestions on this work.

Much appreciated and needed was also the technical support provided by Mr. Graham Hardman, Dr. Jason Wood and Mr. John Hodrien.

Valuable discussions and ideas for the evaluation were suggested by Dr. Derek McGee and Dr. Roy Ruddle. I also especially thank Dr. James Handley for helping in the evaluation rehearsal and John Hodrien for general assistance in setting up the environment during the actual evaluation. Special thanks also for Mr. Roberto Ramirez from Leeds General Infirmary to have arranged the meeting and for his support throughout the evaluation day.

Many thanks to my fellow colleagues and friends that somehow helped or contributed to this research, or to a more enjoyable life experience in Leeds. I especially thank the friendship and support of our dear friends Kleos, Lúcia and Júlia.

I would also like to acknowledge the following people who offered insight and ideas for this research: Ms. Samara Alzaidi, Dr. Hamish Carr, Professor Charles Hansen, Dr. Markus Hadwiger and Professor Tim David.

Special thanks to Dr. David Duke and Professor Nigel John for their thorough read of this thesis and for their valuable suggestions given during the examination.

Finally, I give my most deep thanks to my dearest ones that wholeheartedly supported me from the other side of the pond - Esther, Saul and Renata - and to the one that has been always by my side, giving unconditional support and enduring the best and the worst of this wonderful experience - my wife Elisa, to whom I dedicate this thesis.

Declarations

Some parts of the work presented in this thesis have been published in the following articles:

Marcelo Cohen and Ken Brodlie, "Focus and Context for Volume Visualization", *Theory and Practice of Computer Graphics 2004*, (2004) 32–39.

Marcelo Cohen and Ken Brodlie and Nick Phillips, "Hardware-accelerated Distortion for Volume Visualization in Medicine", *Proceedings of the* 4^{th} *IEEE EMBSS UK and RI Postgraduate Conference in Biomedical Engineering and Medical Physics*, (2005) 29–30.

Contents

List of Figures

List of Tables

Part I

Introduction and Literature Review

Chapter 1

Introduction

Imagination or visualization, and in particular the use of diagrams,
has a crucial part to play in scientific investigation.
- Rene Descartes, 1637.

THE TERM *visualization*, as Ware [163] describes, means *the construction of a visual image in the mind* (*Oxford English Dictionary*, 1973). But it has also come to mean something more tangible: the graphical representation of data or concepts. Contrary to what many would assume, this new meaning is not recent and not even related to computing: for instance, early examples of visualization can be found in Chinese cartography by the year of 1137 [33].

Hence visualization once was a manually intensive task, involving ink and paper. But the advent of the computing era has brought a substantial advantage: the ability to process, and subsequently visualize, large amounts of data. This ability allows us to concentrate on interpreting and understanding the data, but the human being will always have a role in effective visualization - in the end, it is us who judge which is the best way of visually representing our data.

Eventually, visualization found its way to a number of fields, especially the traditional sciences such as physics, biology, chemistry and medicine. For instance, in medicine visualization techniques are extensively used to create detailed visual representations of the human body - this can be used for diagnostic purposes or even during a surgical procedure to assist the surgeon in guiding instruments to a specific region.

While existing commercial imaging systems do provide useful visualization facilities, there remains a continuing need to explore novel ways of presenting the complex anatomy of the human being. The aim of this thesis is to study an innovative approach to exploring volumetric datasets, resulting from the combination of known visualization techniques. This is demonstrated in a medical application, allowing the investigation of an anatomical feature in detail, while still keeping visible surrounding structures.

1.1 Scope

Increasingly large data sets are being generated by science, engineering and medical applications, either by simulation or direct observation. Direct interpretation of this data is usually impossible, due to two factors: size and complexity. To help solve this problem, the discipline of *scientific visualization* [109] has emerged. This brings together two rather distinct fields: the traditional sciences (physics, biology, chemistry, medicine, etc) and computer graphics. As Collins comments [33], visualization is the *objective* while computer graphics is the *means* to achieve it.

In a similar sense, we have been observing a growth of information being stored electronically, which is mostly a consequence of the Internet's emergence. A considerable amount of that information is not directly related to physical data and maybe is not even numeric, but instead is more abstract data. This kind of data requires a different approach to be effectively visualized, which has led to the creation of a new field: *information visualization* [149]. Making a parallel with scientific visualization, we can possibly say that this new field now brings together the social sciences and computer graphics. Would it be possible to achieve gains by bringing these two fields rather closer together ?

Ultimately there is some overlap between these two disciplines, as one may use ideas from information visualization with scientific data and vice versa (see section 3.4 for volume visualization examples). Because of this, there has been some discussion about this rigid division of disciplines: for example, in a recent panel [136] Johnson suggested:

> "The goal is to create integrated visualization and analysis capabilities that use the best of information and scientific visualization research techniques and to create new integrated *scientific-information* visualization software systems".

In this thesis we shall show how a fundamental concept from information visualization, the notion of focus and context, can be integrated with a traditional scientific visualization technique, in order to enhance its effectiveness.

1.2 Motivation

Although visualization is emerging as a discipline in its own right, it is fundamentally driven by the needs of applications: its value is only evident in association with another discipline such as medicine, astronomy or biology. The starting point is data from the application that we wish to visualize - this data coming from observation or simulation. For instance, in medicine data may come in the form of numerical results from e.g. blood tests. However, a very useful type of data are those which are produced by imaging techniques, i.e. obtained by scanning a subject in some particular way. In the field of neurosurgery, visualization plays a fundamental role: as the object of study is the human brain, surgical procedures must be planned even more carefully - small mistakes can impair the subject's motor skills, memory or even have more serious consequences. The motivation for this work comes from this field: we have approached a consultant neurosurgeon from Leeds General Infirmary (LGI), Nicholas Phillips, in order to better understand the visualization needs from the point of view of a surgeon. We thus learned that a very relevant issue for them is the effective diagnosis of aneurysms, which can be greatly influenced by the speed and quality of the visualization process: for example, the scanning equipment provides reasonable views (according to the neurosurgeon), but rendering is slow and its user interface is non-intuitive and limiting. The neurosurgeon also expressed a desire to use visualization software in the operating theatre - however, he pointed out that most systems are too complex and do not provide adequate user interfaces, suitable for a restrictive environment like the theatre.

Hence the motivation for this work has emerged: the effective visualization of aneurysms, particularly towards its application in the operating theatre. To provide better insight into the issues that this task introduced, this section starts by describing and classifying aneurysms. As diagnosis can be carried out by a number of imaging techniques, we also outline those. Treatment of aneurysms is usually surgical, hence we review the two basic approaches, which directly influenced our own visualization technique. Finally, we briefly critique some previous efforts on visualizing aneurysms and moving visualization to the operating theatre.

1.2.1 Aneurysms

The Circle of Willis (COW) is a circle of arteries at the base of the brain, formed by the junction of the basilar, posterior cerebral, internal carotid and anterior cerebral arteries (see figure 1.1). According to Weir et al [166], an aneurysm is a dilation of an artery caused by the vessel wall yielding and stretching due to the pressure of the blood . Initially

it was believed that aneurysms were a result of congenital defects in the arteries, but as Weir et al further describe, most aneurysms are not congenital and their development is influenced by a number of factors such as age, smoking or hypertension. An aneurysm may at any time break open and cause sudden death from bleeding into the loose tissues or cavities of the body: in the brain, this is called *sub-arachnoid haemorrage*, and the risk of death without surgery can be as high as 63% with a ruptured aneurysm [166]. This can happen in healthy young individuals with no previous symptoms - hence the early and accurate assessment of aneurysms is a very important issue from the medical point of view.

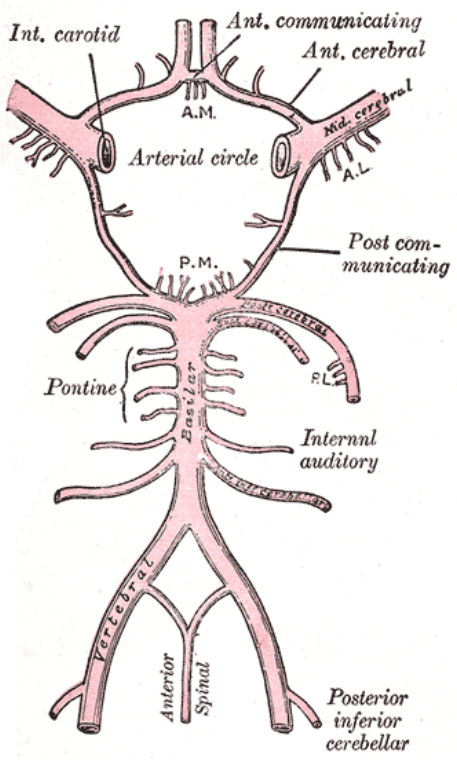


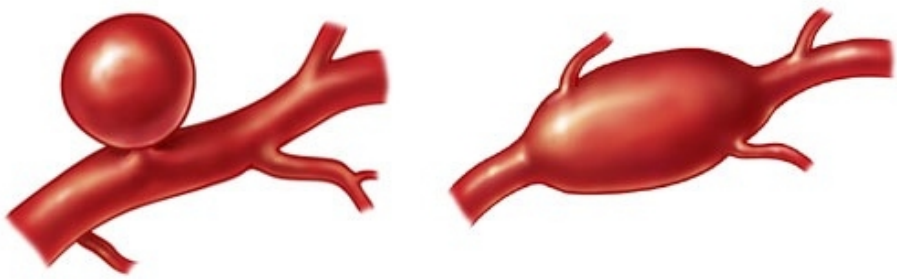
Figure 1.1: The Circle of Willis (from [50]).

Kaye [72] comments that the majority of aneurysms arise at the branch points of two vessels, and 85% occur on the anterior half of the Circle of Willis (top of figure 1.1), with roughly equal frequency on the internal carotid artery, anterior communicating artery and middle cerebral artery - the remaining 15% usually appear on the posterior half, especially on the basilar artery.

Aneurysms usually present themselves as either in a *saccular* or *fusiform* configuration: a saccular aneurysm (also called *berry* aneurysm), as the name implies, has a rounded, sac-like shape (figure 1.2a) - this is the most common type usually found on the Circle of Willis. Saccular aneurysms larger than 2.5 cm in diameter are called *giant*

aneurysms - nevertheless, it is known that aneurysms smaller than 1 cm in diameter have more chance to rupture than larger ones.

On the other hand, fusiform aneurysms (figure 1.2b), often described as *atherosclerotic aneurysms* [63], are caused by a severe form of atherosclerosis which elongates the vessel diameter over sometimes a considerable length. They normally occur on the intracranial circulation, especially on the vertebrobasilar (bottom of figure 1.1) or internal carotid arteries.



(a) Saccular (b) Fusiform

Figure 1.2: Two common types types of aneurysms (from [60]).

1.2.2 Diagnostic Techniques

The diagnosis of aneurysms can be carried out through a number of medical imaging techniques. Puig [129] reports the most commonly used (see figure 1.3):

- Digital Subtraction Angiography (DSA): this is similar to conventional angiography, but captures two sequential images of a patient, before and after the injection of a contrast agent. The DSA technique then subtracts the second image from the first, obtaining a clear picture of just the blood vessels - this gives precise information about vessel morphology, i.e. shape and relationships between vessels. DSA is still considered the "gold standard" in aneurysm diagnosis [11], as it produces the most detailed images compared to the other methods described below.

- Computed Tomography Angiography (CTA): this is based on a series of x-rays emitted around a subject and recorded by detectors at the opposite side. Then an algorithm is applied to reconstruct a 2D image (i.e. slice) from the recorded intensities. Like DSA, CTA requires the use of a contrast agent. At the moment, spiral/helical CTA (SCTA) is the state of the art, where the emitters/detectors rotate

continuously while the subject is moved perpendicularly - this allows the acquisition of an entire volume very quickly, also using multiple sets of emitters/detectors (multi-slice CT). Spiral CTA also requires additional computations (e.g. interpolation) as the slices are obtained by continuous motion.

- Magnetic Resonance Imaging/Angiography (MRI/A): instead of using potentially harmful ionizing radiation, MRI is based on a principle called nuclear magnetic resonance (NMR) [34]. MRI images usually have more contrast detail than a correspondingly CT image - however, they also have a high contrast to noise ratio, hence techniques such as nonlinear anisotropic diffusion [49, 39] have successfully been used to reduce noise levels. MRI also allows the imaging of any plane, not being limited to the axial plane as in CT. However, patients who have a pacemaker or metal implants cannot be imaged, MRI exams also take considerably longer than CT scans and are still very expensive. MRA is similar to MRI, but it removes the signal produced by stationary tissues and detects the presence of fuilds. There are two main types: *phase contrast* gives density and the internal velocity of the blood in the vessels, and *time-of-flight* only registers the presence of flow. However, these techniques are prone to artifacts when the blood flow is slow, hence an alternative approach called "black blood" can be used, which renders the vessels in black.

- Transcranial Doppler (ultrasound): this is based on the Doppler effect [34], used to obtain the speed of blood in vessels. As the blood is constantly moving, it changes the frequency of the ultrasound echoes: the Doppler effect states that a higher frequency is generated if the blood is moving towards the probe and a lower frequency if it is moving away from the probe. The amount of change in the frequency depends on how fast the blood is moving, and it is usually represented by a colour scale. By measuring the speed at some specific point, other locations can be calculated.

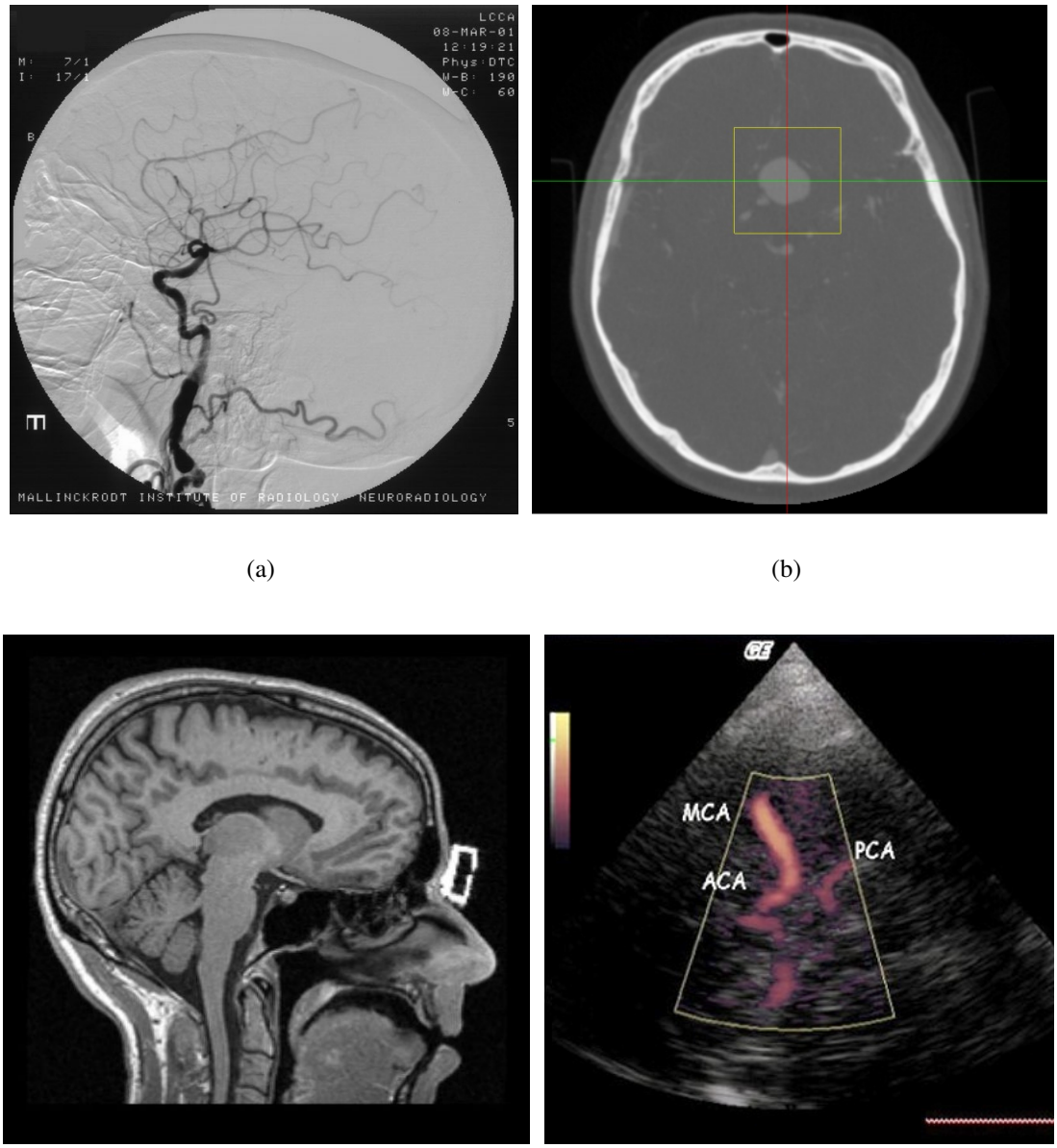
Berenstein and Hartman [14] report the limitations of traditional angiography (DSA):

1. Patient safety related to radiation exposure, volume of contrast administered and risk of stroke;

2. Anatomic ambiguity resulting from inadequate delineation of closely approximated or superimposed structures;

3. Practical considerations such as procedure length and associated costs.

Rotational angiography [19], i.e. angiography performed while rotating the x-ray machine around the body, has been suggested as a better solution to the second problem, but



(a) (b)

(c) (d)

Figure 1.3: Common imaging techniques: (a) digital subtraction angiography (from [62]); (b) computed tomography angiography (see section A.4); (c) magnetic resonance imaging (from [38]); (d) transcranial colour doppler (from [48]).

problems 1 and 3 remain and the manipulation of the acquired images for viewing in a plane other than that in which the images were obtained is usually difficult. As they further describe, the more sophisticated diagnostic techniques such as MRA and CTA have been helpful to correctly evaluate and visualize aneurysms with greater flexibility than other methods such as DSA.

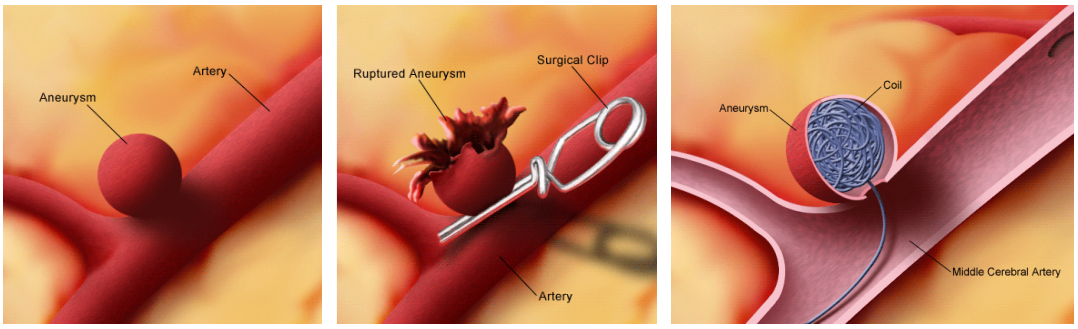
Despite the fact that all of these are 2D imaging methods, it is possible to build a 3D representation of the entire volume, allowing the visualization of the original structures with great detail - this is the volume visualization idea that we describe in chapter 2. Such a 3D view may demonstrate lesion details or severity not visible on planar views, helping to understand the complex vessel anatomy of brain AVMs (*arteriovenous malformations*) which may assist in treatment planning. However, due to the high cost associated to MRA scans they are usually employed with conditions that are harder to diagnose, such as tumours - hence CTA is normally considered as a suitable approach for aneurysm detection and evaluation. In fact, it was the method of choice for our consultant neurosurgeon, thus our visualization system was designed for use with CTA data - although it can be used with other kinds of datasets.

1.2.3 Treatment

There are primarily two options for treating a ruptured aneurysm: *clipping* and *embolization* [63]. Clipping requires open surgery, where the surgeon normally has to use a microscope to locate and precisely place one or more titanium clips at the neck of the aneurysm: this isolates it from the vessel, thus preventing it from receiving blood (see figure 1.4b). It is an effective technique, albeit risky as surgery of the brain is involved.

The clipping procedure was the normally used technique to treat aneurysms, until embolization was introduced. This was possible due to developments in the field of *interventional neuroradiology* [16], where imaging techniques are used to guide instruments to a specific region that needs intervention. The embolization procedure (also called *endovascular coiling*) consists in introducing a catheter through e.g. the femoral artery - the catheter is then advanced to the neck of the aneurysm. Once at the correct position, thin platinum coils are sent through the catheter - these coils fill the space inside the aneurysm, causing thrombosis (see figure 1.4c). Embolization has the significant advantage that open surgery is not needed, and frequently just a local anesthetic is required at the site of insertion. It also allows the treatment of aneurysms which are not suitable for clipping, e.g. those that are too large.

Until very recently, the benefits of embolization versus clipping were inconclusive, as the technique was quite new. But in a large scale clinical trial involving medical centres in the UK and Europe, Molyneux et al [112] reported that coil embolization was found to have less risk (7.4%) of severe complications than open surgery with clipping. However, they have also discovered that although the risk of re-bleeding is low in both cases, it is more common in endovascular coiling.



(a) (b) (c)

Figure 1.4: Treatment possibilities for aneurysms: (a) unruptured aneurysm; (b) clipping: a surgical clip is placed at the neck of the aneurysm, cutting off its blood supply; (c) embolization: platinum coils are inserted into the aneurysm through a catheter inserted at the femoral artery (adapted from [157]).

1.2.4 Medical Visualization of Cerebral Aneurysms

Considerable research has been carried out into the medical visualization of aneurysms: we shall now describe a selected subset. Research has shown that the gold standard for visualization of aneuryms is the 3D reconstruction approach (i.e. isosurfacing, see section 2.4). There are two reasons for this: first, the contrast used during the exam creates a clear distinction between the vessels and surrounding tissues (but there are issues with CTA and bone, see section 2.4 for a discussion), making it easier to extract vessel surfaces. Second, while the availability of texture mapping in consumer graphics cards now makes direct volume rendering (see section 2.3) competitive in many cases, the use of larger datasets is still an issue. The rendering quality of direct volume rendering approaches is starting to improve, but 3D reconstruction is still considered the best method to visualize fine detail in aneurysm images. However, the pre-processing time for reconstruction can be long, and until very recently, the rendering performance was also low due the large number of generated polygons. Nevertheless, recent research such as the work of del Rio et al [36]

has shown that the rendering can be optimized to reach similar speeds as direct volume rendering.

The use of 3D reconstruction is widely supported by a number of publications. For instance, Adams et al [4] used MRA data as a base 3D reconstruction, with good results when compared to MIP (maximum intensity projection, see section 2.6.3) rendering. Bartz et al [9] presented a system which used rotational angiography images to create images of a virtual angioscopy, i.e. views from inside the vessels. Later, Adams et al [3] compared MRA, planar views, MIP and 3D reconstruction of DSA data. They found that planar views and 3D reconstruction images were comparable to those obtained by DSA, and MIP images were poor in comparison to the other approaches. Reconstruction also allows the measurement of vessels, which was highlighted by Bruijns [19] in his work, where he was able to extract the shape of vessels in a semi-automatic way, from 3D reconstructed structures obtained through rotational DSA.

It is also clear that the idea of 3D reconstruction from rotational DSA is starting to be considered a possible replacement for traditional DSA, showing improved superiority in aneurysm visualization [59, 153]. This was confirmed later by Piotin et al [126], who published a study where the accuracy and precision of CTA, MRA and rotational DSA were evaluated: they found that in general, rotational DSA is the most accurate and precise method, followed by CTA and finally MRA. More recently, this was supported by Beck et al [11], who studied 155 cases of saccular aneurysms exclusively with 3D reconstruction of rotational DSA. Nevertheless, CTA is still a good alternative: Kato et al [71] showed that it is a faster and less risky approach for the patient. They also demonstrated later [70] that for regular sized aneurysms (i.e. those that are not too small), CTA has the same efficacy as DSA or rotational DSA, with the advantages already mentioned above. This was also suggested by Tsuchiya et al [156], who indicated that helical multi-slice scanners have enough spatial resolution to be compared to DSA.

But regardless of the visualization technique, adequate performance is essential for interactive visualization. Fast rendering was once limited to advanced workstations - hence some research has focused on using remote visualization techniques to obtain acceptable rendering speed when studying aneurysms. For example, Engel et al [42] created a system where a local machine was used to interact with the user, sending commands to a remote workstation, which in turn produced high quality display images - these images were then sent back to the local machine. Their system also offered a hybrid approach, where a low quality rendering was carried out by the local machine during user activity - the remote machine sent the high quality image only when the user was idle. This helped to minimize the user interface lag. At the time, they had used the system in 8 different cases,

with satisfactory results. Another similar example is the work of Luke and Hansen [103], who combined data compression and low latency techniques to achieve high frame rates.

Another relevant issue is the delivery of medical data to the operating theatre. In this context (but not related to aneurysms), we highlight the work of John et al [64, 65], who recently presented the first application of a remote visualization system that was effectively used during hepato-pancreatic surgery within the operating theatre. This is an attractive approach, as it lessens the requirements of the local machine, which is used just to display the rendered data and interact with the user. In fact, this is the most scalable solution: as data sets keep improving in size, there is a limit to what a single graphics card will be able to handle. However, this creates the necessity of a fast and reliable networking system, which may not always be readily available. We note that in contrast to their approach, our work focuses on local visualization, i.e. using a local computer to carry out the entire visualization procedure - LGI currently does not have the required network infrastructure to support remote visualization.

Further examples of visualization applied to medicine can be found in the survey by Vidal et al [160].

1.3 Objectives

From the motivation, we can propose three main objectives for this thesis:

1. To develop an approach to volume visualization that helps medical professionals to achieve clear visualization of aneurysms;

2. To create a visualization system suitable for use in a medical environment, particularly as a pre-operative planning tool;

3. To carry out an evaluation of this visualization approach and the developed system.

Objectives 1 and 2 are directly related to the motivation: those are the main driving forces of this research. The visualization approach, however, must be thoroughly evaluated to be considered “effective”. Evaluation has typically had too low a priority in visualization research, hence we include this as our third objective.

1.4 Methodology

During the course of this work, we have employed a cyclical research methodology common to much scientific work, as described by Leedy and Ormrod [92]: all research starts

with a problem or question, which must be clearly defined. Problems are sometimes subdivided into smaller, more appropriate and tractable subproblems. These lead the researcher to gather data related to the problem. The data is then interpreted and may lead or not to a tentative solution of the problem. The researcher then formulates a hypothesis, or a guiding question. To support a hypothesis, frequently more data has to be gathered and evaluated, which can in turn generate more hypotheses. At this point, a cycle is complete: in fact, as Leedy and Ormrod comment, this is more accurately depicted as a helix. This helical cycle goes on until a conclusion is reached - either answering, not answering or partially answering the original question.

Applying this methodology to our research, we have started with the desire of working in a practical field, preferably one that would have immediate usefulness - hence we selected medicine. In order to have a real life problem, we then contacted a consultant neurosurgeon and together identified a possible problem to solve: the visualization of aneurysms. The neurosurgeon was also particularly interested in using visualization technology within the operating theatre.

These issues led to a review of the field of volume visualization. As we describe in chapter 2, there are many ways to visualize volumetric data, hence it was crucial to correctly understand the implications, advantages and disadvantages of each one: we needed to find techniques that offered an acceptable balance between speed and image quality.

A prototype system was then created, having in mind the necessities pointed out by the neurosurgeon: speed, ease of use and flexibility in handling different dataset formats (like JPEG or DICOM images). This prototype was presented to the consultant neurosurgeon, who provided initial feedback on features and suitability for the task. From this feedback, we identified additional research opportunities: enhancement of aneurysm visualization and the necessity for higher rendering speed. These led to the next stage of investigation.

The field of information visualization has lots of interesting ideas oriented towards the display of abstract data (non-numerical), but we decided to investigate whether the use of a particular concept would be suitable for volume visualization: the idea of focus and context (reviewed in chapter 3). As a consequence of this study, we proposed a taxonomy to classify focus and context approaches applied to volume visualization (described in chapter 5). On the practical side, initial results were promising but speed was still an issue, as the application of focus and context techniques imposed a higher computational load on the system. With the feedback of the neurosurgeon, another issue was identified: anatomical structures (e.g. the skull bone) could easily get in the way of the aneurysm, thus preventing a clear view of it. These two issues created a new branch of research.

This branch initially took form as understanding the programmability of the modern graphics processor (see chapter 4) and its implications for focus and context visualization. We carried out a study of a number of different algorithmic implementations, which are described in chapter 6. Based on that, we proposed a unified framework, where additional effects could be obtained along with volumetric distortion - one of those gave us a solution to the occlusion problem described above (see chapter 5).

A final assessment stage consisted of user evaluation of the system by a number of medical professionals. Observational and survey data was gathered and then analyzed. Finally, we summarized the results, identified the contributions of this research, its limitations and proposed future work.

1.5 Research Hypotheses

In summary, during the course of following the research methodology outlined in section 1.4, we have identified the following research hypotheses:

1. Is focus and context a suitable approach for enhancing the visualization of medical data, especially aneurysms ?

2. Can we exploit the graphics processor to create innovative effects and also to achieve acceptable rendering performance ?

3. Can we maintain interactive rendering speeds and still provide a clear view of a specific anatomical feature, regardless of viewpoint ?

1.6 Contribution

From the motivation exposed in section 1.2 and the objectives described in section 1.3, we can now state the contributions of this thesis:

- Development of a taxonomy for focus and context applied to volume visualization, where the ideas of distortion and refinement/accuracy are integrated;

- Design of a unified framework for focus and context effects applied to volume rendering;

- Creation of a number of hardware-accelerated methods for implementing volumetric effects, plus a comparative analysis in terms of features, flexibility and performance;

- User evaluation of the system by the potential end users (medical professionals).

1.7 Outline

This document is organized in four parts and a number of chapters, as follows:

- *Part I - Introduction and Literature Review*: this part contains this introductory chapter and the remaining chapters related to the literature review:
  - **Chapter 1** presents the scope and motivation, and describes the objectives, methodology and contributions of this thesis;
  - **Chapter 2** examines research on volume visualization, critiquing the many approaches in order to identify the most suitable method to our problem;
  - **Chapter 3** reviews work on the focus and context idea, from the early ideas to more comprehensive views, also relating it to volume visualization;
  - **Chapter 4** appraises the evolution of the graphics processor, with emphasis on the programmability aspects - which was crucial to the development of the *VolFocus* system.
- *Part II - Framework*: this part is comprised of **Chapter 5**, which presents our taxonomy for focus and context in volume visualization and later introduces a versatile framework to achieve distortion and extra effects with volume data;
- *Part III - The VolFocus System*: this part incorporates the chapters that relate the previously described framework to its implementation:
  - **Chapter 6** chronologically describes all the research on hardware-accelerated methods that was carried out to implement the framework;
  - **Chapter 7** critically examines the general development of the *VolFocus* system, concentrating on the user interface issues;
- *Part IV - Evaluation and Conclusions*: this part brings together the evaluation of the system and the overall conclusions for this thesis:
  - **Chapter 8** appraises the evaluation methodology, planning and the actual procedure, and carries out a critical analysis of the results;
  - **Chapter 9** reviews the goals of this thesis, summarizes the results and contributions, and suggests future work.

Chapter 2

Volume Visualization

2.1 Overview

THIS CHAPTER DESCRIBES the underlying concepts, techniques and algorithms commonly used for the visualization of volumetric data. We will emphasize the discussion of the texture-based approach (see section 2.9), as it is the base of the implementation (see chapter 6 for details) of our focus and context framework (chapter 5). Finally, we review some of the research that has aimed to join both focus and context and volume rendering techniques.

2.2 Volume Data

Before describing how the visualization is performed, we must define what is a volumetric dataset. We consider a 3D continuous function (or signal):

$$f(x,y,z); \qquad x,y,z \in \mathbb{R}$$

In medical imaging, for example, this function represents the subject being examined. The scanning equipment then *samples* this function, i.e. approximates it as a discrete volume, composed of *voxels*. Following the definition by Lichtenbelt et al [99], we understand a voxel as a point in space (with an infinitesimal size). Elvins [40] classified volumetric datasets as *rectilinear*/*curvilinear* and *regular*/*irregular*: a rectilinear dataset

is one where the voxels are arranged on a Cartesian grid (as opposed to curvilinear); and a *regular* dataset is one where the spacing between voxels is constant (as opposed to irregular). We can also view a volumetric dataset as a stack of 2D slices (as typically happens in medical applications) - in this case, even with regular datasets, the distance between slices can be different than in voxels within a slice: this characterizes an *anisotropic* dataset (as opposed to *isotropic*).

In visualization, in order to reconstruct the continuous function from the discrete data, we create an estimate $F(x,y,z)$ of f through a process of filtering of interpolation (see section 2.10.1).

2.3 Classification of Volume Visualization Methods

Extensive research has been done on the visualization of 3D datasets, but according to Brodlie and Wood [17] there are primarily three main approaches:

- Surface extraction and reconstruction: these methods extract surfaces of constant scalar value from the volume data (see section 2.4) - they usually have a long preprocessing stage, but rendering is fast as modern graphics hardware can efficiently draw polygons;
- Slicing: this produces a 2D slice through the domain of the 3D data (see section 2.5);
- Direct volume rendering: this class of methods deal with the raw 3D data, employing algorithms that build an image without requiring the creation of an underlying geometric structure. Rendering performance and image quality vary according to the algorithm, and examples of direct volume rendering methods include: ray casting (section 2.6), shear-warp factorization (section 2.7), splatting (section 2.8) and texture mapping (section 2.9).

The next sections describe each of these methods in detail.

2.4 Surface Extraction and Reconstruction

These methods work best when there is a defined structure within the volume - medical imaging is probably their main application. By applying an algorithm to extract surfaces from the image data, this class of methods allows a precise visualization of almost any structure, at the cost of generating a potentially large number of polygons.

In 1987 Lorensen and Cline proposed the seminal work on isosurfacing: the *marching cubes* algorithm [102]. Given a user-specified threshold value, the volume data is traversed a cell at a time - we name *cell* a set of eight neighbouring voxels. Each vertex in a cell can have a scalar value greater or less than the threshold: this produces 2^8 different combinations, which can be stored in a table - each combination determines which polygons (if any) should be created for that particular configuration. The algorithm then computes the correct vertex of each polygon along the cell edge by linearly interpolating the value of the two voxels that are connected by that edge. Lorensen and Cline also suggested that these 256 cases could be reduced to 15 canonical cases, by reflecting and symmetrically rotating some configurations. However, later research from Nielson and Hamann [119] demonstrated that such reduced cases can produce inconsistencies, like holes in the generated surface.

In the context of our application, it can be hard to use a surface extraction method: an aneurysm is part of a blood vessel, and these appear as bright voxels in CTA images (as a contrast agent is used). However, bone also appears as light regions due to the density of the material - hence correctly differentiating vessels from bone is still much a research issue. A segmentation algorithm such as region growing [127] could be used, but it requires no contact between the bone and vessel voxels, and this is not always the case. Surface extraction also requires pre-processing, which is not always desirable when one wants to quickly visualize a dataset. Finally, as Roettger et al note [140], even recent hardware-accelerated methods still do not provide acceptable performance for interactive exploration. Because of these reasons, we shall focus our discussion on direct volume rendering methods.

2.5 Slicing

This is the simplest approach in terms of complexity, as it obtains a 2D slice from the 3D data. The slice can either be taken from planes parallel to one of the coordinate planes, which produces the classic orthographic views - in medical terminology, these are called axial (top), saggital (front) and coronal (side) views - or in arbitrary orientations. This is also called *multiplanar reformation* (MPR) and as Vidal et al [160] mention, it is a very useful tool and widely used in medicine. They also observe that radiologists are trained to build a mental image of a tridimensional structure such as blood vessels, by looking at sequential slices - we have also confirmed this ability during the evaluation of our system (see chapter 9).

To solve the problem of following non-planar structures, *curved planar reformation* (CPR) was introduced [68] - this creates a planar view by following a previously identified vessel centreline and e.g. projecting a thin slice of the voxels along the way (this is called a *projected* CPR). This allows a single view of the entire vessel, but also has some problems, e.g. the projection prevents the user to measure the vessel length. Later Kanitsar et al [69] improved the original CPR approach, sampling the vessel along a spiral trajectory around the centreline - this produced much better results, and allowed the entire vessel to be displayed with no occlusions.

2.6 Ray Casting

Ray casting was the first approach that implemented a direct volume rendering technique (see section 2.3). In this case, the volume is usually described as a particle cloud [111], where each particle has a certain density, and thus absorbs and/or reflects the incident light according to its surface properties - we consider that a particle can also emit light (this is an emission and absorption optical model, without scattering, as proposed by Max [107]). The process of defining these properties is normally called *classification*, and we give a more detailed description in section 2.10.2.

This is an image-based technique, as it works from the output image pixels: we imagine that a ray is cast from the viewer position through each pixel in the output image. The ray traverses the volume and accumulates the absorption and emission of voxels as it goes through (figure 2.1). However, as the volume data is discrete, it must be *resampled* (or interpolated/filtered) to obtain values at intermediate locations - the common approach for this is trilinear interpolation (see section 2.10.1 for further discussion), and the ray is normally sampled at equidistant locations.

The base of this method is a mathematical formulation called the volume rendering integral, proposed by Kajiya and Herzen [67] and described in section 2.6.1. In section 2.6.2 we describe how the method is implemented in practice and in section 2.6.3 we present a commonly used, alternative approach to full integration (*MIP*). Finally, section 2.6.4 presents some recent advances.

2.6.1 The Volume Rendering Integral

We follow the initial explanation as described by Bartz [7]: the volume rendering integral (equation 2.1) gives us the amount of light of wavelength λ for each pixel in the image plane (I_λ), defined by a ray cast from position $\vec{x}$ (in the image plane) with direction $\vec{r}$

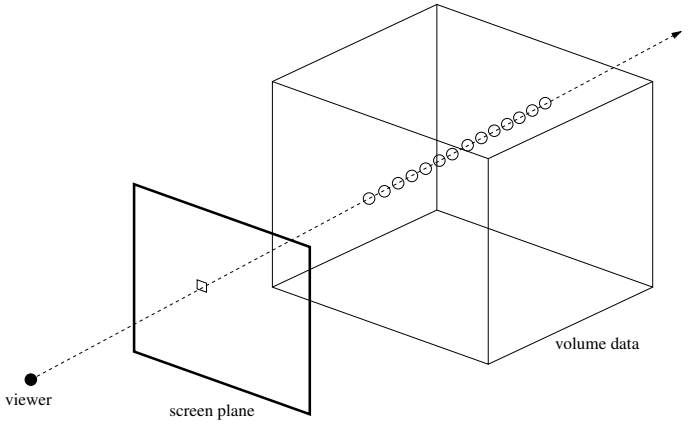


Figure 2.1: Volume rendering through ray casting: a ray is cast through a pixel on the output image and traverses the volume, integrating the emission and absorption of each voxel in its way.

- the length of this ray is given by L. The integral then accumulates the contribution of each sample at position s in the ray, using the colour $C_\lambda(s)$ and the light extinction coefficient $\mu(s)$ [107] - which is essentially the opacity. This result is then attenuated by the exponential opacity term $e^{(-\int_0^s \mu(t)dt)}$, which in itself accumulates the combined opacities of all particles between s and the viewer, for this ray:

$$I_\lambda(\overrightarrow{x}, \overrightarrow{r}) = \int_0^L C_\lambda(s)\mu(s)e^{(-\int_0^s \mu(t)dt)}ds \tag{2.1}$$

2.6.2 Practical Ray Casting

In practice, a numerical approximation of equation 2.1 is used - as proposed by Levoy [95] - and if we assume that the data is unit spaced and define $C_\lambda(i)$ and $\alpha(i)$ as the colour and opacity of the sample at position i, we can express it as:

$$I_\lambda = \sum_{i=0}^{n} C_\lambda(i)\alpha(i)\prod_{j=0}^{i-1}(1-\alpha(j)) \tag{2.2}$$

Depending on the needs of the application, at this point we may use *shading* to apply a lighting model to each voxel - this requires the computation of the voxel normal, which can be carried out by the estimation of the gradient at that voxel (see section 2.10.3 for details). As Brodlie and Wood [17] further describe, equation 2.2 can recursively be computed by working on a single sample at a time and accumulating the colour and opacity -

this is called *front to back compositing*. We treat colour and opacity separately and define two equations that accumulate their contributions:

$$\begin{aligned} C'_i &= C'_{i-1} + (1-\alpha'_{i-1})\alpha_i C_i \\ \alpha'_i &= \alpha'_{i-1} + (1-\alpha'_{i-1})\alpha_i \end{aligned} \tag{2.3}$$

Equation 2.3 can be translated as follows: from the previously computed opacity and colour along the ray (C'_{i-1} and α'_{i-1}) and the current values at this position (C_i and α_i), we calculate the resulting colour and opacity for this position: C'_i and α'_i. Note that it is possible to optimize this procedure, as we can stop traversing the ray when the accumulated opacity reaches 1.0 - this is called *early ray termination* as proposed later by Levoy [96]. However, this form of compositing has implications for hardware-based rendering: for example, we need to store the opacity values in the framebuffer, which means that the hardware must have support for *destination alpha* - this is only supported by workstations-class graphics cards. Because of this, another simpler formulation can be used which carries out the compositing procedure from the last sample in the ray towards the viewer, i.e. from n to zero. This is called *back to front compositing* and in this case, only the colour needs to be accumulated to obtain the same result:

$$C'_i = \alpha_i C_i + (1-\alpha_i) C'_{i+1} \tag{2.4}$$

Now in equation 2.4, the resulting colour at each position (C'_i) is computed from the current value at this position (C_i) and the resulting value computed in the previous step (C'_{i+1}), i.e. in the next position along the ray.

As part of his basic research, Levoy also proposed that the time-consuming volume rendering process could be done by adaptive refinement [97], i.e generating a first image with a ray for each k pixels and interpolating the results to obtain the intermediate pixels. Then k would be reduced for each subsequent refinement, until we reach one ray per pixel (or in case of supersampling, more than a ray cast for every pixel) - this allowed the user to interrupt the process and adjust the viewpoint, in case the resulting image was deemed not adequate.

2.6.3 Maximum Intensity Projection

A simplified approach is to find just the maximum intensity value in the ray, instead of performing the accumulation described earlier: this is called *maximum intensity projec-*

tion (MIP) and is commonly used to visualize medical data obtained from traditional x-ray angiography or better methods like MRA or CTA, as the objective is to highlight blood vessels, which usually present themselves as high contrast voxels. The advantage is that there is no need to define optical properties such as colour and opacity for the voxels (the so-called transfer function, see section 2.10.2) - but depending on the dataset, the depth information in the resulting image can be misleading. As noted by Brodlie and Wood [17], the challenge in MIP rendering is to achieve high quality and high performance - for example, a fast Java-based software implementation was proposed by Mroz et al [114]. Later, Hauser et al [56] proposed a system that enabled the combination of different rendering techniques for different subsets of the data, including MIP (see also section 3.4). The present state of the art (in terms of performance) is the implementation by Mora and Ebert [113], based on octrees.

2.6.4 Recent Advances

Accelerating the ray casting algorithm has always been a major research theme: Knittel [78] proposed a system based on the Pentium III CPU where a number of optimizations were possible by exploiting the SIMD features of the MMX instruction set and also considering a multiprocessor architecture. Large scale data is still limited to CPU-based ray casting, hence recent work such as Grimm et al [51]: they showed that a common notebook computer (Pentium M 1.6 GHz) could achieve a frame rate of 2.5 frames per second while rendering a large (512 x 512 x 1202) dataset.

A lot of the recent work has been dedicated to accelerating ray casting through graphics hardware. Possibly one of the first attempts was the development of the *VolumePro* board by Pfister et al [124]: this was a dedicated graphics card capable of performing ray casting - in fact, it is based on a modified version of the shear-warp algorithm - at interactive speeds (256^3 dataset at 30 frames per second), and offering advanced features such as gradient estimation, classification, and per sample lighting.

As the graphics processors evolved (see section 4.3), it became possible to achieve true ray casting with consumer-level hardware. The pioneering work was developed by Purcell et al [130]: they first described how ray tracing could be mapped to common graphics hardware and also analyzed the performance of a ray casting implementation on possible next generation hardware through a simulator. The first GPU-based ray casting implementation was later proposed by Roettger et al [141]: their implementation incorporated the ideas of both early ray termination and empty-space skipping as optimization schemes, and volumetric clipping. A similar implementation was described by

Krüger and Westermann [83]. Both approaches relied on multi-pass algorithms, i.e. more than one rendering pass was needed to perform the entire procedure - however, Roettger et al also used pre-integrated classification (see section 2.10.2). Recently, Stegmaier et al [152] demonstrated that with current GPUs it is possible to achieve the same results with a single-pass algorithm (see figure 2.2), also allowing sophisticated effects such as ray reflection and refraction, and transparent isosurfaces.

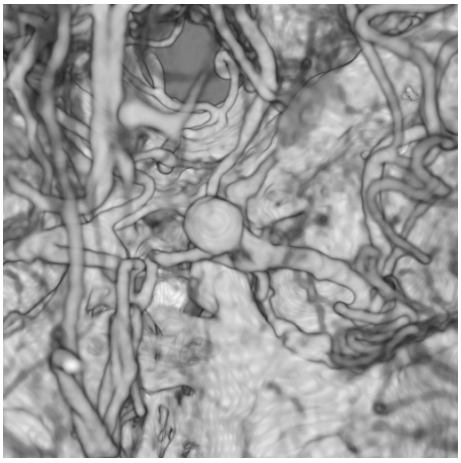


Figure 2.2: Accelerated ray casting with the GPU (using the implementation from [152]): a high-contrast dataset (see section A.2) is rendered on a 512^2 display window at 4 frames per second, but with high quality results.

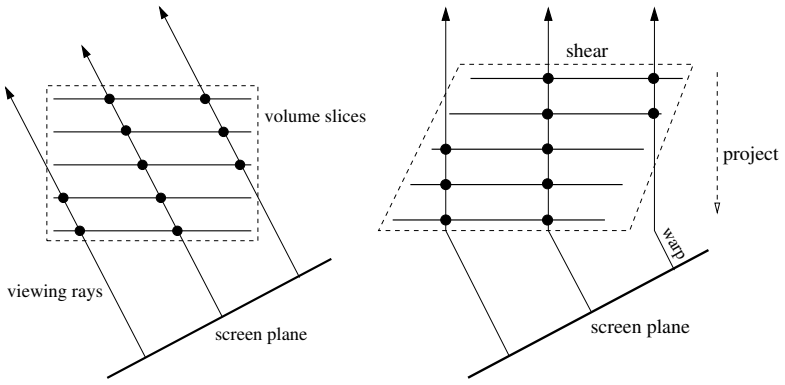
2.6.5 Discussion

Ray casting is still the method that provides the best possible quality in volume rendering and a lot of research has been carried out to increase its performance (see section 2.6.4). However, even with GPU-based approaches, performance is still not at the level of faster methods such as the texture-based ones (see section 2.9). This may change in the near future, as the hardware is becoming increasingly fast and also permitting more accurate representations of the data. For instance, GPU-based ray casting relies on textures to store temporary data - hence advances in texture representation such as floating point textures with higher precision, combined with floating point frame buffers, will allow more precise computation and compositing.

2.7 Shear-Warp Factorization

The shear-warp factorization method, proposed by Lacroute and Levoy [86] is a clever way of quickly computing the volume rendering integral: instead of casting a ray through

each screen pixel (as in the ray casting method), the volume is split into parallel slices and those are sheared and projected into an intermediate image, in such a way that the viewing rays become perpendicular to the slices. Then a 2D affine warp is carried out in the intermediate (distorted) image and the resulting image is sent to the screen (figure 2.3). This description works for parallel projections: if we are using a perspective projection, then the shear is combined with a scaling factor that depends on each slice distance from the viewer.



(a) Original volume and viewing rays

(b) Shearing and warping

Figure 2.3: Volume rendering through shear-warp factorization (from [86]), parallel projection: in (a) the original volume slices and viewing rays are shown whilst in (b) the slices have been sheared, hence aligning the viewing rays.

The greatest advantage to this method is that the warp is applied just to a final image and not to individual slices, therefore it is fast; and note that the shearing can also now be performed very efficiently by the graphics hardware (e.g. with textures). In fact, the shear-warp method is probably the fastest software-based volume rendering approach - Lacroute also described an even more efficient, parallelized version of the original algorithm [87]. Some recent work on shear-warp includes the research of Sweeney and Mueller [154], who analyzed the effect of post-interpolated classification and shading, and the use of interpolated, intermediate slices to reduce the aliasing effects (similar in principle to what Rezk-Salama et al [134] have done for texture mapping, see section 2.9.1), along with other aspects.

2.8 Splatting

Splatting is a novel object-order algorithm proposed in 1990 by Westover [168], based on the idea of computing the contribution of each voxel to the final image, and traversing the dataset in a front to back order - hence the algorithm starts from the face of the volume cube which is closest to the observer, and works with parallel slices to that face. The name "splatting" comes from the fact that the procedure can be compared to throwing a snow ball at a glass pane - the ball will spread from the point of impact, where the contribution is highest. Each scalar value (voxel) is first classified through application of the transfer function - the resulting colour and opacity can be shaded by e.g. gradient shading (see section 2.10.3). The next step is the splatting itself: the voxel is projected onto the image plane and a reconstruction kernel (a circular Gaussian) is used to calculate the magnitude of the contribution. This projection is called a *footprint*, and it is scaled proportionally to the size of the volume and to the size of the image plane, and then placed at the centre of the projected voxel in the image plane. This footprint can be pre-calculated and stored in a table for lookup during the actual rendering - this allows the method to be faster than this description suggests. The final step is to blend the current voxel with the image plane at every pixel within the footprint area, using the colour and opacity (from classification) attenuated by the value of the Gaussian at that specific pixel centre. However, this algorithm can produce bleeding artifacts from hidden objects, as the volume rendering integral has been essentially reordered.

Westover then suggested an improvement of his original algorithm [169], where the splats were composited into "sheet buffers" - each sheet buffer is equivalent to a volume slice, aligned parallel to the volume face most parallel to the image plane. The sheet buffers are then accumulated back to front, which is similar to the discretization of the volume rendering integral and the texture-based methods - this reduced the bleeding artifacts from the original method. However, as mentioned by Mueller and Crawfis [115], this introduces brightness variations - they describe these as "popping artifacts", which can be visible when the viewpoint changes. Therefore they also proposed a modification in the sheet buffer algorithm, using image-aligned slices which prevented these artifacts. Mueller et al [116] later demonstrated that the splatting algorithm could be achieved with post-classification (i.e. interpolation followed by classification), which reduced the smoothing effect normally present.

Recently, it became possible to implement the splatting algorithm at near interactive speeds through the graphics processor: an example is the work of Neophytou and Mueller

[118], who carried out early elimination of hidden splats and the skipping of empty spaces using the hardware z-culling to achieve an efficient implementation (see figure 2.4).

Figure 2.4: GPU-accelerated splatting: *Bonsai* dataset (256^3) rendered at 4.9 frames per second on a 400^2 display window (from [118]).

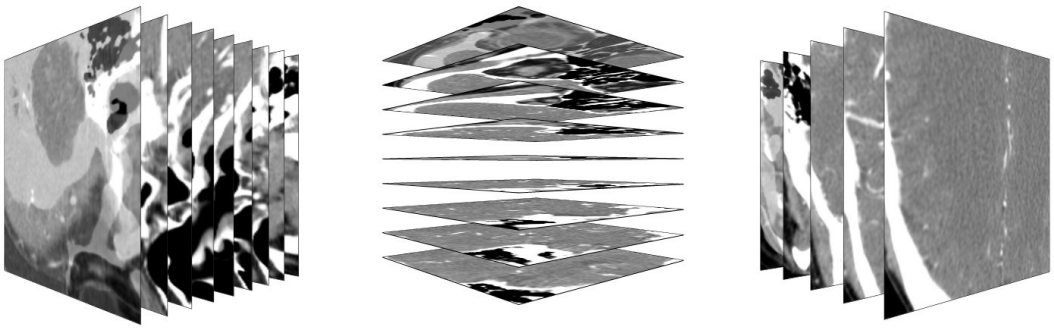
2.9 Texture Mapping

This approach aims to achieve interactive rendering performance by exploiting the texture mapping hardware. In general terms, there are two ways of doing this: either using 2D or 3D textures. The next sections present both approaches, highlighting advantages and disadvantages of each.

2.9.1 2D Textures

The idea of the 2D textures method is very simple, inspired by the earlier work of Lacroute and Levoy [86] (as we described in section 2.7): the original volume is sliced into three sets of 2D planes (images), each set perpendicular to one of the coordinate axes (figure 2.5). Then the bounding box of the volume is computed (which is normally called *proxy geometry*) and a series of texture mapped, parallel polygons are drawn from the back of this box towards the front, along one of the coordinate axes. To decide which axis to use (which also selects the set of textures), the angle between the polygon normals and the viewer is computed: the minimal angle gives the best set of textures to use. This approach automatically provides filtering, as the hardware is capable of bilinear filtering, and back to front compositing (as described in section 2.6.2), as colours and opacities are accumulated when drawing overlapping polygons (i.e. we assume that a transfer function

has been applied to the original data). Note that the number of slices used directly influences both image quality and rendering time - for optimal quality, this can be set to the maximum dimensions of the volume data, but not higher.



(a) X-aligned textures (b) Y-aligned textures (c) Z-aligned textures

Figure 2.5: Volume rendering through 2D texture mapping: the three sets of 2D textures required for this method are presented with a few slices each.

This is a very fast method and as it uses the commonly available 2D texture mapping, does not require specialized hardware. However, this method has some disadvantages: first, it requires three times the amount of texture space as the original data. Second, if the viewpoint or viewing direction changes, the texture set being used can abruptly be switched to another, and this causes a visible change in the final result. Finally, depending on the viewpoint, aliasing artifacts at the edge of polygons may be visible due to an insufficient sampling rate - as noted by Rezk-Salama [133], this cannot be solved as the number of polygons is fixed in this algorithm. Because of this, he proposed a solution based on multitextures [134], where the artifacts were minimized by interpolating the texture values between two adjacent slices.

As this 2D method is fast and can be easily implemented with almost all graphics hardware, some research has been done to allow this kind of visualization over networks, in particular the Internet. For example, Hendin et al [57] proposed a system based on Java and VRML to allow the visualization of volumes over the Internet, also providing isosurfacing integration into the rendered volume. Later, Engel and Ertl [41] created a similar system, but with multiple user capabilities (such as tagging of volumes and offering a chat feature) and also incorporating compression to minimize network traffic.

2.9.2 3D Textures

This idea was first suggested by Akeley [5], and Cabral et al [24] published the seminal work on 3D texture-based volume rendering, complemented by Wilson et al [170] providing the mathematical equations required to define 3D texture coordinates, and Cullip and Newmann [35] analyzing and discussing the implementation issues and the advantages of viewpoint-aligned versus object-aligned slices. In this case, as the hardware can directly work with 3D data, it is just a matter of loading the volume as a 3D texture and drawing texture-mapped polygons - now orthogonal to the viewing direction, again from the back towards the front of the proxy geometry. This process is illustrated in figure 2.6.

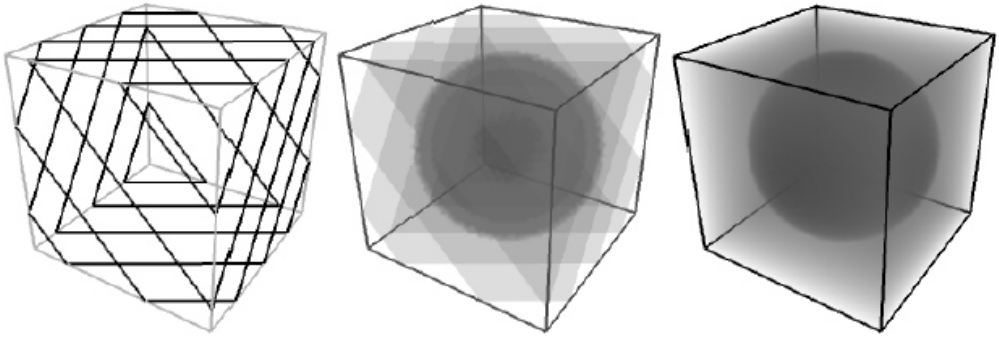


Figure 2.6: Volume rendering through 3D texture mapping (from [167]): a few slicing polygons are shown in the left image, and the texture-mapped result is presented in the centre image; finally, the right image displays the same volume, but rendering with a much greater number of slices.

However, as the polygons now are not necessarily aligned with a coordinate axis, the actual rendering algorithm is slightly more complex. It can be described by the following steps [61]:

1. Transform the proxy geometry vertices from *world* to *view* coordinates (see section 4.2);

2. Identify the minimum (z_{min}) and maximum (z_{max}) z coordinates of the vertices in *view* coordinates. Create n slicing polygons, using equidistant spacing from the viewer, where n is computed by dividing the maximum number of slices (see below) by the sampling rate;

3. For each polygon from z_{max} to z_{min}:

   (a) Check for all intersections with the edges of the proxy geometry. Store intersections in a vertex list - there will be at most six intersections (see middle polygon in figure 2.7);

(b) Sort intersections in clockwise or counterclockwise order - this can be done by first computing the centre of the polygon formed by all intersected vertices and then calculating the angle between a vector from this centre to the first vertex on the x axis (as reference), and a vector from the centre to each vertex;

(c) Create triangles from the sorted vertices (*tessellation*) and associate texture coordinates to each vertex - in OpenGL, the 3D texture coordinates for each vertex can be obtained by e.g. the texture generation functions (*texgen*) or by applying the same transformation to the texture matrix;

(d) Draw the texture-mapped triangles.

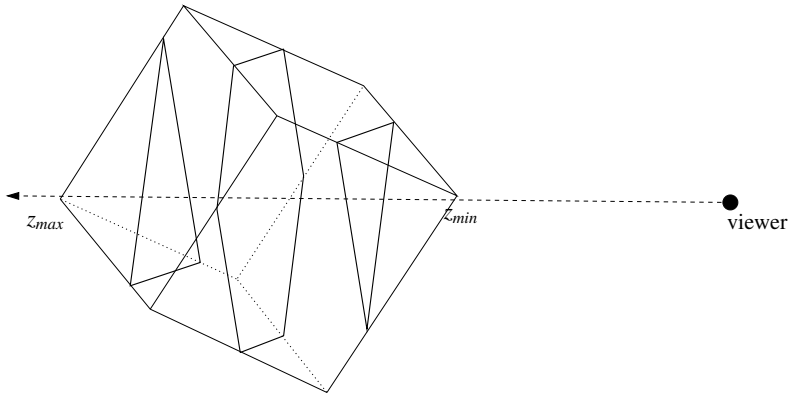


Figure 2.7: View-aligned (3D texture) volume rendering: three slicing polygons are shown through the proxy geometry - the two triangles can be directly drawn but the polygon at the centre will have to be tessellated, as it has six vertices.

The maximum number of slicing polygons can be computed according to the longest possible diagonal through the volume. If (dim_x, dim_y, dim_z) are the dimensions of the volume in each direction, then the diagonal will measure $\sqrt{dim_x^2 + dim_y^2 + dim_z^2}$ - note that this will be higher than any of the original volume dimensions. And as more slices require more rendering time, this is not desirable. Hence an *adaptive* algorithm can be used [21], which calculates the optimal number of slices according to the viewing direction: first the dot products between the viewing vector, and each coordinate axis scaled by the respective volume dimension are computed. The sum of these dot products will give the number of voxels that are visible along the viewing direction, so to obtain the optimal number of slices we just need to divide this result by the sampling rate. For example, supposing that we are viewing the volume directly from the front, then the dot product for the x and y axes will be zero and for the z axis it will be equal to the depth of the volume, i.e. dim_z. But as the observer starts moving to the right, the dot product for the z axis will become progressively smaller, while increasing for the x axis.

A final observation regards the effect of the sampling rate over the opacity [43]: if we use fewer slices, then the opacity of each voxel will have to be increased so we maintain the same overall intensity in the final image. This can be done by a scaling formula, where α is the original opacity, $s0$ is the original sampling rate (e.g. 1), s is the desired sampling rate and α' is the corrected opacity:

$$\alpha' = 1 - (1 - \alpha)^{\frac{s}{s0}}$$

2.9.3 Discussion

The 2D texture mapping method is very fast and is also available in most older graphics hardware. However, it requires three sets of textures, which adds considerably to its memory requirements - as opposed to the 3D textures method, which needs just a single 3D texture. The 3D textures method also automatically provides trilinear interpolation, whereas the 2D method only provides bilinear interpolation. This could be improved with the work described in [134] (see section 2.9.1), but the aliasing artifacts are just minimized and not completely eliminated. Another consideration is the sampling rate: because of the 2D slices, it is not constant in the 2D textures method, varying according to the viewpoint - obviously with 3D textures, this does not happen and the sampling rate is constant (at least for parallel projections and a reasonable approximation for perspective views). The slicing procedure itself leads to artifacts in the 2D case, especially as the angle between the viewing vector and the slice normal increases.

In summary, the texture mapping methods, in particular the 3D textures version, are possibly the fastest approaches to carry out direct volume rendering, efficiently exploiting the graphics hardware and allowing interactive rendering rates - this is essential for many applications, especially in medicine. However, if higher image quality is desirable then methods such as ray casting (section 2.6) or splatting (section 2.8) should be used instead.

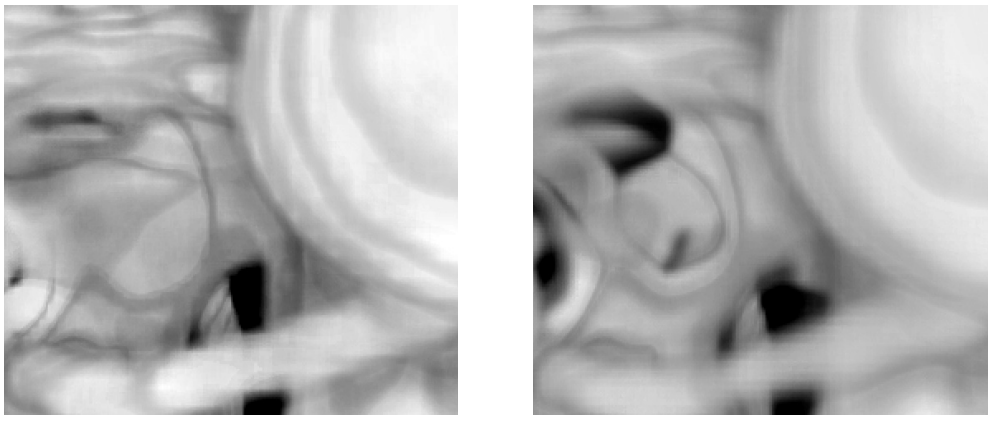
2.10 Revisiting the Direct Volume Rendering Pipeline

This section describes in more detail the stages of the direct volume rendering pipeline and also presents some research that has been done to improve specific stages. The pipeline described informally in section 2.6 is composed of the following stages: resampling/filtering, classification, shading and integration. The following sections present the three initial stages of the pipeline, as the integration stage was thoroughly described in sections 2.6.1 and 2.6.2.

2.10.1 Filtering

Generally speaking, filtering is in reality two separate stages: sampling and interpolation: first, a voxel is sampled from the volume, along the viewing ray for e.g. ray casting. This depends on the sampling distance, and also applies to the other methods: for example, in the texture-based methods, the sampling distance is the distance between slicing polygons. However, as the volume is discrete data, interpolation must be carried out to reconstruct a continuous signal: as already mentioned in section 2.6, trilinear interpolation is commonly used. For software-only solutions, a fast implementation was proposed by Hill [58], and in modern graphics hardware the process of sampling texture data and trilinearly interpolating it takes place as a single step.

Until very recently, higher order interpolation was impossible to achieve without a considerable performance loss. However high order interpolation became feasible as the graphics hardware improved: Hadwiger et al [52,53] describe a method to achieve bicubic and tricubic B-spline interpolation, using multiple textures and multiple rendering passes. Recently, Sigg and Hadwiger [147] presented a method to obtain cubic B-splines with a single rendering pass, using 1D textures to very efficiently encode the filtering kernel - their method is also applicable to gradient computation, hence improving shading quality. Nevertheless, our experience is that this still cannot be done at truly interactive rates (see figure 2.8 and also section 2.11 for a discussion), therefore for some applications trilinear interpolation still remains more suitable.



(a) (b)

Figure 2.8: Filtering approaches in 3D texture-based volume rendering, using the high contrast dataset (see section A.2), 512^2 display window: (a) shows trilinear interpolation (35 frames per second); (b) displays third order interpolation (10 frames per second, using algorithm from [147]).

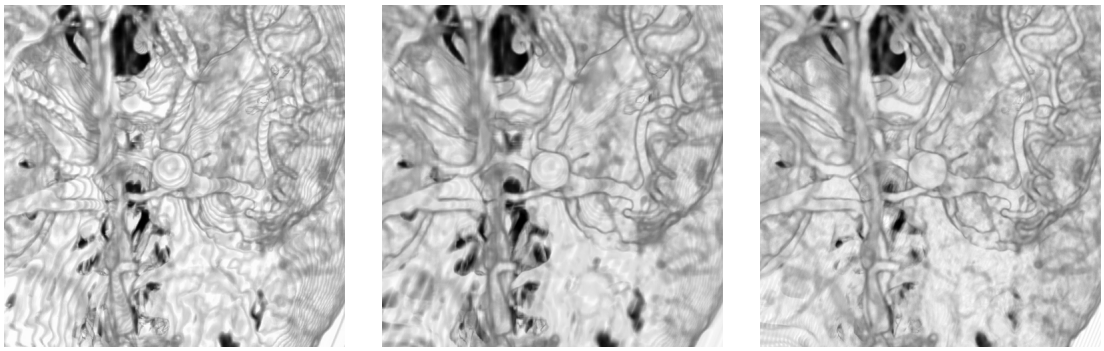
2.10.2 Classification

The reason we carry out classification is to identify features of interest in the data, and this process is commonly understood as the mapping of voxel values to emission and absorption coefficients. This process is generally specified by *colour and opacity transfer functions*, which map from each source scalar value to a colour and an opacity - we call this a *1D transfer function*. However, we may consider more information than just the scalar value: in fact, this was proposed at an early stage in Levoy's seminal work [95] on ray casting, where he suggested to use the magnitude of the gradient (see section 2.10.3) combined with the scalar value to define the domain of a *2D transfer function* - this can potentially help the transfer function to better distinguish boundaries in the data.

One may ask why we need a transfer function, when we could simply store colour and opacity, i.e. the optical properties, in the volume itself. Kniss [43] gives two good reasons: first, if we need to change these optical properties, we will have to do it for every voxel in the volume, which is time-consuming and prevents interactive updates; second, it has been shown [47] that doing classification before filtering (*pre-classification*) produces a smoothing effect, which in turn was found out to eliminate high frequencies (i.e. sharp peaks) from the transfer function [44, 43] thus producing visual artifacts. This is not desirable, hence the commonly accepted method is to carry out *post-classification*, i.e. doing classification after filtering the data. Finally, we can add a third reason to justify the use of transfer functions: volume data stored as RGBA (i.e. without a transfer function) requires roughly four times the memory of the same volume stored as density data plus a separate transfer function.

In 2001, Engel et al [44] introduced the concept of *pre-integrated classification*, i.e. splitting the numerical integration into two separate integrations - one for the scalar field and another for the transfer function. This was implemented with texture-based volume rendering in the following way: we define a *slab* as the space between two consecutive slicing polygons - if we now trace the viewing ray through the slab, the contribution of this ray segment will be equivalent to the integration of the colours and opacities mapped to the voxels at the front and at the back slices that made up this slab. Therefore, pre-integrated classification means that we pre-compute all possible combinations and store them in a 2D table, indexed by the scalar values at the front and at the back. In practice, this is stored as a separate 2D texture and accessed during rendering. Evidently, this is slower than the normal texture-based method as two samples per voxel will have to be obtained, in addition to another sample from the pre-integrated texture - nevertheless, Engel et al showed that fewer slices are required to obtain comparable quality images to normal texture methods (see figure 2.9 for a comparison). Finally, as we already mentioned

in section 2.6, pre-integration is not limited to texture-based methods: both Roettger et al [141] and Stegmaier et al [152] have also used it in their ray casting implementations.



(a) (b) (c)

Figure 2.9: Effect of pre-integration in visual quality and performance of the 3D texture-based method, 512^2 display window: (a) few slices, not pre-integrated (38 frames per second); (b) few slices, pre-integrated (16 frames per second); (c) many slices, not pre-integrated (7 frames per second).

The specification of a transfer function can be a complex procedure - this has led to some research on how to automate it: for example, Marks et al [106] proposed the idea of "design galleries", where the user is presented with an interface where he/she could select the most appropriate transfer function, according to a broad and varied set automatically generated by the system. Fully automatic generation is still a very active research field, but some effort focused on semi-automatic generation, such as the work of Kindlmann and Durkin [75], where they suggest that by detecting boundaries (using the first and second derivatives) between different features of the volume, the opacity of the transfer function could be increased at those points. Their method was later improved by Rezk-Salama et al [135], who proposed a system to allow automatic adjustment of transfer functions based on reference templates, i.e. transfer functions that have been previously created for a specific type of data - the algorithm then automatically adjusted this template to datasets of the same type, by non-linear distortion.

Instead of automating the generation of transfer functions, some research aimed instead to help the user interactively define it: Patten and Ma [123] proposed a graph-based interface for the visualization of transfer functions, where changes in parameters created new nodes and the resulting graph allowed the user to visualize clearly his/her most important modifications along the way. The recent work of Kniss et al [76, 77] focused on multidimensional transfer functions, and specifically on direct manipulation widgets that both provided information about the data and also allowed quick changes in the func-

tion. However, this requires significant training to be effectively used - and as Vidal et al suggested [160], the usability of multidimensional transfer functions is still an issue.

2.10.3 Shading

Shading is the process of applying lighting effects to a classified voxel. This is based on the use of a local illumination model, and normally the Phong lighting model [125] is employed, based on the combination of ambient, diffuse and specular terms. As the Phong lighting model is well known, we will not describe it in detail here but instead we will focus on its application for volumetric data.

The main issue in applying a lighting model is the determination of the surface normal, which depends on the shape of the surface: with polygonal data this is straightforward and a cross product between two vectors on the surface gives us the normal vector. However, in volume data we do not have a surface, hence an approximation is used as the normal: the *gradient vector*. It is defined as the first order derivative of a scalar field $f(x,y,z)$ [43] and gives us a measure of how quickly the scalar values changes:

$$\nabla f = (f_x, f_y, f_z) = \left(\frac{\delta}{\delta x}, \frac{\delta}{\delta y}, \frac{\delta}{\delta z} \right)$$

From the gradient vector, we can extract its scalar magnitude which gives the local variation of intensity:

$$|\nabla f| = \sqrt{{f_x}^2 + {f_y}^2 + {f_z}^2}$$

The magnitude is not normally used in the lighting computations, but it can be employed during the classification stage to help find boundaries between materials (as we mentioned in section 2.10.2).

There are many ways of computing the gradient - perhaps the simplest is the *central differences method*, which computes the directional derivatives as follows:

$$\begin{aligned} f_x(x,y,z) &= f(x+1,y,z) - f(x-1,y,z) \\ f_y(x,y,z) &= f(x,y+1,z) - f(x,y-1,z) \\ f_z(x,y,z) &= f(x,y,z+1) - f(x,y,z-1) \end{aligned}$$

Other gradient estimation methods were later developed, such as the Sobel operator [99], which uses the 26 neighbouring voxels and being considerably more computationally expensive, is correspondingly more accurate. Once the gradient has been computed, it is stored e.g. in a 3D texture to be used during the rendering procedure. As

pointed out by Engel [43], this has implications on the accuracy of the lighting computations, as the gradient must also be interpolated. With recent graphics hardware it became possible to compute gradients on-the-fly, i.e. during the actual rendering process [43], but this reduces rendering performance, as additional texture fetches will be required to compute e.g. the central differences. Figure 2.10 shows the visual results of volumetric shading when using 3D texture mapping - note that performance is much lower than the non-shaded version (figure 2.9a).

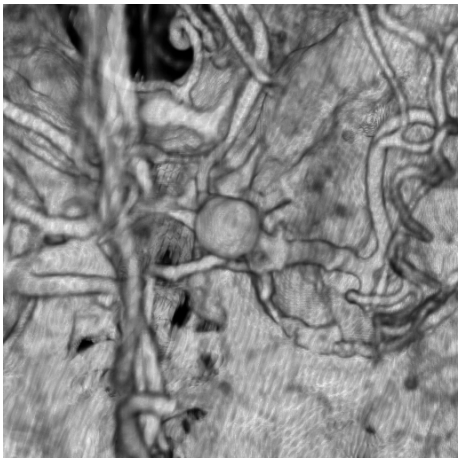


Figure 2.10: Visual effect of volumetric shading in 3D texture volume rendering, using an ambient, diffuse and specular lighting model (4 frames per second, using the implementation from [139]), 512^2 display window.

2.11 Choosing a Rendering Method

In order to decide which method to use as a basis for our implementation, two factors had to be taken into account:

- Image quality: ideally, we wanted to achieve the highest possible image quality that accurately represents the feature of interest, i.e. the aneurysm;
- Speed: as the end users of our system will be medical professionals, high rendering speed was crucial. Also any pre-processing should be avoided, as the users would like to visualize the dataset by simply loading it.

Considering the image quality factor alone, we have demonstrated that the best method is ray casting. However, even with GPU-based methods, frame rates still are much lower than texture-based volume rendering (see figure 2.2). Another option was to implement

pre-integrated classification, but this reduces the performance of the rendering procedure and our system aims to emphasize interactive volumetric effects. Nevertheless, we have conducted rendering experiments with implementations of both texture-based pre-integrated volume rendering (see figure 2.9) and for completeness, GPU-based ray casting (see figure 2.2) as well - although the latter only recently became available, as described in section 2.6.4.

Another issue was whether to use shading or not: although shading can increase the realism of a volumetric rendering, it adds complexity to the rendering process - gradients must be computed either before (increased pre-processing time) or during the actual rendering, and in any case the rendering performance will be slower than a non-shaded method (see figures 2.9 and 2.10 for a comparison). Considering the end user point of view, i.e. medical professionals, they frequently want to see the data as is - and shading can induce misleading interpretations, also increasing the complexity of the user interface (e.g. people normally do not care about positioning or orienting light sources).

Therefore, we concluded that the method which provided the overall best balance between quality and rendering speed is the 3D texture mapping, without pre-integration or shading. This is the basis for the distortion methods further described in chapter 5.

2.12 Summary

In this chapter we have reviewed the research and issues on volume visualization, focusing our attention on high quality, fast rendering approaches. We also mentioned advantages and disadvantages of each method, and at the end, provided a comparative analysis which helped us to define the base method for our implementation (which is described in chapter 6). In the next chapter of this thesis (chapter 3), we will review the research on focus and context, which is the key idea behind our work, and its relation and application to volume rendering.

Chapter 3

Focus and Context

3.1 Overview

In some applications there is too much data to be displayed at once on a computer display (or the display resolution may be insufficient for practical use), hence a simple and widely used solution is to apply a magnification factor to get a closer view of a specific region. But by doing so, it is equally easy to get lost in the dataset. This is generally called *loss of context*, because we are no longer able to visualize the entire dataset. When we zoom in, we are *focusing* on a certain feature that is of interest. In the field of information visualization this problem is called *focus and context* [149] and a number of successful solutions have emerged. The challenge is to find a way of looking at a high level of detail at this area of focus, without losing the overall context - in this chapter, we review the research on this problem, from the initial distortion ideas to later more comprehensive views. Finally, we demonstrate how these ideas moved from 2D to 3D and survey their application in volume visualization.

3.2 Distorted Presentation

According to Leung and Apperley's [94] taxonomy of presentation techniques for large graphical data spaces, it is possible to categorize techniques in two different ways: *non-distorted* and *distorted views*. Examples of non-distorted approaches include scrolling

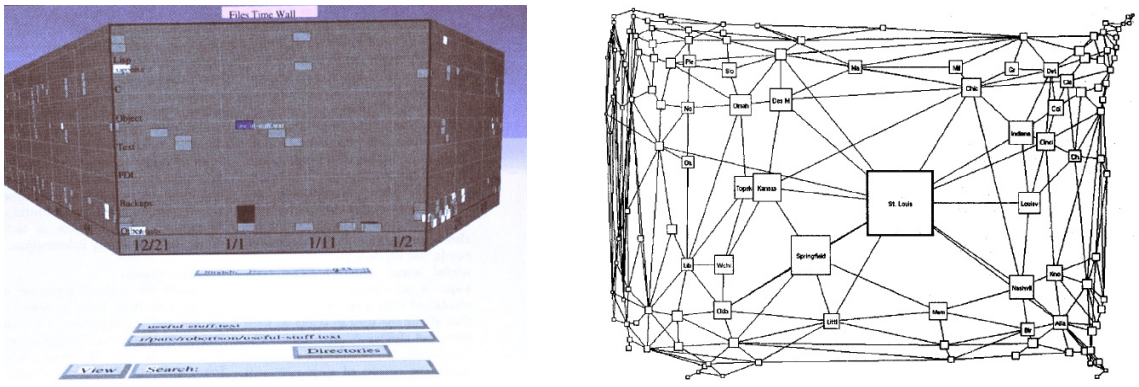
and zooming, and common hierarchical views, where part of the information is hidden. Carpendale et al [29] refers to these as *sufficient context approaches*, i.e. context filtered according to the interest of the viewer. For instance, the *DragMag* technique introduced by Ware and Lewis [164] displays an image where a magnified region is presented on top of it, hence some of the context is sacrificed. These and other non-distorted approaches can be adequate for small datasets, but once we have larger amounts of data they may not be the most suitable, as presenting the data in an effective way can be challenging. Because of this, we will focus our review on distorted presentation techniques.

3.2.1 Methods for Non-Hierarchical Data

One of the first ideas to solve the focus and context problem for non-hierarquical data is the use of distortion-oriented displays. One of the first ideas was proposed by Spence and Apperley: the *bifocal display* [150], where an information space was "folded" to allow simultaneous viewing of all data. This concept was implemented by Mackinlay et al as the *perspective wall* [104]: using a three-dimensional perspective effect (figure 3.1a), it was possible to display the entire information space, showing the region of interest on the front wall and the remaining regions on the left and right walls (distorted by the perspective projection). A two-dimensional extension of the idea was used earlier by Leung [93], to assist in the presentation of topological networks.

In a similar way, Furnas [46] introduced the concept of a *fisheye view*: by the suppression of irrelevant data (defined by a *degree of interest* function) it was possible to present a large number of structures - he demonstrated this with textual trees, C program code and a calendar. This has led to considerable further research: Sarkar and Brown [144] were the first to suggest the application of Furnas' text-based fisheye view in a graphical fashion (figure 3.1b), implementing the technique to effectively display large graphs. They later proposed a "rubber sheet" metaphor [145], where one could use handles to selectively enlarge regions of a 2D layout (e.g. a graph or a diagram) - this also allowed multiple regions of interest. Later, Rao and Card [131] applied the fisheye idea in their *table lens* visualization, but combined graphical (in the context regions) and textual (in the focus region) representations to allow more data to fit. Soon it was realized that the fisheye idea was suitable for different applications. For instance, Bederson [12] created the *fisheye menus*, where entries of a long menu were displayed with different font sizes, according to a fisheye view. More recently, Baudisch et al [10] proposed a web browser where the contents could be compressed vertically, again using the fisheye idea.

Although the fisheye display is based on a single focus point, earlier work was done in the field of cartography to use multiple focal points (*foci*) by Kadmon and Shlomi [66]: the *polyfocal* display. In fact, as Leung and Apperley [94] report, it was originally defined by a single focus but can be extended to multiple ones, which affects the computation time and comprehensibility of the resulting distortion. They describe the polyfocal display as based on a function which produces a high magnification at each focal point, surrounded by an area of progressively less magnification - the fisheye display being as a special case of the polyfocal projection. The notion of multiple foci was further extended by Carpendale et al [26]: they proposed a tool (*3D pliable surfaces*) where the user could push or pull multiple points on a 3D surface - these points affect the surface like control points in a parametric surface. If now we imagine this surface projected onto a plane, this produces a magnification effect which is dependent on how much each focal point was pushed or pulled. Later, Carpendale and Montagnesse [30] suggested a framework for unifying the presentation space: the *Elastic Presentation Framework* (EPF). This was based on 3D projective geometry and could replicate any distorted or non-distorted effect - they also introduced a lens library, which provided different magnification effects. More recently, Carpendale et al [29] proposed a method to achieve high levels of magnification (e.g. up to fifty times), using auxiliary functions to alter the behaviour of the magnification function - their approach also allowed lenses on top of other lenses, multiplying the effect.

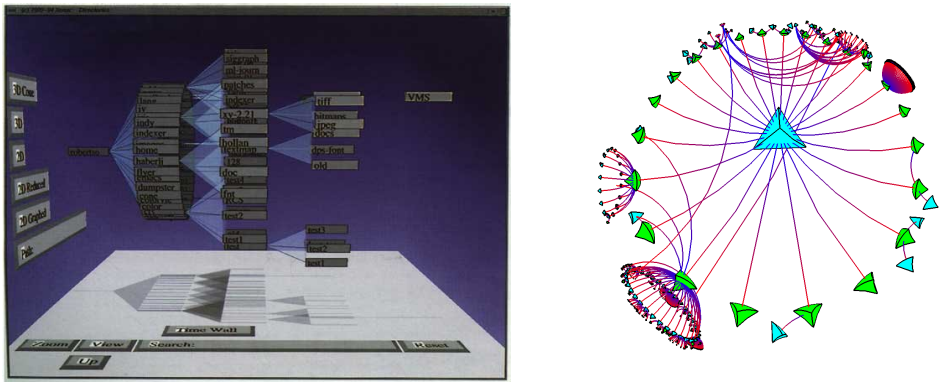


(a) (b)

Figure 3.1: Early focus and context methods based on distortion: (a) perspective wall (from [104]); (b) graphical fisheye views (from [144]).

3.2.2 Methods for Hierarchical Data

Hierarchical datasets such as large file trees usually require specific techniques to be effectively visualized. Such approaches include the work of Robertson et al [138], who suggested the idea of "cone trees" (figure 3.2a) to display large, hierarchical datasets: one of their examples was the visualization of a complex UNIX directory tree - they demonstrated that it could fit a screen if displayed with a 3D representation where the nodes of a subdirectory are positioned in 3D around their parent. They also showed that animation plays a fundamental role: for example, when the user selected a node, the tree smoothly rotated to display it - this allowed the user to better perceive the relationships between the nodes, whereas an abrupt change would make this much harder to understand. Later, Lamping and Rao [90] proposed a computationally simpler but highly effective technique: the *hyperbolic browser*. They positioned the nodes of the hierarchy on a hyperbolic plane and then mapped this plane onto a circular region on the display - this allows a smooth blending between focus and context, and to change the focus, one just needs to move the plane, so layout of the nodes is carried out just once. As Keahey later described [73], the key idea of the hyperbolic approach is the mapping from the infinite Euclidean space into a finite disk, where the space is bigger near the centre of the disk and smaller near the periphery. The basic 2D hyperbolic browser was extended later by Munzner and Burchard [117], who proposed a new method using 3D hyperbolic space and demonstrated this with visualization of the World Wide Web and file systems (figure 3.2b) - they also described it as having an effect similar to a fisheye view.

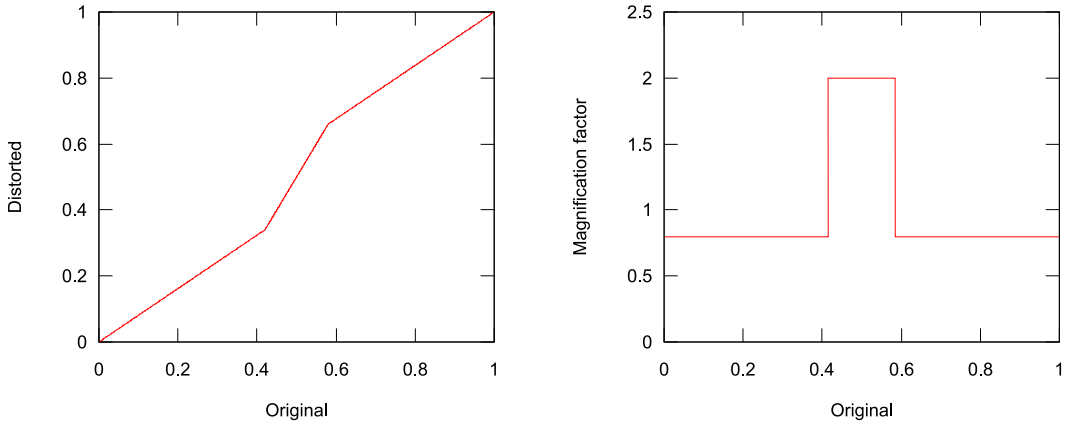


(a) (b)

Figure 3.2: Using focus and context to view hierarchical data: (a) cone trees (from [138]); (b) 3D hyperbolic file system browser (from [117]).

3.2.3 Classifying Distortion

In their seminal review, Leung and Apperley [94] classified distortions into *Cartesian* (where each coordinate of each data point is distorted based on the scalar difference between it and the corresponding coordinate of the centre of focus) and *polar* (where each point of the data is distorted based on the actual (vector) distance between it and the centre of focus). Distorted views are created through the application of a mathematical function to the original, undistorted data, the *transformation function.* From that we can obtain a corresponding *magnification function*, which is the derivative of the transformation function: the magnification function gives a profile of magnification or de-magnification factors throughout the data. For example, figure 3.3 presents the transformation (a) and magnification functions (b) of the bifocal display.



(a) (b)

Figure 3.3: Understanding the bifocal display through transformation or magnification functions: (a) transformation function - focus region is located in the centre, but effect is not readily understandable; (b) magnification function - the same focus region can be easily seen as a region with a higher magnification factor.

They also considered that each method can have either a *continuous* or a *piecewise continuous magnification function*. A continuous magnification function defines some degree of magnification at the centre of focus and decreases the magnification continuously with distance from the central point (such as a fisheye view). In a piecewise continuous magnification function, the function itself is made up of a finite number of continuous sections - in this case, the magnification function can also be classified as either *constant* or *varying*. A constant magnification function is one where the magnification does not change within a single section (e.g. bifocal display), whereas in a varying magnification

function, the magnification may change both within certain sections and between sections (e.g. perspective wall). Figure 3.4 presents some examples of continuous and piecewise continuous distortions and their effects on data presentation.

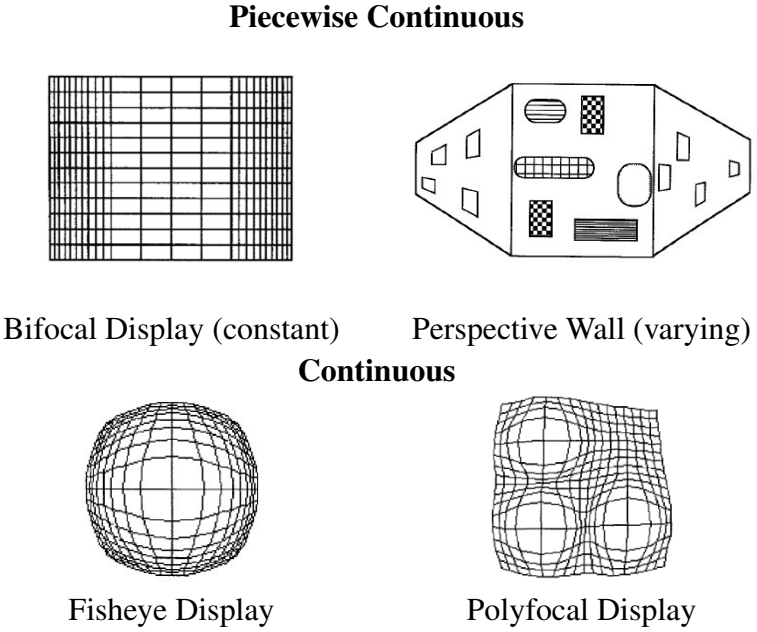
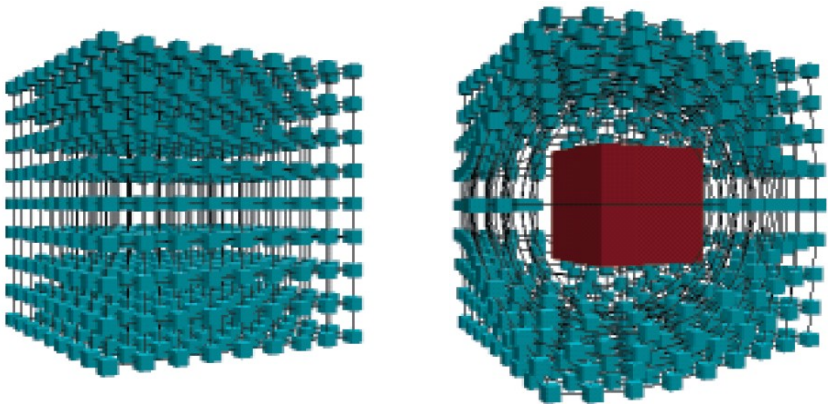


Figure 3.4: Classification of distortions according to the behaviour of the magnification function (from [94]).

3.2.4 Distortion in 3D Space

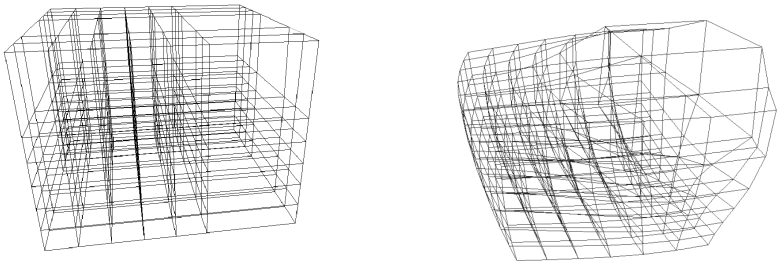
For 3D datasets, the foundations were laid by the research of Carpendale et al [25, 27] on the effects of distortion techniques on 3D data. Their work indicated that even though it is possible to obtain different views of the focus region (e.g. through rotation), there is an additional complication when viewing 3D data: occlusion of the focus region (figure 3.5a). Hence they proposed a distortion technique (*visual access distortion*) that allows a clear visual path to this region (figure 3.5b) - in fact, we shall see later in this thesis that we have achieved a similar result with our approach. Also important is their analysis on the comprehensibility of distortions [28], based on human perceptual skills: they proposed that visual cues play a fundamental role on our understanding of distorted presentations. For instance, a grid can be used to help us visualize both the focus region and the distortion effect; colour and shading can also be greatly effective to convey that kind of meaning. This was further evaluated and demonstrated by Zanella et al [172] through a practical study.



(a) (b)

Figure 3.5: The occlusion problem in 3D distortions: (a) original lattice; (b) Gaussian radial visual access distortion technique (from [27]).

Independently of Carpendale's research, Winch et al [171] presented the *3D Cartesian fisheye display* and the *3D polar fisheye display*, the latter of which extends the 2D counterpart proposed earlier by Sarkar and Brown [144]. Interestingly, their so-called Cartesian fisheye display is, in fact, a 3D extension of the bifocal display: it defines a 3D focus region (where magnification is constant), and the remaining regions exhibit magnification or de-magnification according to the position on each axis - in contrast to their polar display, which uses the radial distance from the centre of magnification to the point under consideration to determine the magnification factor. However, they presented results just as proof of concept and did not experiment with large scale data or real applications. Figure 3.6 presents both approaches using a 12 x 12 x 12 dataset.



(a) (b)

Figure 3.6: 3D fisheye distortions proposed by Winch: (a) cartesian; (b) polar (from [171]).

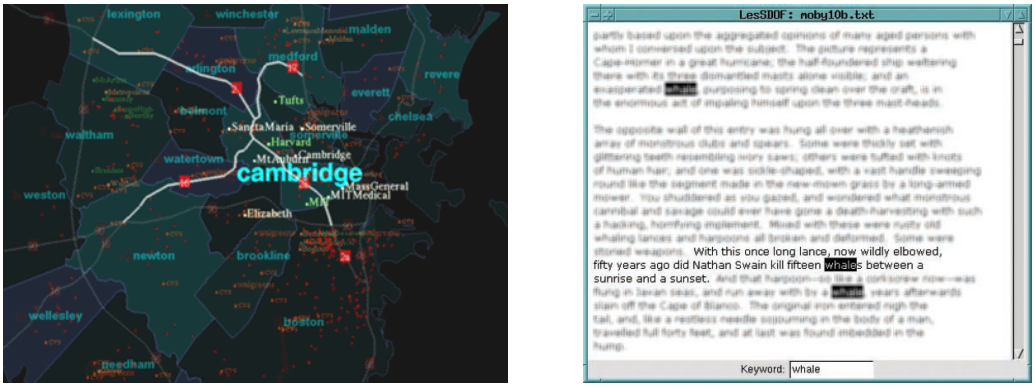
3.3 Beyond Distortion

Distortion is possibly the most common approach to focus and context, but research has shown that other visual cues can be used as well. For example, in the *GeoSpace* system proposed by Lokuge and Ishizaki [101], complex geographical data is effectively viewed through the careful use of colour and opacity to highlight selected features and fade out unnecessary context (figure 3.7a). Another interesting concept is the *magic lenses* introduced by Bier et al [15]: these allow regions within a dataset to be shown with a different visual representation (e.g. wireframe instead of solid, figure 3.7c), which can be useful to both navigate through the data and achieve better understanding of it. A significant advantage is that the lenses do not require additional screen space. More recently, Kosara et al [80, 81] suggested the idea of *semantic depth of field*: depth of field is defined as the zone given by the minimum and maximum distance from a lens where objects will be in perfect focus [91]. It normally depends on the focal length and aperture of the lens: a wide aperture produces a shallow depth of field whilst a narrow aperture produces large depth of field - also due to optical properties, longer focal lengths give correspondingly less depth of field. Hence the idea of semantic depth of field is to apply this blurring effect, but according to the importance or relevance of each object (figure 3.7b). It was later evaluated through some user studies [82], which suggested that it is a very effective way for guiding the viewer's attention.

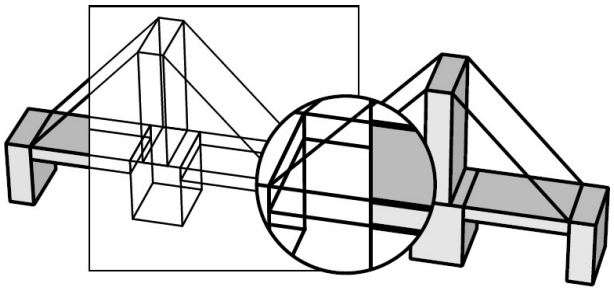
In our taxonomy for focus and context for volume visualization (first published in [31]) - see section 5.2 for the full discussion - we propose the idea of refinement of the focus region as an accuracy attribute to achieve a focus and context effect. We develop these ideas further in the full framework presented in section 5.3 - this exploits the ideas of highlighting through colour and facilitates viewing of the focus region through modulation of opacity. More recently, Hauser [55] suggested a generalization of focus and context visualization. He introduced the idea of *graphic resources*, which can be unevenly used to achieve different effects: space (i.e. distortion), opacity/colour (e.g. *GeoSpace*), frequency and style. The frequency resource is essentially mapped to the idea of semantic depth of field, whereas style means displaying the focus and context with different visual styles, as in e.g. magic lenses.

3.4 Applications in Volume Rendering

In this section we highlight research that combines the idea of focus and context with volume visualization. This is not a recent idea: back in 1990, Levoy and Whitaker [98]



(a) (b)



(c)

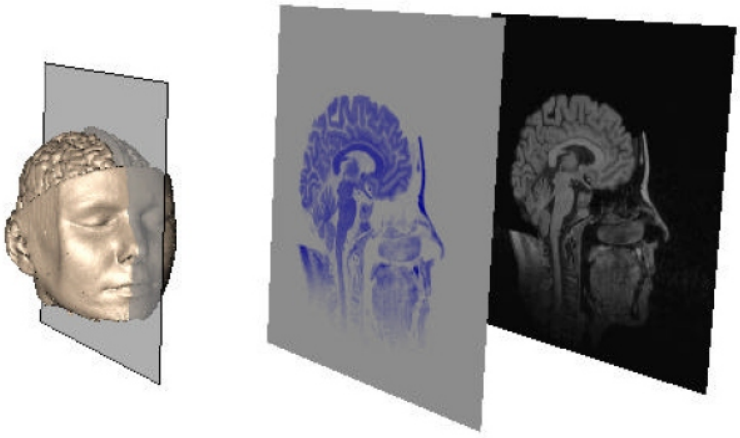
Figure 3.7: Achieving focus and context effects with techniques other than distortion: (a) using colour and opacity changes to highlight a feature of interest (from [101]); (b) semantic depth of field applied to a text viewer (from [81]); (c) magic lenses showing different visual representations (wireframe and 2D magnifier, from [15]).

proposed a ray casting gaze-directed technique, where the quality of the rendering varied according to the user's gaze (using a head mounted display and a position/orientation tracker). This was based on the fact that the spatial acuity of the human eye is highest in the fovea (a region in the centre of the retina) and progressively reduces toward the periphery of the visual field - they exploited this by adjusting the sampling distance for each ray cast, thus producing less quality in regions that fell outside the foveal area.

Concerning the slicing technique, Van der Heyden et al [159] proposed an interesting system that allowed the visualization of a large number of MRI slices at once, by selectively adjusting the scale of each slice according to its importance and using layout ideas initially developed by Carpendale et al [26]. This was extended later by Kuederle et al [85], who evaluated and compared a detail-in-context with a thumbnail display - they found out that both approaches were effective, depending on the user needs and ap-

plication. Some research combined slice views with volume representations: examples include *ExoVis* by Tory and Swindels [155], who display a selected slice as a planar "wall" projected away from the volume - hence minimizing the need for separate planar views. Figure 3.8 presents some of these approaches.

(a)



(b)

Figure 3.8: Slicing-based focus and context effects: (a) detail-in-context display of medical images (from [85]); (b) planar views as projections in the *ExoVis* system (from [155]).

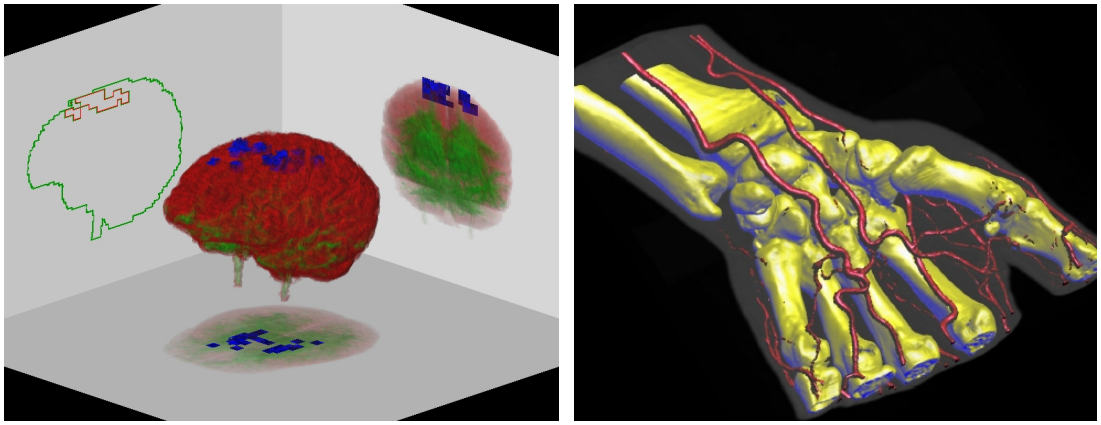
The 3D texture mapping algorithm is very effective, but needs a lot of texture memory. This is a lesser problem now, as graphics cards are increasingly getting better. But citing medical technology as an example case, resolution of the imaging devices (scanners) is constantly improving, so more texture memory will likely be required to render a 3D dataset. Thus some research has been done on optimizing texture utilization or combining multiple textures - some of these are directly related to focus and context ideas. An

example is the work of LaMar et al [88]: they introduced a multiresolution approach to volume rendering, based on a hierarchical texture organized with octrees - in their work, the region of interest is rendered with a higher resolution texture whereas the regions away from that region are rendered with progressively lower resolutions. However, their algorithm did not guarantee that the interpolation between resolution levels was correct. This was improved later by Weiler et al [165]: their algorithm was designed to minimize rendering artifacts due to interpolation and also used a hierarchical data structure. These two algorithms also deal quite effectively with the texture memory problem, but at the expense of losing image detail outside their focus area (figure 3.9).

Figure 3.9: A multiresolution approach to volume rendering: left image shows dataset in full resolution; centre image exhibits lower resolution detail in the back; right image uses four levels of detail, greatly reducing the use of texture memory (from [165]).

Combining different visual representations is also a common way of achieving focus and context effects: for example, König and Gröller [79] proposed a system where additional views of the same volume were presented in "mirrors" (i.e. planes) positioned orthogonally in relation to the volume - but whilst the volume was displayed with a direct volume rendering technique, for instance, the mirrors could display MIP or contours, thus providing extra information. Later Hauser et al [56] presented a system where one could segment a dataset and associate different representations with each part - they showed that texture-based volume rendering could be combined with e.g. MIP or surface rendering. This was achieved by defining a rendering procedure with two compositing levels: local (objects with the same kind of visual representation) and global (compositing amongst all representations). These two ideas as presented in figure 3.10.

Focus and context effects can also be obtained by modulating the opacity of each voxel, according to some factor: an example is the work of Viola et al [161], who introduced the idea of importance-driven volume rendering, i.e. associating an importance

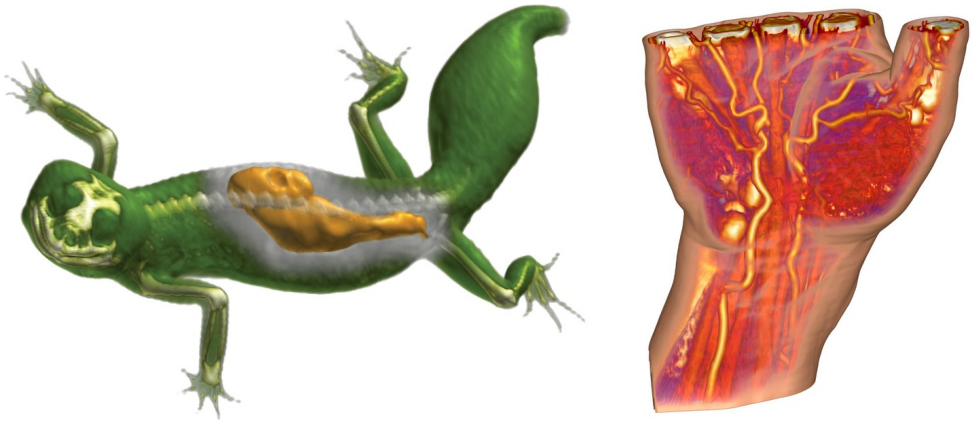


(a) (b)

Figure 3.10: Combining multiple visual representations to achieve focus and context effects: (a) "magic mirrors": the left wall exhibits a contour display and the right and bottom walls use different transfer functions (from [79]); (b) two-level volume rendering: vessels are rendered with isosurfacing, bones with tone shading and the skin with MIP (from [56]).

value to segmented parts of the dataset, allowing the rendering process to remove or suppress those parts of lesser importance - their method was based on a ray casting software renderer. However, this relies on previous segmentation and manually associating the importance values. An automated approach was recently proposed by Bruckner et al [18]: in their method, the opacity of each voxel is attenuated by a model that combines the gradient magnitude with the actual shading intensity resulting from lighting computations, amongst other variables. The advantages of their approach are that no focus region needs to be set, and the user just needs to classify the different materials using colour, leaving the opacity to be computed by their model. Figure 3.11 shows the results of these two methods.

Some of the more recent research has been directed towards the idea of focus and context through volumetric deformations or distortions (figure 3.12): LaMar et al [89] were the first to introduce a texture-based volume lens to explore volumetric data, allowing the enlargement of a specific region inside the volume. We have done previous experiments with volume distortion [31], by dividing a volume into smaller regions and scaling those, adapting Winch's [171] idea of a 3D Cartesian bifocal distortion (for details, see section 6.2). This was implemented purely as a change in a traditional texture-based volume renderer, in such a way that it considered the presence of a focus region to correctly split the original data. As such, the method did not offer enough flexibility or speed to be



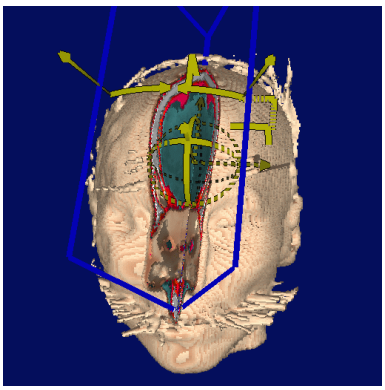
(a) (b)

Figure 3.11: Focus and context by the modulation of opacity: (a) importance-driven volume rendering, where an internal organ (focus) shows through other tissues (from [161]); (b) illustrative context-preserving volume rendering: vessels and other structures are visible through the skin (from [18]).

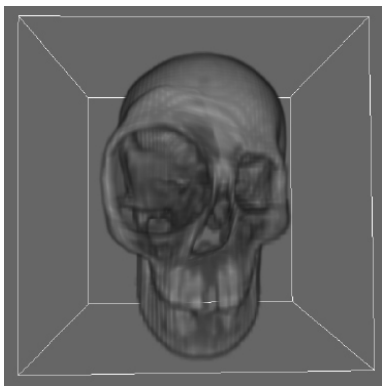
used in practice, being limited to small datasets and bifocal distortion only. In a rather different approach, McGuffin et al [110] developed a distortion method based on surgical metaphors such as cutting, peeling and spreading. Implementing these metaphors through direct manipulation techniques in 3D and rendering point primitives, his method achieved interactive speeds. Later, Wang et al [162] proposed the idea of a "magic lens" for volume visualization, where they used ray casting through the GPU to implement a number of volumetric effects based on optical modelling of a lens - this also exploited the idea of importance-driven volume rendering. However, their method did not provide interactive performance with large datasets (although the authors mentioned that further optimization is possible). Very recently, Brunet et al [20] developed a distortion method based on 3D offset textures (i.e. textures that store offsets to other texture coordinates - as suggested in [43]), which allowed e.g. the implementation of McGuffin's surgical metaphors at interactive speeds.

3.5 Summary

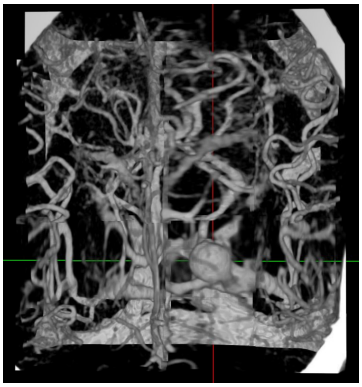
In this chapter we have reviewed the ideas behind focus and context and also related them to our application field, volume visualization. In the next chapter, the last in part 1, we shall review the evolution and programmability of the graphics processor - a fundamental aspect for the actual implementation of this research.



(a)

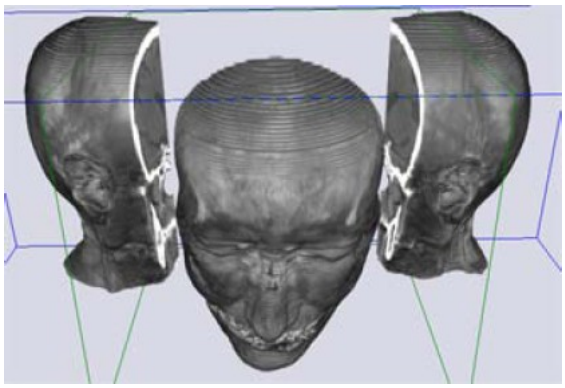


(b)



(c)

(d)



(e)

Figure 3.12: Focus and context effects through volumetric distortion: (a) hinge spreader tool, which pushes voxels to the side (from [110]); (b) circular volumetric lens (from [89]); (c) 3D bifocal distortion from our initial research in [31]; (d) "magic lens" effect of feature-based magnification (from [162]); (e) a leafer tool implemented with offset textures (from [20]).

Chapter 4

Hardware Developments

4.1 Overview

THIS CHAPTER PRESENTS a review of the graphics pipeline, a brief history of the graphics processing unit (GPU) and how modern graphics hardware can be programmed - this is exploited later in this thesis, as we use hardware-accelerated methods to achieve focus and context and other novel effects (see chapter 6 for details). We also present a brief review of a new research field: the use of GPUs for general purpose computation.

4.2 The 3D Rendering Pipeline

In this section we review the rendering pipeline for 3D graphics - biased towards OpenGL, as it is the industry standard and is also the base of our implementation. In general, the rendering pipeline can be seen as a sequence of stages (or transformations), that convert from a 3D model (defined in *model coordinates*) to a 2D image on the screen [37]:

- Modelling Transformation: this stage applies geometric transformations such as translation, rotation and scaling to the *model coordinates*, in order to obtain *world coordinates*;
- Viewing Transformation: using the viewing parameters (viewer position and orientation), the *world coordinates* are converted to *view coordinates*;

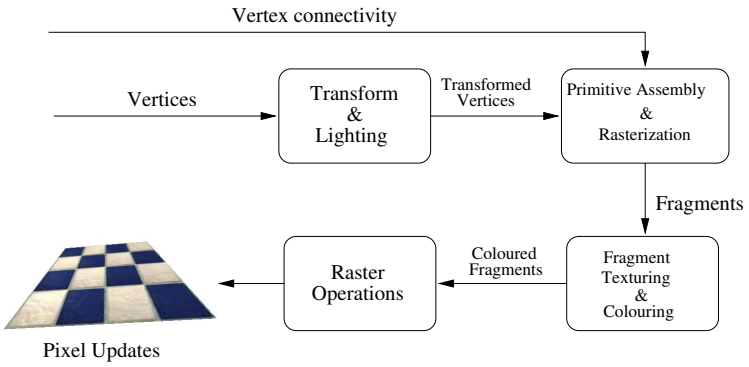


Figure 4.1: OpenGL fixed functionality pipeline.

- Projection Transformation: this stage applies the desired projection (e.g. orthographic or perspective) to the view coordinates and produces *projection coordinates*, i.e. 2D coordinates on the projection plane;
- Normalization Transformation and Clipping: at this stage, the projection coordinates are usually *normalized* for efficiency purposes and then *clipped* against the viewing volume (alternatively, the 3D clipping can be done in the viewing transformation stage);
- Viewport Transformation: this stage converts from the *normalized*, *clipped coordinates* to the *viewport coordinates* (i.e. pixel positions) on screen;
- Scan conversion (or rasterization): now the 2D positions (vertices) are converted to raster form, to draw and/or fill all the necessary pixel locations and create the final shape.

This conceptual pipeline does not take into consideration additional stages such as texturing or lighting - these are normally dependent on implementation, but lighting is usually computed after the viewing transformation and texturing normally takes place after rasterization. For example, OpenGL defines a *fixed functionality pipeline* as figure 4.1 shows.

This pipeline is called “fixed”, as there are no programmable components. All elements of the conceptual pipeline are present, but they may appear in a different form or be grouped together:

- Transform and lighting: this corresponds to the modelling and viewing transformation stages, along with lighting calculations - it receives the original vertices and outputs transformed vertices (in view coordinates). In fact, when we refer to the

term "vertex" in this context, we mean all the associated data such as the actual coordinates, normal vector, texture coordinates and colour;

- Primitive assembly and rasterization: this essentially covers the remaining stages of the conceptual pipeline. Using vertex connectivity information (i.e. the type of graphics primitive being drawn), the transformed vertices are first assembled into the desired primitives - this is necessary, as the clipping algorithm is different for e.g. points, lines or polygons. Next the vertices are clipped to the viewing volume and projection takes place[1], optionally followed by culling (if enabled). Rasterization happens last, but in OpenGL it produces *fragments* - smaller units that contain all information needed to draw a pixel on screen, such as colour, texture coordinates and depth values;

- Fragment texturing and colouring: in this stage, texture mapping is applied (if enabled) to each fragment produced by the rasterization. Also other operations take place, such as fog or colour sum (combining a fragment primary colour with its secondary colour);

- Raster operations: these are the operations that can reject or modify the fragment data before it is sent to the screen, such as depth/stencil/alpha tests or alpha blending.

In the next section, we revisit briefly the history of the GPU, in order to better understand how this fixed pipeline has been improved over the years.

4.3 The Graphics Processing Unit

Before 1995, some pioneering work in workstation-class machines introduced the idea of hardware-assisted rendering, with capabilities such as texture-mapped polygonal shading, line anti-aliasing [6], accumulation buffer [54], linear texture filtering and 3D textures, stereo graphics in a window and full screen anti-aliasing [5]. At that time, the graphics hardware in a consumer PC was called a "graphics accelerator" and it was basically limited to the drawing of 2D shapes: there was no texture mapping in hardware and the practically the entire visualization pipeline (except rasterization) was done by the CPU. Between 1995 and 1998 the first hardware capable of true texture mapping and z-buffering

[1] In OpenGL, this is a two-step procedure, consisting of projection and perspective division, the latter which divides the x, y and z coordinates by the homogeneous coordinate w.

started to appear, but some computationally intensive stages of the pipeline (such as modelling/viewing transformations and lighting calculations) were still performed by the CPU (figure 4.2).

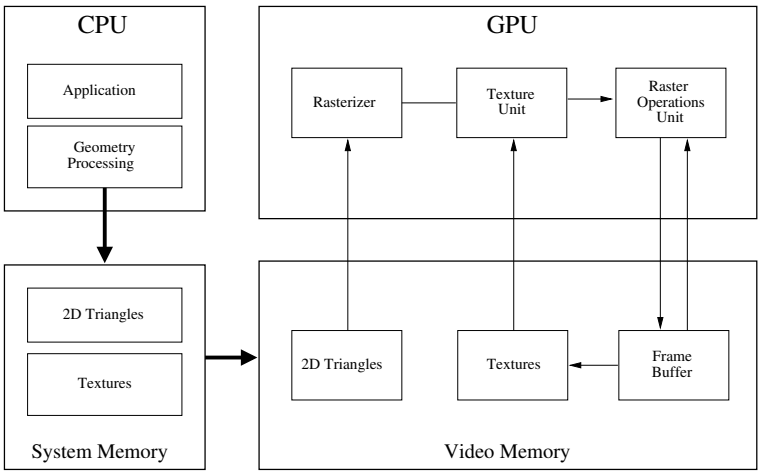


Figure 4.2: Typical PC graphics architecture between 1995-1998 (from [146]).

The first consumer-level graphics processing units (GPUs) came out between 1999 and 2000: these were now able to carry out 3D computations and thus implement viewing transformations, lighting and other stages of the pipeline (figure 4.3). Also the hardware could support more than one texture unit (i.e. multitexturing), and a rudimentary form of programming was present as units called *register combiners*: these allowed the user to chain the outputs of the texture units, combining them according to some operation such as adding, scaling, or discarding colour values and performing dependent texture reads, i.e. the output of one texture unit is used as the input (texture coordinates) for the next unit.

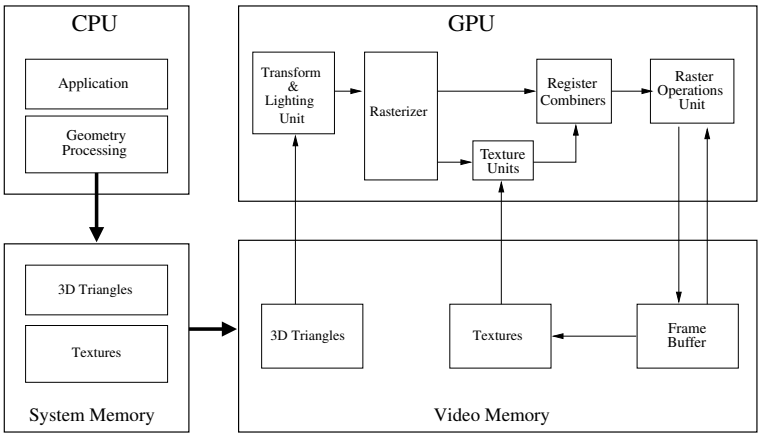


Figure 4.3: GPU architecture between 1999-2000 (from [146]).

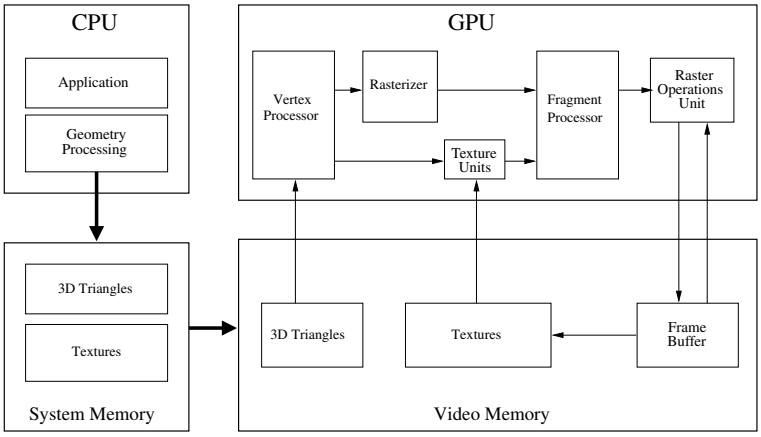


Figure 4.4: GPU architecture in 2004 (from [146]).

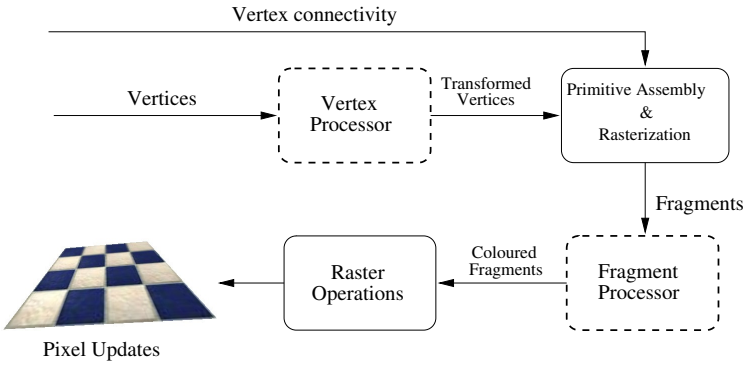


Figure 4.5: OpenGL programmable pipeline.

For the consumer market, most advances in graphics hardware were driven by the entertainment industry: the increasing demands of game publishers and users. Therefore from 2001, the hardware became increasingly programmable, and flexible units started to replace certain stages of the fixed functionality pipeline. At first, the transformation and lighting stage (see figure 4.1) was replaced by a *vertex processor* [100]. This is a programmable unit that operates on the original vertex data (for details, see 4.4.1). By 2004 (figure 4.4), a second specialized unit became available: the *fragment processor*, which is capable of carrying out diverse operations per fragment (for details, see 4.4.1). Consequently, the fixed functionality pipeline was replaced by a *programmable pipeline*, as shown in figure 4.5.

4.4 Programming the GPU

As the level of programmability of the graphics hardware has increased over the years, the programming task has evolved as well. For example, the architecture based on register

combiners offered limited programmability, based on specific OpenGL extensions in the form of function calls to setup parameters. When the true programmable units started to appear, a new level of programmability was needed, so a specialized assembly language was created. However, assembly code is hard to learn, write and maintain - hence a high-level language was a necessity. The use of high-level languages for rendering was not new: years before programmable hardware was available, *RenderMan* [158] allowed the precise description of lighting, surface features and procedural texturing - even today, it is still widely used for photorealistic, non-realtime graphics. Later in 1998, Olano et al [122] created *PixelFlow* (a SIMD graphics computer), the first system capable of using a shading language in realtime. Then in 2001, Proudfoot et al [128] presented the first shading language integrated with a compiler, capable of converting from a high-level source to machine code in a variety of graphics hardware - this eventually evolved into NVIDIA's *Cg* language [105, 45]. Although *Cg* is a very capable language and still in widespread use at the moment of writing, it requires the installation of a SDK containing a compiler and other tools, and it is also a proprietary language, developed and maintained by a hardware vendor. Because of this, an open standard similar to OpenGL was desirable: from an earlier white paper by 3Dlabs [2], development then started of what later became known as *OpenGL Shading Language* (*GLSL*).

Language syntax in *Cg* and *GLSL* is very similar, however in *Cg* the compiler reads high-level source code and produces an equivalent assembly (which must be then converted to machine code), while in *GLSL* the compiler produces machine code directly from the high-level source code. Another advantage of *GLSL* is that the compiler has been integrated into the OpenGL 2.0 API, so no additional SDK needs to be installed to compile or run *GLSL*-based code. Having said that, this can also be seen as a disadvantage: by hiding the resulting assembly from the end user, *GLSL* does not allow any kind of further optimization, whereas if using Cg one can review the assembly source and modify it if necessary. In the end, both the integration into the core OpenGL API and vendor independence of *GLSL* has led us to choose it as the language of choice for development during this research.

4.4.1 OpenGL Shading Language

As seen in section 4.3, the graphics hardware currently has two specialized processing units that can be programmed by the user: vertex and fragment processors. Hence we define code that runs on these units as *vertex shaders* and *fragment shaders*, respectively. Vertex shaders are pieces of code that substitute the normal transformation and lighting

operations for each vertex supplied by the user's program - if a vertex shader replaces some of these operations, it has to implement all the remaining ones. For example, supposing that the user wants to change the vertex colour according to some function, he/she will have to perform the lighting computations as well, as this is done in the same stage of the fixed pipeline.

In a similar way, fragment shaders are code that replace the fixed functionality which affected fragment values - recalling that each fragment is composed of a screen coordinate, a depth value and other attributes such as texture coordinates and colour. These values are generally obtained through linear interpolation of the parameters set for the primitive vertices, during rasterization (however, fragment shaders cannot change the final screen coordinate - this is fixed by the rasterization stage). A fragment shader can, for example, be used to compute per pixel (Phong) lighting. But a major application of fragment shaders is to access and manipulate texture data in many different ways.

For a practical example, let us consider a non-photorealistic shader that creates a cartoon-like effect (adapted from [142]): the objective here is just to demonstrate how a vertex shader and a fragment shader can be used together. In this effect, instead of applying a full lighting model we associate ranges of light intensity to specific colours. Normally we use a small number of colours to produce the best effect, and in this case, we will store them in a texture that will be read in the fragment shading stage. First, listing 4.1 presents the vertex shader:

Listing 4.1: Vertex shader to achieve a cartoon-like effect.

```
varying vec3 normal, lightDir;
void main()
{
  lightDir = normalize(vec3(gl_LightSource[0].position));
  normal = gl_NormalMatrix * gl_Normal;

  gl_Position = ftransform();
}
```

This shader declares two *varying* variables: *normal* and *lightDir*. These are variables that later will be interpolated by the rasterization stage - in this case, the normal vector and the light direction. The latter is computed first, by simply normalizing the first OpenGL light source position (*gl_LightSource[0].position*) - we assume that this is a directional light. The normal vector in viewing coordinates is computed next, by multiplying the normal matrix (a matrix provided by GLSL which corresponds to the inverse transpose of the *modelview* matrix [142]) by the normal vector associated to this vertex. Finally, the transformed vertex position is computed, through the *ftransform* function - this simply

multiplies the original vertex by the *modelview* (which encodes the modelling and viewing transformations in OpenGL) and projection matrices, producing the same result as the fixed functionality.

At this point, we must remember that the output of this stage is sent to the primitive assembly and rasterization stage. Only then fragments will be created and fed to the fragment processor - and then the fragment shader shown in listing 4.2 will be run:

Listing 4.2: Fragment shader to achieve a cartoon-like effect.

```
varying vec3 normal, lightDir;
uniform sampler1D colours;

void main()
{
  vec3 n = normalize(normal);
  float intensity = max(dot(lightDir,n),0.0);

  gl_FragColor = texture1D(colours, intensity);
}
```

Note that we declare again the same *varying* variables: this is a requirement of the compiler, so it will know how to link user-defined vertex shader outputs to the fragment shader inputs. We also declare a *uniform sampler1D variable colours*: *uniform* variables denote values that do not change while defining the vertices for a primitive, e.g. between *glBegin* and *glEnd* calls; a *sampler1D* represents a texture unit, accessed through 1D coordinates - we assume that this texture unit contains the texture where each texel is one of the different colours that our cartoon shader uses. The first step is to normalize the normal vector: this is necessary, as the rasterization does not guarantee that the interpolated value will be already normalized. Then we compute the light intensity of this fragment, using the dot product between the light direction and the normal vector. Finally, we access the 1D texture using the computed intensity as texture coordinate and store that as the resulting colour for this fragment (*gl_FragColor*). Therefore if we are using a texture with four elements, the visual result would be similar to figure 4.6.

Figure 4.6: Visual result of a cartoon-like shader.

4.4.2 General Purpose Computation on Graphics Hardware

This is a new field of research, which has emerged from the realization that a GPU is essentially a massively parallel machine. Each fragment can be seen as an independent processing unit, and the combined processing power of thousands of fragments is much higher than current CPUs - Buck and Purcell [23] reported that a GPU could achieve a performance roughly equivalent to a 10 GHz Pentium processor. They also comment that another important factor is the much higher bandwidth of a GPU compared to a CPU (e.g. 25.3 Gb/s in the GeForce FX 5900 Ultra card compared to 5.96 Gb/s in the Pentium 4).

Considerable research has already been done in this field. For example, Hoff et al [74] showed how a graphics processor could be helpful to compute Voronoi diagrams. Later, McCool et al [108] developed a language for shader meta-programming, which could be integrated in a C++ program and run transparently with or without specialized hardware - they demonstrated that the Julia set fractal could be computed very efficiently. Later, Kruger et al [84] showed that a GPU could also be used to solve linear equations and proposed direct solvers for sparse matrices. A recent example is the work of Buck et al [22], who proposed a language designed for stream computing, i.e. using the GPU as a streaming processor - they demonstrated with image segmentation and ray tracing, achieving a speed up of roughly seven times faster than comparable CPU implementations.

4.5 Summary

This chapter has presented the main aspects related to the programming of a modern graphics processor - in particular, the growing use of shading languages as efficient and reliable programming tools. This closes part I of this document - in part II, we shall describe the actual research on distortion methods that were integrated into the development of the *VolFocus* system.

Part II

Framework

Chapter 5

A Framework for Focus and Context in Volume Visualization

5.1 Overview

THIS CHAPTER DESCRIBES the research that has been carried out to explore the transfer of focus and context ideas from information visualization to volume visualization, towards the creation of a generic framework for focus+context applied to volume visualization. In section 5.2 we present a taxonomy of focus and context methods for volume visualization, in terms of nature of focus region, type of focus effect and space in which the effect is applied. Then section 5.3 describes the framework derived from this initial research.

5.2 A Taxonomy

Most information visualization techniques create an abstract representation of the original data. For example, a user's file system can be represented as icons on a flat wall. To help comprehension, the representation can be distorted in order to make better use of screen space. In the wall example, we can bend the two ends of the wall backwards and view in perspective (the so-called perspective wall).

By contrast, volume visualization typically creates a representation which reflects a real spatial relationship amongst the data. We can apply distortion techniques but we need to be aware that we are changing those spatial relationships. Therefore this is one of the research goals: to find out how big an issue this is. Winch [171] has already proposed the extension of the distortion ideas to 3D, but he did not experiment with volume data. Nevertheless, his findings show that there are potentially a number of advantages to the approach when dealing with large scale 3D data.

In chapter 3, we have explained how the focus and context (F+C) concept in information visualization allows a user to study fine detail in a restricted region of a large dataset, without losing the context of the overall data space. In this section, we aim to develop a taxonomy for F+C in volume visualization, building a classification of existing techniques and identifying possible new approaches. We shall concentrate on direct volume rendering and we shall classify an approach in terms of three attributes:

- **Focus**: this defines the nature of the focus region, which in 2D can either be a point or a circular or rectangular region. If the Visualization Space is 3D, then we see the focus as a point, a spherical or a cuboid volume. In the case of a point, the focus effect is greatest at a single point, and falls off with distance towards the boundary of the visualization space. In the case of a region/volume, the focus effect is constant within the specified region/volume, and then falls off with distance towards the boundary.

- **Effect**: this defines the way in which the detail of the focus is achieved, and can be by refinement or by distortion, or by a combination of both. Refinement is an accuracy attribute, and indicates that the rendering within the focus region is carried out to higher accuracy (but there is no spatial distortion). Distortion on the other hand, achieves the focus effect by spatial deformation of the Visualization Space – the focus region is magnified. It is possible to combine both, where the greater spatial extent of the focus region is accompanied by higher accuracy rendering in that region.

- **Visualization Space**: this defines the space in which the F+C operation is applied, and can be either image space or object space, or indeed both. By object space, we are thinking in terms of the dataset being classified and shaded as though it were a block of inhomogeneous coloured gel – this gel is our object which is rendered.

This taxonomy therefore yields twenty-seven (3 x 3 x 3) different approaches, based on the values of these three attributes. Some approaches are well known, others are relatively

new. We look at each in turn, as an instance of the triple {Focus, Effect, Visualization Space}:

1. { Point, Distortion, Image Space}: this is the well-known fisheye view of Furnas [46], and later Sarkar and Brown [144], applied to the image created by a volume rendering algorithm. The distortion is a simple magnification process, with pixels being replicated (or removed) from the original image. This can be applied as a post-processing step on the original image.

2. {Rectangular Region, Distortion, Image Space}: this is the bifocal display approach of Spence and Apperley [150], with similar effect to the above.

3. {Circular Region, Distortion, Image Space}: this is a variant of the Spence and Apperley bifocal display.

   Having made this distinction between point and region/volume, for simplicity we now treat them together in what follows, considering point as limiting case of circular/spherical region/volume (see table 5.1 for a summary):

4. {Point/Region, Refinement, Image Space}: here the accuracy of each pixel in the image is related to its distance from the focus region. This approach has its origins in the flight simulation area, where high resolution insets were blended in to a low resolution background. We can do exactly the same with volume rendering, simply by generating inset images at higher resolution. This is also related to Levoy's "Volume Rendering by Adaptive Refinement" paper [97] where areas of greater image complexity are refined by casting more rays.

5. {Point/Region, Distortion and Refinement, Image Space}: this combines approaches 1/2/3 and 4. As well as carrying out the spatial distortion, we also increase the relative accuracy within the focus region, either by increased sampling or by interpolation from existing samples. For example, our implementation of texture-based methods in 2D (see section 6.3) falls into this category.

6. {Point/Region, Distortion, Object Space}: in this approach, we apply a 3D spatial distortion to the dataset. Within a 3D focus region, the object (that is, the portion of gel within the focus region) is magnified by a certain factor (greater than 1), and the remainder is magnified with decreasing factors as we approach the boundary of the gel. A variation of this approach was presented by LaMar et al in their

magnification lens paper [89]. This is also explored further in this thesis, for both point and region.

7. {Point/Region, Refinement, Object Space}: in this approach, we model the object, i.e. gel, at higher accuracy within the focus region, before the gel is rendered. An example of this method is LaMar's work on multiresolution volume visualization [88].

8. {Point/Region, Both, Object Space}: this combines 6 and 7 above. The texture-based methods of chapter 6 fall into this category, as we apply a distortion to the texture and the resulting resampling that is carried out by the GPU hardware uses trilinear interpolation.

9. {Point/Region, Distortion, Both}: this has not been studied, but we could envisage the image distortion being applied as a post-processing step.

10. {Point/Region, Refinement, Both}: this is basically the idea in Levoy's gaze directed volume rendering [98]. Refinement in image space is achieved by higher density of rays cast, and refinement in object space is achieved by higher sampling along the ray. Note that the focus region in object space is a frustum evolved from the focus region in image space.

11. {Point/Region, Both, Both}: this would combine 9 and 10 above.

Table 5.1: Examples for the taxonomy of focus and context applied to volume visualization: assuming focus as generically point/region/volume (approaches not studied are indicated by a "?").

Vis. Space \ Effect	Distortion	Refinement	Both
Image	Fisheye [46, 144] and bifocal display [150]	Adaptive refinement [97]	Our 2D texture-based methods
Object	Magnification lens [89]	Multiresolution [88]	Our 3D texture-based methods
Both	?	Gaze-directed [98]	?

5.3 The Framework

We have proposed in section 5.2 (also published in [31]) a taxonomy for focus and context in volume visualization, in terms of three attributes: nature of the focus region (a single point, a circular or rectangular region), effect (refinement/accuracy, distortion or both) and visualization space (image, object or both). Recently, as we described in section 3.3, Hauser [55] proposed the idea that a number of different graphics resources (space, opacity, colour, frequency and style) can be employed to distinguish the focus and context regions on a dataset. Here we describe a unified framework, which aims to simultaneously apply some of these resources, providing great flexibility. The framework has three elements (or effects): mapping, highlighting and attenuation. We can associate this new framework with our taxonomy and with Hauser's work: the mapping effect is related to all attributes of the taxonomy and also related to Hauser's *space* graphics resource; highlighting is related to Hauser's *colour* graphics resource; finally, attenuation is related to Hauser's *opacity* graphics resource - the framework does not have any association with Hauser's *frequency* or *style* resources. The following sections present the three elements - mapping, highlighting and attenuation - in detail.

5.3.1 Mapping Effect

It is appropriate to understand a mapping effect as a transformation function, which modifies the coordinates of each voxel in order to obtain the "distorted" values, in such a way that this will achieve an enlargement or compression, depending on the region. However, as the effect does not change the original dimensions of the dataset, the edge voxels must be kept fixed. Following the taxonomy proposed in section 5.2, it seemed natural to begin the research with existing methods. Hence this section starts by discussing the implementation of two 3D distortion (mapping) methods based on 2D counterparts: 3D Cartesian bifocal and 3D Cartesian fisheye. To demonstrate the flexibility of the framework, we implemented a third method also based on a 2D idea: the volume lens. All of these methods can be classified, according to the taxonomy, as *{Point/Region, Distortion, Object Space}* - as there is no refinement.

Before talking about the methods, we must bear in mind an important implication related to the fact that we are dealing with discrete data (voxels): let us consider that a simple way to achieve volume distortion is by creating a new volume and then applying the mapping function to the original coordinates, which will give us the correct distorted coordinates in the new volume. However, this would not work: if we copy a smaller original region to a larger distorted one (in the case of magnification, for example), this

direct mapping would leave holes in the volume as not all voxels in the destination region would be filled. In the same way, if we copy a larger original region to a smaller distorted one, the direct mapping would copy more than one original voxel to the same destination one - these are the same problems that happen with image warping. The obvious solution is to use the inverse of the mapping function and work backwards, from the distorted coordinates to the original ones, i.e. we traverse the new volume and apply the inverse mapping function to obtain the original coordinates. Using each computed position, we fetch the corresponding voxel from the original volume and store it in the new one. This process is better illustrated in figure 5.1, which shows its application to a 2D image.

Furthermore, it is worth noting that although this is essentially a nearest neighbour sampling, there is nothing that prevents a better interpolation method (e.g. trilinear) from being used. In that case, the methods would be classified as *{Point/Region, Both, Object Space}* - which is exactly what happens when they are implemented with hardware-based texture mapping (which typically uses trilinear interpolation), as will be explained in chapter 6.

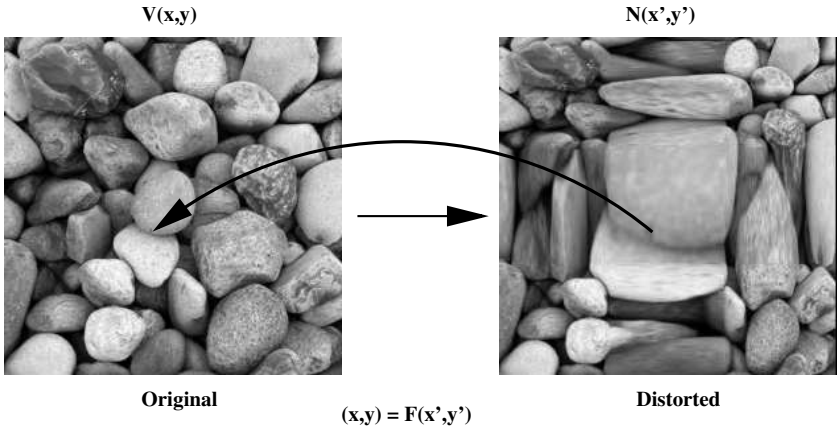


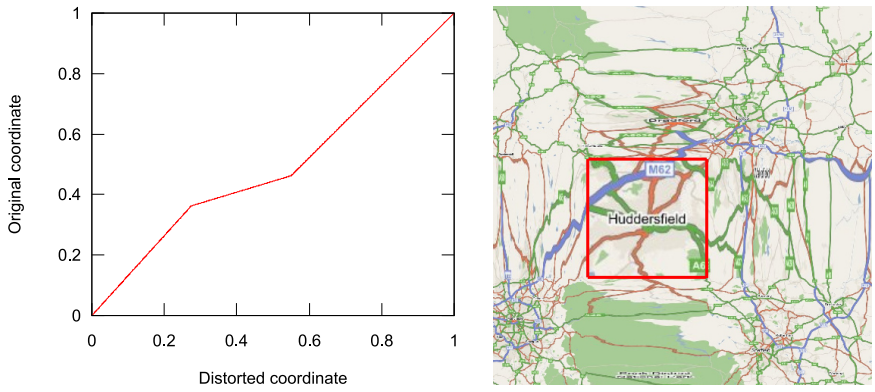
Figure 5.1: Inverse mapping with an image.

5.3.1.1 3D Cartesian Bifocal

A bifocal distortion is similar to a bifocal display [150], but in 3D: it creates a region of uniform magnification (focus region) where the feature of interest is. As the focus and context approach aims to visualize the entire dataset, we must compress everything outside the focus region.

It is simpler to use three distinct mapping functions, one for each dimension, instead of a single function depending on x, y and z coordinates - we shall use normalized coordinates to simplify computations. For clarity, figure 5.2 shows just the transformation function being applied to the x dimension (a) and the corresponding visual result in 2D (b)

- note that in (a) the horizontal axis displays the distorted coordinate, while the vertical axis shows the transformed (original) one resulting from the inverse mapping.



(a) Mapping function (b) Visual result

Figure 5.2: Bifocal distortion in 2D, applied to a map example.

The mapping function is directly adapted from Winch's work [171], and depends on three parameters:

- Centre of focus (x_f, y_f, z_f)
- Magnification factor (mag) - determines how much the focus area will be enlarged
- Size of each half of the focus region (sx, sy, sz) in the original volume

The first step is to compute six scaling factors: for each axis and for each side of the focus region. These factors are needed to determine how much the volume is compressed to each side of the focus region. Figure 5.3 shows how the factors for the x axis are obtained. Let Δ_{x+} be the distance from the right edge of the original focus region (x_{max}) to the boundary; $\Delta_{x'+}$ be the corresponding distance after the distortion. Using similar notation for the left side, we define scaling factors:

$$scale_{x+} = \frac{\Delta_{x'+}}{\Delta_{x+}} \qquad scale_{x-} = \frac{\Delta_{x'-}}{\Delta_{x-}}$$

(same for y and z)

Then, for each voxel $N(x', y', z')$ in the new volume (distorted), the following algorithm calculates the corresponding original voxel coordinate $V(x, y, z)$, using the scaling factors and the limits of both regions:

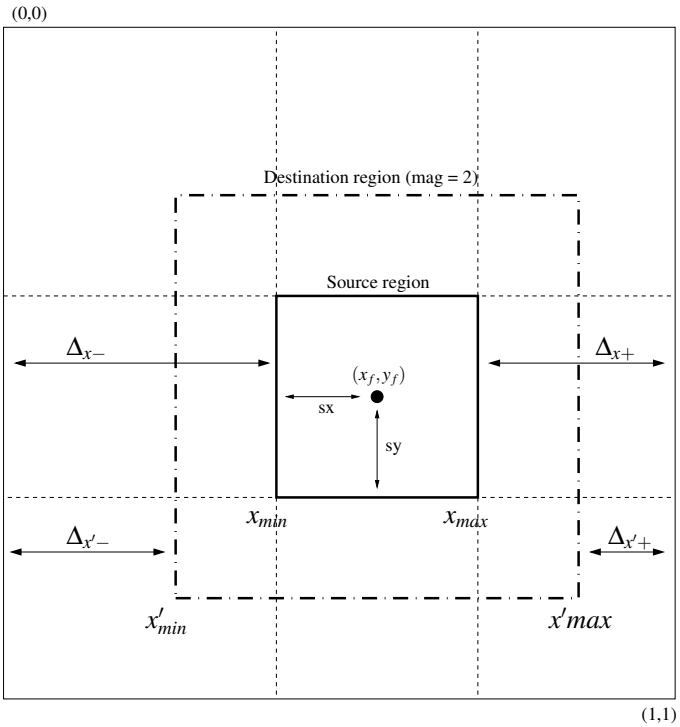


Figure 5.3: Scaling factors for the 3D bifocal Cartesian distortion.

- if x' is outside the focus region:
  - if $x' < x_f - mag \cdot s_x$ (left side): $x = \frac{x'}{scale_{x-}}$
  - if $x' > x_f + mag \cdot s_x$ (right side): $x = 1 - \frac{(1-x')}{scale_{x+}}$
- if x' is inside the focus region, just apply the magnification factor: $x = \frac{x' - x_f + mag \cdot x_f}{mag}$ (the formulation is the same for y' and z')

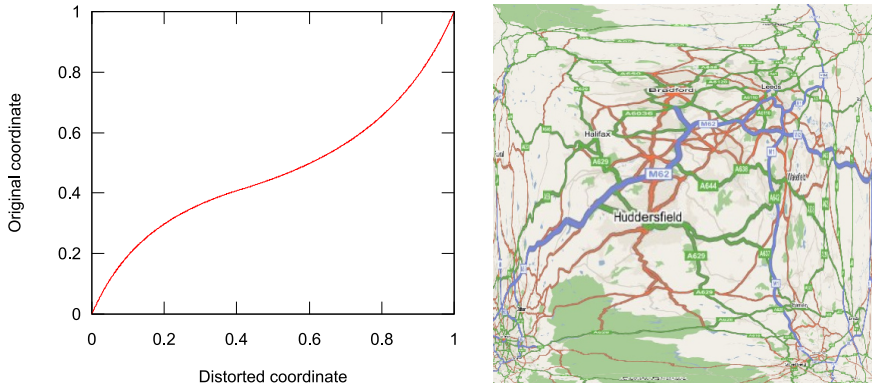
This distortion is interesting as it produces a clear separation between focus and context regions, which can be helpful to distinguish between them. However, this can also be seen as a disadvantage due to the abrupt changes in the mapping function.

5.3.1.2 3D Cartesian Fisheye

Like the previous method, this can also be seen as an extension of a two dimensional technique, the fisheye display [46,94,144]. In this case there is no focus region, but a *focal point* (i.e. a voxel) - the idea is to present the data in a continuous way, without a clear distinction between focus and context regions. The magnification factor - here specified

by a *distortion factor* - is highest at this focal point, and decreases as we move towards the edges of the dataset. This is interpreted in a different way from the magnification factor of the bifocal display, as when the distortion factor is zero, no magnification is taking place.

As before, the mapping function for the x dimension can be seen in figure 5.4a and the visual result in 5.4b.



(a) Transformation function (b) Visual result

Figure 5.4: Fisheye distortion in 2D, applied to the same map example.

Adapted from Sarkar & Furnas work [144, 46], this method depends on two sets of parameters:

- Centre of focus (x_f, y_f, z_f)
- Distortion factor (d)

Like the Cartesian bifocal method, we use the distances between the centre of focus and each dataset boundary. So considering the x axis, there are two distances - right (Δ_{x+}) and left (Δ_{x-}):

$$\begin{aligned} \Delta_{x+} &= 1 - x_f \\ \Delta_{x-} &= x_f \end{aligned}$$

(same for y and z)

Then, by traversing each voxel in the distorted volume $N(x', y', z')$ the following algorithm calculates the voxel coordinate in the original volume $V(x, y, z)$:

- if $x' < x_f$

$$x = x_f - \frac{\Delta_{x-}(x_f - x')}{\Delta_{x-} + d(\Delta_{x-} + x' - x_f)} = x_f - \frac{x_f(x_f - x')}{x_f + dx'}$$

- if $x' > x_f$

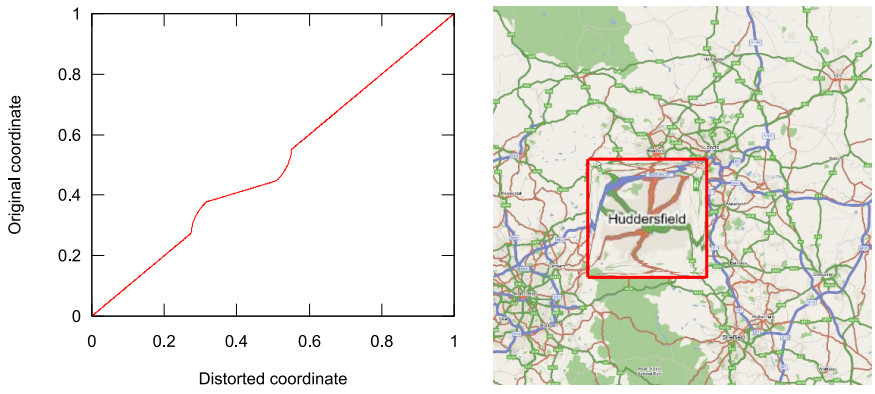
$$x = x_f + \frac{\Delta_{x+}(x' - x_f)}{\Delta_{x+} + d(\Delta_{x+} - x' + x_f)} = x_f + \frac{(1 - x_f)(x' - x_f)}{1 - x_f + d(1 - x')}$$

(same for y and z)

5.3.1.3 Volume Lens

The third mapping effect - called volume lens - is probably the most intuitive one, as it replicates the behaviour of a real lens: magnification is limited to a specific region and the rest of the dataset is largely unaffected. In order to achieve this, the mapping function must contain a narrow transition zone, so that we can move gradually from the non-magnified outside to the highly magnified inside.

Figure 5.5 presents both the transformation function for the lens (a) and its visual result (b). The transition zone was modelled as a quadratic curve between the edges of the non-magnified and magnified regions - we believe this is enough to achieve the desired effect.



(a) Transformation function (b) Visual result

Figure 5.5: Lens distortion in 2D, applied to the same map example.

If we consider just inside the focus region, the mapping function for the lens effect is similar to the one used in the bifocal method, as the magnification is uniform in most of it - except in the transition zone, where the magnification factor will rapidly drop to 1 as

we move to the outer region. The transition zone is specified as a percentage (*trans*) of the distorted focus region size and thus we define three additional limits (figure 5.6):

- Distance from the centre of focus to the edge of the focus region:

$$\Delta_{Lx} = x_f - x'_{min}$$

- Distance from the centre of focus to the start of the transition region:

$$\Delta_{Tx} = trans \cdot \Delta_{Lx}$$

- Distance from the start of the transition region to the edge of the focus region:

$$\Delta_{Rx} = \Delta_{Lx} - \Delta_{Tx}$$

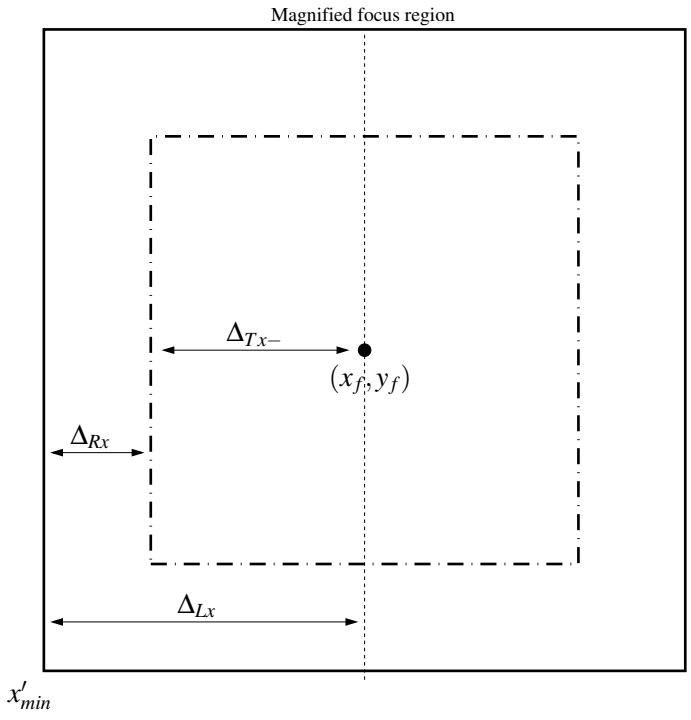


Figure 5.6: Limits needed for computing the lens factor in the x dimension: note that these are calculated taking into account the final, magnified size of the focus region - the original focus region (not magnified) is not being shown for clarity.

Once these limits are obtained, the process is similar to the previous methods: for each voxel $N(x', y', z')$ in the distorted volume, we compute the corresponding original voxel coordinate $V(x, y, z)$ by carrying out the following steps:

- if x' is outside the focus region, then:

$$x = x'$$

- if x' is within the focus region, we first apply the magnification factor:

$$x = \frac{x' - x_f + mag \cdot x_f}{mag}$$

  - if x' is also in the transition zone, we compute a *lens factor*:

$$\lambda = \begin{cases} \frac{(x_f - \Delta_{Tx} - x')}{\Delta_{Rx}}, & x' < x_f - \Delta_{Tx} \\ \frac{(x' - x_f - \Delta_{Tx})}{\Delta_{Rx}}, & x' > x_f + \Delta_{Tx} \end{cases}$$

It is easy to observe that the lens factor will be zero when x' is exactly at the edge of the transition zone (e.g. $x_f - \Delta_{Tx}$ for the left side) and it will be one when x' is at the edge of the focus region (e.g. x'_{min} for the left side). Hence the lens factor enables us to perform a linear combination of the (already computed) original x coordinate with the distorted x' coordinate - we actually use the square of the factor to obtain a quadratic behaviour in the result:

$$x = \lambda^2 \cdot x' + (1 - \lambda^2) \cdot x$$

5.3.1.4 Mapping Effects in 3D

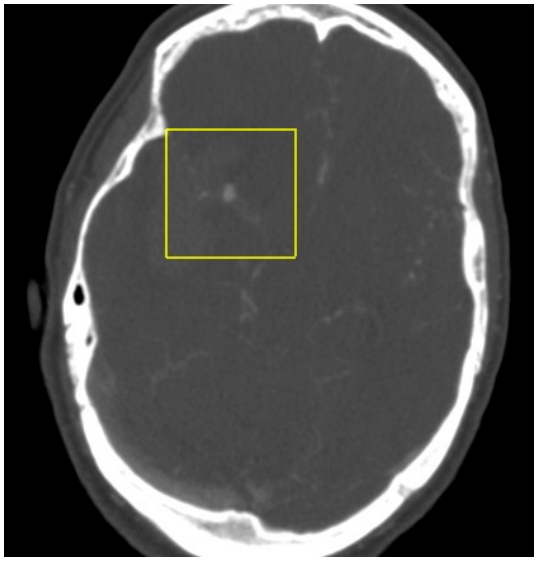
To demonstrate how the methods perform with real data, we now present some results obtained with a DICOM medical dataset, consisting of 512 x 512 x 183 slices - this dataset contains CTA data of a cerebral aneurysm.

Figure 5.7 shows a single slice of the dataset, where the aneurysm can be seen in the marked focus region: in (a), the aneurysm is not magnified and presents itself as a faint, small white structure; whereas in (b), (c) and (d) the application of each mapping method can be seen: bifocal, fisheye and volume lens. From these images, it is clear that all methods produce a good magnification of the focus region. However, the fisheye method creates a high distortion in the voxels close to the edges of the dataset. As for the volume lens, the contents of the transition zone are also very compressed and thus it is rather difficult to visualize any detail in that region. Even so, if the user wants to enlarge just

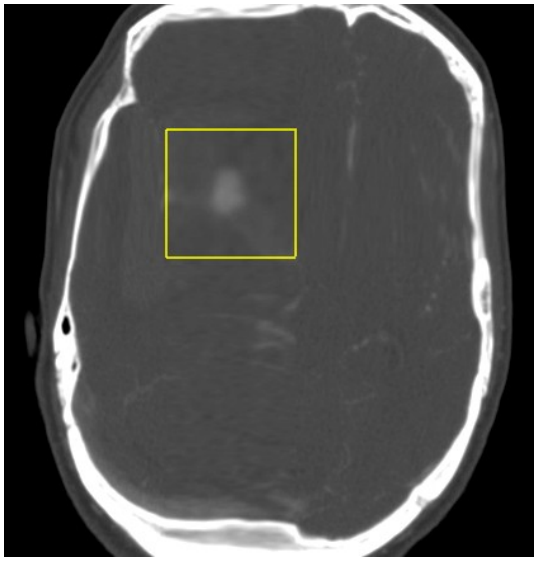
the focus region, then the volume lens provides an optimal view. Note that in these 2D examples, the mapping effects are applied using just the functions for x and y.

We can make similar observations when considering a volume rendered view of the same dataset (figure 5.8): in this case, the distortion effects of the fisheye method are even more apparent, but as the magnification function does not exhibit discontinuity in slope, the spatial relations of the dataset are better preserved. This medical example is particularly good to illustrate this point: as the aneurysm has formed in an artery, it is important to be able to visualize this particular vessel without any discontinuity - this is not possible with the volume lens, as the regions outside the focus region are not magnified. Again with the lens method, the user loses the ability to visualize detail in the transition zone - but this is an acceptable compromise, if one does not want to have distortion outside the focus region. Nevertheless, the lens is still a very useful exploration tool, as it works in 3D space, i.e. affects a defined 3D region within the dataset, regardless of the viewpoint.

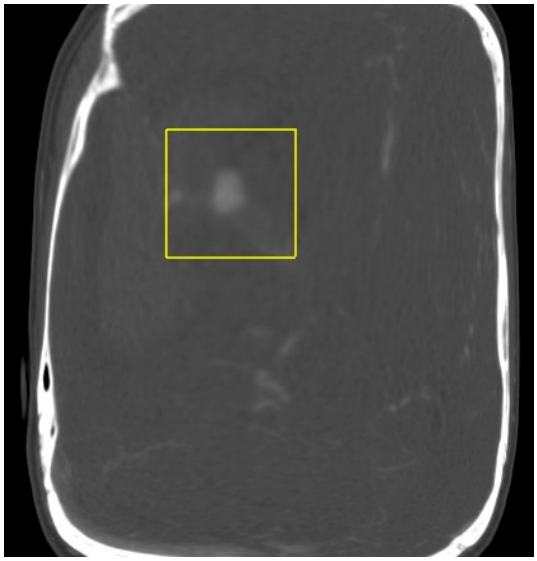
Therefore the user can position the lens, for example, behind other structures and still use it to magnify that region - note that this is a different approach than the one implemented by Wang et al [162], as their lens affects everything that is directly in front of the viewer.



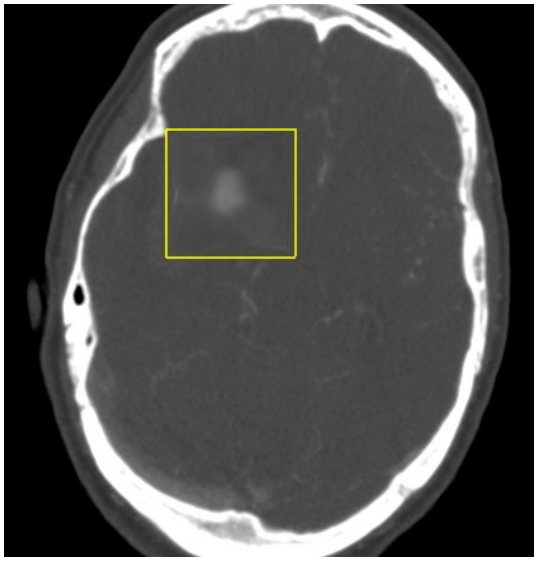
(a) Normal, no effect



(b) Bifocal effect

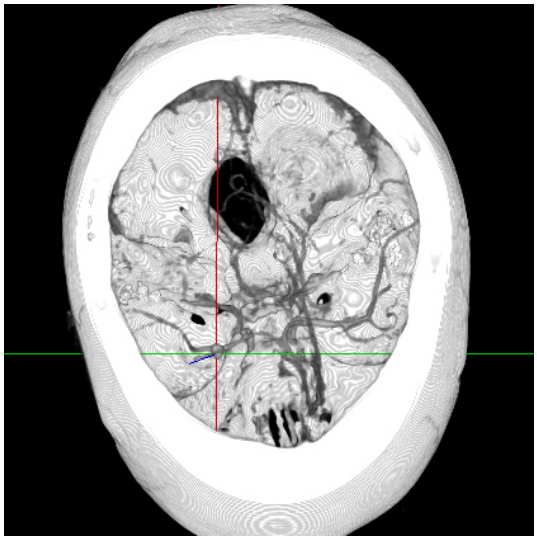


(c) Fisheye effect



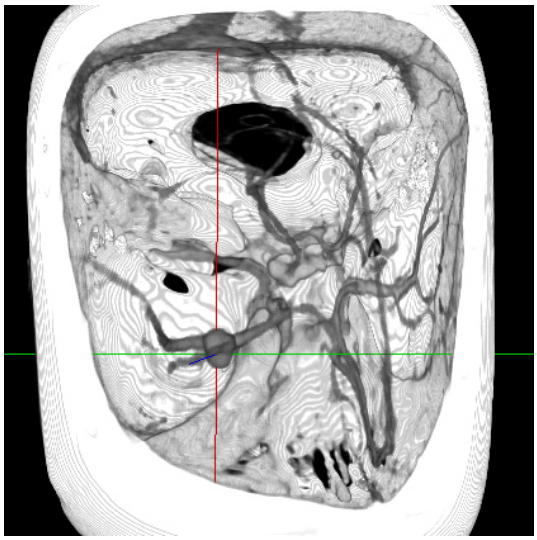
(d) Volume lens effect

Figure 5.7: An example of applying the mapping effects to the same 3D dataset, single slice view: in (b) and (d) the magnification factor is 2.5; in (c) the distortion factor is 1.5 (roughly equivalent to the magnification factor in the others) - note that in this case the actual focus region is not used, just its central point.



(a) Normal, no effect

(b) Bifocal effect



(c) Fisheye effect

(d) Volume lens effect

Figure 5.8: An example of applying the mapping effects to the same 3D dataset, volume rendered view: in (b) and (d) the magnification factor is 2.5; in (c) the distortion factor is 1.5.

5.3.2 Attenuation Effect

When dealing with volume data, it is common that the feature of interest is located deep inside the volume - this can make it difficult to visualize properly. For example, consider a 3D dataset of the human brain: the major problem here is the skull, which will always be around the inner structures. Even with a suitable transfer function, it may not be possible to remove it and preserve the desired feature at the same time (e.g. in the case of CTA data). This led to the inclusion of the *attenuation* effect in the framework, which provides an *opacity scaling factor* for each voxel - 1 meaning no attenuation (original opacity) and 0 meaning full attenuation (fully transparent). The exact behaviour is specified by an *attenuation function*.

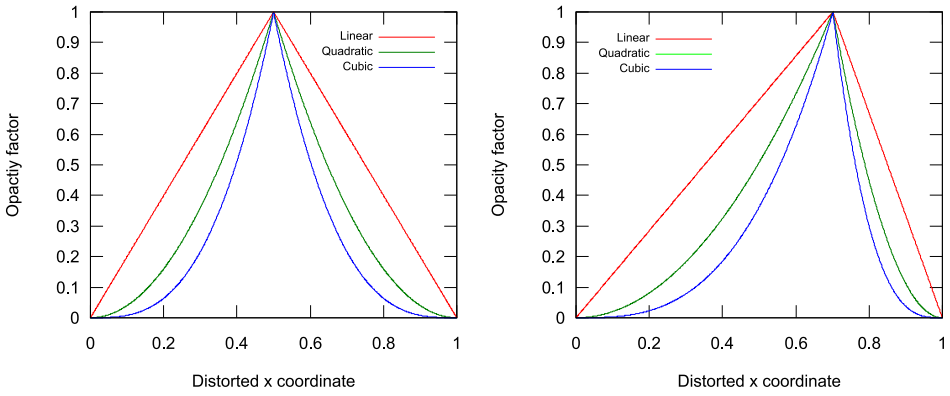
One may imagine that suitable attenuation could be created by a function which is proportional to the distance of each voxel from the centre of focus - however, this does not help much if the focus region is close to the edge of the dataset, as the voxels at the edge will not be attenuated enough in order for the focus region itself to show through. Also this would produce attenuation inside the focus region, which is not desirable. A possible solution is to use the ratio between two distances from the closest dataset edge: to the current distorted coordinate and to the closest distorted boundary of the focus region. Hence for any $x \in [0,1]$ the attenuation value ($atten_x$) is computed by:

$$atten_x = \begin{cases} \frac{x}{x'_{min}} & x < x'_{min} \\ \frac{1-x}{1-x'_{max}} & x > x'_{max} \end{cases} \tag{5.1}$$

We must note, however, that if we are using the fisheye mapping effect then there is no focus region and therefore, we would replace x'_{min} and x'_{max} in equation 5.1 with x_f. Also note that the linear function generally creates a rather gradual effect, hence we can use the function squared or cubed and obtain a better attenuation effect - we define these as *quadratic* and *cubic attenuation functions*. Observe that equation 5.1 guarantees that regardless of the position of the distorted focus region/centre of focus, the opacity factor at the edges of the dataset will be zero - this helps the focus region to be revealed, especially if it is close to an edge of the dataset. To exemplify this concept, figure 5.9 shows how the functions behave when the focus region is at the centre and when it is closer to the right edge, and also presents the visual result in 2D for each case - this is for a fisheye mapping, so the centre of focus is used to compute the function.

Note that the *combined* opacity factor for a specific voxel is given by the multiplication of the opacity factors for each dimension, i.e:

$$atten_{xyz} = atten_x \cdot atten_y \cdot atten_z \tag{5.2}$$



(a) Focus centre at 0.5

(b) Focus centre at 0.7

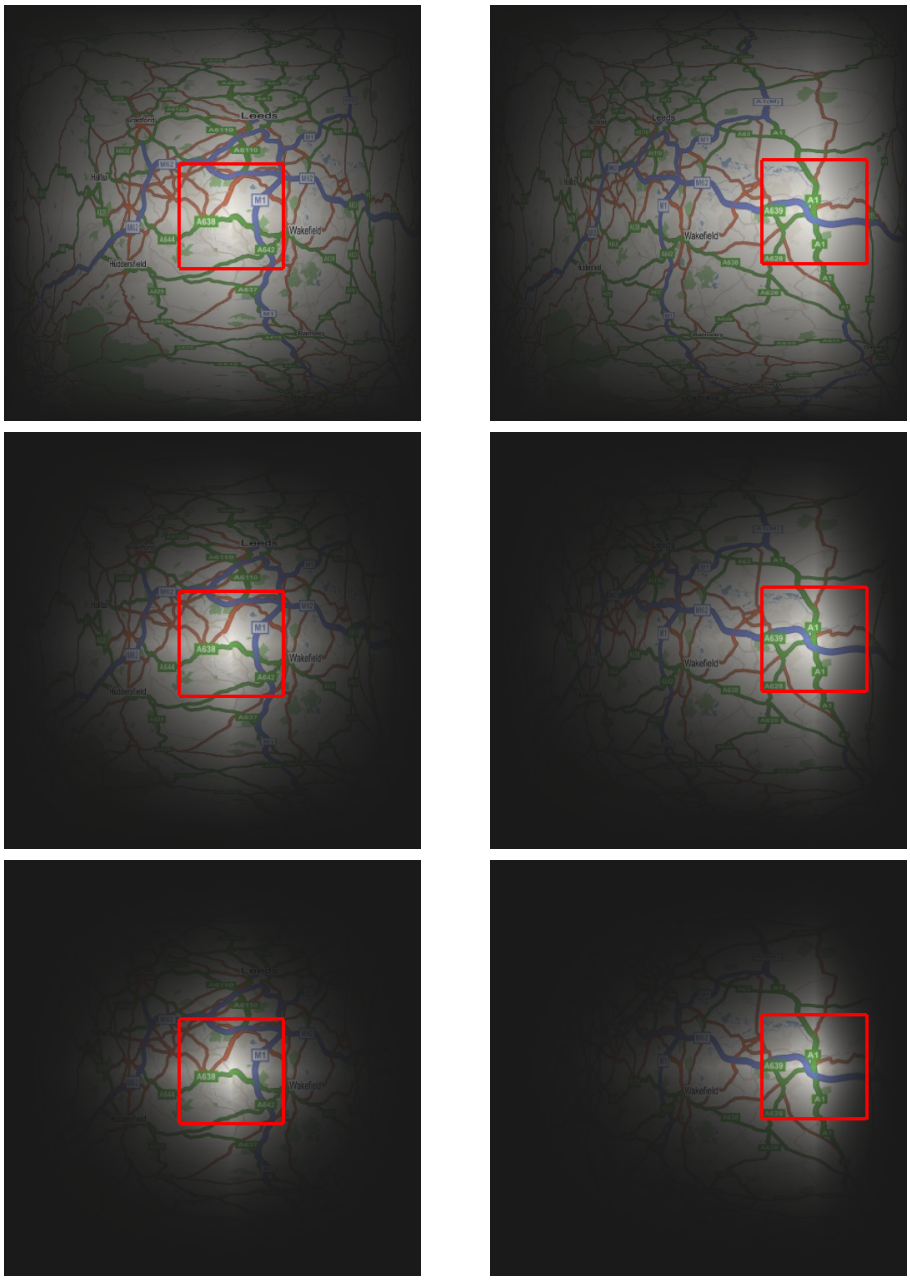


Figure 5.9: Attenuation functions and the influence of the focus centre in fisheye distortion: linear (red, top image), quadratic (green, middle image) and cubic (blue, bottom image).

For most cases this solution is acceptable, however there are situations when volume data close to the focus region should be more attenuated: hence we can consider less than the full distance from a boundary of the focus region to the closest dataset edge. For this, we define $dist_{xyz+}$ and $dist_{xyz-}$ as the set of distances from each boundary of the *attenuation* region to the corresponding closest dataset edge (see figure 5.10 for a graphical view). These values are then subtracted from the previously computed distances to determine the behaviour of the attenuation function - note that the resulting value is clamped at zero. Therefore equation 5.1 is adapted as follows:

$$atten_x = \begin{cases} max\left(\frac{x-dist_{x-}}{x'_{min}-dist_{x-}}, 0\right) & x < x'_{min} \\ max\left(\frac{1-x-dist_{x+}}{1-x'_{max}-dist_{x+}}, 0\right) & x > x'_{max} \end{cases} \tag{5.3}$$

Note that values of $dist_{x+/-}$ greater than zero can be used if the volume data close to the focus region is more opaque, hence the attenuation region will start closer to the focus region. If these values are zero then we obtain the same behaviour as equation 5.1.

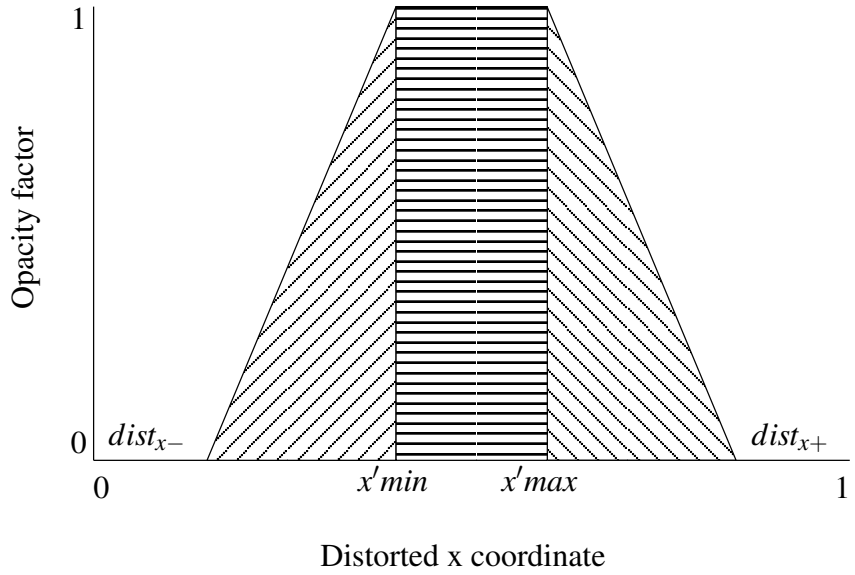


Figure 5.10: Adding distance components to the attenuation computation: the attenuation regions are indicated with the diagonal patterns.

This approach yet has a problem: the primary objective of attenuation is to remove voxels which are potentially in front of the focus region, but as it is presented, it removes voxels from *all* directions - which is not always desirable, as some voxels behind the focus region may contain useful information. Therefore, an alternative function can be computed taking into account the viewing direction in 3D space - this has the goal of

creating a region of transparency only in front of the focus region, while keeping the remaining regions unchanged.

In this case, the procedure is slightly more complex: first the current viewpoint is used to compute a *gaze vector* ($\overrightarrow{gaze}$), which points from the centre of focus to the viewer $(x_{view}, y_{view}, z_{view})$:

$$\begin{aligned} gaze_x &= x_{view} - x_f \\ gaze_y &= y_{view} - y_f \\ gaze_z &= z_{view} - z_f \end{aligned}$$

This vector is then normalized:

$$\overrightarrow{gaze} = \frac{\overrightarrow{gaze}}{|\overrightarrow{gaze}|}$$

Once normalized, the gaze vector is used to adjust the resulting opacity factor as follows, e.g. for x:

- If $gaze_x < 0$ (viewer looking from the left):

$$atten_x = \begin{cases} (1 - |gaze_x|) + \frac{x - dist_{x-}}{x'_{min} - dist_{x-}} |gaze_x| & x < x'_{min} \\ 1 & x > x'_{max} \end{cases} \tag{5.4}$$

- If $gaze_x > 0$ (viewer looking from the right):

$$atten_x = \begin{cases} 1 & x < x'_{min} \\ (1 - |gaze_x|) + \frac{1 - x - dist_{x+}}{1 - x'_{max} - dist_{x+}} |gaze_x| & x > x'_{max} \end{cases}$$

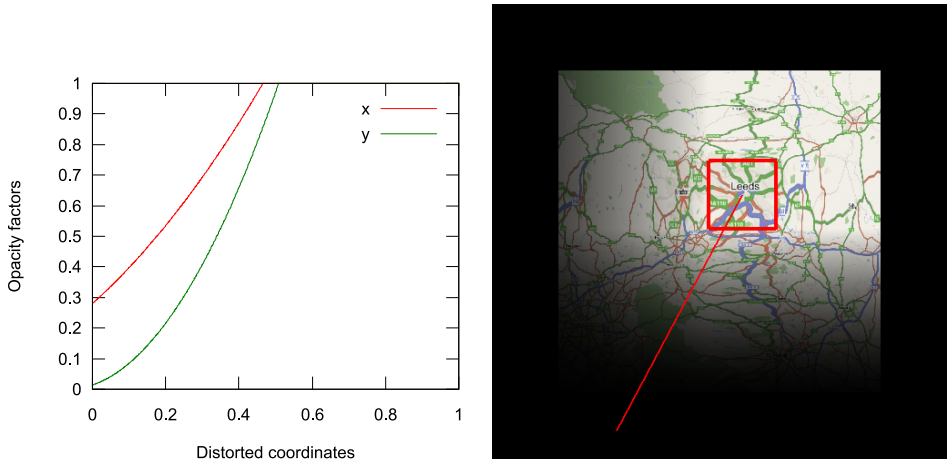
- The final result is raised to the power (p) of the viewpoint-based attenuation function - this can be 1 for linear, 2 for quadratic and so on. However, as we are removing just what it is in front of the focus region, higher power values usually work better (e.g. 8 or higher).

$$atten_x = (atten_x)^p$$

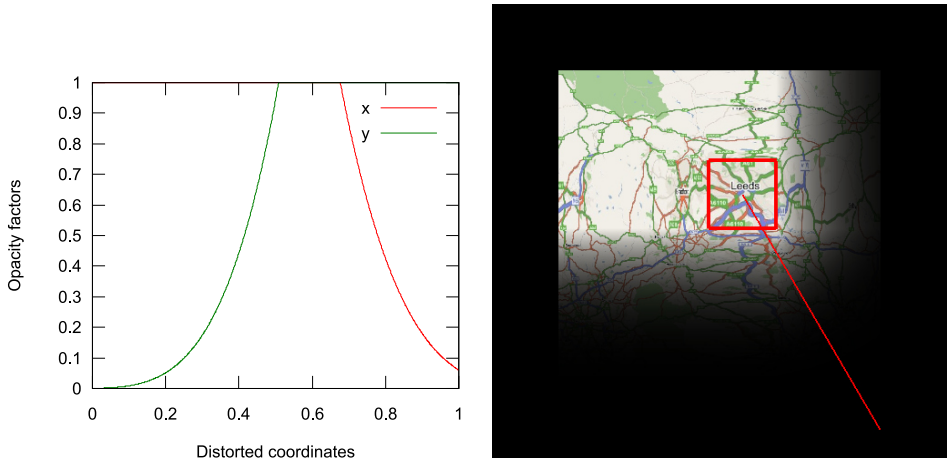
For example, if we are viewing the volume from the left side (negative x gaze) and the current voxel is further to the right than the focus centre ($x > x_f$) then equation 5.4 guarantees that the opacity factor at that voxel will be 1 (i.e. no attenuation) - this works, as from the current viewpoint the voxel is behind the focus centre, so it should be normally displayed.

To help understand how this works, figure 5.11 presents the viewpoint-based attenuation in the map example (2D), now applied along with a bifocal distortion: as we consider that the viewer is outside of the image, the gaze vector is indicated by the red line. From the figure, it is easy to see how well the method performs: in 5.11a/b, the viewer is at the bottom and to left of the focus region - but more towards the bottom than to the left side. This makes the x opacity function (quadratic in this example) to start from 0.3 instead of zero - as there is no need to completely remove voxels in x. From the point when either x or y reach the corresponding focus region edge, the functions map to one so there is no longer any attenuation. In 5.11c/d, x and y produce almost identical, but reversed functions as the viewing angle is nearly 45 degrees - note that in this case we used a higher attenuation power (4), to illustrate the difference in the visual result. Finally, figure 5.11e/f shows an example where the viewer is totally to one side of the focus region: this evidently creates a gradual function for the y axis, with almost no attenuation throughout - in contrast to the corresponding function for the x axis, which presents a very steep rise and therefore most of the image to the right of the focus region will be completely attenuated.

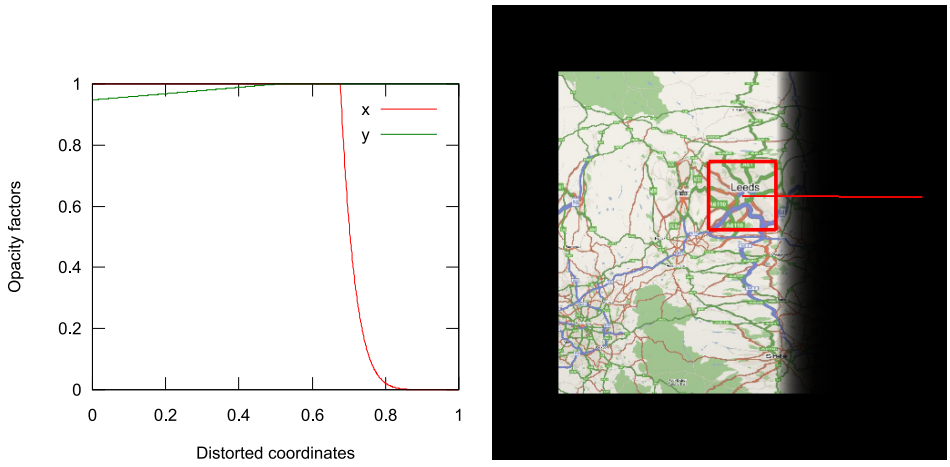
In order to fully understand how effective this effect can be when applied to a 3D dataset, figure 5.12 shows a volume rendered view of the same medical dataset we used earlier. Even though we have set an appropriate colour map to present the aneurysm, this alone is not enough to allow a clear view of the focus region as the skull bone gets in the way for most viewpoints - if the colour map was set to remove the skull bone, the required opacity range would also remove the aneurysm, due to the contrast used during the acquisition procedure. In (a), the dataset is shown without any attenuation, whilst in (b) a linear distance attenuation function has been applied, but it is not effective to completely attenuate the bone, and in (c) the attenuation function is computed according to the viewing direction, thus fully removing the skull bone and preserving more of the structures behind the focus region as well - note that in this case it is the combined viewpoint-based attenuation with an attenuation power of 15 that creates the best effect.



(a) (b)

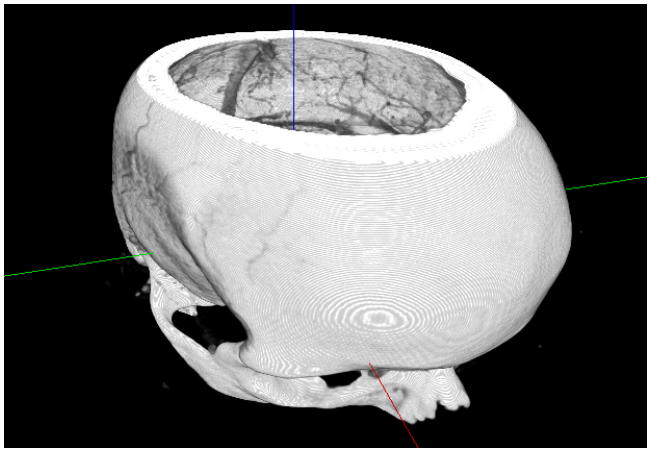


(c) (d)

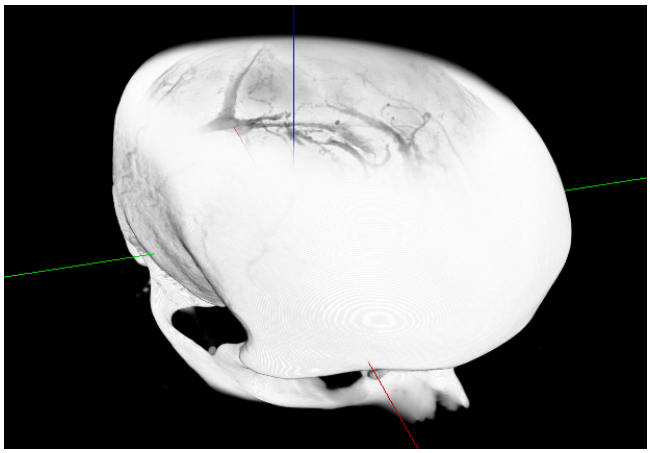


(e) (f)

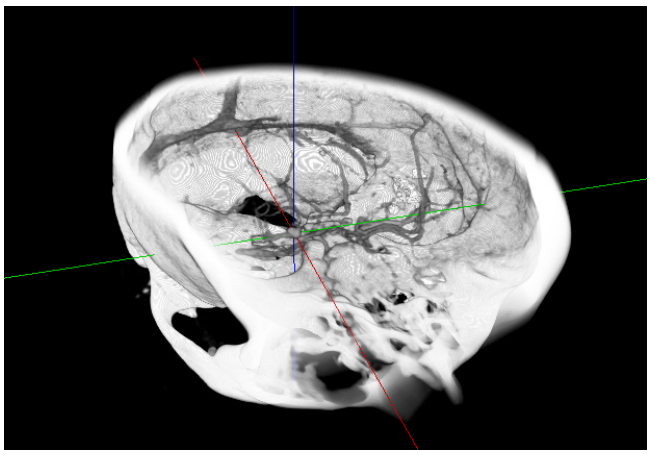
Figure 5.11: Viewpoint-based attenuation in 2D ($dist_{xy}$=0): in (a/b), the power of the attenuation function is 2; in (c/d), the power is 4; and in (e/f), the power is 8.



(a) No attenuation



(b) Linear attenuation



(c) Viewpoint-based attenuation

Figure 5.12: Using attenuation to remove occluding features: in (b) the linear attenuation is not enough to completely remove the skull bone; in (c) the viewpoint-based attenuation is much more effective, using an attenuation power of 15.

5.3.3 Highlighting Effect

It is often desirable to be able to highlight what we are interested in. For example, we use bold or italic text to highlight important words; we use colour to highlight features on an image or graph. So it seems natural that we can also use some sort of highlighting when dealing with volume data, hence the framework includes a *highlighting* effect - this uses a specific colour (e.g. green) to apply colour to each voxel, according to an associated highlight value - from zero (no highlight, original colour) to one (full highlight, combined colours). A *highlighting function* controls the exact behaviour - for example, a simple way of computing the value could be to use the distance from the voxel to the centre of focus or edge of focus region. As we did with the other effects, the highlighting function is defined independently for x, y and z, and we use half of the distance for the basic, linear function, as there is normally no need to highlight the whole dataset and we experimentally found out that it is a suitable range. Hence for x:

$$highlight_x = max(1 - 2\left|x_f - x\right|, 0)$$

The *max* function guarantees that the resulting value will never be less than zero (no highlight). We can also raise this result to a power, to obtain quadratic or cubic functions. Alternatively, the highlight could be restricted to the focus region: in this case, the highlighting function could be a constant.

The actual colouring effect is obtained by multiplying the highlighting values from each dimension, in a similar way to the combined attenuation value (equation 5.2):

$$highlight_{xyz} = highlight_x \cdot highlight_y \cdot highlight_z$$

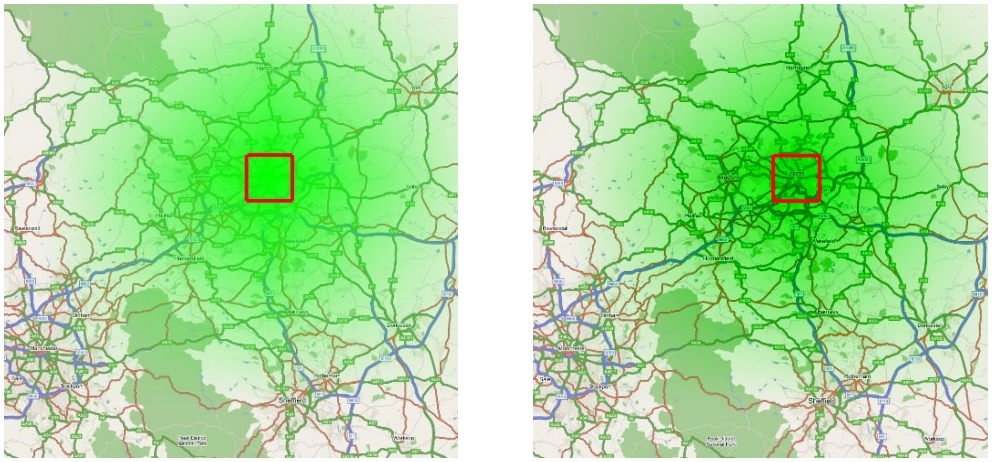
Then considering V_{rgb} as the original voxel colour and H_{rgb} as the highlighting colour defined by the user, we compute the resulting colour as follows - all colours defined as normalized RGB values:

$$V_{rgb} = max\left(V_{rgb} - (1 - H_{rgb}) \cdot highlight_{xyz}, (0,0,0)\right) \tag{5.5}$$

For example, if the highlighting colour is green - i.e. $H_{rgb} = (0,1,0)$ - and the original voxel colour is white - i.e. $V_{rgb} = (1,1,1)$ - then the final colour will be given by $(1,1,1) - (1,0,1) \cdot highlight_{xyz}$. In this case, it is easy to see that depending on the combined highlighting value, we will have a different shade of green as the final colour.

One may ask why we did not use a common linear blending function such as $V_{rgb} = highlight_{xyz} \cdot H_{rgb} + (1 - highlight_{xyz}) \cdot V_{rgb}$: that function would eventually (e.g. near the

focus centre) replace the original voxel colour with the highlighting colour, and that is not what we want. To illustrate this, figure 5.13 compares the application of such a blending function with the method presented in equation 5.5.



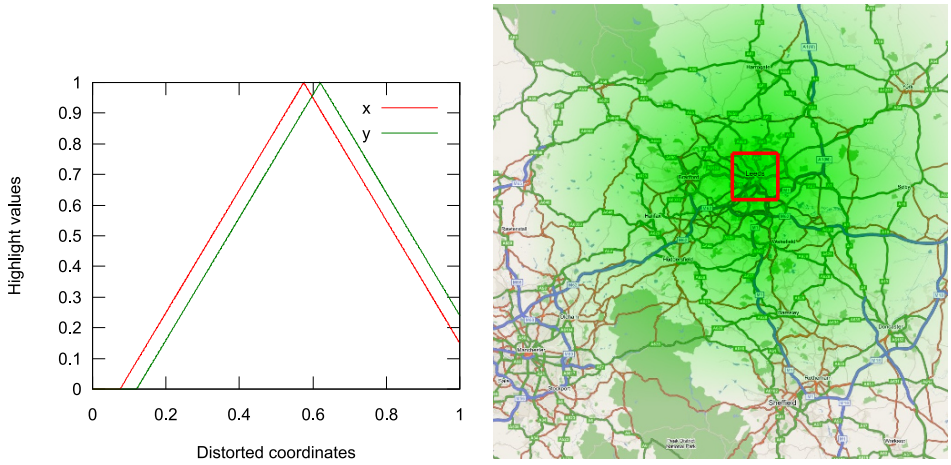
(a) Normal blending (b) Colourising method

Figure 5.13: Comparing the effect of highlighting with a normal blending function (a) and a colourising method (b).

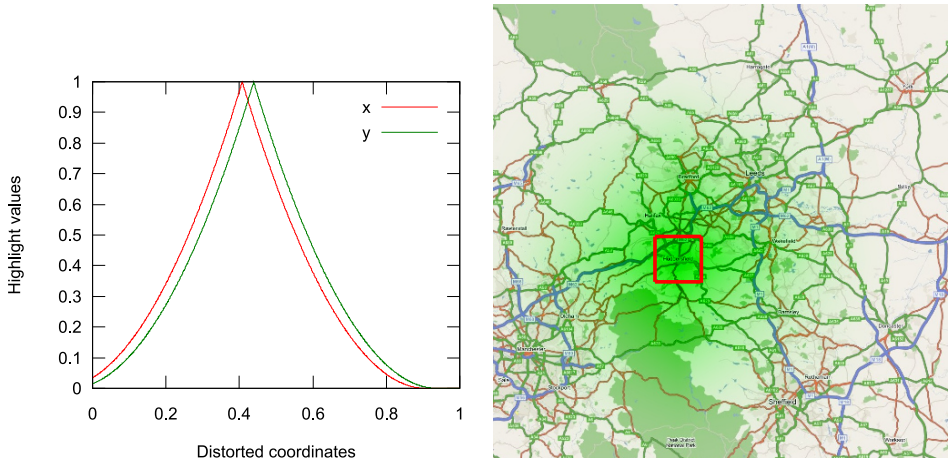
Next, figure 5.14 presents how different highlighting functions and positions of the centre of focus have influence in the final result, using the 2D map example. Finally, figure 5.15 shows how the highlighting effect can be used in 3D, to enhance the visualization of the aneurysm in a different medical dataset. In this example, the contrast used during image acquisition produced a dataset with very bright vessels - this is good for diagnostic purposes, but at the same time can be confusing as many structures may be visible (and some may not be desirable, as the extra bone for example). As this dense data normally is displayed in grey scale, the highlighting method works particularly well.

5.4 Summary

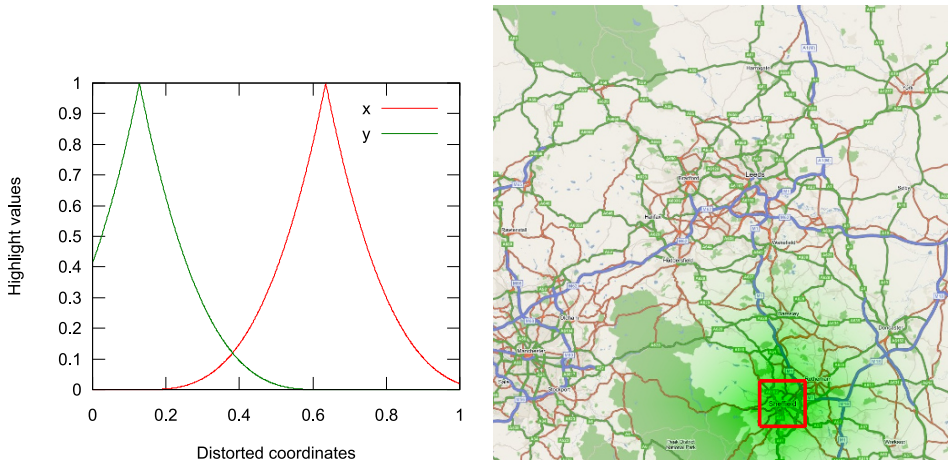
This chapter has presented a taxonomy and a flexible framework to achieve volumetric distortion as well as other novel effects. We have described in detail how the framework is modelled as independent functions for each coordinate dimension and provided examples of the framework application with 2D and 3D datasets. The following chapters report how the theoretical framework was implemented: chapter 6 reveals the research on hardware-accelerated distortion methods and chapter 7 presents how these methods were integrated into a volume rendering system, with emphasis on the user interface issues.



(a) (b)

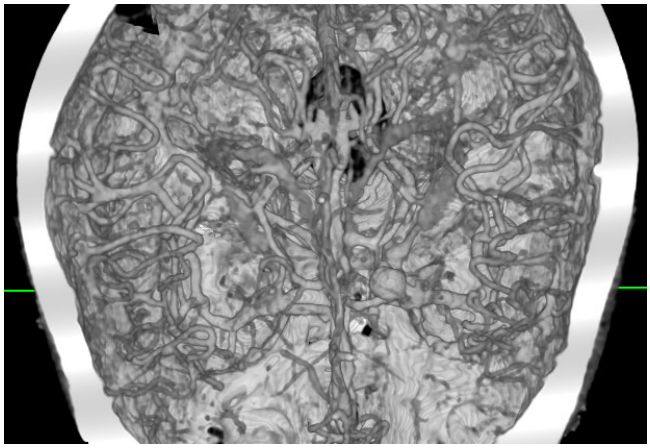


(c) (d)

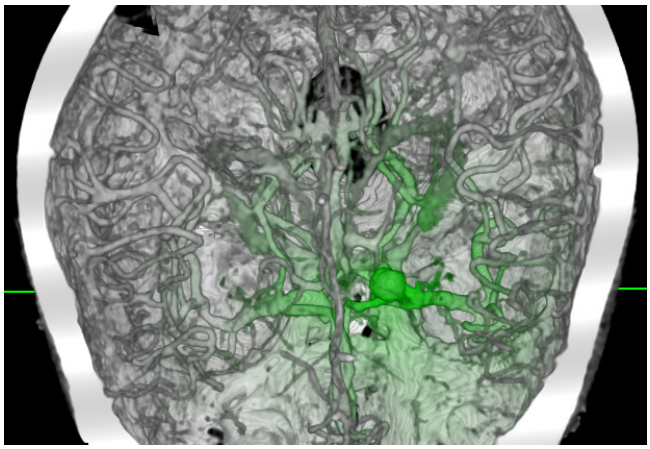


(e) (f)

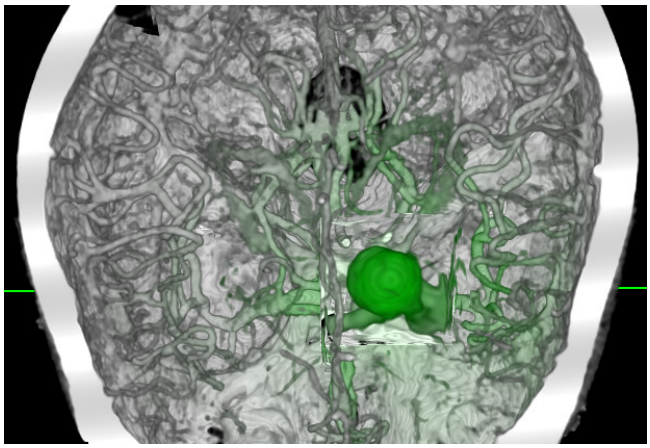
Figure 5.14: Highlighting in 2D: in (a/b), the highlighting functions are linear; in (c/d), the highlighting functions are quadratic; and in (e/f), the highlighting functions are cubic.



(a) No effect



(b) Quadratic highlighting



(c) Volume lens and cubic highlighting

Figure 5.15: Applying the highlighting effect in a confusing dataset: in (a) the sheer number of vessels makes it difficult to visualize the aneurysm; in (b) with a quadratic highlighting function the aneurysm and some nearby arteries can be clearly seen; in (c) this is combined with the volume lens.

Part III

The *VolFocus* System

Chapter 6

Hardware-accelerated Methods

6.1 Overview

THIS CHAPTER PRESENTS, in chronological order, the evolution of the distortion methods used within the *VolFocus* system. This covers all the development that has been done for the system, from the adaptation of an existing volume renderer to the advanced methods based on GPU programming. A comparative analysis of all methods will be presented at the end of this chapter. As most methods share common parameters, we define them below:

- Centre of focus: (x_f, y_f, z_f)
- Magnification factor: *mag*
- Half of the focus region size: (s_x, s_y, s_z)
- Scaling factors for bifocal distortion: $(scale_{x-}, scale_{x+})$, $(scale_{y-}, scale_{y+})$ and $(scale_{z-}, scale_{z+})$
- Maximum voxel coordinate in each axis: $(vmax_x, vmax_y, vmax_z)$, from 0 to the respective dimension minus one.

All parameters that depend on computations were already described in section 5.3.1 so the mathematical procedures will not be repeated here, except in cases where they are useful to clarify the explanation.

6.2 Bricking and Scaling

When our research began, the graphics hardware was limited to textures whose dimensions were a power of two, so in order to work with volumes with different sizes it was necessary to split the volume into smaller parts, each one of which respects that limitation. This operation is called *bricking*, and the volume parts are then commonly referred to as *bricks*. Each brick will have its own texture, extracted from the full volume. In our case, we used the OSCVR library [21], which already provided the bricking algorithm. Nevertheless, it was not possible to apply the formulation described in section 5.3.1.1, as the original, full-sized texture will not be available to each brick.

Therefore an alternative approach was developed: the full volume is first split into the 27 regions outlined in section 5.3.1.1. Each region u is defined primarily by:

- Minimum and maximum limits, before distortion: $(min_{xu}, min_{yu}, min_{zu})$ and $(max_{xu}, max_{yu}, max_{zu})$
- Minimum and maximum limits, after distortion: $(dmin_{xu}, dmin_{yu}, dmin_{zu})$ and $(dmax_{xu}, dmax_{yu}, dmax_{zu})$

To find the distorted limits, one must first compute the new boundary of the focus region after distortion, as figure 6.1 shows in 2D. As these values are in voxel coordinates, to simplify further calculations we also normalize the coordinates so the volume occupies the unit cube. Hence for the x dimension, the boundary of the focus region would be obtained by:

$$\begin{aligned} x'_{min} &= \frac{x_f - mag \cdot sx}{vmax_x} \\ x'_{max} &= \frac{x_f + mag \cdot sx}{vmax_x} \end{aligned}$$

The limits of the focus region are then used in the specification of the limits of all other regions. For example, considering the region marked as R in the figure, the original limits for x would be $min_{xR} = x_{min}$ and $max_{xR} = x_{max}$, while the distorted limits for x would be $dmin_{xR} = x'_{min}$ and $dmax_{xR} = x'_{max}$.

The whole idea behind this method is the correct scaling and positioning of each region, hence now we need to find out which scaling factor to use for each dimension in this region. The six scaling factors - $(scale_{x-}, scale_{x+})$ and the respective ones for y and z - are computed using the formulation explained in section 5.3.1.1. Then we select the

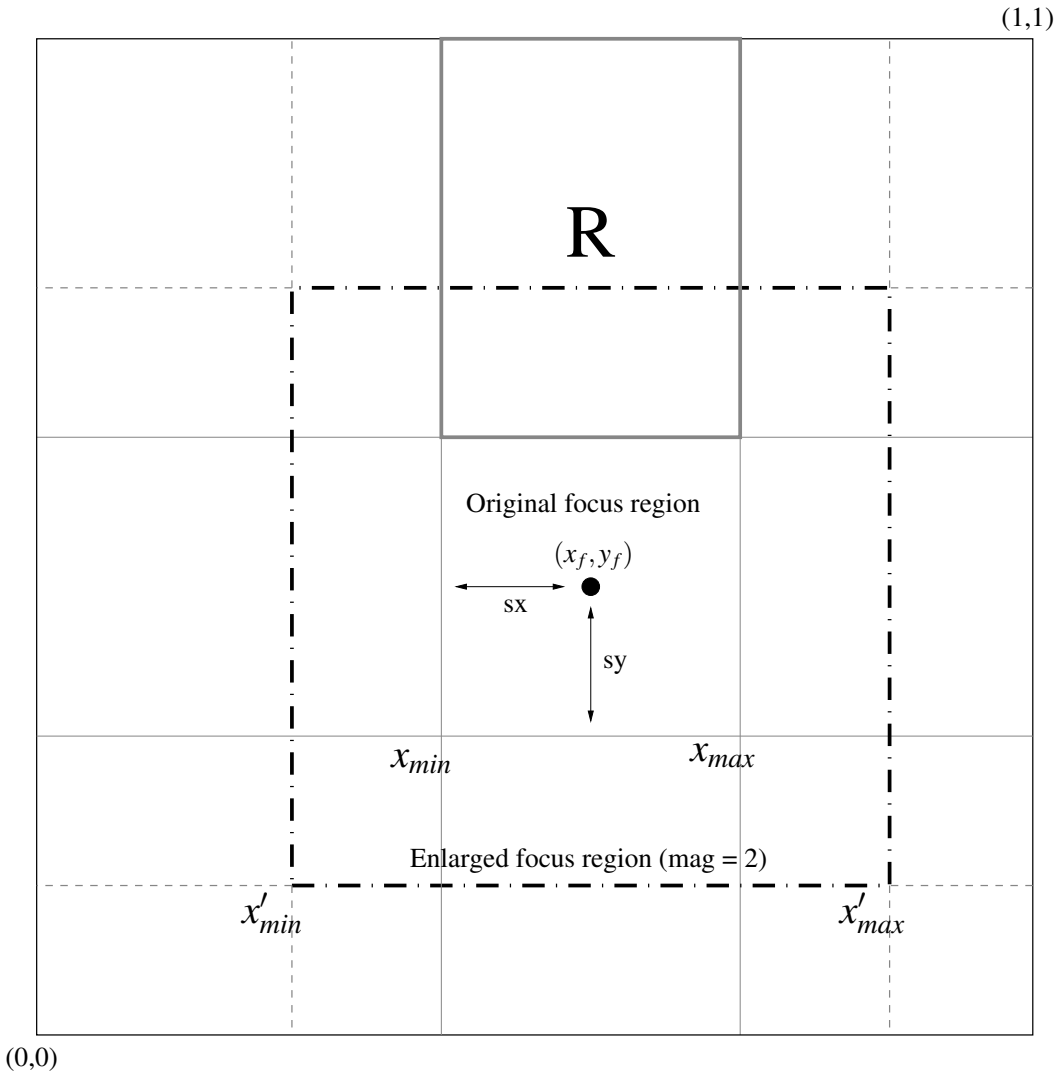


Figure 6.1: Calculating the distorted boundary (x'_{min}, x'_{max}) of the focus region in 2D: (x_{min}, y_{min}) denote the original boundary (without distortion), (xf, yf) is the region centre, (sx, sy) are half of the dimensions of the original region and *mag* is the magnification factor being applied.

appropriate ones from the set already computed, according to the region being considered. So if we use the same region R as an example, the scaling factors for x and y would be:

$$\begin{aligned} S_{xR} &= mag \\ S_{yR} &= scale_{y+} \end{aligned}$$

as the region R is within the focus region in x but after its limits in y.

However, as we cannot guarantee that the lower left corner of the region is at the origin of the coordinate system, we then compute a translation to move that corner to the origin (see figure 6.2a):

$$tx1_R = -min_{xR}$$

Now the region can be scaled, using the scaling factors selected before (figure 6.2b). Finally, the lower left corner of the region - now at $(0,0)$ - must be translated to the corresponding distorted position (figure 6.2c), which naturally is:

$$tx2_R = dmin_{xR}$$

In summary, the algorithm has the following preprocessing steps:

- Split the volume into 27 regions
- Compute the six scaling factors (see section 5.3.1.1)
- For each region u
  - Determine the minimum/maximum limits (normal and distorted)
  - Obtain the scaling factors for x, y and z
  - Obtain the two translations for each dimension
  - Split each region into the number of bricks needed to tessellate it
  - Create all required bricks, associating the region number to each

The rendering algorithm will then:

- Sort all bricks in back to front order
- To draw each brick:

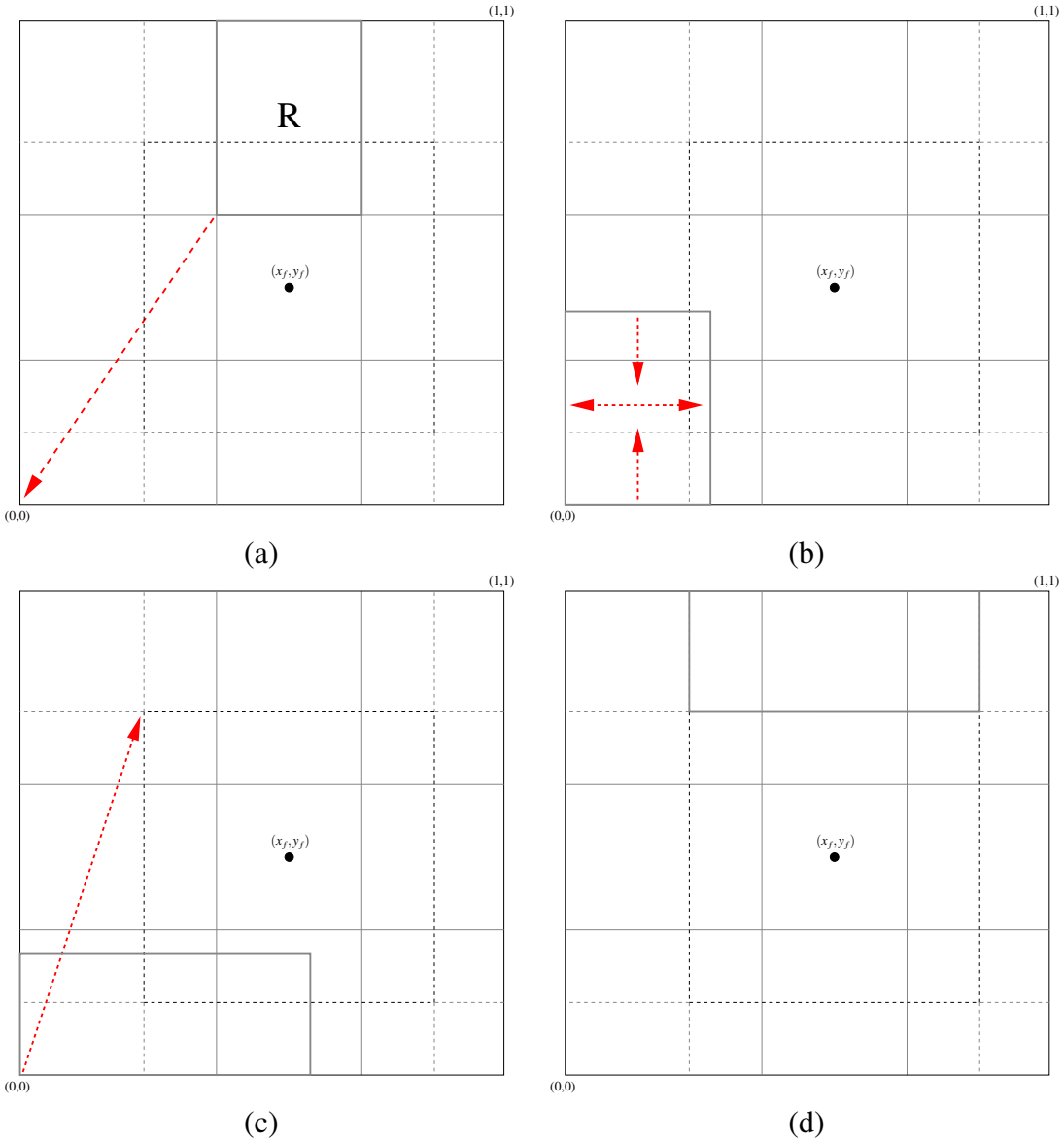


Figure 6.2: Bricking and scaling method in 2D: computing the translation to move the region lower left corner to the origin (a); result after translating the region (b); scaling the region according to its scaling factors - in this case, an enlargement is taking place in the x dimension and a compression in the y dimension - and computing the translation to move the region to the final position (c); final region correctly positioned and scaled (d).

- Identify which region u the brick is in (from the stored region number)
- Apply a translation of $(tx1_u, ty1_u, tz1_u)$
- Apply a scale of (Sx_u, Sy_u, Sz_u)
- Apply a translation of $(tx2_u, ty2_u, tz2_u)$
- Render the brick normally

Finally, it is important to note that this process will produce many more bricks than the quantity originally needed to tessellate the full volume. But the data is still the same, no distortion takes place on the bricks themselves. It is only during the rendering process of each brick that the distortion will be applied, using the previous algorithm.

This bricking method has the following disadvantages:

- it requires the volume to be split into 27 sub-volumes, which means that new texture memory will have to be allocated and portions of the original texture will need to be copied to the texture memory associated with each region - this requires processing time.
- it is very difficult to implement non-linear distortion methods such as the 3D Cartesian fisheye, as each brick can only be scaled linearly (through trilinear interpolation by the graphics hardware).

Because of those disadvantages, further research was targeted at the development of better methods. The availability of new GPUs with advanced programming capabilities (such as fragment shading) allowed us to design methods that exploited different approaches to achieve distortion. These methods also try to put less overhead on the CPU, in the search of an optimal balance between CPU and GPU use. We have used OpenGL Shading Language (*GLSL*) [142], in order to achieve graphics hardware independence and to ease maintenance of the code.

6.3 Direct Computation

In section 5.3.1 we explained why inverse mapping is needed to achieve seamless distortion, i.e. working from a distorted voxel space to obtain non-distorted coordinates. In practice, we can adapt this idea to use texture mapping: in this case the *distorted* voxel coordinates map to *original* texture coordinates and the *original* voxel coordinates map to *distorted* texture coordinates. Hence this method is based on the principle that it is

possible to achieve volume distortion by simply altering the texture coordinates of each fragment being drawn on screen. As the fragment shader gets the interpolated, original texture coordinates, it is simple to create a function that applies the bifocal distortion according to the region that the fragment is located in. This was first attempted with a 2D texture, to verify the feasibility of the process (listing 6.1).

The GLSL code starts by defining some **uniform** variables, which are values that can be changed by the user's program. Those variables are called uniform because they cannot be changed inside a primitive - in contrast with **varying** variables which represent interpolated values. The following are the uniform variables needed:

```
uniform sampler2D texUnit; // the texture unit
uniform vec2 focus;        // centre of focus region
uniform vec2 xlimits;      // region limits in x (xmin,xmax)
uniform vec2 ylimits;      // region limits in y (ymin,ymax)
uniform vec2 scalex;       // scaling factors for context regions (x)
uniform vec2 scaley;       // scaling factors for context regions (y)
uniform float mag;         // magnification factor
```

In the above code section, the *vec2* variables are arrays with two elements. Likewise, it is possible to use *vec3* or *vec4* if needed. The *sampler2D* type is used to specify a texture unit, which is necessary to later perform texture lookup. Now the first step is to get the original texture coordinates. This is done by accessing a varying variable called *glTexCoord[0],* which stores the interpolated texture coordinate for the first set - OpenGL supports multiple sets of texture coordinates.

```
vec2 tex = gl_TexCoord[0].xy;
```

Then a series of tests is performed to identify if those coordinates lie within the limits of the focus region in each coordinate direction (defined with *xlimits* and *ylimits*) - each test will give a result less than zero in that case. Thus for the *x* direction:

```
float xtest = (tex.x - xlimits[0])*(tex.x - xlimits[1]);
```

Now it is just a matter of using the result to identify the correct region: either before, within or after the focus region. Once that is done, the texture coordinate for the *x* axis is adjusted accordingly, following the model outlined in section 5.3.1.1.

```
if (xtest<=0) tex.x = (tex.x - focus.x + mag * focus.x)/mag;
else if(tex.x < xlimits[0]) tex.x = tex.x/scalex[0];
else tex.x = 1-((1-tex.x)/scalex[1]);
```

Then the same procedure is repeated to obtain the new *y* texture coordinate:

```
float ytest = (tex.y - ylimits[0])*(tex.y - ylimits[1]);
```

```
if(ytest<=0) tex.y = (tex.y - focus.y + mag * focus.y)/mag;
else if(tex.y < ylimits[0]) tex.y = tex.y = tex.y/scaley[0];
else tex.y = 1-((1-tex.y)/scaley[1]);
```

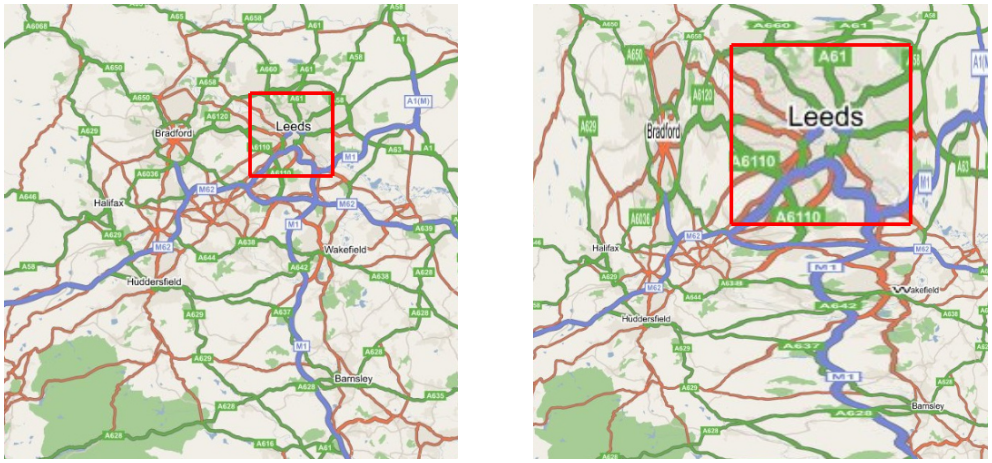
Finally, the last step is just to fetch the texture colour using the the new texture coordinates. The variable *gl_FragColor* is used to store the fragment colour so it will show on the screen.

```
gl_FragColor = texture2D(texUnit, tex);
```

Listing 6.1 presents the complete code and the results of this method are shown in figure 6.3 - they reproduce exactly the application of the mathematical method outlined in section 5.3.1.1. The 2D example of figure 6.3 is useful in itself (and has wider application as the figure shows) but the power of our approach is its application to 3D.

Listing 6.1: Distortion through direct computation of new texture coordinates with a fragment shader (2D)

```
uniform sampler2D texUnit;       // the volume texture unit
uniform vec2 focus;              // centre of focus region
uniform vec2 xlimits;    // region limits in x (xmin,xmax)
uniform vec2 ylimits;    // region limits in y (ymin,ymax)
uniform vec2 scalex;     // scaling factors for context regions (x)
uniform vec2 scaley;     // scaling factors for context regions (y)
uniform float mag;       // magnification factor
void main(void)
{
        vec2 tex = gl_TexCoord[0].xy;

        float xtest = (tex.x - xlimits[0])*(tex.x - xlimits[1]);
        if (xtest<=0) tex.x = (tex.x - focus.x + mag * focus.x)/mag;
        else if(tex.x < xlimits[0]) tex.x = tex.x / scalex[0];
        else tex.x = 1-((1-tex.x)/scalex[1]);

        float ytest = (tex.y - ylimits[0])*(tex.y - ylimits[1]);
        if(ytest<=0) tex.y = (tex.y - focus.y + mag * focus.y)/mag;
        else if(tex.y < ylimits[0]) tex.y = tex.y = tex.y / scaley[0];
        else tex.y = 1-((1-tex.y)/scaley[1]);

        gl_FragColor = texture2D(texUnit, tex);
}
```



(a) normal (b) distorted

Figure 6.3: 2D distortion using fragment shader (direct computation approach).

Thus logically, the next step is to attempt the same procedure with a 3D texture. It is simple to adapt the previous code to work with a 3D texture (see listing 6.2): first the *sampler2D* uniform variable becomes a *sampler3D* one, then focus must be specified by a 3D coordinate and finally, two uniforms are added - one for the z limits (*zlimits*) and another for the z scaling factors (*scalez*). Also in order to present a semi-transparent volume, the colour information must have an alpha component. This is accomplished through the use of a lookup table, in the form of a one dimensional RGBA texture (*sampler1D)*. Later the code is changed accordingly, so the fetched texture data is used as the index for this lookup table**.**

The rest of the code remains mostly the same, but a section is added after the computation of the y texture coordinate, in order to do the same procedure to the z texture coordinate:

```
float ztest = (tex.z - zlimits[0])*(tex.z - zlimits[1]);
if(ztest<=0) tex.z = (tex.z - focus.z + mag * focus.z)/mag;
else if(tex.z < zlimits[0]) tex.z = tex.z = tex.z / scalez[0];
else tex.z = 1-((1-tex.z)/scalez[1]);
```

The last step is to use the fetched texture data as the index for accessing the lookup table:

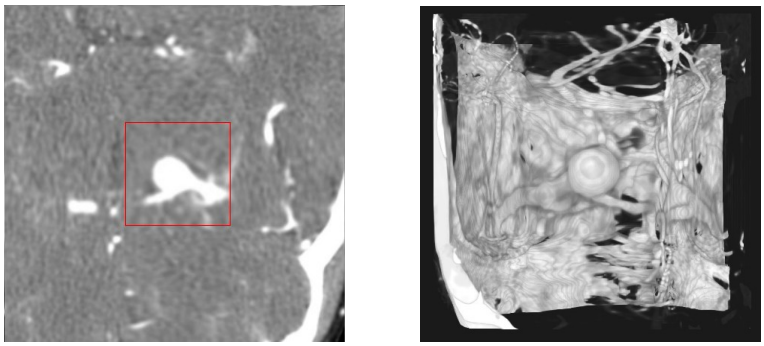
```
float luminance = texture3D(texUnit, tex).r;
gl_FragColor = texture1D(colourUnit,luminance);
```

Listing 6.2: Directly computing new texture coordinates (3D)

```
uniform sampler3D texUnit;        // the volume texture unit
uniform sampler1D colourUnit;     // the colourmap texture unit
uniform vec3 focus;               // centre of focus region
uniform vec2 xlimits;             // region limits in x (xmin,xmax)
uniform vec2 ylimits;    // region limits in y (ymin,ymax)
uniform vec2 zlimits;    // region limits in z (zmin,zmax)
uniform vec2 scalex;     // scaling factors for context regions (x)
uniform vec2 scaley;     // scaling factors for context regions (y)
uniform vec2 scalez;     // scaling factors for context regions (z)
uniform float mag;                // magnification factor
void main(void)
{
        vec3 tex = gl_TexCoord[0].xyz;

        float xtest = (tex.x - xlimits[0])*(tex.x - xlimits[1]);
        if (xtest<=0) tex.x = (tex.x - focus.x + mag * focus.x)/mag;
        else if(tex.x < xlimits[0]) tex.x = tex.x / scalex[0];
        else tex.x = 1-((1-tex.x)/scalex[1]);

        float ztest = (tex.z - zlimits[0])*(tex.z - zlimits[1]);
        if (ztest<=0) tex.z = (tex.z - focus.z + mag * focus.z)/mag;
        else if(tex.z < zlimits[0]) tex.z = tex.z / scalez[0];
        else tex.z = 1-((1-tex.z)/scalez[1]);

        float ytest = (tex.y - ylimits[0])*(tex.y - ylimits[1]);
        if(ytest<=0) tex.y = (tex.y - focus.y + mag * focus.y)/mag;
        else if(tex.y < ylimits[0]) tex.y = tex.y = tex.y / scaley[0];
        else tex.y = 1-((1-tex.y)/scaley[1]);

        float luminance = texture3D(texUnit, tex).r;
        vec4 colour = texture1D(colourUnit,luminance);
        gl_FragColor = colour;
}
```

Figure 6.4 presents the results, both in 2D and 3D of applying the technique to a medical dataset. The 3D view was generated by drawing all the slices (128 in this case) from the volume, in back to front order towards the viewer - this is the common approach used by a volume renderer. Although the 2D view is rendered with an adequate speed, the 3D view is not (see table 6.2 in section 6.9 for details). This happens because the number of fragments is much greater and the fragment shader has to compute the new texture coordinates for every single fragment. Being simply a direct application of the distortion

formulae, in practice this approach is not really suitable for execution on a GPU. It could be made faster by reducing the number of slices being drawn - but at the expense of less quality in the final image. An advantage of this method is that any function, be it distortion-oriented or not, could be potentially implemented by changing the code - but this is not simple nor flexible enough for practical use (and as we demonstrated, does not offer adequate performance).



(a) 2D view (b) 3D view

Figure 6.4: 3D distortion using fragment shader (direct computation approach).

6.4 Region Encoding

Fragment shaders are fast because they exploit the inherent parallelism of the GPU: multiple fragments are potentially processed simultaneously by the fragment processor and thus they can execute their code as fast as possible - this is like a SIMD[1] processing model. Long and complex fragment shaders take longer to execute and therefore, reduce the overall rendering speed. The code presented above also contains branches, which are costly because they prevent the fragment processor from doing exactly the same operations on all fragments - although the latest graphics hardware is now capable of true branching.

The common way of dealing with this limitation is through the use of auxiliary textures. As textures can encode anything and not just colour information, they can be used to do a number of different operations, including branching. This can be done as follows: consider that three new textures are created along with the volume texture. Each one of these will be used to encode the regions within the volume, for each dimension, using the RGB colour components. All textures will be one dimensional (1D), because they will be used to test single coordinates (x, y or z). Consequently, the length of the first texture will

[1] Single instruction, multiple data.

be the volume length; the length of the second texture will be the volume height and the length of the third texture will be the volume depth. For each pixel in one such texture, the R component will indicate if that pixel coordinate is within the focus region, the G component will indicate if that pixel coordinate is before the focus region and likewise, the B component will indicate if that pixel coordinate is after the focus region. This is encoded as a 0 (false) or 255 (true) colour component.

Considering a volume whose dimensions are 256 x 256 x 128, the focus centre at the centre in x and z, slightly off-centre in y (using a reasonably sized region) and the magnification factor as two, figure 6.5 presents an example of how these three textures would be filled:

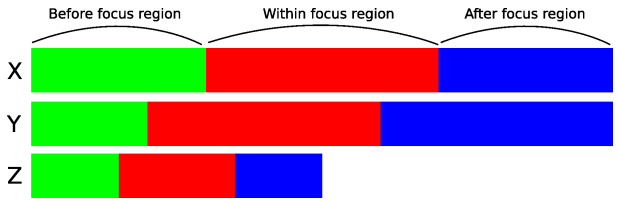


Figure 6.5: Auxiliary textures used to discover the region where a texture coordinate (x, y or z) lies.

Hence now the code is changed to account for the new textures. But it actually becomes simpler than before, as the region limits are no longer needed:

```
uniform sampler3D texUnit;     // the volume texture unit
uniform sampler1D xUnit;       // the x auxiliary texture unit
uniform sampler1D yUnit;       // the y auxiliary texture unit
uniform sampler1D zUnit;       // the z auxiliary texture unit
uniform sampler2D colourUnit; // the colourmap texture unit
uniform vec3 focus;            // centre of focus region
uniform vec2 scalex;           // scaling factors for context regions (x)
uniform vec2 scaley;           // scaling factors for context regions (y)
uniform vec2 scalez;           // scaling factors for context regions (z)
uniform float mag;             // magnification factor
```

In order to avoid the branching, the code now fetches a colour from the three auxiliary textures for x, y and z texture coordinates, respectively. Then it simply makes a multiplication of each colour component (i.e. the encoded region) by the corresponding distortion code for that region and adds all three multiplications together:

```
vec3 tex = gl_TexCoord[0].xyz;

// Fetch the encoded region data
vec4 rx = texture1D(xUnit, tex.x);
vec4 ry = texture1D(yUnit, tex.y);
```

```
vec4 rz = texture1D(zUnit, tex.z);

// Now apply the distortion according to the desired region
tex.x = (rx.r * (tex.x - focus.x + mag *focus.x)/mag) +
        (rx.g * (tex.x / scalex[0])) +
        (rx.b * (1-((1-tex.x)/scalex[1])));
tex.y = (ry.r * (tex.y - focus.y + mag *focus.y)/mag) +
        (ry.g * (tex.y / scaley[0])) +
        (ry.b * (1-((1-tex.y)/scaley[1])));
tex.z = (rz.r * (tex.z - focus.z + mag *focus.z)/mag) +
        (rz.g * (tex.z / scalez[0])) +
        (rz.b * (1-((1-tex.z)/scalez[1])));
```

This works because for each pixel in a texture, there will be always just a single colour component that will not be zero, so the remaining components will not have any effect on the sum. The visual results are exactly the same as before, but the rendering on older cards (such as the Quadro FX 3000) is actually a bit slower, even though there are no longer branches on the code. This can be explained by two reasons: first, there are more texture fetch operations per fragment (five: three to get the region data, one to get the intensity value and another one to perform colour lookup). Second, the overall complexity of the mathematical method is higher, as all terms of the equation will be computed even though two terms will always evaluate to zero. Because of these two reasons, we can formulate the hypothesis that the overhead and additional fetching operations are greater than the benefit of data paralellism. However, this balance changes for newer graphics cards (such as the GeForce 6200 and Quadro FX 4400, see section 6.9) and the method is slightly better than the one presented in section 6.3 for these cards.

6.5 New Volume Texture

Both previous approaches suffer from either lack of parallelism in the code design (direct computation method, section 6.3) or intensive computation of new texture coordinates (both). This leads to a slowdown during the rendering, and as fragment shading speed depends greatly on the number of pixels being drawn, full screen performance is greatly reduced (see section 6.9). Hence the need for a method that solves both shortcomings: code must be designed for correct GPU parallelism exploitation and the fragment shader itself must be simplified as well. The only way to accomplish that is by pre-calculating some of the required data beforehand, instead of doing everything inside the fragment shader - in a similar fashion to what was done in the region encoding method (section 6.4).

Now let us assume that the distortion fragment shader is applied just once, and the results stored in an auxiliary texture. Then it will be just a matter of performing the volume rendering process using that texture. This two-step method is actually possible, but some care must be taken as explained below:

1. To create the auxiliary texture, we render all the slices on the screen, as flat planes (whose size is exactly the slice size) and using the distortion fragment shader (see listing 6.2). The only change in the algorithm is that we draw using the original luminance (no colour mapping takes place at this step). For each slice rendered, we copy the screen contents to the auxiliary texture, to the correct position according to the slice depth.

2. Using the auxiliary texture as source volume, we perform volume rendering using a simple colour mapping fragment shader. This will just fetch the colours and opacities from the colourmap texture, for each fragment.

Step 1, which takes most of the time, must be repeated when any of the distortion parameters is changed (focus centre, size of focus region, magnification factor). This method would be much faster if it was possible to render the slice directly to the texture but at the moment this feature is not available on any graphics card[2]. Nevertheless, the visual results are exactly the same as the direct computation method, but the rendering itself is much faster - as the texture is only recomputed when the distortion parameters change, interactive viewpoint changes are now possible.

6.6 3D Offset Texture

In this section an alternative method is presented, in that (unlike 6.5) it uses the CPU (instead of the GPU) to pre-calculate some of the required data. A single auxiliary 3D texture is used to encode the **actual offset amount** required for each voxel in the volume, as the colour components R (x offset), G (y offset) and B (z offset) - the general idea for this was initially proposed by [43]. For simplicity, we are assuming that the auxiliary texture is capable of storing floating point values - this is supported on most current GPUs.

Considering that the volume dimensions are given by (*width*, *height*, *depth* - in voxel space), to build the auxiliary texture one must go through each voxel of the original volume (x_i, y_j, z_k) - ranging from $(i = 0...vmax_x, j = 0...vmax_y, k = 0...vmax_z)$ - and carry out the following procedure:

[2]The *framebuffer objects* extension only works for 1D and 2D textures at the moment of writing.

1. Compute the original texture coordinates, by dividing the voxel positions by the volume limits. This is necessary as the texture coordinates must be normalized:

$$\begin{aligned} x_{orig} &= \frac{x_i}{vmax_x} \\ y_{orig} &= \frac{y_j}{vmax_y} \\ z_{orig} &= \frac{z_k}{vmax_z} \end{aligned}$$

2. Compute the distorted texture coordinates, applying the distortion formulae to the original ones. Let us consider just the x coordinate - in the following, x_f is the focus region centre, sx is half the size of the focus region and $scale_{x+}, scale_{x-}$ are the scaling factors. The same process is applied to y and z:

$$x_{dist} = \begin{cases} \frac{x_{orig}}{scale_{x-}} & x_{orig} < x_f - sx \\ 1 - \frac{1-x_{orig}}{scale_{x+}} & x_{orig} > x_f + sx \\ \frac{x_{orig} - x_f + mag \cdot x_f}{mag} & x_f - sx \leq x_{orig} \leq x_f + sx \end{cases} \tag{6.1}$$

3. Compute the *differences* between the original and distorted texture coordinates:

$$\begin{aligned} \Delta x &= x_{orig} - x_i \\ \Delta y &= y_{orig} - y_j \\ \Delta z &= z_{orig} - z_k \end{aligned}$$

4. Store the computed differences as R,G,B colour components for the corresponding voxel:

$$\begin{aligned} R_{ijk} &= \Delta x \\ G_{ijk} &= \Delta y \\ B_{ijk} &= \Delta z \end{aligned}$$

Once the auxiliary texture has been successfully built, the fragment shader just needs to fetch the offset amount for each voxel in each direction (x, y and z), subtract them from the original texture coordinates and use the resulting values to fetch the actual volume voxel (listing 6.3). Note that the actual implementation is more optimized than this description suggests, as it uses vector operations of the shading language.

Listing 6.3: Distortion using a floating-point auxiliary texture as offset

```
uniform sampler3D texUnit;       // the volume texture unit
uniform sampler3D deltaUnit;     // the displacement texture unit
uniform sampler1D colourUnit;    // the colourmap texture unit
void main(void)
{
        // Obtain the original texcoord
        vec3 tex = gl_TexCoord[0].xyz;

        // Fetch and subtract the displacement
        tex -= texture3D(deltaUnit, tex);

        // Fetch the luminance from the volume texture
        float luminance  = texture3D(texUnit, tex).r;

        // Finally, fetch the colour from the colourmap
        gl_FragColor = texture1D(colourUnit,luminance);
}
```

Like the method presented in 6.5, this is much faster than the direct computation method but there are two disadvantages:

1. More processing time is needed to actually generate the 3D offset texture, whereas the previous method uses the GPU and is considerably faster. The GPU itself could still be used to compute the offset texture, but the rendered data would have to be copied back to the offset texture. However, as already mentioned in section 6.5, the current generation of GPUs do not support this operation (*framebuffer objects*) with 3D textures.

2. More texture memory is required by this method, as the offset texture must contain floating-point RGB data.

Having said that, this method allows for the design of **any** volume distortion function, not being limited to the bifocal distortion presented here.

However, if the GPU being used does not have support for floating point textures, an adaptive method can be used instead. The procedure is split into two passes: during the first pass, we find out the minimum ($min_{x,y,z}$) and maximum ($max_{x,y,z}$) difference values within the dataset. This is accomplished by performing the first three steps of the original procedure. Then in the second pass we change the fourth step to the following, which converts the floating point difference to a signed byte value, considering the needed range for each coordinate:

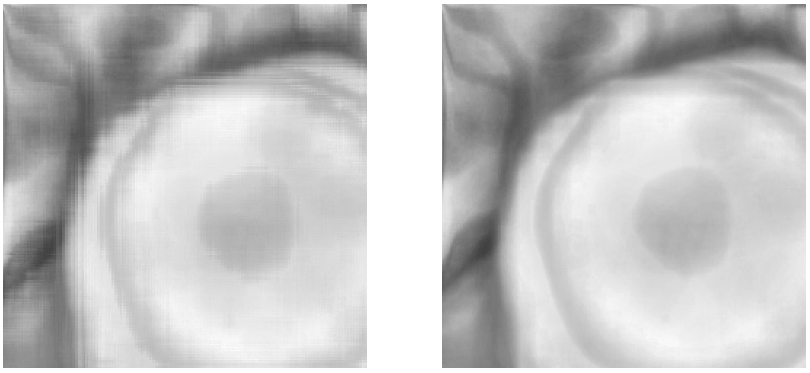
$$
\begin{aligned}
R_{ijk} &= \frac{\Delta x - min_x}{max_x - min_x} \cdot 255 \\
G_{ijk} &= \frac{\Delta y - min_y}{max_y - min_y} \cdot 255 \\
B_{ijk} &= \frac{\Delta z - min_z}{max_z - min_z} \cdot 255
\end{aligned}
$$

Now the fragment shader must be adapted as well, because it will have to convert the offset (stored as signed bytes) to the correct range before subtracting it from the original texture coordinates. The range is defined by the uniform vectors *mindif.xyz* and *maxdif.xyz*. (listing 6.4):

Listing 6.4: Distortion using a signed byte auxiliary texture as offset

```
uniform sampler3D texUnit;      // the volume texture unit
uniform sampler3D deltaUnit;    // the displacement texture unit
uniform sampler1D colourUnit;   // the colourmap texture unit
uniform vec3 mindif, maxdif;    // the min/max texcoord diffs
void main(void)
{
        // Obtain the original texcoord
        vec3 tex = gl_TexCoord[0].xyz;

        // Fetch the texture displacement
        disp = texture3D(deltaUnit, tex);

        // Fetch, convert to the the correct
        // floating point range and subtract
        tex -= (maxdif - mindif)*disp.xyz + mindif;

        // Fetch the luminance from the volume texture
        float luminance  = texture3D(texUnit, tex).r;

        // Finally, fetch the colour from the colourmap
        gl_FragColor = texture1D(colourUnit,luminance);
}
```

This adaptive method has a further disadvantage: in cases of extreme magnification (either a large magnification amount or getting too close to the volume) some artifacts will start to show up due to the limited precision of the signed byte representation (see figure 6.6). On the other hand, the signed byte texture (one byte per voxel) requires much less memory than the floating point texture (either two or four bytes per voxel, depending on floating point precision).



(a) Adaptive version (b) Floating point version

Figure 6.6: Distortion using a 3D offset texture: close view of the focus region, where sampling artifacts can be seen in the adaptive version of the method (which uses a signed byte representation).

6.7 3 x 1D Offset Textures

This approach is in fact a simplification of the previous method outlined in section 6.6: instead of using a single, three-dimensional texture to encode the offset for each voxel, it is possible to use three unidimensional textures, one for each coordinate axis (also published in [32]). Thus the size of the first texture will be the volume width, the size of the second texture will be the volume height and the size of the third texture will be the volume depth. Also it is important to note that in this method, the textures only need to have a single colour component, which normally is used to store intensity values (this is called a *gray scale texture*).

To build each of these textures, we must go through each coordinate and carry out a procedure similar to that was previously used in the 3D offset method (section 6.6). Therefore, for the x axis texture the procedure would be as follows:

1. Compute the original texture coordinate, by dividing the current x coordinate by the x axis limit:

$$x_{orig} = \frac{x_i}{vmax_x}$$

2. Compute the distorted texture coordinate, applying the distortion formulae from equation 6.1 to the original one;

3. Compute the *difference* between the original and distorted texture coordinate:

$$\Delta x = x_{orig} - x_i$$

4. Store the computed difference as an intensity colour component for the corresponding position:

$$I_i = \Delta x$$

For the y and z axis, the procedure would be exactly the same. Listing 6.5 presents the new method.

The main advantages of this method in contrast with the 3D offset one are the increased speed to generate the textures (time is negligible) and low memory consumption. However, this method is less flexible than the previous one: each original x coordinate will always generate the same distorted x coordinate (same for y and z). In practice, this allows the modelling of distortions like bifocal and fisheye, but other effects - such as the volume lens - would not be possible, as they would require that some regions have no distortion at all. This disadvantage was solved through the extension of this method, as we describe in the next section.

Listing 6.5: Distortion using three unidimensional auxiliary textures to compute the offset

```
uniform sampler3D texUnit;       // the volume texture unit
uniform sampler1D xUnit;         // the x displacement texture unit
uniform sampler1D yUnit;         // the y displacement texture unit
uniform sampler1D zUnit;         // the z displacement texture unit
uniform sampler1D colourUnit;    // the colourmap texture unit
uniform vec3 mindif, maxdif;     // the min/max texcoord diffs
void main(void)
{
   vec3 tex = gl_TexCoord[0].xyz;
   vec3 disp;
   // Grab the texture displacements
   // and subtract it from the original texcoord
   disp.x = texture1D(xUnit, tex.x).x;
   disp.y = texture1D(yUnit, tex.y).x;
   disp.z = texture1D(zUnit, tex.z).x;
   tex -= disp;

   // Grab the luminance from the volume texture
   // and fetch the colour from the colourmap
   float luminance = texture3D(texUnit, tex).r;
   gl_FragColor = texture1D(colourUnit,luminance);
}
```

6.8 Extended 3 x 1D Offset Textures

Following all the previous research, this method aims to implement the complete focus and context framework as described in chapter 5. In order to achieve this, we have chosen the method that offered the best compromise in flexibility versus performance (for a comparative analysis see section 6.9): that is the 3 x 1D offset method (section 6.7). The method was extended in the following way: instead of using three floating-point luminance textures, we now employ three RGBA floating-point textures, again one for each dimension. The key idea is that all effects of the framework for a specific dimension (mapping, highlighting and attenuation) can be mapped to a colour component on the corresponding texture (see figure 6.7).

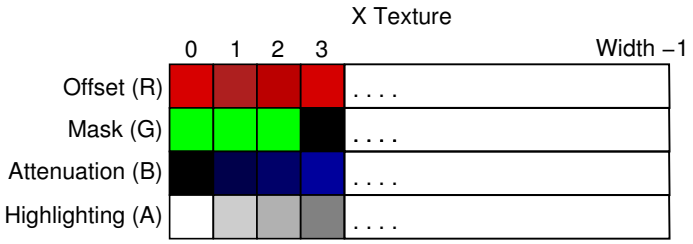


Figure 6.7: Mapping the framework effects to a 1D texture: *offset* indicates the texture difference between original and distorted coordinates; *mask* provides a mask for applying the offset; *attenuation* and *highlighting* store the attenuation/highlighting values for this particular voxel, respectively.

The mapping effect is naturally stored as the offset - red colour component - as in the 3 x 1D method. At the same time, a *mask* value is also created, which indicates whether the effect should be applied for that particular voxel or not. The mask is then stored in the green colour component of the corresponding texture. For example, in the case of the bifocal and fisheye distortions, the mask is always 1, as the effect will be applied throughout the volume. But in the case of the volume lens, the effect must be only applied inside the focus region, so the mask will contain 0 on all voxels outside it. The attenuation and highlight values are then computed, according to the selected functions (e.g. linear, quadratic or cubic and whether using viewpoint-based attenuation or not - see sections 5.3.2 and 5.3.3 for details) and stored in the blue and alpha colour components of the texture.

Once the textures have been created, the fragment shader then performs the following steps:

1. Fetch the encoded effects for x, y and z at the present voxel.
   As the textures are 1D, we must fetch the encoded effects for each dimension, using the corresponding texture coordinate.

```
vec3 tex = gl_TexCoord[0].xyz;
vec4 xdisp = texture1D(xUnit, tex.x);
vec4 ydisp = texture1D(yUnit, tex.y);
vec4 zdisp = texture1D(zUnit, tex.z);
```

2. Compute the final offset values for x, y and z.
   If the voxel is within the region defined by the 3D mask (i.e. within each of the 1D regions), then the mapping is applied.

```
vec3 disp;
disp.x = xdisp.r * ydisp.g * zdisp.g;
disp.y = ydisp.r * xdisp.g * zdisp.g;
disp.z = zdisp.r * xdisp.g * ydisp.g;
```

3. Obtain the new texture coordinates, by adding the final offset values for x, y and z to the corresponding original texture coordinate.

```
tex -= disp;
```

4. Fetch the voxel luminance using the new texture coordinates.

```
float luminance = texture3D(texUnit, tex).r;
```

5. Compute the resulting attenuation factor by doing component-wise multiplication of the attenuation values for x, y and z.

6. Compute the resulting highlighting factor by doing component-wise multiplication of the highlight values for x, y and z. Note that steps 5 and 6 are optimized as a single operation by the vector operations available in GLSL:

```
vec2 litealpha = xdisp.ba * ydisp.ba * zdisp.ba;
```

7. Obtain the colour from the colour map and combine it with the highlight colour, using the resulting highlighting value.

```
vec4 basecolour = texture1D(colourUnit, luminance);
basecolour -= lite_colour * litealpha.g;
```

8. Store the final colour, setting the final alpha as the original alpha of the voxel, multiplied by the resulting attenuation value.

```
gl_FragColor = vec4(basecolour.rgb, basecolour.a * litealpha.r);
```

The content of the textures must be computed again every time one of the effects is changed, but this has a negligible impact on the application performance.

6.9 Comparative Analysis

This section presents a comparative analysis of all methods discussed in the previous sections. First we consider the memory requirements and flexibility of each method: as seen in previous sections, some methods require considerable memory so this must be taken into account. By flexibility we mean how capable each method is to achieve the framework effects, and at what cost. Table 6.1 presents this information: the *Memory* column indicates how much memory is needed for auxiliary data/textures - for the purpose of this table, we assume that the volume dimensions are 256 x 256 x 128, gray scale data. The other columns indicate whether the effect can be implemented with the respective method or not.

Method	Memory	Bifocal	Fisheye	Vol. Lens	Attenuation	Highlighting
Bricking and Scaling	> 16 Mb[1]	Yes	No	No	No	No
Direct Computation	0 bytes	Yes	Yes[8]	Yes[8]	Yes[8]	Yes[8]
Region Encoding	2304 bytes[2]	Yes	Yes[8]	Yes[8]	Yes[8]	Yes[8]
New Volume	16 Mb	Yes	Yes[8]	Yes[8]	Yes[8]	Yes[8]
3D Offset (int)	48 Mb[3]	Yes	Yes	Yes	No	No
3D Offset (float)	96 Mb[4]	Yes	Yes	Yes	No	No
3 x 1D Offset (int)	768 bytes[5]	Yes	Yes	No	No	No
3 x 1D Offset (float)	1536 bytes[6]	Yes	Yes	No	No	No
Extended 3 x 1D Offset	3072 bytes[7]	Yes	Yes	Yes	Yes	Yes

Table 6.1: Memory requirements and flexibility offered by each distortion method.

[1] At least 16 Mb, as the number of bricks depend on the exact region dimensions.

[2] 3 textures x 256 elements x 3 bytes (RGB)

[3] Original volume x 3 bytes (RGB)

[4] Original volume x 3 bytes (RGB) x 2 bytes (assuming half-precision floats)

[5] 3 textures x 256 elements

[6] 3 textures x 256 elements x 2 bytes (assuming half-precision floats)

[7] 3 textures x 256 elements x 4 bytes (RGBA)

[8] These methods would be possible by modifying the fragment shader

Each method in the table is described below:

- Bricking and scaling: splitting the volume into 27 regions and drawing each one with different scaling and position (see section 6.2);
- Direct computation: directly applying the bifocal distortion formulae in the fragment shader (see section 6.3);
- Region encoding: using three 1D textures to compute the region of each fragment (see section 6.4);
- New volume: using a auxiliary texture to store the distorted result (see section 6.5);
- 3D Offset (int/float): using a auxiliary 3D texture to store the offset (difference between original and distorted texture coordinates - see section 6.6);
- 3 x 1D Offset (int/float): using three 1D textures to encode the offset, instead of a single 3D texture (see section 6.7);
- Extended 3 x 1D Offset: extension of the previous method to allow highlighting and attenuation effects (see section 6.8).

As seen in the table, the direct computation method and region encoding methods (which is based on the former) do not have high memory requirements and could be capable of implementing the entire framework, but that would require the fragment shader to be changed - and therefore performance would be affected. The new volume method, although based on direct computation as well, requires double the original volume memory, so for large volumes it would not be feasible to use it. Finally, with the 3D offset method it would be possible to implement any mapping effect, but the method needs either two or six times the original volume memory and the other effects would not be achievable. So from this table alone, the 3 x 1D offset method and its improvement, the extended 3 x 1D offset, offer a good compromise between memory requirements and flexibility - the latter, still requiring very little memory, allows the complete framework to be implemented.

Nevertheless, to have a clear picture of the advantages and disadvantages of each method, we must also compare their performance. To obtain a spread of performance data, we have taken measurements of frame rate with three different combinations of graphics cards and machines, each roughly corresponding to one of the following levels:

- entry-level graphics: GeForce 6200 running on an Athlon XP 2000+ with 512 Mb RAM;

- mid-range graphics: Quadro FX 3000 running on a dual Xeon 1.7 GHz with 2 Gb RAM[3];
- high-end graphics: Quadro FX 4400 running on a quad Xeon 3.4 GHz with 4 Gb RAM.

Table 6.2 presents the results, obtained with all methods in each machine level above. We used the same example dataset (256 x 256 x 128 voxels). The *Pre* column indicates the time for preprocessing, usually to generate the distortion or to create auxiliary data (in milliseconds, *0** meaning less than 1 ms); and the *Render* column indicates the rendering frame rate, considering a window size of 640 x 480 pixels. Finally, some methods have been implemented both using signed byte values (int) and floating point values (float, where available) - which also produced different results.

Table 6.2: Performance comparison of distortion methods: machine level indicates each graphics card tested (GeForce 6200, Quadro FX 3000 and FX 4400), *Pre* specifies the preprocessing time (if relevant), *Render* is rendering performance in frames per second (averaged from 30 runs with normal system load). Blank cells denote that the specific method is not supported by the hardware and *n/a* indicates that the measurement is not applicable to the method.

Method \ Machine level	GF 6200		FX 3000		FX 4400	
	Pre	Render	Pre	Render	Pre	Render
Bricking and scaling	2925	12.4	2400	14.2	1600	47.4
Direct computation	n/a	8.7	n/a	2.4	n/a	21.3
Region encoding	0*	10.6	0*	2.1	0*	25.5
New volume	2363	29.1	1283	18.2	830	67.5
3D Offset (int)	1784	23.2	876	10.5	560	52.5
3D Offset (float)	—	—	—	—	548	51.3
3 x 1D Offset (int)	0*	23.3	0*	8.2	0*	52.5
3 x 1D Offset (float)	—	—	—	—	0*	59.1
Extended 3 x 1D Offset	–	–	–	–	0*	50

From table 6.2, it is clear that the worst method in all levels is the direct computation one: by not exploiting at all the parallel processing capabilities of the graphics hardware, it has achieved a performance even worse than the bricking method, which has a comparatively long preprocessing step. It is also worth noting that the performance of this method on the entry-level card (GeForce 6200) was much better than on the mid-range

[3]the Quadro FX 3000 used to be high-end, but as it is one generation behind the Quadro FX 4400 (at the time of writing), we consider it mid-range.

card (Quadro FX 3000). This can be explained due to advances in fragment shading hardware - the GeForce 6200, although being a budget graphics card, can run fragment shaders more efficiently than the Quadro FX 3000. But in order to find the best method (in terms of performance), a careful analysis must be done, considering that a "method" is comprised of two separate steps: preprocessing and rendering.

Purely in terms of preprocessing, the best methods are the 3 x 1D offset, extended 3 x 1D offset and region encoding methods: the last has no preprocessing time and the others show a negligible preprocessing time (less than 1 ms). But while the region encoding method loses in performance, the others offer the best interactive rates of update for the distortion parameters (e.g. region and magnification factor). However, they do not provide the best possible rendering speed, requiring additional computations on the fragment shader. Also if we compare the rendering performance of either 3D offset method versus the corresponding 3 x 1D offset method on the same machine level, we find that there is not much difference - apart from the mid-range machine, where the 3 x 1D offset method was almost 20% slower than the 3D offset one. The reason for this is again the less efficient fragment shader implementation on the Quadro FX 3000, where the additional texture fetches slow down the processing. Ultimately, all methods (except for the bricking and scaling) perform worse on the Quadro FX 3000 due to the same reason.

Even so, if we consider just the raw rendering performance then the new volume method always wins: by pre-computing an entirely new 3D texture every time the distortion parameters change, the fragment shader is only used to perform colourmap lookup and thus is very efficient. But we must bear in mind that the usefulness of a method will be dictated by the two figures combined: what is more important, having less preprocessing time or more rendering speed ? Also we must not forget that even if a method offers high performance and little preprocessing time, the flexibility and memory requirements presented earlier must be considered as well.

Therefore we have demonstrated that it is not possible to achieve the best in both aspects, but nevertheless the methods which offer the optimal compromise are the ones based on offset textures. These methods provide a relatively good rendering speed in all machine levels - less than ideal on the Quadro FX 3000, but we consider it outdated hardware now - and more importantly, are able to achieve a seamless, real-time interaction experience for the manipulation of distortion parameters.

Taking all of these ideas into account, we have decided to implement the full range of the framework effects using the 3 x 1D offset method as a base, as that was the one which offered the best compromise between memory, flexibility, preprocessing time and rendering performance - the result was the extended method presented in section 6.8. From table 6.2, it is clear that even including the attenuation and highlighting effects

(which increase the complexity of the fragment shader), the good performance of the 3 x 1D offset method is maintained. Additionally, we have carried out measurements with this method and different datasets - these are shown in table 6.3.

Dataset	Size	Extended 3 x 1D	Conventional renderer
Aneurysm (cropped)	287 x 311 x 89	51	140
Aneurysm (full)	512 x 512 x 146	18	43.6
Large aneurysm	512 x 512 x 190	13	34.8
Small aneurysm	512 x 512 x 183	13.6	35.8

Table 6.3: Performance comparison of the extended 3 x 1D offset method and a conventional texture-based volume renderer, running on the high-end machine and displaying the volume in a 640 x 480 window (values in frames per second).

We did not experiment with the other methods at this point, as the memory requirements for some of those would be much higher (as we already demonstrated, due to the large size of the original datasets) and because we were interested in achieving the maximum flexibility possible (i.e. implementing all framework effects), while still obtaining interactive rendering performance. We also note that any recent graphics card would be able to implement the extended 3 x 1D method, like e.g. the GeForce 6 or 7 series (with the exception of the lower end cards, which do not support floating point textures).

6.10 Summary

In this chapter we presented the chronological research on hardware-accelerated methods carried out to implement the focus and context framework described in chapter 5. At the end, we offered a comprehensive comparative analysis that justified why the extended 3 x 1D offset method is the only one really suitable to be used in the volume rendering system, as that method is capable of providing all the framework effects. Next, chapter 7 will describe the user interface and actual development issues of the *VolFocus* system.

Chapter 7

VolFocus User Interface

7.1 Overview

FROM THE VERY beginning of this research, an important issue in the *VolFocus* system was its user interface: as already mentioned in the motivation for this work (see section 1.2), the neurosurgeon lacked a tool that was easy to use, flexible enough to handle different medical formats and able to present information in a clear and effective way.

The design principles that went into the user interface are outlined as follows:

1. Clear display of information such as crucial detail showing the correct left/right orientation for slices (which in medical systems is normally inverted).

2. Ease of use: no complicated commands, menus, preferences or options.

3. Mouse operation: this is essential if the system is to be used in the operating theatre, as a keyboard would be cumbersome and impractical (although the mouse is not the best input device to be used in a sterile environment, see section 9.6.3).

4. Facility to handle different image formats: the system must be able to read DICOM data, the standard for medical data, and other common formats such as sets of JPEG or PNG images.

7.2 Evolution of the GUI

In the following sections the design issues, changes and adaptations that the graphics user interface (GUI) has undergone since the first version are described in detail (for a table of the actual milestones see section B.2). The pictures were captured from the Linux version of the system - note that it can also be compiled and run on any platform where the *wxWidgets* library and OpenGL are available (e.g. Windows).

7.2.1 General Interface

From the onset, the system was designed to offer an interface based on the commonly used concept of four viewing panes: three orthographic (slice) views and one 3D view showing the volume. The slice views are labelled according to medical terminology as axial (top), saggital (lateral) and coronal (frontal). Each one can also be maximized and displayed in full screen mode - this is useful when one wants, for example, to focus one's attention on a specific view (axial, for example). Figure 7.1 presents the general system interface:

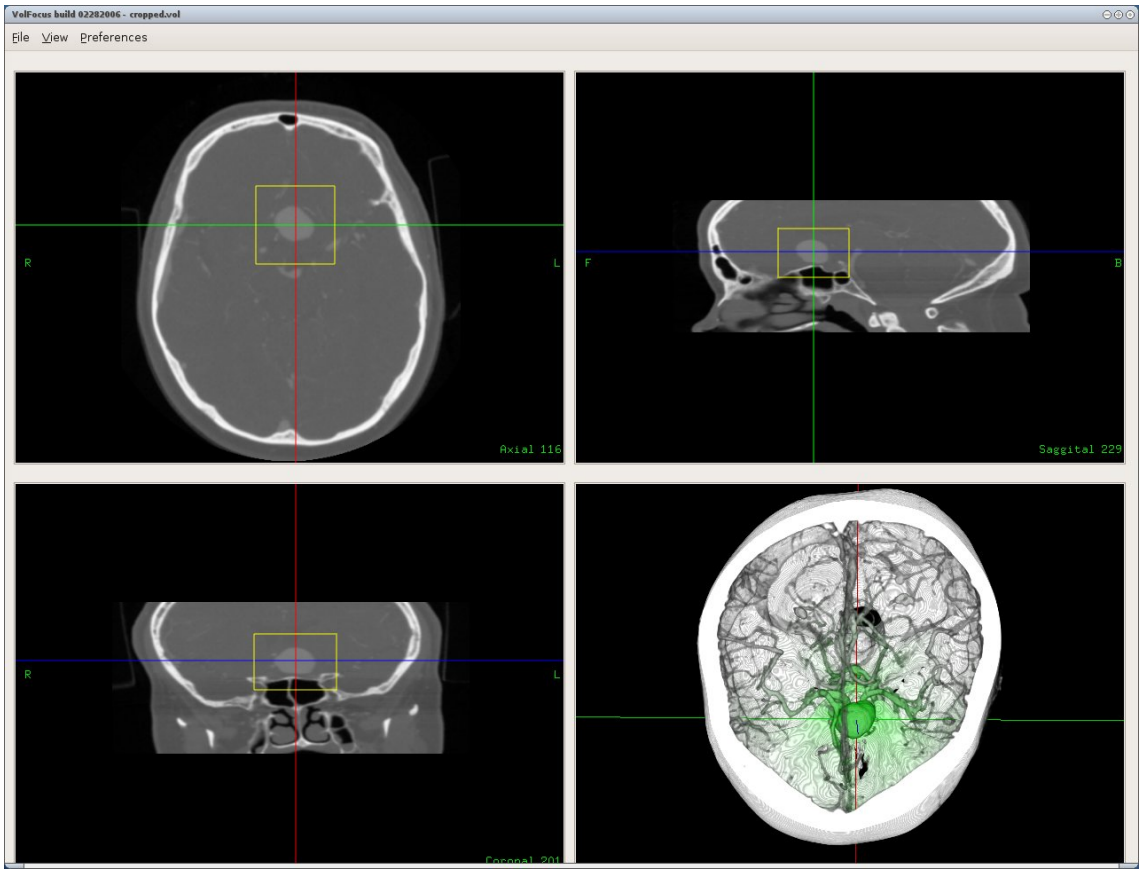


Figure 7.1: *VolFocus* user interface: general view.

The interface has a conventional main menu to perform common operations: the *File* menu (figure 7.2) allows the user to load and save volume data, colour maps and *system snapshots* - which are files that store the system configuration: filename of the current volume, colour map and all framework parameters selected by the user.

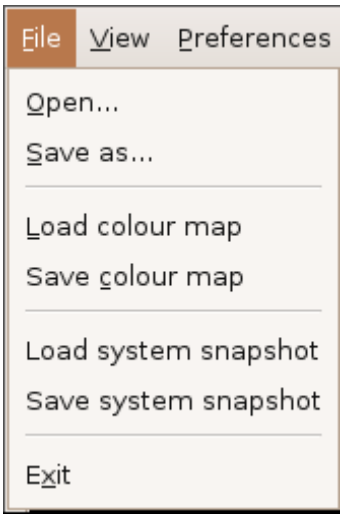


Figure 7.2: *VolFocus File* menu.

The loading of volumes is particularly important: as stated in section 7.1, it had to be flexible enough to allow the importing of different data formats, including sets of individual images such as JPEG or PNG. The first data format that the system was able to load was the one natively used by the OSCVR library, called VOL: this format uses a text file with a header (*.vol*) that describes the volume (size, channels, etc), while the actual volume is stored as raw (binary) data in a separate file - see section B.3.1 for a description of the file format.

However, it takes some effort to convert existing data to this format - especially if the data is presented as a set of images. In this case, one would have to convert the data to raw files, concatenate them into a volume file and write an appropriate header for it. Hence the system allows the user to load sets of images in formats such as JPEG or PNG as a volume. Additionally, the DICOM image format is also supported through the *MedCon* library [120]. This functionality is extremely important, as frequently the medical professional will just have images extracted from the scanner equipment and would like to see them as a volume, without needing to convert them. The system requires all images from the same set to be in a specific directory, so the user will just need to select one of the images from the list (figure 7.3):

After user selection of one file, the system automatically builds the list of image files to be loaded using the following template:

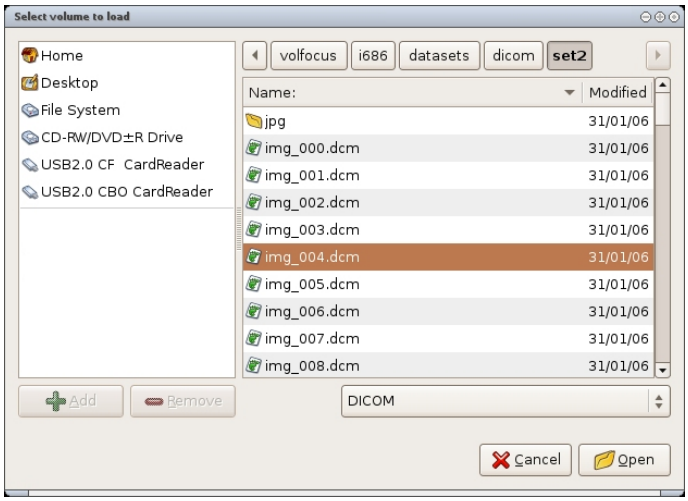


Figure 7.3: Loading a set of DICOM images in*VolFocus* (Linux version).

- The *basename* is comprised of the substring from the start of the filename to the first numeric or underscore "_" character;
- The *number of significant digits* is obtained from the position after the first numeric/underscore character to the dot preceding the suffix;
- The *suffix* indicates the type of image being read (in the figure, *dcm*).

Then it is just a matter of going backwards in the directory until we find the first image that respects the above template and then going forwards until we find the last one. Once this range is found, the system loads each image in sequence, constructing the full volume in memory.

The File menu also allows the user to save the current volume in the VOL format mentioned earlier - this is helpful as it enables the user to keep the entire volume in just two files and also is faster to load than individual images. Loading and saving colour maps is also achieved through this menu, and a simple text file format is used (*.lut*, see section B.3.2). Finally, it is possible to save and restore the entire system state, which is stored in a snapshot file (*.sna*, see section B.3.3).

As for the *View* menu (figure 7.4), it allows the user to see the system log window (where debugging messages are shown) and to view textual data extracted from DICOM files - in this case, the system shows the data present on the first DICOM slice that was loaded.

The last menu - *Preferences* - provides common options (such as setting the background colour) and since milestone 10 (December 2005) it includes settings for the frame-

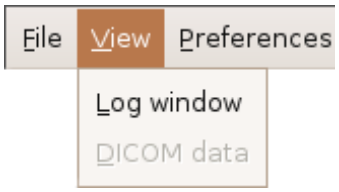


Figure 7.4: *VolFocus View* menu.

work mapping, attenuation and highlighting effects. To set the mapping effect, one just has to select the desired effect in the *Distortion* sub-menu (figure 7.5):

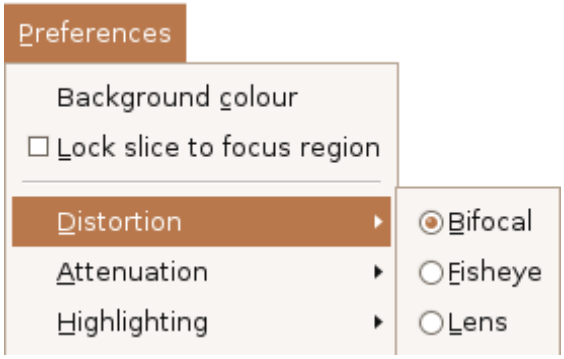


Figure 7.5: *Distortion* options in *VolFocus*.

The *Lock slice to focus region option* was added after the evaluation, following a suggestion from the interventional radiologist (see section 8.5.2 for details): this forces the slice views to be coordinated to the centre of the focus region, i.e. when the focus region is moved in a particular view, the centre of focus is used to update the current slice in the other 2D views.

The *Attenuation* and *Highlighting* menus, shown in figure 7.6, allow the user to activate/deactivate each effect and to select the desired function: linear, quadratic or cubic. Also in the *Attenuation* menu, there is an option to enable/disable the viewpoint-based attenuation mode - when this is active, the power of the attenuation function is controlled by the turning the mouse wheel while in the volume view. The mouse wheel is also used in combination with any mouse button: in this case, it adjusts the percentage of the distance from the edges of dataset to the start of the attenuation region.

7.2.2 Slice Views

The slice views present relevant information simultaneously: display of the current slice (figure 7.7A), a message and notification area (B), slice orientation (C), axes indicating the current slice in the other views (D), and a self-hiding toolbar of transparent “buttons”

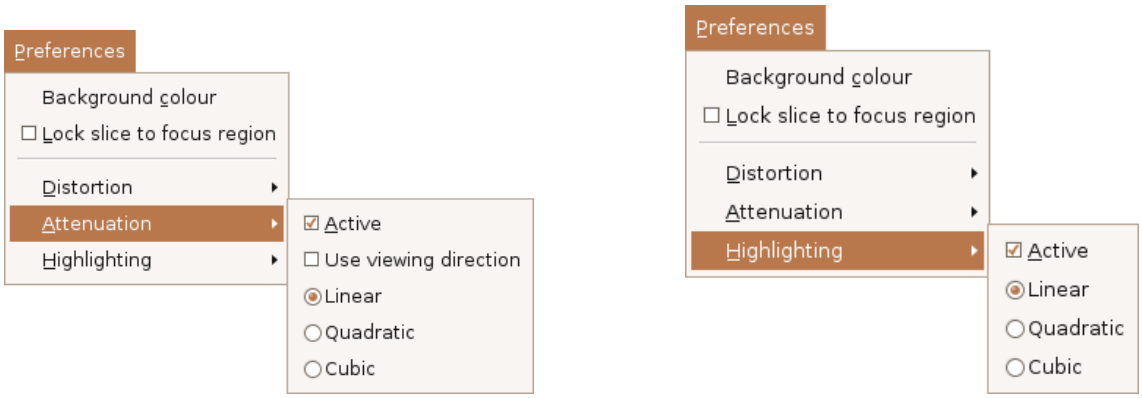


Figure 7.6: *Attenuation* and *highlighting* sub-menus.

(E), used to indicate specific system states. Some of these elements, such as the button interface, are explained later in this section.

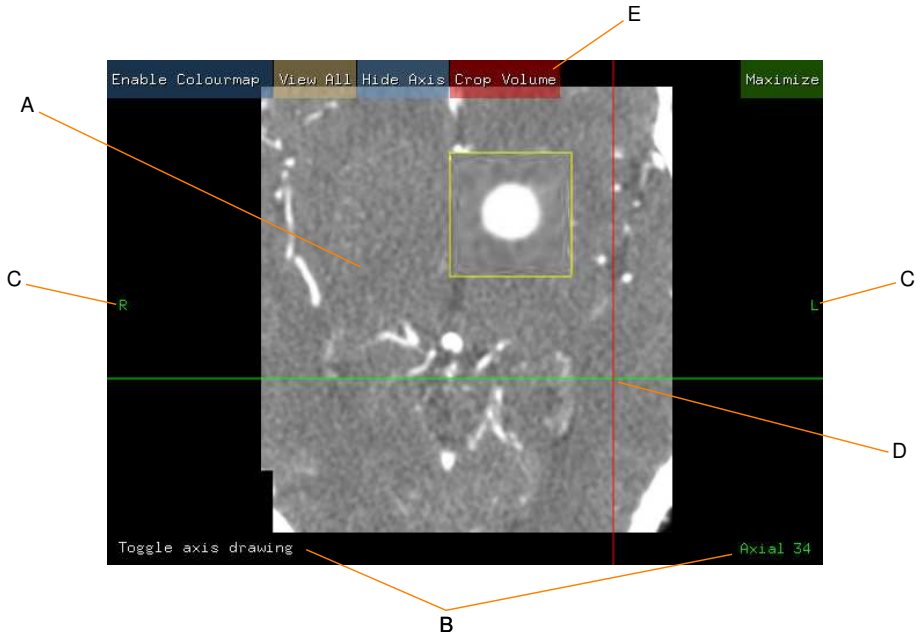


Figure 7.7: Slice view interface elements: slice display (A), message/notification area (B), slice orientation (C), slice axes (D), and toolbar (E).

The use of the mouse, as already mentioned in section 7.1, was one of the important design decisions: if the prototype was to be used in the operating theatre, this had to be simple, effective, and easy to learn and remember. We proposed the following navigation scheme:

- left button and drag up/down: change the currently displayed slice
- middle button and drag: pan the slice view

- right button and drag up/down: zoom in/out the slice view

From milestone 5 (June 2004, see section 7.2.4 for details), the right button + drag combination was also used to change the magnification factor when the mouse pointer was inside the focus region.

The idea of the button toolbar came from the fact that we wanted to maximize the screen space available to display the slice - a conventional toolbar would take up precious space. The toolbar only appears when the mouse pointer is within the top region of the window (i.e. the region occupied by the toolbar when displayed) and as the buttons are semi-transparent, they do not get in the way of the underlying display. The visual representation of the buttons evolved over time, from a very simple row of labels to reactive buttons that can act as toggle or action buttons, and can also indicate different states. This evolution is displayed in figure 7.8:

Figure 7.8: Slice view toolbar evolution: small buttons (A), renamed labels and better colouring (B), final version (C).

In (A), the buttons used in milestone 5 (June 2004) are shown:

- *NAV*: navigation mode, the default mode of operation
- *ROI:* region of interest mode, where the ROI would be displayed and could be changed by the user
- *ALL:* reset the view to display the entire slice
- *AXIS:* toggle axis drawing
- *F+C:* enable/disable focus and context mode
- *MAX/MIN:* maximize/minimize the window

At that point, we intended to create icons and replace the labels, but we ended up enlarging the buttons and renaming the labels - as the consultant neurosurgeon suggested that even icons could be confusing for new users. This happened in milestone 8 (June 2005), shown in (B):

- *Mode: Normal/Distorted:* toggle between normal and focus and context mode
- *View All:* same as *ALL*
- *Hide/Show Axis:* same as *AXIS*
- *Maximize/Minimize:* same as *MAX/MIN*

The final change (C) took place in milestone 10 (December 2005), with the inclusion of the *Crop Volume* button (see section 7.2.4 for details) and substitution of the *Mode* button by the *Enable/Disable Colourmap* button, which applied the current colour map to the slice being displayed. The *Mode* button was no longer necessary, as the system ran always in focus and context mode.

7.2.3 Volume View

The volume view presents information in a very similar way to the slice views: the central area displays the volume (figure 7.9A), there is also a message and notification area (B), a set of axes indicating the current slice in the other views (C), and a toolbar (D).

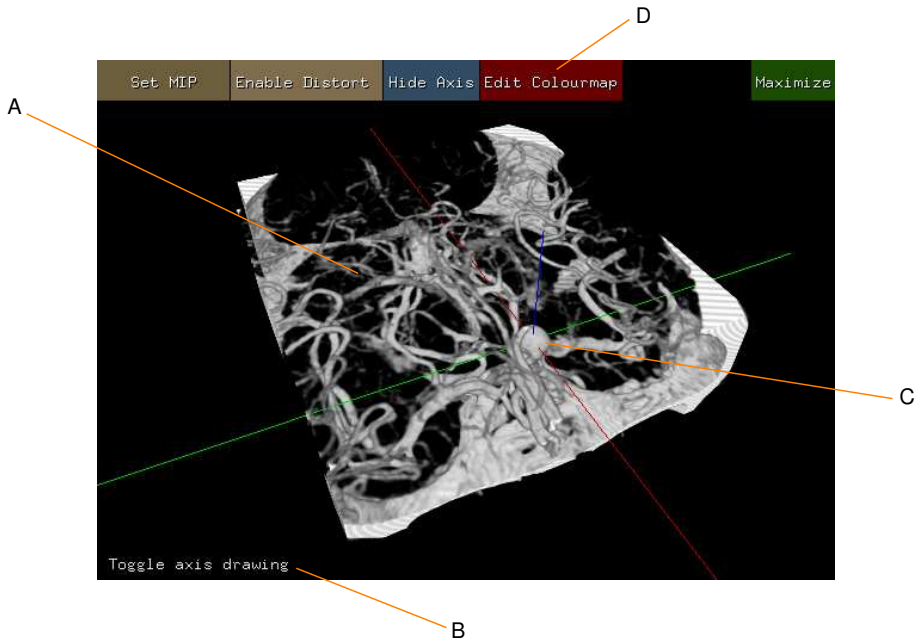


Figure 7.9: Volume view interface elements: volume display (A), message/notification area (B), slice axes (C), and toolbar (D).

The mouse navigation scheme is similar to the one used in the slice views:

- left button and drag: rotate the volume using a virtual trackball;

- middle button and drag: translate the volume in x and y;
- right button and drag up/down: zoom in/out;
- right button and drag left/right: roll the volume around the z axis.

Once again, the toolbar evolved over time, as new functionality was added to the system. Figure 7.10A presents the user interface in milestone 5 (June 2004):

Figure 7.10: Volume view toolbar evolution: small buttons (A), renamed labels and better colouring (B), final version (C).

- *NAV*: navigation mode, default
- *F+C:* enable/disable focus and context mode in the 3D view
- *MIP/CMP*: toggle between maximum intensity projection (*MIP*) and composite rendering (*CMP*)
- *COL:* enable/disable the colourmap editor (see section 7.14 for details)
- *AXIS:* toggle drawing of the axes
- *MAX/MIN:* maximize/minimize the window

The *F+C* toggle button was independent from the respective button in the 2D view, as at that moment the distortion was not being computed at interactive rates, so the user had the option of disabling it in the volume view. Following the same development done in the slice views, the buttons were then enlarged and relabelled, which again happened in milestone 8 (June 2005), shown in (B):

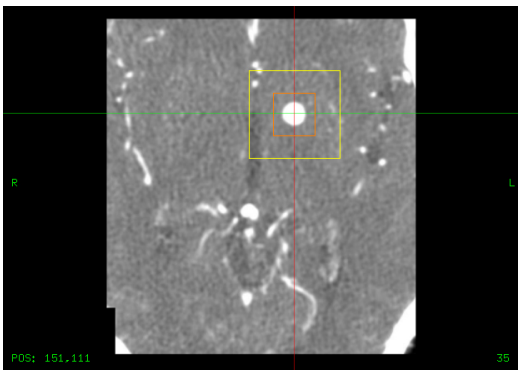
- *Mode: Composite/MIP:* toggle between MIP and composite rendering
- *Normal/Distorted*: enable/disable focus and context mode in the 3D view.
- *Hide/Show Axis:* same as *AXIS*
- *Edit Colourmap/Finish Editing*: same function as *COL*

- *Maximize/Minimize:* same as *MAX/MIN*

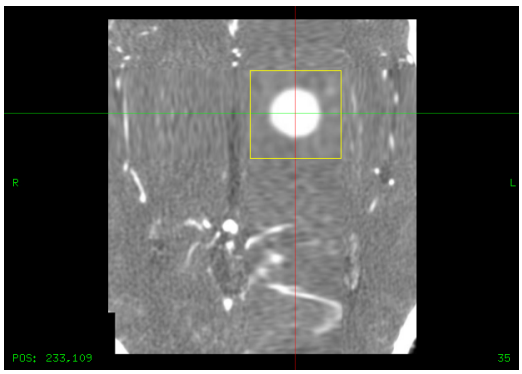
In milestone 10 (December 2005) some buttons were renamed, to reflect the action that would be performed - this was done for consistency: in the previous version, the *Mode* and *Normal/Distorted* buttons indicated the current state. However, the other buttons indicated *actions* to be performed. This was confusing, so the *Mode* button was renamed to *Set MIP/Composite* and the *Normal/Distorted* button was renamed to *Enable/Disable Distort*.

7.2.4 Region of Interest and Distortion

Once the distortion had been included in milestone 5 (June 2004), it was necessary to create some scheme to allow the user to specify the distortion parameters such as the region position/size and magnification factor. The first idea was to have two viewing modes: in the first, we presented two regions to the user (figure 7.11a): the inner region specified the size of the focus area whereas the outer region represented the dimensions of the same area when magnified - hence in this mode the magnification effect was not visible. To see the effect, the user would switch to the second viewing mode, i.e. enabling distortion in the slice views. In that case, a single region would be shown (figure 7.11b) - that region displayed the magnified contents of the inner region. In either viewing mode, the magnification factor was adjusted by dragging the mouse while pressing the right button inside the ROI - in single viewing mode, this just modified the size of the inner region. Finally, the regions could be resized by dragging the corners and each vertical or horizontal line.



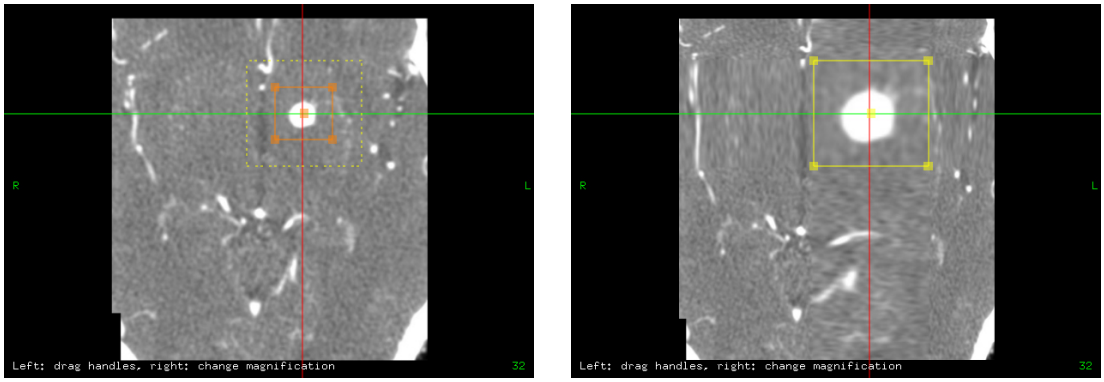
(a) Two regions, normal mode



(b) Single region, focus and context mode

Figure 7.11: Region of interest interface: two regions, both editable.

It was quickly discovered that the two region scheme was rather confusing to the user. We tried to improve it in a modification introduced at milestone 7 (October 2004): the user then could only change the inner region, i.e. the original focus area before magnification. However, the outer region was still displayed so the user could see if he/she tried to move it outside the dataset limits (figure 7.12a). Visible handles were also displayed in the corners and centre, to make it easier to scale and move the focus region. Naturally, the single region mode was still present (figure 7.12b).

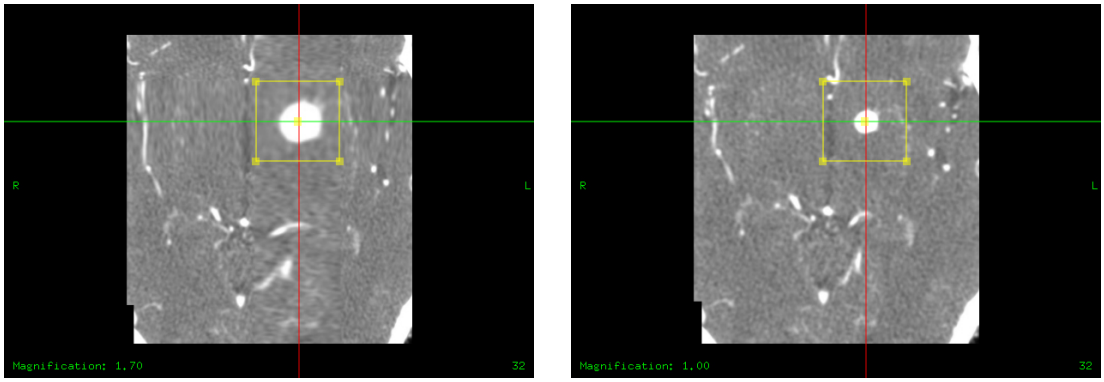


(a) Two regions, normal mode

(b) Single region, focus and context mode

Figure 7.12: Region of interest interface: two regions, only inner region is editable.

Milestone 9 (September 2005) represented a major change in the ROI interface: we realized that displaying the two regions was still counter-intuitive so it was decided that displaying a single region with magnification always being applied would be better, i.e. the single region mode of the previous milestones (figure 7.13a). If the user did not want to use magnification, he/she would simply need to reset it to the neutral value of 1x (figure 7.13b).



(a) Magnified

(b) Not magnified

Figure 7.13: Region of interest interface: single region version.

The *Crop Volume* button, introduced in milestone 10 (December 2005), displays a red region around the entire volume, allowing the user to resize and move it in the same way as the focus region - this was suggested by the consultant neurosurgeon, as an alternative to attenuation (when suitable). When satisfied with the cropping, the user just needs to click again the button (which now displays *Finish Crop*) and the operation will be performed.

7.2.5 Colour Map Editor

The colour map editor was the module that went through the greatest number of changes throughout development, because it was necessary to find an approach suitable for non-technical people such as medical professionals. One of the first design decisions was not to offer complete control over RGB colours, as the system essentially will deal only with grey scale data. This gave the advantage that the colour map editor could be simplified a lot. However, as the colour map is stored internally as RGBA we refer to the "colour" ramp and "opacity" ramp. A ramp specifies a range within the valid data values e.g. from 0 to 255, in which each original data value is mapped to a resulting colour or opacity - essentially, this creates the transfer function used by the volume renderer. We use the term "ramp" as the user is free to change the start and/or end values, but the intermediate values are automatically linearly interpolated. Finally, it was decided to show the colour map temporarily overlaid on top of the volume view, as this is the view where the user will spend most of his/her time while using the system.

Based on these ideas, the first version (figure 7.14) showed the colour and opacity ramps represented as lines over a semi-transparent grey background. Dragging the mouse horizontally changed the start of each ramp (final value was fixed at the end of each ramp): the left button changed colour whilst the right button changed opacity.

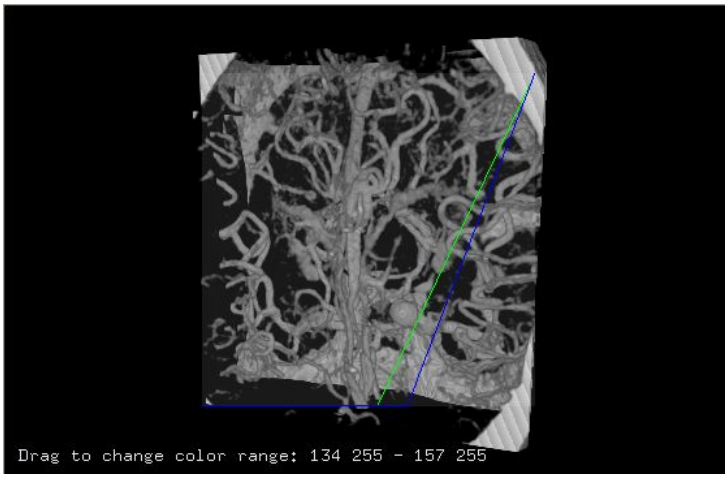


Figure 7.14: Colour map editor (first version).

As the first version was mostly for testing purposes, shortly (milestone 4) a new and improved version was developed (figure 7.15): the ramps were then represented by a grey gradient over a semi-transparent pattern of squares - the usual representation of transparency in painting systems. Arrows pointed to start and end of ranges - the end of each range was still fixed. Later (milestone 9) the arrows could be freely moved, but it was discovered that this was difficult to perform as the arrows themselves were too small.

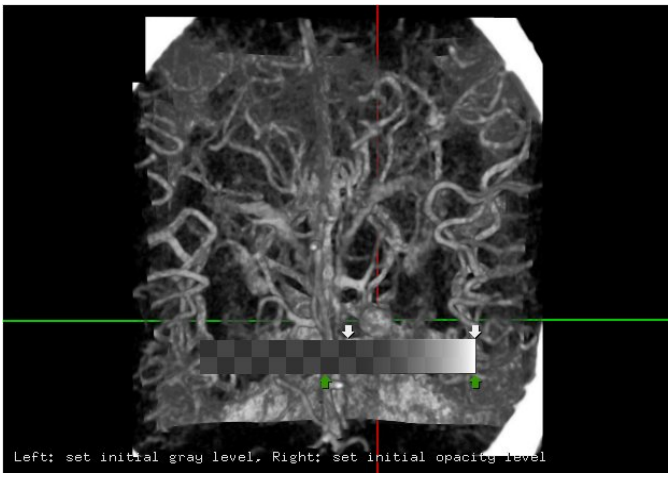


Figure 7.15: Colour map editor (second version).

To solve the problem of dragging the arrows, a third version was produced between milestones 9 and 10, introducing major changes in the colour map visual representation and interaction (figure 7.16): the colour map widget was proportionally enlarged when the volume view was maximized, and it could also be freely resized (by dragging over it with the right button) and moved (by dragging over it with the left button) by the user. This allowed the user to set its position and size to the best possible arrangement.

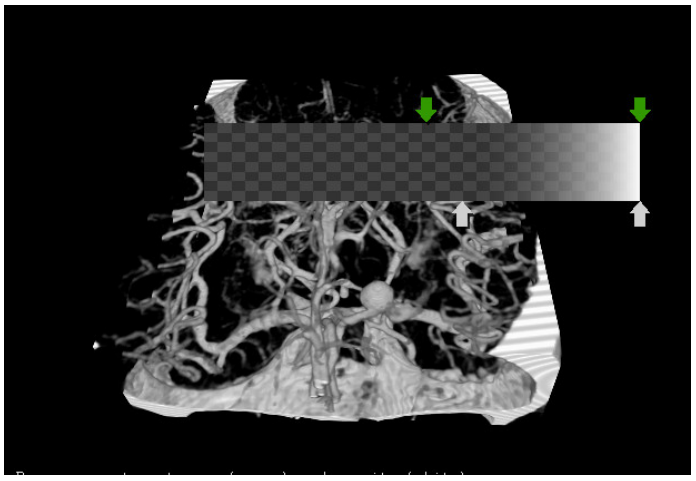


Figure 7.16: Colour map editor (third version).

Milestone 10 introduced the fourth and final version of the colour map editor (figure 7.17), with a substantial change in the way the ramps were controlled: medical visualization systems such as scanner equipment software usually offer a simple approach to colour map editing, using the concepts of *window centre* and *window width*. This is a straightforward way of adjusting brightness and contrast in images and works as follows:

- the window centre (c) specifies the data value which is mapped to the middle grey/half opacity, i.e. the centre of the generated ramp;
- the window width (w) specifies the size of each half from the centre of the ramp, i.e. the ramp always starts at $c - w$ and ends at $c + w$;
- the colour ramp is then generated from black (at position $c - w$) to white (at position $c + w$); similarly the opacity ramp is generated from fully transparent to fully opaque, at the respective positions. Note that the window centre/width are independent for each ramp.

To incorporate the concept into the system, a middle indicator was created to adjust the window centre. The arrows were then drawn over the respective ramp (instead of above or below it), allowing the user to set the window width. Note that moving one arrow forced the other arrow to be moved as well, as they must remain symmetric in relation to the window centre. Finally, as the consultant neurosurgeon had complained that it was not easy to see the effect of colour and opacity in a single area, the ramps were then shown separately, the top one for colour and the bottom one for opacity.

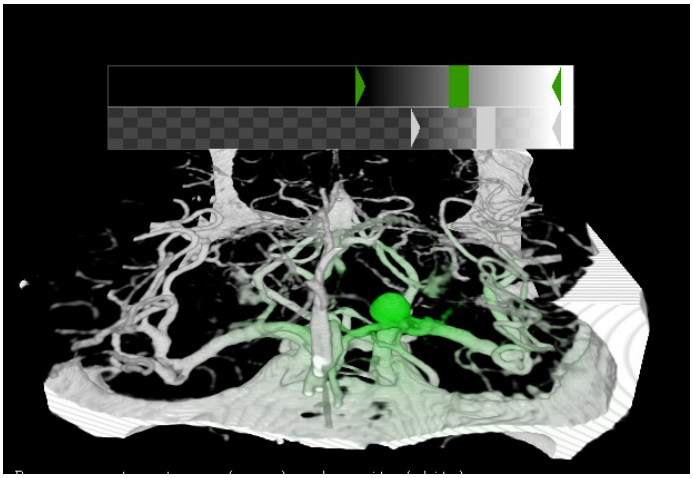


Figure 7.17: Colour map editor (final version).

7.3 A Simulated Case Study

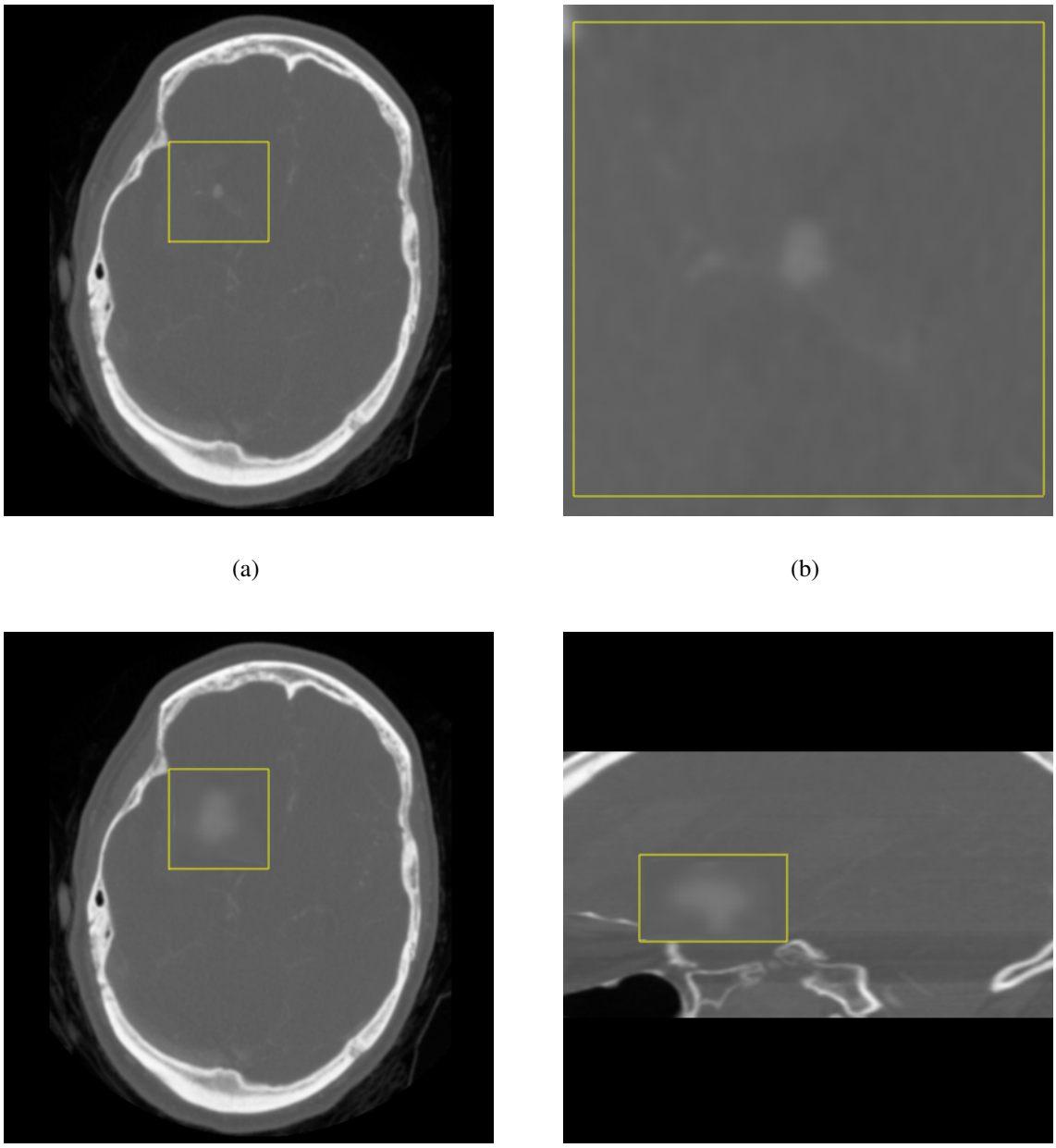
To help understand how the GUI presented in the previous sections could be effectively used in a real scenario, this section presents a simulated case study - this study details the actions that a medical professional would undertake to gather insight about the nature of the aneurysm (see section 1.2 for details). It is based on our combined observations from all the subjects who evaluated the system (see chapter 8 for details).

7.3.1 Locating the Aneurysm in 2D

The user would start by loading the set of images as a volume, as described in section 7.2.1 - we assume that he/she is using the dataset described in section A.3. Once the volume is loaded, the user has the option of either browsing through 2D slices or going straight to the 3D view and trying to find the aneurysm in the volume. In fact, during the evaluation we noticed that most subjects preferred to start off with the axial view as it reflects how the data is originally arranged, i.e. a set of 2D images in that orientation. Thus in this case the user would use the mouse to quickly navigate through the axial slices (figure 7.18a), occasionally zooming in (figure 7.18b) or panning the view to look in more detail. At this point it would be interesting to use one of the mapping effects, e.g. the volume lens (figure 7.18c), instead of simply zooming in. Finally, the user would try to locate the same structure in at least one other slice view, to confirm the diagnostic (figure 7.18d).

7.3.2 Visualizing the Aneurysm in 3D

Once the aneurysm is located in 2D, the next step would be to visualize it in 3D. As the system cannot guess what range of voxel values the user is interested in visualizing, it defaults to a standard colour map which is just a grey/opacity ramp from the minimum (black/transparent) to the maximum values (white/opaque) - this would show the volume as figure 7.19. The system also defaults to disabling the framework effects in 3D, as some users will not require to use them straight away.



(a) (b)

(c) (d)

Figure 7.18: Using the slice views to locate the aneurysm: (a) the focus region is already positioned over a structure which appears to be an aneurysm, however at this distance it is not clearly visible; (b) the user has zoomed in on that area, but lost the view of the surrounding regions; (c) the user has used the volume lens to enlarge the focus region and at the same time is able to see the entire slice; (d) confirming the diagnostic in the saggital view.

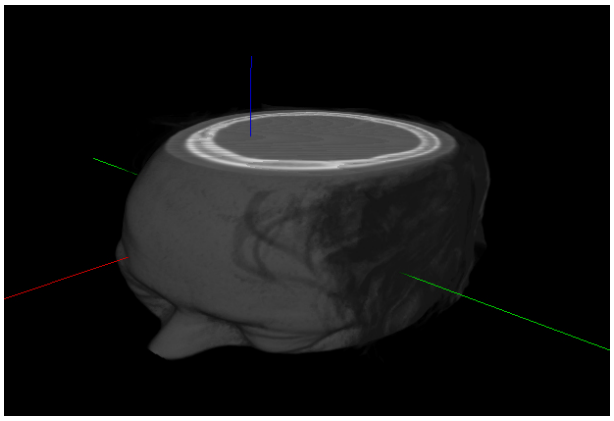


Figure 7.19: The example volume as viewed with a default colour map.

Therefore as the volume appears opaque, one must change the opacity ramp in order to visualize the inner regions of the dataset. The best strategy in this case would be narrowing the range, and then moving the central handle of the opacity ramp until the aneurysm becomes visible. The same can be done to the grey ramp to maximize the contrast, and the result would be similar to figure 7.20.

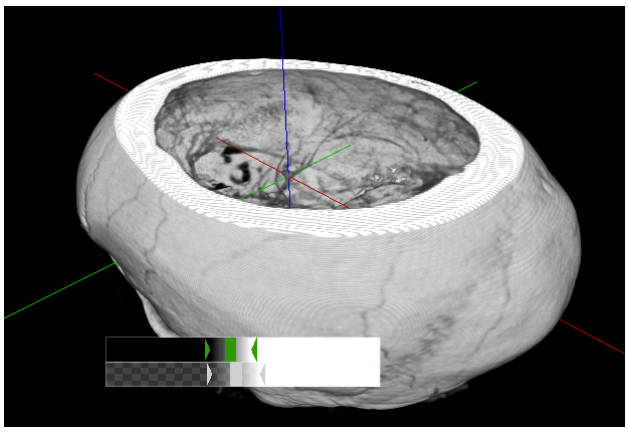
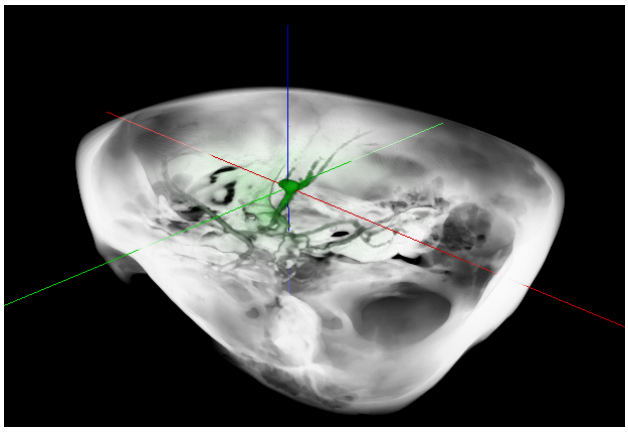


Figure 7.20: Adjusting the colour map to allow visualization of inner structures and to maximize contrast.

7.3.3 Exploiting the Framework Effects

This would be a good moment to enable the framework effects: although the aneurysm is already visible, it is dark due to the way that the colour map was adjusted. To make it appear lighter, it would be necessary to decrease the contrast (i.e. widening the grey ramp) which is not desirable. Alternatively, the user can enable the highlighting effect with a cubic function, for example, to keep the highlight close to the aneurysm.

As the skull bone could easily get in the way, the user would now activate the attenuation effect (with a linear function, for example). Finally, the user would notice an inconvenience with the volume lens[1]: although it gives him/her a magnified view of just the focus region only, it also breaks the connections between arteries that are both in and outside the focus region - as the outside areas are not magnified. Since his/her interest is to visualize the whole vascular tree, to achieve this the user would change the mapping effect to either fisheye or bifocal. Figure 7.21 shows the possible outcome of these changes.

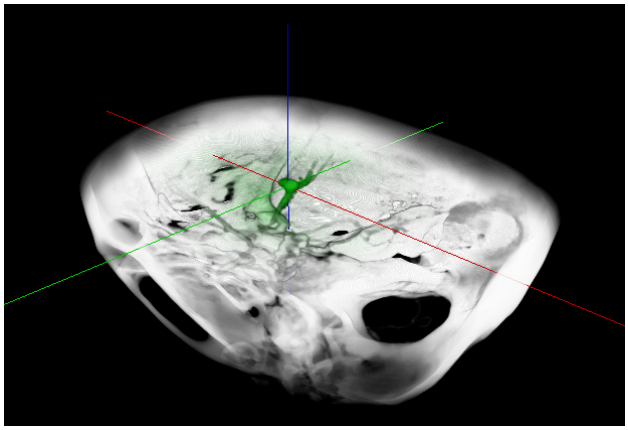


(a)

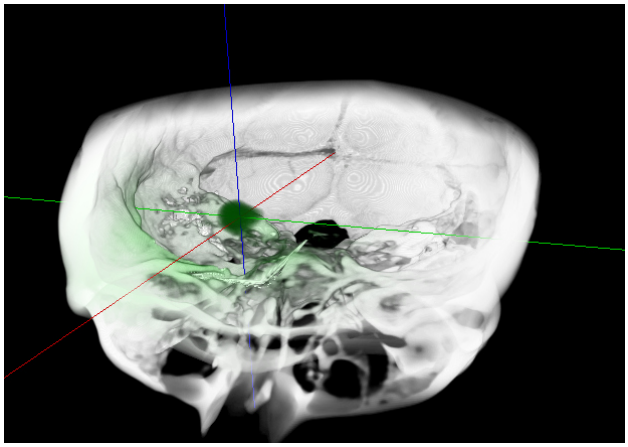
Figure 7.21: Activating highlighting and attenuation to enhance the visualization.

Although the highlighting looks fine, the linear attenuation may not create an effect good enough to be able to remove the bone from any viewpoint - in fact, just rotating the volume could affect it. A possible solution would be to use the quadratic or cubic attenuation functions, but these would remove more regions of the dataset than he/she wants. Hence the solution would be to use the viewpoint-based attenuation, as figure 7.22 shows - that would allow the user to look at the volume from any orientation and still be able to see the aneurysm clearly. Finally, using the mouse wheel, the user would probably adjust the power of the attenuation function to achieve the best effect.

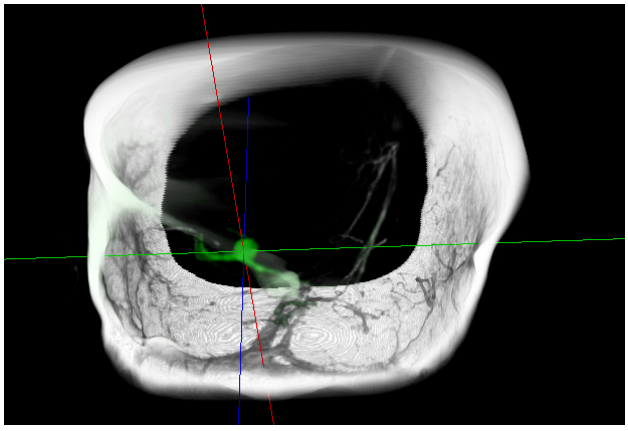
[1]This was also observed during the evaluation.



(a)



(b)



(c)

Figure 7.22: Using viewpoint-based attenuation to achieve a clear visualization of the aneurysm: (a) original view, power of the attenuation function = 6; (b) alternative view, power of the attenuation function = 4 (as less of the volume needs to be removed); (c) underside view, power of the attenuation function = 10 (as there is a lot of bone at the bottom).

7.4 Summary

Here we have presented the user interface and general development issues of the *VolFocus* system, highlighting the main milestones in the system development and critically reviewing the solutions and outcomes for most of the system design decisions. We also demonstrated a simulated case study, trying to show how the system would be used by a medical professional to explore a dataset containing a specific condition (the aneurysm). The next chapters conclude this document: the actual evaluation of the system is reported in chapter 8 and the main conclusions are exposed in chapter 9.

Part IV

Evaluation and Conclusions

Chapter 8

Evaluation

8.1 Overview

THIS CHAPTER DESCRIBES how the evaluation of the *VolFocus* system was planned and conducted. Insight gathered during the evaluation process is reported throughout and in section 8.5, we carry out an analysis of the acceptance of the framework effects by the users, and of the usability of the system as well.

8.2 Rationale and Preparation

The evaluation was envisioned as having two main objectives: to investigate the effectiveness of the volumetric framework in a "real life"-like scenario and to assess the usability of the system with the end users, i.e. medical professionals. The subjects were members of the Department of Neurosurgery at Leeds General Infirmary: an interventional radiologist (IR), four consultant neurosurgeons (CN1-CN4*), three registrars (R1-R3) and one senior house officer (SHO). This provided us with a set of people whose everyday work could benefit from the research - but with different levels of expertise - this is suggested by Rubin [143], who explains that less experienced users can add a lot of insight into the evaluation of a product or system: if they can use a system effectively, this gives an indication that more experienced users can exploit the system better. On the other hand, failure to use the system by less competent users can reveal hints on how to fix funda-

mental design flaws. It is appropriate now to classify our approach: according to Benyon et al [13], this is a *summative* evaluation, as it takes place after implementation of the system.

The first objective of the evaluation was addressed by two main tasks: the first one was to use the tools to gain insight into the nature of the aneurysm present in the dataset (the second being locating the aneurysm). According to the consultant neurosurgeon, this is the crucial aspect when planning for surgery or treatment: there are many different configurations of aneurysms in relation to the surrounding vessels (see figure 1.2), and the operative approach must be adapted for each. For example, in figure 1.2a, the usual procedure is to clamp the saccular aneurysm, preventing the blood from reaching it. However, figure 1.2b shows an entirely different situation: the fusiform aneurysm protrudes from the blood vessel in all directions, so there is not a well-defined neck that could be clamped.

The idea was to provide the subjects with a preset focus region around the aneurysm in the dataset, so they would not have to locate it by themselves (this is covered in the second task). Then they would be asked to try to exploit all the framework effects, in order to help them in determining the present configuration and then rate each one of the effects according to their personal preference. Of particular importance was to find out which mapping effect was the most interesting for the subjects, as this is a key issue of the framework. Nevertheless, we were not able to assess how well the subjects performed the task nor how much time they took to perform it - as they had different levels of expertise (from SHO to CN/IR), this would require a construct validity test, which was not feasible due to time limitations.

The second task of the first objective was to locate the aneurysm, using the navigational tools provided by the system. As the system can be entirely operated by the mouse, it was important to understand how effective the mouse navigation scheme was to the users. In this task, we wanted the subjects to use the 2D views and define an appropriate focus region around the aneurysm, using the region selection tool. This required them to use most of the 2D navigation controls - for example, the usual method is to go through all the axial slices (i.e. top to bottom) until something interesting is found, then go through either the frontal (coronal) or lateral (saggital) views to confirm the diagnostic. This procedure can be done very quickly with the system. Once the aneurysm has been found, then a region of interest can be set.

Finally, the subjects were asked to fill in a form containing those questions in a structured way and a further section on the general usability of the system. The latter provided us with feedback on the graphics user interface and thus helped achieve the second objec-

tive of the evaluation. However, it was not possible to go through a proper usability test due to the subjects' limited availability.

In summary, the procedure outlined in this section can be best described as a combination of *observational* and *survey* evaluations.

8.3 Process

It was felt important to get senior professionals - e.g. consultant neurosurgeons and interventional radiologists - to evaluate the system. However, the main difficulty in conducting the actual evaluation was to gather the subjects together in a single session. But as their department at LGI runs a weekly academic meeting, it was decided that one of those occasions would be suitable. We then gave a short presentation at the end of one of these meetings, where we explained and discussed the motivation for creating the system, and also demonstrated all of the effects provided by the framework. After the presentation, we scheduled some people to return and do an individual evaluation - the same people mentioned in section 8.2.

Each individual evaluation took about 10-15 minutes, during which we asked the subjects to perform the two tasks mentioned in section 8.2 - it must be noted, however, that due to time constraints some of the subjects did not perform the second task. The evaluations were recorded in video as well, with the objective of later reviewing the footage and gathering further insight from actions and comments made throughout the process. Figure 8.1 shows two of the evaluation sessions (IR and CN4*).

(a) Interventional radiologist (IR) evaluation

(b) Consultant neurosurgeon (CN3) evaluation

Figure 8.1: Evaluation sessions

8.4 Results

As mentioned in section 8.2, evaluation results were obtained through a form filled in by the subjects at the end of each evaluation session. Table 8.1 presents a brief summary of the evaluation results collected for the first task (identifying the nature of the aneurysm), where a multi-point scale [13] of *bad* (1) to *excellent* (5) has been used to rate each aspect. This scale was chosen following discussion with the consultant neurosurgeon. The table specifies whether the distortion was considered to be useful or not, and gives a ranking for each method (bifocal, fisheye and volume lens). Similarly, the usefulness of the other two effects (attenuation and highlighting) is presented as well. In the table, CN4* is the consultant neurosurgeon that supported us during the course of the research, and it was felt that his participation in the evaluation was necessary.

Subject	IR	CN1	CN2	CN3	CN4*	R1	R2	SHO	Average
Distortion useful ?	N	Y	Y	Y	Y	Y	Y	Y	–
Bifocal	N/A	4	2	3	5	2	4	4	3.4
FishEye	N/A	5	4	4	4	3	3	3	3.7
Lens	N/A	1	3	2	4	3	4	4	3.0
Effects useful ?	Y	Y	Y	Y	Y	Y	Y	Y	–
Attenuation	5	5	4	4	5	3	4	4	4.2
Highlighting	3	5	3	4	4	3	3	4	3.6

Table 8.1: Summary of evaluation: identifying the nature of the aneurysm

Some subjects also performed the second task - this was limited by their availability - i.e. attempting to locate the aneurysm by themselves, using the tools provided by the system. Table 8.2 presents the results from such task: now the multi-point scale uses ranges from *not helpful/difficult* (1) to *very helpful/very easy* (5), in respect to how each tool was actually employed during the process.

Subject	IR	CN1	CN2	CN3	CN4*	R1	R2	SHO	Average
Navigation	5	5	–	3	4	–	4	–	4.2
Region definition	4	5	–	4	4	–	4	–	4.2

Table 8.2: Summary of evaluation: locating the aneurysm

Each of the subjects was also asked to provide a reason for the ranking being given to each aspect. In section C.2, we present detailed comments as well as notes describing interesting aspects of each evaluation (from subsequent analysis of the video footage).

8.4.1 Usability

Finally, some questions on the prototype usability have also been asked. These basically give an overall view of how the users responded to the user interface, using a mutli-point scale from *strongly disagree* (1) to *strongly agree* (5). Results are presented in table 8.3.

Subject	IR	CN1	CN2	CN3	CN4	R1	R2	SHO	Avg
Easy to use	5	5	4	4	5	3	4	5	4.4
Easy to learn	5	–	3	3	4	4	4	5	4
Pleasant GUI	5	–	4	3	5	3	4	5	4
Clear display of information	4	–	4	4	4	4	4	4	4

Table 8.3: Usability questions

8.5 Analysis

Following is an analysis of the results presented in the previous sections, both in terms of the effectiveness of the framework as well as the usability of the prototype.

8.5.1 Framework Effects

It is easy to see that the majority of users have approved of the mapping effects. The exception was the interventional radiologist (IR): he stressed that the system is a surgical navigational tool, not a tool for interventional radiology. Distortion does not work for him, as he needs to work from the inside out, i.e. through the carotid artery. This is an entirely different approach to the way the surgeons work, which is from outside in, i.e. through the skull bone to reach the aneurysm. Nevertheless this was the opinion of a single subject, thus we cannot say it would be a definitive trend with a larger sample of interventional radiologists. Interestingly, the lens effect, which we initially considered as the most “natural” one, did not have a very good acceptance - most people complained that it prevented a clear view of the connections between the the vessel network and the focus region, making it unsuitable for clinical use. Hence the methods that preserved the entire vessel tree were preferred instead: bifocal and fisheye. Some people appreciated the uniform enlargement offered by the bifocal method, even though the distortion in the surrounding regions was sometimes quite extensive. A similar reaction was obtained with the fisheye method - yet in this case the magnification was not uniform. Surprisingly, the

heavy distortion usually present at the edges of the dataset when using heavy magnification in the fisheye method was of no concern to most of the subjects.

Attenuation and highlighting were both found useful. The attenuation effect was especially liked - this is easy to explain, as it was considered very effective in removing the outer skull bone, which is really useful if planning a surgery or simply visualizing the aneurysm from different viewpoints.

Some people suggested that the system could be very effective as an educational tool and in this case the highlighting effect was thought to be especially valuable. Anatomy is still mostly learnt in 2D and the ability to do it in this "extended" 3D is likely to be really useful to catch students' attention. Finally, some people including a senior consultant (which did not stay for the evaluation) proposed that the system could be used to study tumours as well.

8.5.2 System Usability

As the end users of the system are meant to be medical professionals, good usability is a major concern. Judging from the ratings shown in table 8.3, we can safely say that this objective has been achieved: the system was regarded as very easy to use, requiring minimal training. The uncluttered user interface was well received by all subjects. Of significant importance was the use of the mouse for the entire navigation and menu system, as this offers an easy user interface for pre-operative planning. The subjects reckoned that all major information was clearly displayed, which is also a key aspect in medical visualization systems. For example, the correct orientation of slices (left/right, top/bottom) is essential to correctly visualize medical data.

Some useful comments on usability have been made by the IR as well: he felt that the process of locating the correct slice in each view was unnecessary. For example, if the user is moving the focus region in the axial view, the centre of the region could be used to automatically change the current slice in the other two views. Currently the system requires the user to scan through the slices and then position the focus region, which can be inefficient. This was in fact easy to add as an additional feature - as explained in section 7.2.1. It was also observed that the effects must be activated and then their parameters (linear, quadratic, etc) can be set - this could be done in a single step by changing the menu.

Finally, we must note that although the number of participants in the evaluation was not very high, we believe that choosing the right people had a fundamental importance in obtaining good results. In fact, this is a crucial concern for any kind of usability evalution:

as Rubin [143] mentions, test results will only be valid if end users of the system do participate. He also points out that the number of participants is largely dependent on a number of factors, such as the number of available resources to set up and conduct the test, duration of the test session and availability of the participants.

8.6 Summary

In this chapter we presented the complete evaluation results of the *VolFocus* system. The system was evaluated by eight medical professionals with different levels of expertise, which provided us with rather different views but an overall positive reaction to the system and to the framework effects. Also a complete analysis of the evaluation was developed at the end, highlighting the current advantages and disadvantages of the system.

Chapter 9

Conclusions

9.1 Overview

THIS CHAPTER PRESENTS the conclusions of this thesis, by assessing the contributions, reviewing the objectives, appraising the research hypotheses and finally, proposing some directions for future work.

9.2 Summary of Contribution

In this section we evaluate and summarize the contributions of this thesis. As stated in chapter 1, this thesis introduced a number of contributions as follows:

9.2.1 Taxonomy

We have developed a taxonomy for focus and context applied to volume visualization (see section 5.2), where we integrated the ideas of distortion and refinement/accuracy. The taxonomy allows us to classify existing methods based on these ideas and also served as insight for the creation of the volumetric framework, although the latter did not implement refinement or accuracy in the focus region.

9.2.2 Framework

In section 5.3 we have described a versatile framework which combined the ideas of spatial distortion, colour highlighting and opacity attenuation as a means to provide enhanced visualization of volumetric datasets. In particular for the visualization of aneurysms, the framework was shown to be effective - this statement is supported by the results of the user evaluation. This was a very specific application: we did not validate the framework for other datasets, medical or not. However, we shall discuss the wider applicability of the framework in section 9.5.

9.2.3 Research in Hardware-accelerated Methods

In chapter 6 we have presented detailed descriptions of a number of hardware-accelerated methods to achieve volumetric distortion. These methods were then carefully evaluated, in order to find the most suitable for implementing all the effects of the framework: we assessed the suitability of each method as well as memory requirements and overall rendering performance. At the end, we carried out further experiments with the extended 3 x 1D offset textures method, to confirm its viability when handling larger datasets. We must note, however, that there is some loss of generality with this method, as not every 3D function can be represented - for instance, more than one focus region may be impossible, depending on the region. This limitation is caused by storing separately the functions for x, y and z - this could be solved by using the 3D offset textures method, bearing in mind the necessary extra memory and additional preprocessing time.

9.2.4 User Evaluation of the *VolFocus* System

We have described the planning, execution and results of a user evaluation of the system in chapter 8. This evaluation was undertaken by a selected group of potential end users: an interventional radiologist, consultant neurosurgeons, registrars and a senior house officer. These results provided us with a good indication that the *VolFocus* system is suitable for the task it was designed for: the effective visualization of aneurysms. Ultimately, this is directly associated to the successful implementation of the framework effects. Valuable suggestions also were proposed at this stage, mostly related to the user interface of the system - these are mentioned in the future work section (9.6).

9.3 Assessment of Objectives

In section 1.3 we have described the three objectives of this thesis. We shall now assess those objectives, relating them to the contributions described in section 9.2.

9.3.1 First Objective

The first objective was defined as *to develop an approach to volume visualization that helps medical professionals to achieve clear visualization of aneurysms.* As already mentioned in the contributions, both our experience and the results of the user evaluation lead us to believe that this objective has been reached. Evidently, some of the effectiveness of our method comes from the speed provided by directly programming the graphics hardware. In fact, as we shall show in section 9.5, it may be possible to apply our approach to different situations and datasets with satisfactory results.

9.3.2 Second Objective

We have described the second objective as *to create a visualization system suitable for use in a medical environment, particularly in the operating theatre*. The *VolFocus* system was evaluated by the medical professionals and considered easy to learn and use, whilst at the same time providing valuable information on screen and adequate rendering speed. However, we cannot say that this system is really suitable for the operating theatre as it is now, as this sort of evaluation was not done - we have just the informed opinion of the consultant neurosurgeon to support this. Also, previous work such as the one carried out by John [64] has pointed out that mice are not necessarily the most appropriate input devices in a sterile environment - we discuss alternatives in section 9.6.3. Therefore we see the system as a pre-operative tool for surgery planning, but not as an intraoperative system.

9.3.3 Third Objective

The third objective of this thesis was *to carry out an evaluation of this visualization approach and the developed system*. As already described in the contribution section (9.2.4), we have evaluated both the visualization approach (the framework and its effects) and the *VolFocus* system (usability evaluation). A full usability test could not be carried out, but nevertheless the evaluation has a significant importance for future research on the subject, as we now better understand the role distortion plays in a medical application: it is not

suitable for all applications and tasks, yet it can be very useful (e.g. aneurysm visualization for neurosurgeons).

9.4 Reviewing the Research Hypotheses

In section 1.4, we have described the methodology followed by this research - this has introduced three research hypotheses. We shall now revisit these hypotheses and appraise whether we found a definite answer or not.

1. Is focus and context a suitable approach for enhancing the visualization of medical data, especially aneurysms ? *Yes*. Although distortion is not always suitable, highlighting and in particular attenuation can be very useful in many situations. For example, highlighting is especially effective for confusing datasets such as the example shown in figure 5.15. Attenuation has shown promising results in a number of examples throughout this thesis (see figures 5.12, 7.21, 7.22 and 9.1).

2. Can we exploit the graphics processor to create innovative effects and also to achieve acceptable rendering performance ? *Yes*. The comparative analysis shown in section 6.9 definitely supports this answer - see also tables 6.2 and 6.3 for the performance data.

3. Can we maintain interactive rendering speeds and still provide a clear view of a specific anatomical feature, regardless of viewpoint ? *Yes*. This is possible with the help of the viewpoint-based attenuation, and as demonstrated in section 6.9, it can be achieved at interactive speeds with the 3 x 1D offset textures method.

9.5 Wider Applicability of the Framework

In this section we describe a simple experiment that can show the wider applicability of our approach: figure 9.1 shows mapping, attenuation and highlighting effects applied to a variety of commonly used datasets [8]. Each image on the left displays the adjusted transfer function and the visual result without any effects, whilst the corresponding images on the right show framework effects being applied: we can easily see that the effects increase the comprehensibility of each selected focus region (shown using the highlighting effect). We shall now analyze each case in more detail: in the following examples, we have used viewpoint-based attenuation with varying powers and when necessary, we also adjusted the distance from the boundary of the attenuation region to the dataset edge.

The first case is displayed in figure 9.1a/b: the tooth is a challenging dataset to visualize, as there are a number of structures of interest - dentin, enamel, nerve cavity, etc. In fact, it is commonly visualized with multidimensional transfer functions [76, 77], or alternative approaches as shown in recent work [140]. Nevertheless, by simply doing quick adjustments to the default transfer function and activating viewpoint-based attenuation, we have achieved a much improved view (figure 9.1a/b). This example illustrates how helpful it can be to use less than the full distance from the boundary of the attenuation region to the dataset edge, as introduced in section 5.3.2: the displayed effect is achieved by using 40% of the distance for each axis, combined with a high power attenuation (20).

In the second case (figure 9.1c/d), we used a non-medical dataset: a two-piston engine. In this situation, we wanted to visualize the interior of one of the pistons. Again, an only slightly modified colour map was set, which produced the result shown in figure 9.1c. In this case, attenuation works similar to a cutting plane, allowing us to see the piston - we have also employed the volume lens effect to enhance the visualization.

Finally, the third example (figure 9.1e/f) illustrates that MRI data is also suitable for our approach: here we did not want to isolate a particular anatomical feature such as an aneurysm or tumour, but simply wanted to better visualize the brain through the skull (figure 9.1f). We also selected quadratic highlighting to obtain a larger highlighted area.

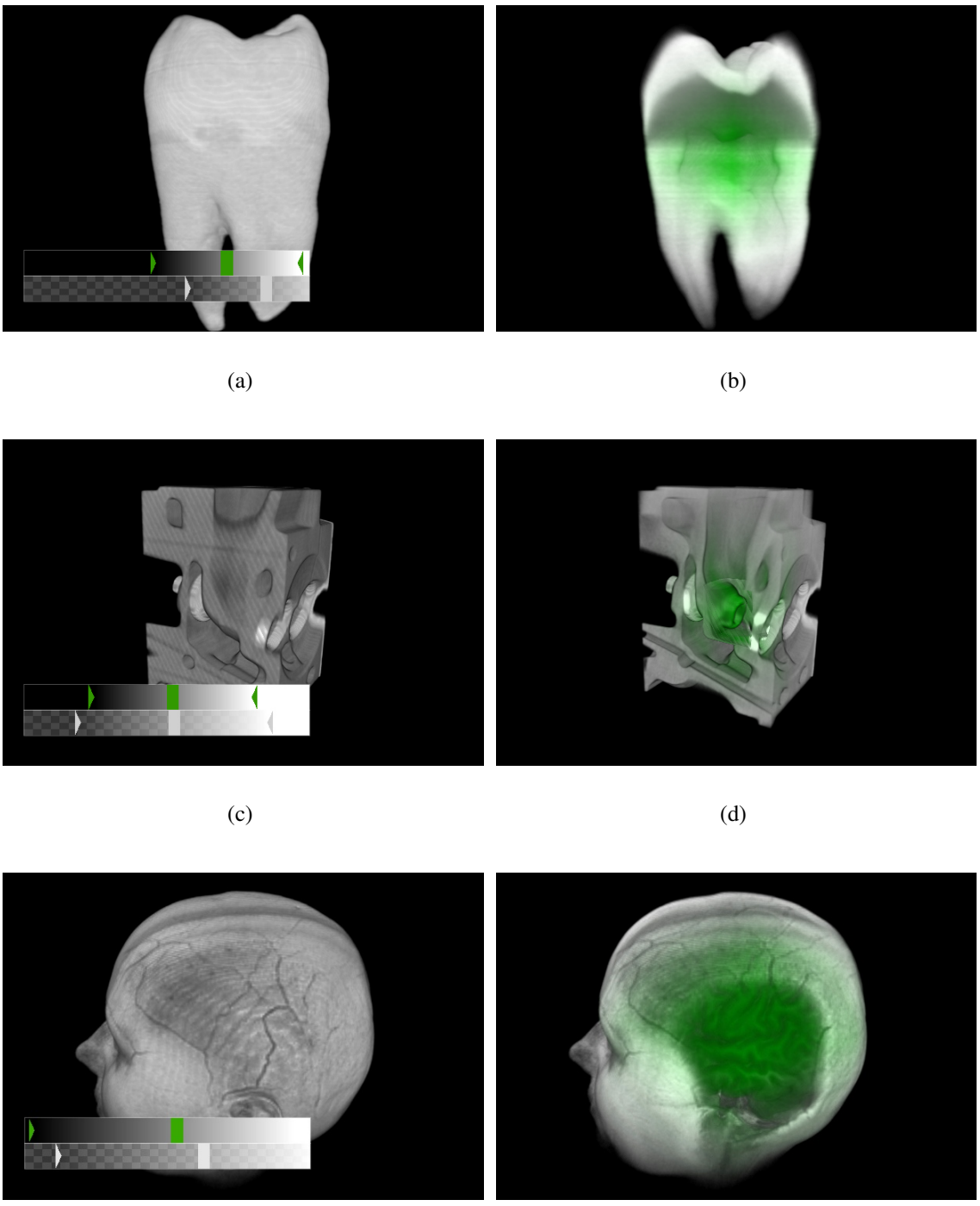
In summary, we believe that the framework is suitable for a number of datasets and different applications. In fact, some of the limitations are more related to the *VolFocus* user interface than the framework itself - for instance, being restricted to linear, quadratic or cubic highlighting, and not being able to adjust the transfer function in RGB.

9.6 Future Work

Although the objectives of this thesis have been achieved and the research hypotheses satisfied, we can foresee three branches for future research: increased rendering quality, further extensions of the framework and additional research towards better usability of the VolFocus system.

9.6.1 Rendering Quality

The quest for increasing the rendering quality is an ever present issue. The *VolFocus* system is no exception: it would be desirable to have better quality in the system, if interactivity would not be sacrificed. However, as we shall demonstrate, this balance is hard to achieve with current technology - it is indeed possible to obtain higher quality



(a) (b)

(c) (d)

(e) (f)

Figure 9.1: Applying the framework effects to a variety of datasets: (a) *tooth* dataset with no effects; (b) same dataset with cubic highlighting and attenuation (power = 20, distance = 40%); (c) *engine* dataset with no effects; (d) same dataset with cubic highlighting, attenuation (power = 8, distance = 100%) and volume lens; (e) MRI *woman head* dataset with no effects; (f) quadratic highlighting and attenuation (power = 12, distance = 100%).

but this implies some performance loss. Nevertheless, we can identify three possibilities for increased rendering quality - better interpolation, pre-integrated classification and ray casting:

- Pre-integrated classification: this is the approach proposed by Engel et al [44], where one can achieve better image quality by pre-integrating colours and opacities within a volume slab (see section 2.10.2 for details). In the context of our framework, pre-integration is costly as we must fetch two texels for the front and back voxels in a slab, hence we also need two extra texture fetches to get the mapping effect, attenuation and highlighting for both voxels. Also if we recall that a pre-integration table indexed by two scalar values is stored in a 2D texture, adding highlighting and attenuation would require a 4D table, which is not possible to implement as a single texture. This would also lead to higher computation time for the table, which would prevent interactive changes either in the colour map or in the framework effects.

- Higher order interpolation: as demonstrated by Sigg and Hadwiger [147], it is already possible to achieve higher order interpolation - but according to our investigation, this has a considerable performance hit (see figure 2.8 for an example). However, an interesting option worth researching would be to apply higher order interpolation just inside the focus region, thus reducing the computational cost - it still remains the question of how to do this selectively inside the fragment shader, without a performance penalty for introducing conditional branches.

- Ray casting: this is also increasingly becoming an attractive option, as it has already been demonstrated by many [141, 83, 152] that it can be efficiently implemented in a fragment shader. Ray casting has the significant advantage that interesting effects can be obtained by changing the sampling rate. For instance, increased accuracy inside the focus region would just require a higher sampling rate than the one used outside that region. However, there is a major issue when we try to apply this idea to our framework: it is not clear how the framework effects could be efficiently fetched by the ray casting algorithm, without a serious speed penalty - we must recall that each voxel can have a different mapping, attenuation or highlighting effect. Finally, even in the conventional GPU ray casting implementation, performance is still an issue (see figure 2.2). Nevertheless, as the hardware matures, we anticipate that future graphics cards will be able to achieve truly interactive speeds.

9.6.2 Framework Extensions

The first idea that could be introduced in the framework is the refinement or accuracy concept: as the focus region is supposedly more important than the remainder of the dataset, it would be sensible to render it with a higher quality. However, as we already discussed in section 9.6.1, at the moment this is a challenging task: there is always a trade-off between rendering quality and speed. Some research has already focused on this idea [88, 165] but not with a more comprehensive framework such as ours.

We have demonstrated the versatility of the framework with the bifocal, fisheye and volume lens approaches, as they are well known ideas and simple to understand. Nevertheless, it would be interesting to introduce more generic distortion formulae: the current ones rely on very few parameters such as magnification/distortion factor and location/size of focus region. Alternatively, different but useful specialized distortions could be implemented, such as the surgical metaphors demonstrated by McGuffin et al [110] - we must note, however, that the limitations of the 3 x 1D approach described in section 9.2.2 still apply.

Finally, we note that the framework does not support the idea of time-varying data. We do believe this is an interesting possibility: can the extra dimension support the concept of focus and context ? For example, one may imagine that in a time-varying dataset we are interested in a specific interval for e.g. visualizing part of the cycle of a beating heart. In this context, we can envision focus and context being applied as compressing time for the non-interesting intervals, in such a way that the "time region of focus" is displayed at normal speed. This and other similar questions could lead to considerable new research.

9.6.3 *VolFocus* Usability

Most of the usability ideas suggested here are consequences of suggestions and observations carried out during the evaluation. A common complaint was that some operations took place in the volume view, whilst other operations took place in the slice views. For example, the focus region and magnification/distortion factor must be set through the slice view interface, whilst the result in 3D can be seen in the volume view. If one is working solely on the volume view, e.g. rotating the volume, and wishes to change the magnification, one has to move back to one of the slice views and carry out this procedure there - this creates a break in continuity of operation, which is not desirable. Hence in the future these operations could also be implemented in the volume view, for example, by directly dragging the focus region in 3D or right-clicking inside it to change magnification. Another complaint was that the focus region should be displayed in the volume view - we

have not implemented that because we believed that it would clutter up the view and the plane axes already give similar information. However, this may help one to visualize the exact region.

The use of menus can be another source of complications: for instance, all effects must be activated through the *preferences* menu - there could be a way of doing the same in the slice or volume interfaces, such as clicking on a special button or a through combination of mouse buttons.

Although the system provides the option to save the current session as a snapshot file, it would be interesting to have a way to save the entire sequence of operations carried out during a session. This would allow one to replay a saved session, in order to e.g. evaluate performance for training purposes or share ideas or plans with a colleague - we highlight the use of VNC [132] or NX [121] as possible short term solutions: these approaches allow one to record a remote session and replay it later.

As mentioned in section 9.3.2, John [64] has reported that the mouse is not the best option for use in the operating theatre - in his work, he used a low cost joystick in a sterile bag, combined with a system that followed a pre-defined sequence of operations, selected by observation of the surgeon's workflow. However, we suggest that newer input devices such as 3D mice [1] could be also suitable and still allow freedom of movement for the surgeon. Alternatively, the traditional "space mouse" with six degrees of freedom could be employed - these devices usually also have a number of command buttons, which could be used to activate/deactivate functions.

We also believe that it would be very interesting to conduct a proper usability test of the *VolFocus* system: this would possibly lead to new discoveries of possible improvements and limitations concerning the user interface. However, this is not an easy task: a proper usability test would require both HCI experts [143] and additional end users of the system.

Finally, it could be useful to carry out a diary study of the system. A diary study, as described by Rieman [137], is based on participants writing down their daily activities on preprinted log forms. If we suppose that the *VolFocus* system is going to be used by a number of medical professionals, these people could fill in the log form each time they used the system. This would provide us with valuable, real life usability data that would be impossible to obtain otherwise. However, Rieman also reports some challenges that this technique introduces: first, the participants must be persuaded to make some effort to record their activities - this cannot be overlooked, and as we also discovered during our evaluation, people are sometimes not very keen to fill evaluation forms. Second, sometimes people do maintain their diaries but record little detail - we have also seen

this during our evaluation. Hence there is a need for the researcher to carry out personal interactions with the participants, keeping close contact with them if possible.

9.7 Final Remarks

It is our view that this thesis has contributed to the field of volume visualization, by introducing a framework that combines focus and context ideas and applies those to a visualization system. We have also demonstrated this framework through a practical application in medicine, and moreover we hope that this thesis may be useful not only for neurosurgeons, but other medical specialties that routinely employ volumetric datasets.

We conclude hoping that this work may inspire others in the constant pursuit for better and more effective data visualization.

Appendix A

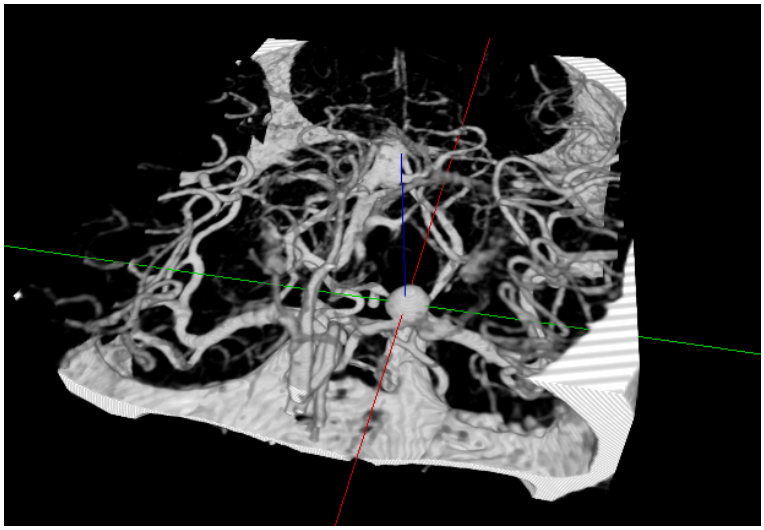
Example Datasets

A.1 Overview

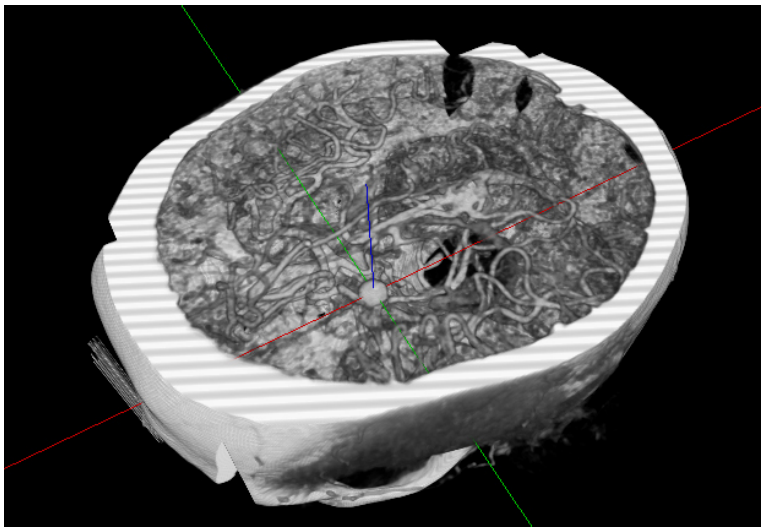
This appendix presents the three datasets we used during the course of the research to carry out the various experiments. All were provided by the consultant neurosurgeon and contain CTA data of cerebral aneurysms.

A.2 High Contrast

This was the first dataset we used in the research: it exhibits a high contrast in the vessels, thus it is possible to get very good detail in them. However, as this was obtained from JPEG images with medical annotation (patient name, etc) we were forced to manually remove the annotation, and therefore the dataset has some artifacts. During the first stages of this research we employed a cropped version of this dataset, as the graphics hardware was not cabable of handling the full volume efficiently - that version is shown in figure A.1(a). Later we were able to use the original dataset, which is displayed in figure A.1(b). Note that this dataset is much harder to visualize, as the skull bone is visible at the bottom.



(a) Cropped to 287 x 311 x 89



(b) Entire volume: 512 x 512 x 146

Figure A.1: A high contrast dataset, both in cropped and full versions.

A.3 Low Contrast, Small Aneurysm

This dataset was extracted directly from DICOM data and shows a very small aneurysm - this was suitable for experimenting with the distortion effects, as the small size of the aneurysm prevents a clear view without zooming in (figure A.2).

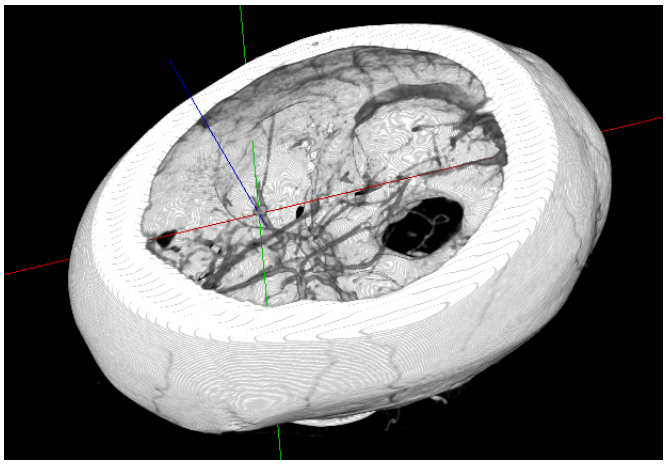


Figure A.2: A low contrast, small aneurysm dataset: 512 x 512 x 183.

A.4 Low Contrast, Large Aneurysm

This was another dataset extracted from DICOM data and this time, shows a large aneurysm ("giant" in medical terminology). In this case, the distortion effects are not very suitable (as the aneurysm is already large), but attenuation and highlighting still can be used with good effectiveness (figure A.3).

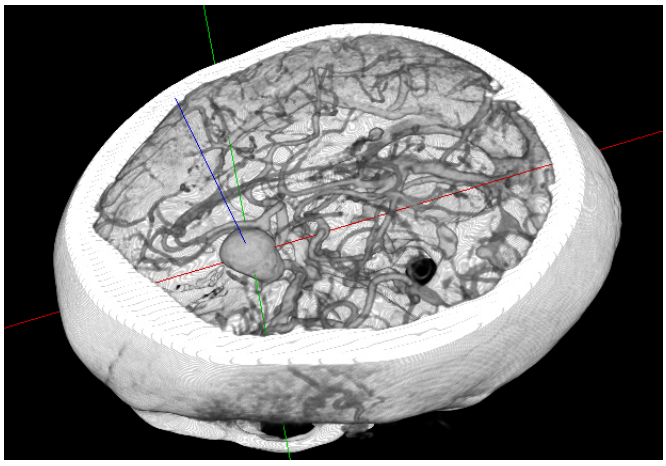


Figure A.3: A low contrast, giant aneurysm dataset: 512 x 512 x 190.

Appendix B

VolFocus Milestones and File Formats

B.1 Overview

This appendix describes the sequence of milestones achieved during development of the *VolFocus* system. Here we also present the various file formats used by the system.

B.2 Development Milestones

The *VolFocus* system went through a number of changes during development, triggered by e.g. new findings in research or additional implemented functionality. Table B.1 summarizes the timeline of the system with emphasis on the GUI, highlighting the major milestones that have been achieved throughout the research and development process.

B.3 File Formats

The *VolFocus* system employs three different file formats to store volume data (.vol), a colour map (.lut) or the entire system status (.sna). in this section, we describe in detail these formats.

B.3.1 Volume file (.vol)

As already mentioned in section 7.2.1, the VOL file format is actually comprised of two separate files: a text file with a header (*.vol*) that describes the volume (size, channels, etc), and the volume data itself, which is stored as raw (binary) data in another file. The structure of the header file is presented in listing B.1:

#	Date	Milestone
1	April 2003	First version with a rudimentary GUI: four-pane view, created with FLTK [151]. Basic navigation: slice change, zooming, panning, maximize/minimize.
2	September 2003	Preliminary experiments with focus and context methods, using IRIS Explorer and finally generating new volumes from existing data (off-line distortion).
3	January 2004	Final version using FLTK: MIP/Composite rendering toggle, drawing axes and very simple colourmap editor.
4	May 2004	First version based on the wxWidgets toolkit [148]. New colourmap editor, showing transparency as a patterned background. System now can read image data (JPEG, etc). First experiments with texture-based distortion, as stand-alone programs.
5	June 2004	Bifocal distortion now possible inside the prototype, a region can be set using the region of interest (ROI) mode - magnification is controlled by the size of the inner region and through the right mouse button. Distortion takes roughly 2-3 seconds to be computed, due to the need of bricking the volume again.
6	July 2004	System now is able to read DICOM data files and display their information in a separate window - this is very important for the integration into the medical scenario.
7	October 2004	Major GUI changes: system is always in navigational mode (NAV) and enters ROI mode only when the mouse is inside the focus region. Only the inner region is editable now, but the outer region is still displayed and scales accordingly. The transparent buttons have better colours and their labels are no longer a 3-letter code.
8	June 2005	Programmable shaders now used to create the distortion effect, following the research on distortion methods (see section 6.9). Distortion now happens at interactive rates, and bricking the volume is not needed anymore as the graphics hardware no longer has texture size limitations.
9	September 2005	Slice views no longer have a “distort” mode - they always distort the data, but start with no magnification. Now a single region specifies the distorted focus area. Colour map editor now has draggable arrows to set the limits.
10	December 2005	New distortion method added, resulting from research on the framework: fisheye distortion. Colour map editor now behaves as a window centre / window width system: middle square sets the centre and distance to either side defines width (symmetric). Prototype now called *VolFocus*.
11	February 2006	Added the volume lens distortion and the two other effects of the framework: attenuation and volume lens - menus have been created to set the parameters.
12	March 2006	System evaluation at Leeds General Infirmary

Table B.1: Milestones in the system development.

Listing B.1: Header to describe volumes in OSCVR .vol format

```
\raw header_size width height depth channels channel_size endianess
raw_filename
```

Where:

- *header_size*: length of header data in the raw volume file (normally 0);
- *width*, *height*, *depth*: volume dimensions;
- *channels*: number of channels in the volume data (normally 1);
- *channel_size*: size in bytes of each voxel (1 for 8-bit data, 2 for 16-bit data);
- *endianess*: should be 0 when dealing with 8-bit data - in the case of 16-bit data[1], this parameter indicates the byte ordering: 0 for little-endian data (least significant byte first) or 1 with big-endian data (most significant byte first).

B.3.2 Colour map file (.lut)

The system stores colour maps in text files, following a very simple structure (see listing B.2):

Listing B.2: File format to store a colour map (.lut)

```
number_of_entries
R0 G0 B0 A0
...
Rn Gn Bn An
```

In listing B.2, R0/G0/B0/A0...Rn/Gn/Bn/An are respectively the red, green, blue and alpha colour components for each entry, from 0 to *number_of_entries* - 1.

B.3.3 Snapshot file (.sna)

Finally, it is possible to save and restore the entire system state, which is stored in a snapshot file (*.sna*) - see listing B.3:

[1]The system is capable of loading 16-bit data, but it is converted internally to 8-bit before rendering.

Listing B.3: File format to store a system snapshot (.sna)

```
SNAPSHOT
volume = volume_file
multi = 0/1/2/3/4
focus = 0/1
rendering = 0/1
distmode = 0/1/2
roi = xmin xmax ymin ymax zmin zmax
mag = mag_factor
axis = 0/1
planes = axial_plane saggital_plane coronal_plane
background = back_red back_green back_blue
thickness = slice_thickness
orthocolour = 0/1
highlighting = 0/1/2/3
attenuation = 0/1/2/3
attendist = 0/1/.../9
viewatten = 0...k
ramps = colour_min colour_max alpha_min alpha_max
colourmap colourmap_file
```

Where:

- *volume_file:* either a .vol file name or a single image in a directory with the full set;
- *multi*: 0 if four-view mode, (1-3) if maximized in slice views, or (4) if volume view;
- *focus*: 0 if effects are not enabled in the volume view, 1 otherwise;
- *rendering*: 0 if in MIP rendering mode, 1 if in composite rendering mode;
- *distmode*: 0 for bifocal, 1 for fisheye and 2 for volume lens;
- *roi*: limits of the focus region in x,y and z
- *mag_factor*: the magnification factor;
- *axis*: 0 if axes are not visible, 1 otherwise;
- *planes*: current slice in each 2D view;
- *background*: background colour in RGB;
- *thickness*: slice thickness adjusted by the user or read from DICOM files;
- *orthocolour*: 0 if colour mapping is not applied in 2D views, 1 otherwise;

- *highlighting, attenuation*: highlighting/attenuation mode - 0 is disabled, 1 is linear, 2 is quadratic, 3 is cubic;
- *attendist*: percentage of distance from each dataset edge to start of attenuation region (0=0%, 1=10%, ... 9=90%)
- *viewatten*: 0 if in distance-based attenuation mode, $k > 0$ if in viewpoint-based mode (k indicating the power of the attenuation function);
- *ramps*: minimum and maximum limits for the colour and opacity ramps, used to position the arrows on the colour map widget;
- *colourmap*: file name containing the colour map.

Appendix C

Evaluation Questionnaires

C.1 Overview

This appendix present a transcript of the complete questionnaires filled in by the subjects during the actual evaluation of the VolFocus system, as well as a number of notes taken throughout the process and following the review of the video footage.

C.2 Transcripts

C.2.1 First subject: IR

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *NO*
  - Why ? (usefulness): *Treatment and analysis of aneurysms depends on acccurate depiction of shape and dimension, therefore distortion is not applicable.*
  - Ratings:

Bifocal	N/A
Fisheye	N/A
Volume Lens	N/A

  - Justification for rating: *N/A*
- Were the special effects useful ? YES
  - Why ? (usefulness): *Highlights aneurysm against surrounding structures such as bone or blood clots.*

  - Ratings: *Attenuation* 5, *Highlighting* 3
  - Why ? (effectiveness): *Manipulation of attenuation is key to visualizing structures which have different x-ray properties.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *5 (very helpful)*
- Why ? *Need to locate aneurysm, but would help if the three views were coordinated to movement of the focus region.*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *4 (easy)*
- Why ? (level of difficulty) *Still have to locate lesion in three views.*

Strengths of the System

- *Excellent for planning neurosurgery.*
- *Would be applicable to other applications.*
- *Very easy to use and very fast.*

Weaknesses of the System

- *Less use for his work*

Additional notes

- Subject first wanted to change the transfer function, as he needed to clearly see the blood clot that surrounded the aneurysm - therefore he quickly activated the colour mapping on the 2D views.
- He complained that navigating through the slices could be tied to movement of the focus region - at the moment, the user needed to first go through the slices in at least two views, then position and resize the focus region.

- He explained that distortion does not work for him, as he works from the inside of a blood vessel to reach the aneurysm and perform his work internally only. He does not need to know the relationship of the aneurysm to the skull, however the exact shape of the aneurysm is crucial for his work.
- He also reckoned that the 3D view can smooth off some details, but it is good for the surgeons, as it gives anatomical views. He is interested in precise measurements, so 2D is the only way for him.
- Finally, the IR pointed out that the program should be useful for other applications, such as viewing tumours. He also mentioned that the system could be linked to their medical data centre and hence be used directly in the operating theatre.

C.2.2 Second subject: CN1

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Enlarges region of interest but keeps surrounding structures in context.*
  - Ratings:

Bifocal	4
Fisheye	5
Volume Lens	1

  - Justification for rating: *Lens not good because deformation in surrounding objects.*
- Were the special effects useful ? YES
  - Why ? (usefulness): *Provide very good view of surgical anatomy.*
  - Ratings:

Attenuation	5
Highlighting	5

  - Why ? (effectiveness): *Not given.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *5 (very helpful)*
- Why ? *Not given.*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *5 (very easy)*
- Why ? (level of difficulty) *Not given.*

Strengths of the System

- *Excellent view of surgical anatomy.*
- *Would be good for surgery preparation and teaching.*

Weaknesses of the System

- *Not given.*

Additional notes

- He wanted to change the transfer function to see the blood clot along with the aneurysm. Played a bit with the 2D views and then with the 3D views.
- He pointed out that there is not a way to reset the magnification and other parameters, such as the position and orientation of the volume in the 3D view - this would be helpful for new users.
- He said that attenuation looked very good, as he normally needs to drill a hole in the skull bone. Highlighting then provided him with a better view.
- He reckoned that distortion is not a problem for the surgeons and it can even help, as they really are interested in seeing the skull and the vessel network at once.

C.2.3 Third subject: CN2

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Focus on areas of interest, i.e. neck of aneurysm, vessels, relationships.*

  - Ratings:

Bifocal	2
Fisheye	4
Volume Lens	3

  - Justification for rating: *Fisheye gives enlargement without losing much detail.*

- Were the special effects useful ? YES
  - Why ? (usefulness): *Allows you to determine the operative approach, i.e. removing bone opacity.*
  - Ratings:

Attenuation	4
Highlighting	3

  - Why ? (effectiveness): *Attenuation is more relevant for surgical approach.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *N/A*
- Why ? *N/A*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *N/A*
- Why ? (level of difficulty) *N/A.*

Strengths of the System

- *Relatively easy to use, but attenuation could be made easier.*

Weaknesses of the System

- *Not given.*

Additional notes

- Subject was not sure what to do: we explained that we were interested in finding whether the distortions are useful or not.
- He liked a lot the fisheye, due to the continuous nature. Was not very impressed with the lens or bifocal.
- He was very interested in getting a good 3D view, as he wanted to see the relationship of the neck of the aneurysm and its diameter.
- Finally, he wanted to use the transfer function editor - but had some trouble understanding the user interface.

C.2.4 Fourth subject: CN3

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Allowed an effective zoom on structure of interest while maintaining a view of surrounding bone/anatomy.*
  - Ratings:

Bifocal	3
Fisheye	4
Volume Lens	2

  - Justification for rating: *Fisheye seemed to give best view of aneurysm whilst still seeing vascular tree and bone.*
- Were the special effects useful ? YES
  - Why ? (usefulness): *The highlighting function was useful to focus on region of interest.*
  - Ratings:

Attenuation	4
Highlighting	4

  - Why ? (effectiveness): *Attenuation was essential to remove surrounding structure which occluded view e.g. skull.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *3 (slightly helpful)*
- Why ? *Helpful to centre axes for program but more natural could be to use volume view to go straight to region of interest by clicking on volume.*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *4 (easy)*
- Why ? (level of difficulty) *Easy once method become known.*

Strengths of the System

- *Responsive and present structures in detail.*
- *Allows quick definition of region of interest.*

Weaknesses of the System

- *Attenuation and magnification controls are in 2D views whilst feedback is seen on volume window.*
- *No way to fully reset magnification or orientation.*

Additional notes

- Subject brought his own datasets, with a few slices of a tumour. Unfortunately there were too few slices to give him a good 3D view - but he was very keen to try with a complete set later.
- He went straight to change the transfer function and asked how it worked.
- He started experimenting with the volume lens, but was not too impressed. Then experimented with fisheye and bifocal.
- He later wanted to remove the bone with attenuation, and asked how the different attenuation functions worked.

- He tried to position the volume as it would be used in the theatre - he reckoned that the system is very helpful to find the orientation of the aneurysm, especially if it is very close to the bone, which is a situation that is very difficult to understand only with CT slices.

C.2.5 Fifth subject: CN4*

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Allows the natural desire to focus on the region of interest without cluttering the periphery.*
  - Ratings:

Bifocal	5
Fisheye	4
Volume Lens	4

  - Justification for rating: *Allows better examination of aneurysm.*
- Were the special effects useful ? YES
  - Why ? (usefulness): *Attenuation is excellent to allow modelling of surgical approach.*
  - Ratings:

Attenuation	5
Highlighting	4

  - Why ? (effectiveness): *Highlighting is good to instantly show region of interest, attenuation models bone removal in surgery.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *4 (helpful)*
- Why ? *Surgeons and radiologists think in terms of three anatomical plains, so it is very helpful.*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *4 (easy)*
- Why ? (level of difficulty) *Having found aneurysm on axial slices, the process is very easy.*

Strengths of the System

- *Very versatile, allows different approaches to examination.*
- *Very easy to use once learnt.*

Weaknesses of the System

- *Most surgeons will need a little time to get used to it.*

Additional notes

- He started his evaluation by trying the different mapping effects, seemed to like bifocal more than the others.
- He recognized the aneurysm on the dataset as one he had already operated on - mentioned that it looked very similar to what he was seeing.
- Most of the time he used a "surgical view", i.e. one that is similar to how the surgeon will see the patient as he/she lies on the operating table.
- Interestingly, he played with the focus region but only in the axial view - probably as the region has been previously set. He changed quite a lot the region size, but not much the magnification factor - he explained that he needed to see the arteries close to the aneurysm magnified as well.
- Really appreciated the viewpoint-based attenuation - reckoned it would be tremendously useful to model the bone removal done in surgery.

C.2.6 Sixth subject: R1

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Allows better visualization.*
  - Ratings:

Bifocal	2
Fisheye	3
Volume Lens	3

  - Justification for rating: *Not given.*
- Were the special effects useful ? YES
  - Why ? (usefulness): *Very useful, application in interpretation in neurosurgery.*
  - Ratings:

Attenuation	3
Highlighting	3

  - Why ? (effectiveness): *Not given.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *N/A*
- Why ? *N/A*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *N/A*
- Why ? (level of difficulty) *N/A*

Strengths of the System

- *Can see it to becoming very useful in practice in the future.*

Weaknesses of the System

- *Will need a lot of education for users.*

Additional notes

- He asked how to move the focus region and whether it was fixed or not.
- Then he wanted to use highlighting and started experimenting with magnification and the volume lens, followed by the attenuation.
- He was a bit lost in the volume view and spent quite some time to get his bearings - as the attenuation was also turned on, he could not see the bone.
- Overall, it was a bit difficult for him to navigate in the 3D view.
- Interestingly, he was the only one that wanted to maximize the view - but then complained that the system got a bit slow.
- At the end, wanted to change the transfer function. Asked how it worked, and we explained and demonstrated how colour and opacity worked together.

C.2.7 Seventh subject: R2

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Allows one to establish relationships of anatomical details of lesion being visualized.*
  - Ratings:

Bifocal	4
Fisheye	3
Volume Lens	4

  - Justification for rating: *Not given.*
- Were the special effects useful ? YES
  - Why ? (usefulness): *Possible to position image (volume) similar to operating positioning.*
  - Ratings:

Attenuation	4
Highlighting	3

  - Why ? (effectiveness): *Allows better visualization of structures.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *4 (helpful)*
- Why ? *Very helpful and user friendly.*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *4 (easy)*
- Why ? (level of difficulty) *Easy to use, similar to other systems we use for neuronavigation, providing three slice views.*

Strengths of the System

- *Easy to use*
- *Does not require specialized hardware.*

Weaknesses of the System

- *Not given.*

Additional notes

- Subject first asked how to navigate with the mouse, as he was uncertain.
- He experimented with slice navigation and tried to zoom in the 3D view.
- Later he tried all distortion methods, and did not like fisheye a lot due to the distortion it creates.
- He wanted to remove the bone, and we showed him how to use attenuation and the different functions.
- He also tried to change the transfer function, and we explained how to do it - but he was a bit lost with the controls and we helped. It took some time for him to fully understand how colour+opacity worked together, but he eventually mastered it.
- He also was the only one that wanted to crop the volume, getting a very small region around the aneurysm.

- He reckoned the system should be very useful for training, teaching and also suggested that it could be useful for visualizing tumours.

C.2.8 Eighth subject: SHO

Task 1: Distortion (Mapping Effect)

- Was it useful to have some ability to distort ? *YES*
  - Why ? (usefulness): *Not given.*
  - Ratings:

Bifocal	4
Fisheye	3
Volume Lens	4

  - Justification for rating: *Not given.*
- Were the special effects useful ? YES
  - Why ? (usefulness): *Enhances contrast of structures of interest - avoids the need for cropping.*
  - Ratings:

Attenuation	4
Highlighting	4

  - Why ? (effectiveness): *Not given.*

Task 2: Navigating through the slices

- How helpful was it to navigate through the three set of slices to find the aneurysm ? *N/A*
- Why ? *N/A*

Task 2: Defining the focus region

- How easy was it to define the region for distorting the aneurysm neighbourhood ? *N/A*
- Why ? (level of difficulty) *N/A*

Strengths of the System

- *Easy to use.*

Weaknesses of the System

- *Not given.*

Additional notes

- She started experimenting with slice navigation and zooming. We showed her how to maximize/minimize the window and then explained that instead of zooming, she could use magnification to enlarge just the focus region e.g. with the lens.
- She tried the other methods, fisheye and bifocal. She seemed to prefer the methods that did not distort the focus region, i.e. bifocal and lens.
- She activated highlighting followed by attenuation, to enhance the view of the aneurysm and remove the bone. The menu options were a bit confusing at first.
- At this point, she decided to reposition the volume to get a view similar to to how the patient would be in the operating theatre.
- She asked whether she could crop the volume instead of using the attenuation, and we explained that she could, but the idea of attenuation is to prevent cropping - there are some cases where it would be impossible to crop without removing important structures (e.g. if the aneurysm is very close to the bone).

Bibliography

[1] 3DConnexion. Space Pilot, Space Mouse and 3D Mouse products. Online. Available at http://www.3dconexxion.com, May 2006.

[2] 3Dlabs. *Open GL 2.0 White Paper*. 1.2 edition, 2002.

[3] W. M. Adams, R. D. Laitt, and A. Jackson. The Role of MR Angiography in the Pretreatment Assessment of Intracranial Aneurysms: A Comparative Study. *AJNR Am J Neuroradiol*, 21(9):1618–1628, 2000.

[4] W. M. Adams, R. D. Laitt, J. Thorne, and A. Jackson. MRA visualization of cerebral aneurysms. *Medica Mundi*, 43(1):2–9, March 1999.

[5] Kurt Akeley. Reality engine graphics. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 109–116, New York, NY, USA, 1993. ACM Press.

[6] Kurt Akeley and Tom Jermoluk. High-performance polygon rendering. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 239–246, New York, NY, USA, 1988. ACM Press.

[7] Dirk Bartz. Advanced virtual medicine: Techniques and applications for virtual endoscopy. In *SIGGRAPH 2002 Course Notes*, 2002.

[8] Dirk Bartz. Volumetric datasets. Online. Available at http://www.volvis.org, May 2006.

[9] Dirk Bartz, Wolfgang Straßer, Martin Skalej, and Dorothea Welte. Interactive exploration of extra-and intracranial blood vessels (case study). In *VIS '99: Proceedings of the conference on Visualization '99*, pages 389–392, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[10] Patrick Baudisch, Bongshin Lee, and Libby Hanna. Fishnet, a fisheye web browser with search term popouts: a comparative evaluation with overview and linear view. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 133–140, New York, NY, USA, 2004. ACM Press.

[11] Jürgen Beck, Stefan Rohde, Joachim Berkefeld, Volker Seifert, and Andreas Raabe. Size and location of ruptured and unruptured intracranial aneurysms measured by 3-dimensional rotational angiography. *Surgical Neurology*, 65(1):18–27, January 2006.

[12] Benjamin B. Bederson. Fisheye menus. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 217–225, New York, NY, USA, 2000. ACM Press.

[13] David Benyon, Gordon Davies, Laurie Keller, Jenny Preece, and Yvonne Rogers. *A Guide to Usability: Human Factors in Computing*. Addison-Wesley, 1st edition, 1993.

[14] A. Berenstein and J. Hartman. Three dimensional arteriography. *Electromedica*, 68(Neuro 2000):27–30, 2000.

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

[16] Hal Blumenfeld. Introduction to Neuroradiology. In *Neuroanatomy Through Clinical Cases*, chapter 4, pages 95–96. Sinauer Associates, 2nd edition, 2002.

[17] Ken Brodlie and Jason Wood. Recent advances in volume visualization. *Computer Graphics Forum*, 20(2):125–148, 2001.

[18] Stefan Bruckner, Sören Grimm, Armin Kanitsar, and Meister Eduard Gröller. Illustrative context-preserving volume rendering. In *Proceedings of EuroVis 2005*, pages 69–76, May 2005.

[19] Jan Bruijns. Semi-automatic shape extraction from tube-like geometry. *Xootic*, 8(3):23–30, January 2001.

[20] Tom Brunet, K. Evan Nowak, and Michael Gleicher. Integrating Dynamic Deformations into Interactive Volume Visualization. In *Proceedings of*

Eurographics/IEEE-VGTC Symposium on Visualization 2006, pages 219–226, 2006.

[21] Jason A. Bryan. The OSC volume rendering / virtual reality library. Online. Ohio Supercomputer Center. Avaliable in http://www.osc.edu/j̃bryan/OSCVR/, 2003.

[22] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for GPUs: stream computing on graphics hardware. *ACM Transactions on Graphics*, 23(3):777–786, 2004.

[23] Ian Buck and Tim Purcell. A toolkit for computation on GPUs. In *GPU Gems: Programming techniques, tips, and tricks for real-time graphics*, chapter 37. Addison-Wesley, 1st edition, 2004.

[24] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 symposium on Volume visualization*, pages 91–98. ACM Press, 1994.

[25] M. Sheelagh T. Carpendale, D. J. Cowperthwaite, and F. D. Fracchia. Distortion viewing techniques for 3-dimensional data. In *INFOVIS '96: Proceedings of the 1996 IEEE Symposium on Information Visualization (INFOVIS '96)*, pages 46–53, Washington, DC, USA, 1996. IEEE Computer Society.

[26] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. 3-dimensional pliable surfaces: For the effective presentation of visual information. In *ACM Symposium on User Interface Software and Technology*, pages 217–226, 1995.

[27] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications: Special Issue on Information Visualization*, 17(4):42–51, / 1997.

[28] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Making distortions comprehensible. In *Visual Languages*, pages 36–45, 1997.

[29] M. Sheelagh T. Carpendale, John Ligh, and Eric Pattison. Achieving higher magnification in context. In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 71–80, New York, NY, USA, 2004. ACM Press.

[30] M. Sheelagh T. Carpendale and Catherine Montagnese. A framework for unifying presentation space. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 61–70, New York, NY, USA, 2001. ACM Press.

[31] Marcelo Cohen and Ken Brodlie. Focus and context for volume visualization. In IEEE Computer Society Press, editor, *Theory and Practice of Computer Graphics*, pages 32–39, 2004.

[32] Marcelo Cohen, Ken Brodlie, and Nick Phillips. Hardware-accelerated distortion for volume visualization in medicine. In University of Reading, editor, *Proceedings of the 4th IEEE EMBSS UK & RI Postgraduate Conference in Biomedical Engineering and Medical Physics*, pages 29–30, July 2005.

[33] Brian M. Collins. Data visualization - has it all been seen before ? In R. A. Earnshaw and D. Watson, editors, *Animation and Scientific Visualization - Tools and Applications*, chapter 1, pages 3–28. Academic Press, London, 1st edition, 1993.

[34] Benjamin Crowell. *Vibration and Waves*, volume 3. Benjamin Crowell, 1st edition, 2005. Also available online at http://www.lightandmatter.com/area1book3.html.

[35] Timothy J. Cullip and Ulrich Newmann. Accelerating Volume Reconstruction with 3D Texture Hardware. Technical Report TR93-027, Department of Computer Science, University of North Carolina, Chapel Hill, May 1994.

[36] A. del Rio, J. Fischer, D. Bartz, and W. Straßer. Fast Rendering of Large Encoded Isosurfaces from Uniform Grid Datasets. In *Vision, Modeling, and Visualization (VMV) 2005*, pages 71–78, 2005.

[37] Donald Hearn and M. Pauline Baker. *Computer Graphics with OpenGL*. Pearson Prentice-Hall, Upper Saddle River, NJ, 3 edition, 2004.

[38] Mark Dow. Example slice, MRI head. Online. Available at http://scalespace.com/slices/MRI_head_slice.shtml, May 2006.

[39] Jiang Du, Sean B. Fain, Tianliang Gu, Thomas M. Grist, and Charles A. Mistretta. Noise reduction in MR angiography with nonlinear anisotropic filtering. *Journal of Magnetic Resonance Imaging*, 19(5):632–639, April 2004.

[40] T. Todd Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, 1992.

[41] Klaus Engel and Thomas Ertl. Texture-based volume visualization for multiple users on the world wide web. In Michael Gervaut, Dieter Schmalstieg, and Axel Hildebrand, editors, *Virtual Environments '99. Proceedings of the Eurographics Workshop in Vienna, Austria*, pages 115–124, 1999.

[42] Klaus Engel, Thomas Ertl, Peter Hastreiter, Bernd Tomandl, and K. Eberhardt. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *VIS '00: Proceedings of the conference on Visualization '00*, pages 449–452, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[43] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron E. Lefohn, Christof Rezk Salama, and Daniel Weiskopf. Real-time volume graphics. In *SIGGRAPH '04: Proceedings of the Conference on SIGGRAPH 2004 course notes*, New York, NY, USA, 2004. ACM Press.

[44] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In ACM SIGGRAPH and Eurographics, editors, *Graphics Hardware*, 2001.

[45] Randima Fernando and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003.

[46] G. W. Furnas. Generalized fisheye views. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 16–23. ACM Press, 1986.

[47] C. Gasparakis. Multi-resolution multi-field ray tracing: a mathematical overview. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 199–206, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[48] GE Medical Systems. GE Healthcare - Brochure - Transcranial Color Doppler. Online. Available at http://www.gehealthcare.com/usen/ultrasound/products/cmetcd.html, May 2006.

[49] G. Gerig, O. Kubler, R. Kikinis, and F. Jolesz. Nonlinear anisotropic filtering of MRI data, 1992.

[50] Henry Gray. *Anatomy of the human body*. Bartelby.com, 2000. Online. Available at http://www.bartleby.com/107/.

[51] Sören Grimm, Stefan Bruckner, Armin Kanitsar, and Eduard Gröller. Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In IEEE, editor, *IEEE Symposium on Volume Visualization and Graphics*, pages 1–8. IEEE, October 2004.

[52] Markus Hadwiger, Thomas Theußl, Helwig Hauser, and Eduard Gröller. Hardware-accelerated high-quality filtering on pc hardware. In IEEE Signal Processing Society, editor, *VMV 2001*, pages 105–112, 520, Stuttgart, Germany, November 2001. University of Stuttgart.

[53] Markus Hadwiger, Ivan Viola, and Helwig Hauser. Fast convolution with high-resolution filtering. Technical Report TR-VRVis-2002-001, Vienna University of Technology, January 2002.

[54] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 309–318, New York, NY, USA, 1990. ACM Press.

[55] H. Hauser. Generalizing focus+context visualization. In *Proceedings of the Dagstuhl Seminar on Scientific Computing 2003*, pages 305–327, 2005.

[56] H. Hauser, L. Mroz, and M.E. Italo Bischi, G.and Groller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, Jul-Sep 2001.

[57] Ofer Hendin, Nigel W. John, and Ofer Shochet. Medical volume rendering over the WWW using VRML and Java. In *Proceedings of Medicine Meets Virtual Reality 1998*, 1998.

[58] S. Hill. *Graphics Gems IV*, pages 521–525. AP Professional, 1994.

[59] A. Hochmuth, U. Spetzger, and M. Schumacher. Comparison of three dimensional rotational angiography with digital subtraction angiography in the assessment of ruptured cerebral aneurysms. *American Journal of Neuroradiology*, 23(1):199–205, 2002.

[60] Hyman-Newman Institute for Neurology (Neurosurgery). Cerebral aneurysms. Online. Center for Endovascular Surgery. Avaliable in http://neuro.wehealny.org/endo/cond_aneurysms.asp, 2005.

[61] Milan Ikits, Joe Kniss, Aaron Lefohn, and Charles Hansen. Volume Rendering Techniques. In *GPU Gems: Programming techniques, tips, and tricks for real-time graphics*, chapter 39, pages 667–692. Addison-Wesley, 1st edition, 2004.

[62] Internet Stroke Center. Cerebral Angiography. Online. Available at http://www.strokecenter.org/pat/diagnosis/angio.htm, May 2006.

[63] Bryan Jennett and Kenneth W. Lindsay. Surgery for vascular lesions. In *An Introduction to Neurosurgery*, chapter 7, pages 142–171. Butterworth-Heinemann, 5th edition, 1994.

[64] N.W. John. High performance visualization in a hospital operating theatre. In *Proceedings of Theory and Practice of Computer Graphics 2003*, pages 170–175, June 2003.

[65] N.W. John, R.F. McCloy, and S. Herrman. Interrogation of Patient Data delivered to the Operating Theatre during Hepato-Pancreatic Surgery using High Performance Computing. *Computer Aided Surgery*, 9(6):235–242, 2004.

[66] Naftali Kadmon and Eli Shlomi. A polyfocal projection for statistical surfaces. *The Cartographic Journal*, 15(1):36–41, June 1978.

[67] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1984. ACM Press.

[68] Armin Kanitsar, Dominik Fleischmann, Rainer Wegenkittl, Petr Felkel, and Meister Eduard Gröller. Cpr - curved planar reformation. In *VIS '02: Proceedings of the conference on Visualization '02*. IEEE Computer Society, Washington, DC, USA, 2002.

[69] Armin Kanitsar, Rainer Wegenkittl, Dominik Fleischmann, and Meister Eduard. Gröller. Advanced curved planar reformation: flattening of vascular structures. In *IEEE Visualization 2003*, pages 43–50. IEEE, October 2003.

[70] Y. Kato, K. Katada, M. Hayakawa, M. Nakane, Y. Ogura, K. Sano, and T. Kanno. Can 3D-CTA Surpass DSA in Diagnosis of Cerebral Aneurysm ? *Acta Neurochirurgica*, 143(3):245–250, April 2001.

[71] Y. Kato, H. Sano, K. Katada, Y. Ogura, M. Hayakawa, N. Kanaoka, and T. Kanno. Application of three-dimensional CT angiography (3D-CTA) to cerebral aneurysms. *Surgical Neurology*, 52(2):113–122, August 1999.

[72] Andrew Kaye. Subarachnoid haemorrhage. In *Essential Neurosurgery*, chapter 9, pages 169–187. Churchill Livingstone, 1st edition, 1991.

[73] T. A. Keahey and E. Robertson. Techniques for non-linear magnification transformations. In *Proceedings of the IEEE Symposium on Information Visualizationr 1996*. IEEE Computer Society Press, 1996.

[74] Kenneth E. Hoff, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques*, pages 277–286, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[75] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In IEEE Computer Society, editor, *IEEE 1998 Symposium on Volume Visualization*, page 8. ACM SIGGRAPH, 1998.

[76] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *IEEE Visualization 2001*, pages 255–262, San Diego, CA, USA, October 2001. IEEE Computer Society Press.

[77] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Multi-dimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, page 270.285, July 2002.

[78] Gunter Knittel. The ultravis system. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 71–79, New York, NY, USA, 2000. ACM Press.

[79] Andreas König, Helmut Doleisch, and Meister Eduard Gröller. Multiple views and magic mirrors - fMRI visualization of the human brain. Technical report, 1999.

[80] R. Kosara, S. Miksch, and H. Hauser. Semantic depth of field. In *IEEE Symposium on Information Visualization 2001 (InfoVis 2001)*, San Diego, CA, USA, 22–23 2001.

[81] R. Kosara, S. Miksch, and H. Hauser. Focus+context taken literally. *Computer Graphics and Applications*, 22(1):22–29, Jan/Feb 2002.

[82] R. Kosara, S. Miksch, H. Hauser, J. Schrammel, V. Giller, and M. Tscheligi. Useful properties of semantic depth of field for better f+c visualization, 2002.

[83] Jean Krüger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization 2003*, page 38, Washington, DC, USA, 2003. IEEE Computer Society.

[84] Jean Krüger and Rüdiger Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics*, 22(3):908–916, 2003.

[85] Oliver Kuederle, Kori M. Inkpen, M. Stella Atkins, and M. Sheelagh T. Carpendale. Interacting with image sequences: detail-in-context and thumbnails. In *Proceedings of Graphics Interface 2001*, pages 111–118, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society.

[86] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94*, pages 451–458. ACM Press, 1994.

[87] Phillipe Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):218–231, September 1996.

[88] Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution techniques for interactive texture-based volume visualization. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 355–362, San Francisco, 1999.

[89] Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. A magnification lens for interactive volume visualization. In *IEEE Pacific Conference on Computer Graphics and Applications*, pages 223–232, 2001.

[90] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[91] Michael Langford. *Basic Photography*. Focal Press, 7th edition, 2000.

[92] Paul Leedy and Jeanne Ellis Ormrod. What is Research ? In Kevin Davis, editor, *Practical Research: Planning and Design*, chapter 1, pages 3–13. Merrill Prentice-Hall, 7th edition, 2001.

[93] Y. K. Leung. Human-computer interaction techniques for map-based diagrams. In G. Salvendy and M. Smith, editors, *Designing and Using Human-Computer Interfaces and Knowledge-based Systems*, pages 361–368, Amsterdam, 1989. Elsevier.

[94] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1(2):126–160, 1994.

[95] Marc Levoy. Volume rendering: Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, pages 29–37, May 1988.

[96] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

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

[98] Marc Levoy and Ross Whitaker. Gaze-directed volume rendering. In *Computer Graphics*, volume 24, pages 217–223, March 1990.

[99] Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi. *Introduction to volume rendering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[100] Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 149–158, New York, NY, USA, 2001. ACM Press.

[101] Ishantha Lokuge and Suguru Ishizaki. Geospace: an interactive visualization system for exploring complex information spaces. In *CHI '95: Proceedings of the*

SIGCHI conference on Human factors in computing systems, pages 409–414, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[102] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 163–169. ACM Press, 1987.

[103] Eric J. Luke and Charles D. Hansen. Semotus visum: a flexible remote visualization framework. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 61–68, Washington, DC, USA, 2002. IEEE Computer Society.

[104] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 173–176. ACM Press, 1991.

[105] William R. Mark, R. Steven Glanville, Kurt Akeley, and Mark J. Kilgard. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. Graph.*, 22(3):896–907, 2003.

[106] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: a general approach to setting parameters for computer graphics and animation. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 389–400, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[107] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[108] Michael D. McCool, Zheng Qin, and Tiberiu S. Popa. Shader metaprogramming. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 57–68, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[109] B. H. McCormick, T. A. DeFanti, and M. D. Brown. Visualization in scientific computing. *Computer Graphics*, 21(6), November 1987.

[110] Michael J. McGuffin, Liviu Tancau, and Ravin Balakrishnan. Using deformations for browsing volumetric data. In *IEEE Visualization 2003*, pages 401–408, October 2003.

[111] Michael Meißner, Jian Huang, Dirk Bartz, Klaus Mueller, and Roger Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the 2000 IEEE Symposium on Volume Visualization*, pages 81–90. ACM Press, 2000.

[112] A. J. Molyneux, R. S. Kerr, L. M. Yu, M. Clarke, M. Sneade, J. A. Yarnold, and P. Sandercock. International subarachnoid aneurysm trial (ISAT) of neurosurgical clipping versus endovascular coiling in 2143 patients with ruptured intracranial aneurysms: a randomised comparison of effects on survival, dependency, seizures, rebleeding, subgroups, and aneurysm occlusion. *Lancet*, 366(9488):809–817, Sep 2005.

[113] Benjamin Mora and David S. Ebert. Low-complexity maximum intensity projection. *ACM Transactions on Graphics*, 24(4):1392–1416, 2005.

[114] Lukas Mroz, Helwig Hauser, and Eduard Gröller. Interactive High-Quality Maximum Intensity Projection. *Computer Graphics Forum*, 19(3):341–350, September 2000.

[115] Klaus Mueller and Roger Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 239–245, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[116] Klaus Mueller, Torsten Möller, and Roger Crawfis. Splatting without the blur. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 363–370, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[117] Tamara Munzner and Paul Burchard. Visualizing the structure of the world wide web in 3d hyperbolic space. In *VRML '95: Proceedings of the first symposium on Virtual reality modeling language*, pages 33–38, New York, NY, USA, 1995. ACM Press.

[118] N. Neophytou and K. Mueller. GPU accelerated image aligned splatting. In *Fourth International Workshop on Volume Graphics*, pages 197–242. IEEE, June 2005.

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

[120] Erik Nolf. Medical image conversion library (MEDCON). Online. Avaliable in http://xmedcon.sourceforge.net, 2005.

[121] NoMachine. NoMachine NX - Linux Terminal Server, Thin Client Access and Management Software. Online. Available at http://www.nomachine.com, May 2006.

[122] Marc Olano and Anselmo Lastra. A shading language on graphics hardware: the pixelflow shading system. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1998. ACM Press.

[123] J. Patten and Kwan-Liu Ma. A graph-based interface for representing volume visualization results. In *Proceedings of Graphics Interface '98*, pages 117–124, 1998.

[124] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 251–260, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[125] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[126] Piotin M., Gailloud P., Bidaut L., Mandai S., Muster M., Moret J., and Rufenacht D. A. CT angiography, MR angiography and rotational digital subtraction angiography for volumetric assessment of intracranial aneurysms - an experimental study. *Neuroradiology*, 45(6):404–409, 2003.

[127] W. K. Pratt. *Digital Image Processing*. John Wiley and Sons, 3rd edition, 2001.

[128] Kekoa Proudfoot, William R. Mark, Svetoslav Tzvetkov, and Pat Hanrahan. A real-time procedural shading system for programmable graphics hardware. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 159–170, New York, NY, USA, 2001. ACM Press.

[129] Anna Puig Puig. Cerebral blood vessels modeling. Technical Report LSI-98-21-R, Universitat Politècnica de Catalunya, 1998.

[130] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[131] Ramana Rao and Stuart K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proc. ACM Conf. Human Factors in Computing Systems, CHI*. ACM, 1994.

[132] Real VNC Ltd. Real VNC (Virtual Network Computing). Online, Available at http://www.realvnc.com, May 2006.

[133] Christof Rezk-Salama. *Volume Rendering Techniques for General Purpose Graphics Hardware*. PhD thesis, Universität Erlangen-Nürnberg, December 2001. 209 pp.

[134] Christof Rezk-Salama, Klaus Engel, M. Bauer, and G. Greiner. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware 2000*, pages 109–118, 2000.

[135] Christof Rezk-Salama et al. Automatic adjustment of transfer functions for 3d volume visualization. In *Proc. of Vision, Modelling and Visualization (VMV) 2000*, page 8, 2000.

[136] Theresa-Marie Rhyne, Melanie Tory, Tamara Munzner, Matt Ward, Chris Johnson, and David H. Laidlaw. Information and scientific visualization: Separate but equal or happy together at last. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 115, Washington, DC, USA, 2003. IEEE Computer Society.

[137] John Rieman. The diary study: a workplace-oriented research tool to guide laboratory efforts. In *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–326, New York, NY, USA, 1993. ACM Press.

[138] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194. ACM Press, 1991.

[139] Stefan Roettger. The Versatile Volume Viewer. Online. Available at http://www9.informatik.uni-erlangen.de/Persons/Roettger, 2003.

[140] Stefan Roettger, Michael Bauer, and Marc Stamminger. Spatialized transfer functions. In *Eurographics - IEEE VGTC Symposium on Visualization 2005*, pages 271–278. Eurographics Association, June 2005.

[141] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation 2003*, pages 231–238. Eurographics Association, 2003.

[142] Randi J. Rost. *OpenGL Shading Language*. Addison-Wesley, 2 edition, 2006.

[143] Jeffrey Rubin. *Handbook of Usability Testing*. John Wiley and Sons, 1st edition, 1994.

[144] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *CHI'92*, volume 3:7, pages 83–91. ACM, May 1992.

[145] M. Sarkar, S. S. Snibbe, O. J. Tversky, and S. P. Reiss. Stretching the rubber sheet: a metaphor for viewing large layouts on small screens. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 81–91, New York, NY, USA, 1993. ACM Press.

[146] Chris Seitz. The evolution of GPUs. Online. Available in http://developer.nvidia.com/object/china_2004_presentations.html, 2004.

[147] Christian Sigg and Marcus Hadwiger. *GPU Gems 2: programming techniques for high-performance graphics and general-purpose computation*, chapter 20, pages 313–329. Addison-Wesley, 1st edition, 2005.

[148] Julian Smart et al. The wxWidgets toolkit. Online. Available in http://www.wxwidgets.org, 2006.

[149] R. Spence. *Information Visualization*. ACM Press, 1st edition, 2001.

[150] R. Spence and M. D. Apperley. Database navigation: an office environment for the professional. *Behaviour and Information Technology*, pages 43–54, 1982.

[151] Bill Spitzak et al. The Fast Light Toolkit (FLTK). Online. Available in http://www.fltk.org, 2006.

[152] Simon Stegmaier, Magnus Strengert, and Thomas Klein. A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Proceedings of Volume Graphics 2005*, pages 187–195, 2005.

[153] T. Sugahara, Korogi. Y., Nakashima. K., Hamatake. S., Honda. S., and M. Takahashi. Comparison of 2D and 3D digital subtraction angiography in evaluation of intracranial aneurysms. *American Journal of Neuroradiology*, 23(15):45–52, 2002.

[154] Jon Sweeney and Klaus Mueller. Shear-warp deluxe: the shear-warp algorithm revisited. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 95–ff, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[155] Melanie Tory and Colin Swindells. ExoVis: An Overview and Detail Technique for Volumes. Technical Report SFU-CMPT-TR2002-05, Computing Science Dept., Simon Fraser University, 2002.

[156] K. Tsuchiya, S. Katase, J. Hachiya, and Y. Shiokawa. Volume-rendered 3d display of mr angiograms in the diagnosis of cerebral arteriovenous malformations. *Acta Radiologica*, 44(6):675–679, 2003.

[157] University of Chicago Hospitals. Cerebral Aneurysms. Online. Available at http://www.uchospitals.edu/online-library/content=P08249, 2006.

[158] Steve Upstill. *The RenderMan Companion : A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley, 1 edition, 1990.

[159] J. E. van der Heyden, M. S. T. Carpendale, K. Inkpen, and M. S. Atkins. Visual presentation of magnetic resonance images. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 423–426, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[160] F.P. Vidal, F. Bello, K.W. Brodlie, N.W. John, D. Gould, R. Phillips, and N.J. Avis. Principles and applications of computer graphics in medicine. *Computer Graphics Forum*, 25(1):113–137, 2006.

[161] Ivan Viola, Armin Kanitsar, and Meister Eduard Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization'04*, pages 139–145, 2004.

[162] Lujin Wang, Ye Zhao, Klaus Mueller, and Andy Kaufman. The magic volume lens: An interactive focus+context technique for volume rendering. In *IEEE Visualization 2005*, pages 367–374. IEEE Society Press, October 2005.

[163] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2nd edition, 2004.

[164] Colin Ware and Marlon Lewis. The DragMag image magnifier. In *Conference companion on Human factors in computing systems*, pages 407–408. ACM Press, 1995.

[165] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level–of–detail volume rendering via 3d textures. In *Proceedings of Volume Visualizatioon 2000*, pages 7–13. ACM Press, 2000.

[166] Bryce K. A. Weir, R. Loch Macdonald, J. Max Findlay, Bruce Mielke, and Robert Wollman. Pathology, Development and Haemodynamics of Cerebral Aneurysms and Arteriovenous Malformations. In Alan Crockard, Richard Hayward, and Julian T. Hoff, editors, *Neurosurgery: the science basis of clinical practice*, volume 2, chapter 47, pages 675–694. Blackwell Science, 3rd edition, 2000.

[167] Rüdiger Westermann and Thomas Ertl. Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH 98*, pages 169–178. ACM SIGGRAPH, 1998.

[168] Lee Alan Westover. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376, New York, NY, USA, 1990. ACM Press.

[169] Lee Alan Westover. *Splatting: a parallel, feed-forward volume rendering algorithm*. PhD thesis, Chapel Hill, NC, USA, 1991.

[170] Orion Wilson, Allen VanGelder, and Jane Wilhelms. Direct Volume Rendering via 3D Textures. Technical Report UCSC-CRL-94-19, 1994.

[171] D. Winch, P. Calder, and R. Smith. (focus+context)3: distortion-oriented displays in three dimensions. In *User Interface Conference, 2000. AUIC 2000*, pages 126–133, 2000.

[172] A. Zanella, M. S. T. Carpendale, and M. Rounding. On the effects of viewing cues in comprehending distortions. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 119–128. ACM Press, 2002.