DIPLOMARBEIT

Advanced Raycasting for Virtual Endoscopy on Consumer Graphics Hardware

ausgeführt am Institut für Computergraphik und Algorithmen
Technische Universität Wien
in Kooperation mit dem
VRVis, Zentrum für Virtual Reality und Visualisierung

unter Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
in Kooperation mit
Dipl.-Ing. Dr.techn. Markus Hadwiger
Dipl.-Math. Dr.techn. Katja Bühler

von

Henning Scharsach

Matr. Nr.: 9551348

A - 1020 Wien, Wittelsbachstr. 4 / 16

Wien, im April 2005

This thesis is dedicated to the memory of Daniela Rhomberg

Abstract

Volume rendering techniques for medical applications face a number of problems that restrict the applicable techniques to a handful of established algorithms. Developing a virtual endoscopy application further narrows the choice due to the very specific demands of such a system.

First, being able to move the viewpoint into the dataset and providing correct renderings incorporating the wide field of view optical endoscopy cameras usually deliver is a challenging task at a time when many of the available professional solutions like TeraRecon's VolumePro boards are still restricted to orthogonal rendering. Second, the extreme perspective distortion of the image leads to an amplification of visible sampling artefacts, making it necessary to employ special techniques to deal with this problem.

Third, highly interactive framerates are not a welcomed feature but an absolute necessity, since the possible intra-operative environment makes immediate response to certain actions essential. And last, correct visualization and intersection of the endoscopic tools have to be ensured in order to provide the surgeon with an adequate representation of the environment.

In the past, there has always been a trade-off between functionality, interactivity and high-quality renderings resulting in systems either being able to produce interactive visualizations that lack the necessary detail and correctness of the representation, or high-quality renderings that have to be generated off-line in a tedious process that makes real-time adaptations impossible.

This thesis presents an approach that attempts to meet all the demands on a virtual endoscopy system by creating a rendering framework that allows for interactive framerates for almost every possible dataset, quality setting and rendering mode. To achieve this, a number of specialized techniques is incorporated that extend the basic rendering pipeline in numerous ways.

As virtually all of the different approaches to real-time visualization of volume datasets, raycasting on consumer graphics hardware faces its own problems and pitfalls. This is why separate sections of this thesis are dedicated to solutions to these problems that make the approach as versatile as possible.

Finally, results and real-life images of the raycaster are presented, which is already used in medical practice in pre-operative planning for neuro-surgery.

Kurzfassung

Systeme zur Volumensvisualisierung von medizinischen Datensätzen müssen eine Vielzahl unterschiedlicher Probleme lösen, was die Anzahl verfügbarer und anwendbarer Visualisierungsalgorithmen deutlich limitiert. Virtuelle Endoskopie stellt durch die spezielle Art der erzeugten Bilder noch höhere Anforderungen an die Applikation, womit die Auswahl passender Visualisierungstechniken weiter eingeschränkt wird.

Ein virtuelles Endoskopiesystem muss in der Lage sein, den extremen Sichtwinkel und die starken perspektivischen Verzerrungen, die bei den verwendeten optischen Endoskopiesystemen auftreten, adäquat zu simulieren. Dies ist besonders problematisch, da viele verfügbare professionelle Systeme wie TeraRecons VolumePro-Boards immer noch auf orthogonale Projektion beschränkt sind oder eine perspektivische Projektion nur mit speziellen Algorithmen approximieren können. Zweitens führt die erwähnte perspektivische Verzerrung zur Verstärkung von sichtbaren Diskretisierungsartefakten, was spezielle Techniken erfordert, um dieses Problem in den Griff zu bekommen.

Drittens stellen interaktive Bildfrequenzen keine wünschenswerte Erweiterung mehr dar sondern sind absolut unerlässlich, wenn es darum geht, in der intra-operativen Navigation sofort auf kleine Bewegungen und Richtungsänderungen zu reagieren. Außerdem muss noch für eine korrekte Visualisierung der endoskopischen Werkzeuge gesorgt werden, um dem Arzt eine realitätsgetreue Repräsentation des Umfeldes zu präsentieren, auf die er adäquat reagieren kann.

In der Vergangenheit musste ein Kompromiss zwischen Funktionalität, Interaktivität und Qualität der Visualisierung gefunden werden. Dies hat zu der Entwicklung von Systemen geführt, die entweder interaktive Bilder in ungenügender Qualität und mit teilweise fehlenden Details erzeugen konnten, oder Animationen von hochqualitative Darstellungen vorberechnen mussten, was jede Art von Echtzeitinteraktion unmöglich macht.

Diese Arbeit präsentiert einen Visualisierungsansatz, der den verschiedenen Ansprüchen an ein Virtuelles Endoskopiesystem gerecht zu werden versucht. Es wird ein System vorgestellt, das interaktive Bildfrequenzen für alle Arten von Datensätzen in jeder Qualitätsstufe und verschiedenen Rendermodi ermöglicht. Um das zu erreichen wurden eine Reihe spezialisierter Techniken implementiert, die den grundsätzlichen Algorithmus in vielfacher Weise erweitern.

Wie die meisten Visualisierungsansätze, so hat auch hardware-basiertes Raycasting mit eigenen Problemen zu kämpfen. Diese Probleme werden im Laufe dieser Arbeit untersucht und zum Großteil beseitigt, um das Gesamtsystem so vielseitig wie möglich zu machen.

Zuletzt werden noch Resultate aus der täglichen Praxis präsentiert, wo das System schon erfolgreich in der prä-operativen Planung von neurochirurgischen Eingriffen eingesetzt wird.

Contents

1. Introduction

Modern GPUs offer a degree of programmability that opens up a wide field of applications far beyond processing millions of triangles at ever increasing speed. Raycasting is one of these applications that can make heavy use of the built-in features of today's graphics cards. With the possibilities offered by this technology, there is a lot of room for new techniques that do not simply convert existing algorithms to the GPU, but use the very strengths of this architecture to create more realistic images at interactive frame rates.

Figure 1.1: Example Rendering of a CT scan of a human head. This quality can be achieved at near interactive framerates (approx. 12 fps on a GeForce 6), and the user can navigate into the dataset at any time for virtual endoscopy applications.

1.1 Problem Statement and Objectives

Rendering of volume datasets for virtual endoscopy applications is a computationally expensive task, mostly because of the need for perspective projection from a viewpoint within the volume. To be able to do this in realtime, most applications either convert the volume to a triangle mesh with marching cubes or, in the best case, use iso-surface rendering. Of course, both is not an optimal solution for virtual endoscopy, where the medical doctor wants to get an idea with what kind of tissue he is dealing and what is *behind* the thin structure in front of him. Furthermore, certain areas of interest should always be visible (e.g. a tumor that needs to be removed), and an existing segmentation of the dataset should not be a prerequisite.

A perspective DVR seems like the obvious solution to this, but the need for highly interactive framerates to get a good estimation of the position of different objects made this approach infeasible so far. However, with modern graphics cards exceeding the computational power of CPUs for highly parallelizable tasks and offering a more flexible feature-set than ever before, realtime high-quality perspective DVR is not impossible to achieve anymore.

This master thesis presents an approach to hardware based raycasting in the fragment shader of a shader model 3 compatible graphics card that not only allows for both orthogonal and perspective projection, but enables the user to move the viewpoint into the dataset for virtual endoscopy views. This hardware-based approach can also be used to correctly intersect the rendered dataset with normal OpenGL geometry, allowing arbitrary 3D-meshes, pointers or grids to be rendered in the same scene. This is especially important for virtual endoscopy again, because both the endoscope and the attached tools have to be visualized as well and should of course blend seamlessly into the rendered scene. Furthermore, a couple of specialized raycasting techniques is presented that further improve rendering speed, image quality and applicability of this approach, making this raycaster versatile enough for almost every possible visualization demand.

Special attention is paid to the biggest problem of GPU-based approaches - the limited amount of available video RAM - and how it can be circumvented by applying a cached blocking scheme that loads only blocks of interest into the video memory.

1.2 Structure of this Thesis

Before getting into the implementation details, this thesis gives a quick overview of the necessary fundamentals and different visualization techniques in chapter 2. Apart from comparing image and object order approaches, a quick introduction in raycasting and virtual endoscopy is given and similarities and differences of CPU and GPU based algorithms are identified. This should give the reader a better understanding of what demands a certain technique can satisfy and where strengths and weaknesses of different approaches are. Furthermore, it should explain the choice of a hardware based raycaster for the goal we were trying to achieve.

Chapter 3 will then present the basic raycasting algorithm and explain the idea behind most of the techniques presented in this thesis, focusing on the special structure of the underlying setup compared to software-based approaches.

In chapter 4, the algorithm will be extended by various techniques that improve rendering speed to achieve the goal of interactive framerates - even for large datasets and demanding transfer functions. Furthermore, a special technique will be presented that allows to correctly intersect the volume with arbitrary OpenGL geometry. This technique also assures that no parts of the volume are rendered that are hidden behind these structures, thus even enhancing rendering speed when adding geometry to the scene. The last part of the chapter is devoted to fly-through applications like virtual endoscopy and the necessary modifications to the rendering pipeline.

Image Quality will be the primary concern in chapter 5, where techniques are presented that make the image more appealing by removing certain artifacts without imposing a huge impact on overall rendering performance. Here again, special attention is paid to possible fly-through applications, that move the viewport very close to thin structures and are prone to producing sampling artifacts. With surface shaded DVR, a new rendering mode is introduced that is especially useful in virtual endoscopy applications and combines the advantages of shaded and unshaded DVR.

Chapter 6 will focus on the results we were able to achieve and compare different speeds at various quality and resolution settings. This should give a better understanding of the speedup this algorithm provides over conventional techniques. Also image-quality comparisons will be made to better understand the benefits of the presented techniques in a real-world environment.

At last, chapter 7 will conclude the results achieved and provide a short outlook into our future work and other possible applications that we are looking into. These include further development of iso-surface shaded DVR and deferred shading, leading to an even more flexible rendering pipeline, as well as extensions to the memory management and support for segmented datasets.

2. Fundamentals and State of the Art

Numerous algorithms for realistic rendering of volume datasets have been published, each of them with its very own advantages and problems. The following chapter gives a quick introduction into the different kinds of algorithms and their respective applications. The top down order of introductory sections is meant to give an insight on why exactly this algorithm was chosen for our visualization demands and what other possibilities exist.

The first section gives an answer to the first question that arises when implementing a volume rendering algorithm: With a given type of datasets and visualization demands, what kind of algorithm suits our needs best? In our case, the decision for an image-based approach directly leads to the next section, which gives a short introduction on raycasting, the available techniques and the possible implementations.

After the decision for a certain technique, a closer look is taken at the field of application: The third section is all about virtual endoscopy, the specific problems and available systems. After that, the next question is whether the algorithm should be implemented on the CPU or the GPU. CPU and GPU-based approaches are also sometimes referred to as software or hardware implementations of an algorithm, referring to the fact that many of the specific graphical instructions in such an algorithm have to be divided up into many simple instructions on the CPU while they can be immediately carried out 'in hardware' on a graphics card - this is one of the main advantages the GPU offers.

The fact that a GPU is not as versatile as a CPU heavily influences the decision, and it definitely makes no sense to implement algorithms on the graphics card that can not take advantage of the specific features. In fact, once the decision for a GPU-based algorithm is made, there are only two techniques left that have proven to be applicable and take heavy advantage of the texturing capabilities of the graphics card. Therefore the last section of this chapter compares these two specific tech-

niques, though this is actually again a comparison of an image order and an object order approach.

2.1 Visualizing 3D Datasets

Generally speaking, algorithms for rendering of volume data can be divided into two main categories: Image order and object order approaches. In the past, both of these techniques have proven to be useful for specific applications, though image order approaches seemed to be more generally applicable because the computational complexity scales with image rather than object size, which often makes it a better choice for applications where the exact size and structure of the volume is unknown.

2.1.1 Image order approaches

As the name suggests, image order approaches iterate through the parts of the image that should finally be generated and try to find all possible contributions to this part. With these parts being screen pixels most of the time, the algorithm tries to find all objects that change the appearance of a certain pixel. The most widely known image order approach is raycasting [Levoy, 1988], where a ray is cast through each pixel and sampled at regular intervals. The contributing samples of the volume are composited to a final pixel color, which in the end is a (more or less exact) approximation of the integral along that ray.

Primary advantage of this algorithm is the fact that it is much more dependent on screen resolution than on object size, which also makes it easily scalable by reducing the resolution for quick in-between renders. Also, the number of objects in the volume does not influence rendering speed as much as with object order approaches (this situation changes a little bit with the implementation of techniques like early ray termination and empty space skipping, see chapters 3.1 and 4.1).

Obvious disadvantage of image order approaches is that, if no further measures are taken, very sparsely populated volumes will be rendered a lot slower than with object order approaches, because a lot of pixels will be checked (and thus a lot of rays started) that never even hit an object. On the other hand, techniques like early ray termination make sure that in the case of very dense volumes, only those objects are rendered that really contribute to the final image.

2.1.2 Object order approaches

In contrast to image order approaches, object order approaches iterate through all parts of the object - in most cases voxels - and determine their contribution to the final image. The most popular object order approach is splatting [Westover, 1990, 1991, Wilhelms and A., 1991], where a footprint of the current object is generated and 'splatted' onto the image plane.

This works particularly well if there are only very few non-empty voxels inside a large volume. In all other cases, the probability is quite high that a substantial part of the objects will not even be visible in the final image, thus wasting a lot of computational effort.

Apart from speed concerns, object order approaches can have a substantial advantage in terms of memory consumption. This is mainly due to the fact that virtually all

image based approaches rely on some kind of regular grid to store the data, thus reserving the same amount of memory for empty and non-empty voxels. In the case of object order approaches, usually only non-empty voxels are stored, which can reduce memory demands significantly if the volume is not heavily populated.

However, another main drawback of this approach is that the appearance of the footprint limits the quality of the final image - zooming in on a splatted image will reveal the structure of the footprint quickly, making the quality of the precomputed kernel essential for good results [Westover, 1990]. Though various extensions of this algorithm have been published [Mueller et al., 1999, Mueller and Yagel, 1996, Müller et al., 1999, Huang et al., 2000], the quality of the magnification is still inferior to image-based approaches in most cases.

2.2 Raycasting Fundamentals

The basic software raycasting algorithm, as proposed by Marc Levoy in his initial publication on raycasting [Levoy, 1988], divides the process of image generation into six distinct steps, as shown in Figure 2.2. It should be noted that in order to retrieve correct color values, the voxel colors have to be premultiplied by their respective opacities before resampling, which might not immediately be obvious when looking at the pipeline [Wittenbrink et al., 1998a].

The six raycasting steps are:

Figure 2.1: Simple raycasting algorithms provide the ability to achieve high-quality visualizations of transparent surfaces.

1. The preparation of volume densities along a regular grid, resulting in voxel values for each discrete position
2. The classification of voxels, mapping each voxel density to a respective opacity value
3. Resampling of sample opacities at the discrete sampling positions along the ray
4. The shading, mapping each voxel density to a color value
5. Resampling of voxel colors at the discrete sampling positions along the ray
6. The Compositing step, calculating a final pixel color from the vector of shaded samples and respective opacities

Figure 2.2 provides a good overview over the steps that have to be carried out for color values and opacities. The first step is to prepare the volume densities for further processing, arranging the acquired values at certain positions inside the volume along a regular grid (several techniques have been introduced in the meantime that extend this approach to other grid types like unstructured grids [Weiler and Ertl, 2001, Westermann, 2001]). This step might include correction for nonorthogonal sampling grids, patient motion while scanning or even contrast enhancements, interpolation of additional samples or pre-filtering of noisy data. Figure 2.3 shows how voxels

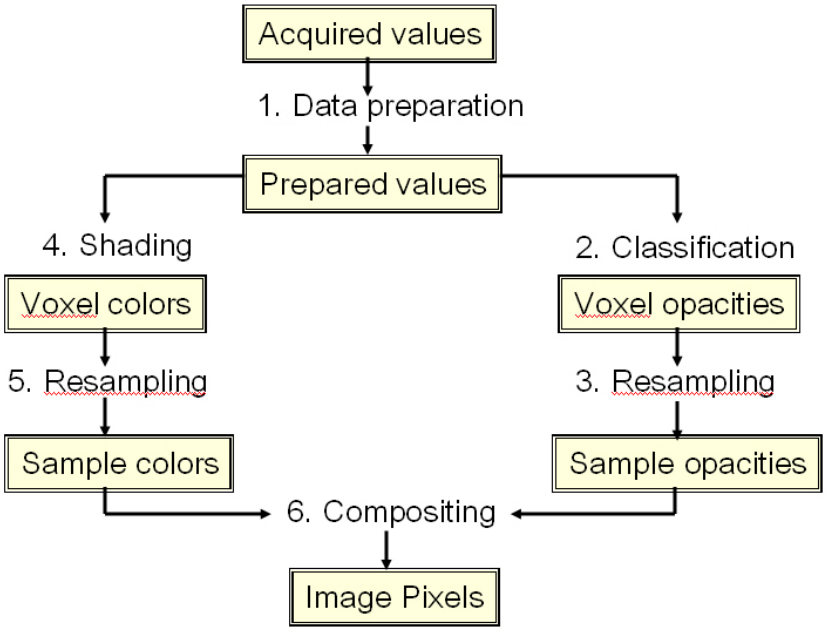

Figure 2.2: The basic raycasting pipeline, with the six steps for preparation, classification, shading, resampling of opacities and colors and finally compositing of samples.

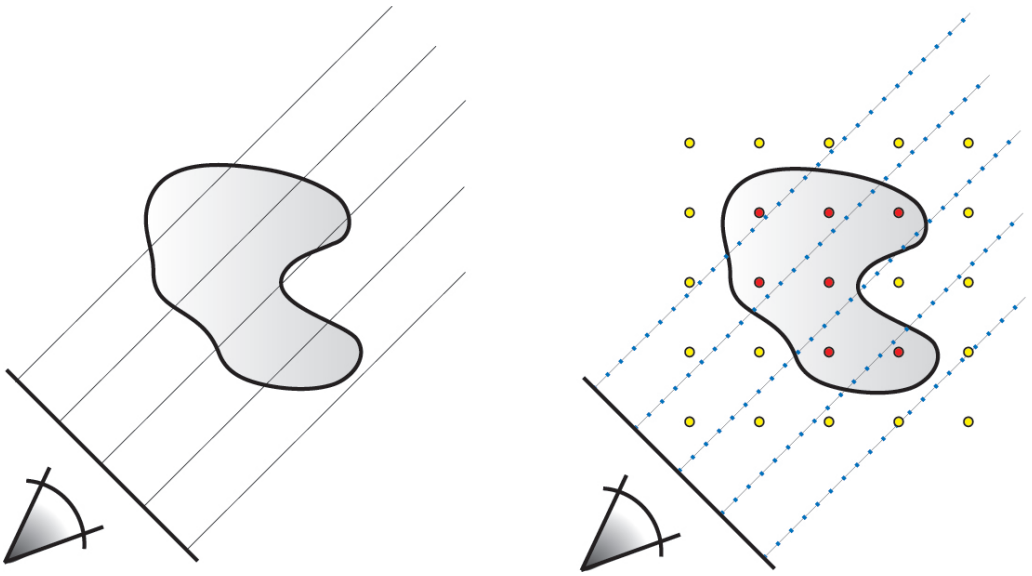

Figure 2.3: Preparing the volume densities for further processing results in values arranged on a regular grid, which simplifies further calculations and prepares the volume for rays being cast through.

of certain density are aligned on a regular grid to facilitate further processing and prepare for later resampling along the rays started from the view plane.

The output of this step is an array of prepared values which is again used as input for the shading and classification steps. In the case of shading, phong shading is used regularly because it represents a good trade-off between speed and quality. A phong-model incorporating an approximation of depth-cueing would look as follows:

$$c_\lambda(\mathbf{x_i}) = c_{p,\lambda} k_{a,\lambda} +$$

$$\frac{c_{p,\lambda}}{k_1 + k_2 d(\mathbf{x_i})} \left[k_{d,\lambda}(\mathbf{N}(\mathbf{x_i})\cdot\mathbf{L}) + k_{s,\lambda}(\mathbf{N}(\mathbf{x_i})\cdot\mathbf{H})^n \right]$$

where

- λ always denotes the respective color channel,
- x_i is the current sample location,
- c is the color value of the pixel,
- c_p the light color of a parallel light source,
- k_a is the ambient coefficient,
- k_d is the diffuse coefficient,

- k_s is the specular coefficient,
- n is the exponent for specular highlights,
- k_1 and k_2 are constants for approximation of depth-cueing,
- $d(x_i)$ is the perpendicular distance from picture plane to voxel location,
- **L** is the normalized light vector,
- **V** is the viewing vector in the direction of the observer and
- **H** is the half-vector between **V** and **L**.

The surface normal **N** is given by

$$\mathbf{N}(\mathbf{x_i}) = \frac{\nabla f(\mathbf{x_i})}{|\nabla f(\mathbf{x_i})|}$$

where the gradient ∇f is approximated using central differences:

$$\nabla f(\mathbf{x_i}) = \nabla f(x_i, y_j, z_k) \approx$$

$$\left(\frac{1}{2}\Big[f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k) \Big], \right.$$

$$\frac{1}{2}\Big[f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k) \Big],$$

$$\left. \frac{1}{2}\Big[f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1}) \Big] \right).$$

The classification performs the essential step of assigning each voxel a respective opacity value. This opacity value can be a function of various parameters, like voxel density, normal vector direction or gradient magnitude. Standard raycasting modes would include setting the opacity above a certain threshold to 1. This results in rendering of the first intersection with a value above the threshold along the ray, commonly referred to as iso-surface raycasting or first-hit-raycasting. Another common classification strategy is the simple definition of opacities for all density values via a transfer function, resulting in a visualization of translucent tissue that is used primarily for direct volume rendering (i.e. the accumulation of all color values along the ray, see the explanation of compositing below). Including the normal vector

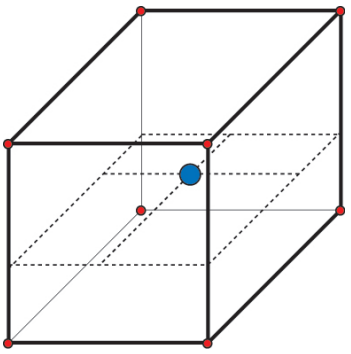

Figure 2.4: Trilinear Interpolation uses the eight neighbouring voxels to calculate an approximation of the density value at a certain sample position.

or gradient magnitude into the classification function is primarily used for non-photorealistic renderings and thus mostly found in specialized applications where these strategies provide a better insight into certain structures.

With shading and classification strategies defined, the actual algorithm is performed by casting rays into the volume and resampling the voxel densities at evenly spaced locations along that ray. The color and opacity values are usually trilinear interpolated from the eight voxels closest to each sample location (see Figure 2.4). This provides a good trade-off between simple nearest-neighbour interpolation (always take the value from the closest voxel) and more complex filter kernels like tricubic interpolation, that yield better results at higher computational demands [Hadwiger et al., 2001, Marschner and Lobb, 1994, Mitchell and Netravali, 1988].

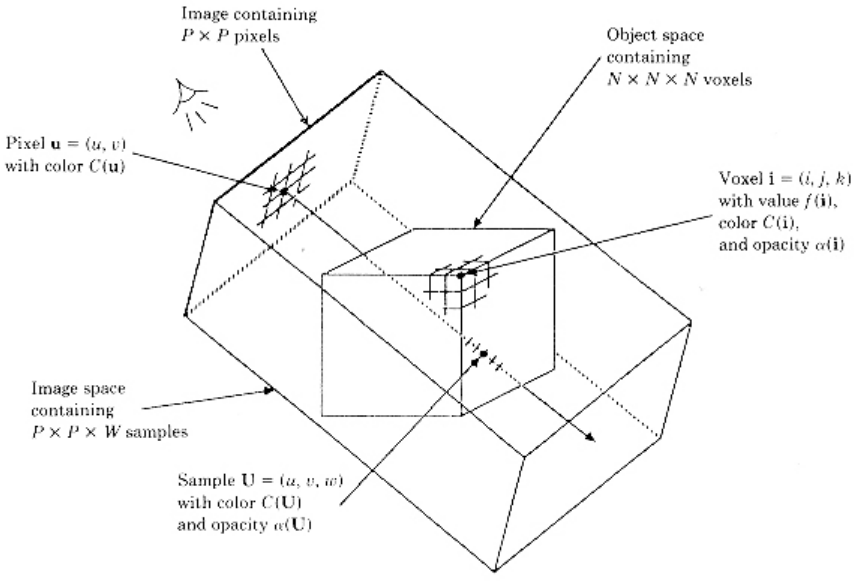

Figure 2.5: The basic raycasting algorithm casts rays from the viewing plane through every screen pixel, always calculating the coordinate translation from image space to object space. Image taken from [Levoy, 1988].

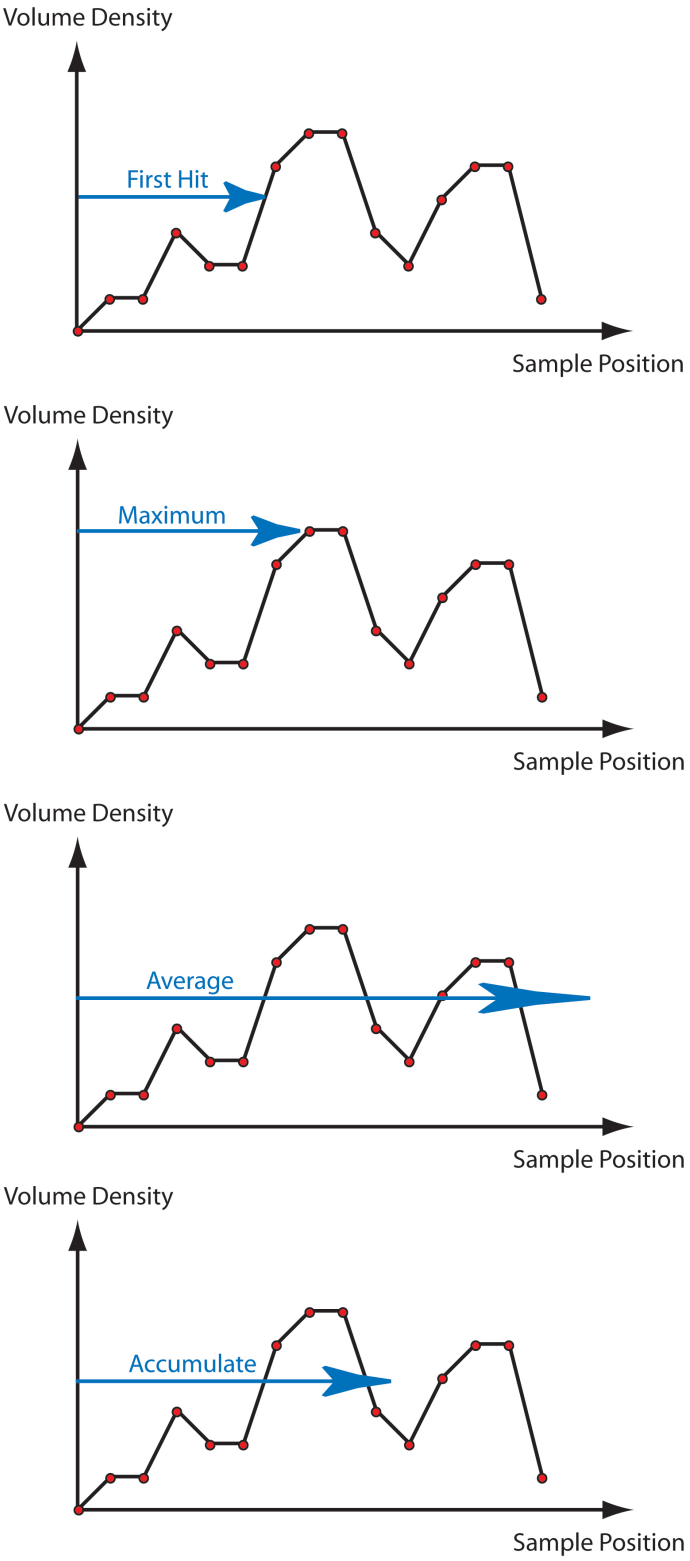

Figure 2.6: These four compositing strategies result in the rendering modes known as first-hit raycasting (iso-surface raycasting), Maximum Intensity Projection (MIP), Averaging and Direct Volume Rendering (DVR) (top to bottom).

Finally, these color and opacity values have to be composited to the final pixel color. In order to exploit strategies like early ray termination front-to-back compositing is usually used, starting the ray at the viewing plane and casting rays through every single pixel until a certain alpha threshold near 1 is reached, as shown in Figure 2.5. Figure 2.5 also illustrates the different coordinate systems used in the process of volume rendering. Front-to-back compositing calculates the summed pixel color by adding further samples according to the following formula:

$$C_{out} = C_{in} + (1 - O_{in})C_v O_v$$
$$O_{out} = O_{in} + (1 - O_{in})O_v$$

where C_{in} and O_{in} are the input color and opacity values before adding the current sample, C_{out} and O_{out} are the output color and opacity values after adding the current sample, and C_v and O_v are the color and opacity values of the sample point (i.e. the result of the trilinear interpolation of classified and shaded samples).

Depending on the compositing strategy, different rendering modes can be achieved, as shown in Figure 2.6.

2.3 Virtual Endoscopy

Minimally invasive procedures have gained increasing importance in medical practice because of the - in many cases - faster (and thus cheaper) process, the often easier and less painful way in which inner organs can be reached and the faster recovery of patients which reduces the overall risk and helps to keep clinical costs low. These procedures have proven particularly useful in surgery, neurosurgery, radiology and many other fields.

In most cases, these procedures are performed using an endoscope, which is a fiber optic of small diameter which serves as a light source, with a small camera and one or more additional tools attached to it. All these tools need to be small enough to fit through small holes in the tissue or tiny vessels. At the same time they have to provide the necessary functionality and - most important of all - a manageable way of handling them. This not only imposes a challenge for the design of suitable tools, but especially affects the way the mini cameras work.

Figure 2.7: This image demonstrates a typical endoscopic view that can be retrieved from the inside of regions that would otherwise be difficult to reach. Image taken from [Neubauer et al., 2004].

In order to provide a sufficient large opening angle at a small enough size, the spezialized lenses used in these cameras deliver a fish eye view of the environment. Besides the difficulties imposed by the small scale of the tools, this distorted view together with the limited amount of light makes controlling and navigation a difficult task. The limited flexibility of the endoscope, the limited depth perception and the necessity to constantly clean the camera lens impose additional challenges.

Furthermore, the effect of a tiny mistake in endoscopic surgeries can be devastating: Since the endoscopic approach is often used in cases where an open surgery is not easily possible, there is a high probability that the region of interest can not easily be reached in case of serious complications, such as strong bleeding. Also, the fact that an open surgery is not possible suggests that the region of interest is surrounded by tissue that can not be cut open or should not be hurt at all, like important nerves.

These facts imply that endoscopic procedures have to be carefully planned in order to avoid any complications, and medical doctors should be given the oportunity to practice the process in a most life-like environment as often as possible. Virtual endoscopy has proven to be an important tool in both of these applications, and its use has been discussed in various publications [Bartz, 2005, Auer and Auer, 1998, Auer et al., 1997].

Besides training on real specimens, virtual endoscopy provides a convenient and cheap alternative to practice the course of the surgery and has the advantage of already providing a visualization of the *real* data, which makes exact pre-operative

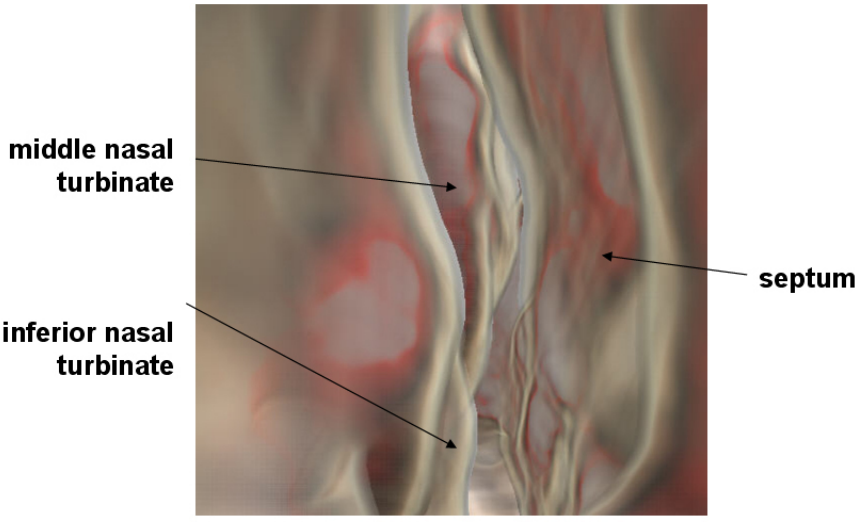

Figure 2.8: Virtual endoscopy gains increasing importance as a tool for teaching, diagnosis, pre-operative planing and even intra-operative navigation. Interactive DVR raycasting could provide additional insight, providing medical doctors with a more detailed representation of the environment. Image courtesy of S. Wolfsberger, Department of Neurosurgery, Medical University Vienna.

planning possible. This visualization is based on a 3D scan of the respective body region, like a CT (Computed Tomography) or MRI (Magnet Resonance Imaging) scan or a rotational angiography.

The resulting data from one (or more) of these scans is visualized in a way that allows interior views of the dataset, mimicking the real environment as closely as possible. Current systems either strive for interactive rendering of iso-surfaces (from polygonal representations generated with marching cubes or with an accelerated iso-surface raycaster), or high-quality renderings that have to be generated offline and can later be viewed without further possibilities for interaction. Though interactive direct volume representations would be highly desirable because of additional expressiveness semi-transparent surfaces provide and the possibility to visualize objects of interest without prior segmentation of the dataset, no system has yet been presented that is capable of delivering sufficient quality at truly interactive framerates [Bartz, 2005].

Applications for virtual endoscopy systems are not limited to pre-operative planning and practicing with endoscopic tools. They may also include teaching and diagnostic purposes as well as the possibility for intra-operative navigation. This supports medical doctors with an additional, computer-generated view of the current position and orientation of the endoscope, providing additional information about surrounding tissue and non-visible parts of the body.

2.3.1 Requirements

Virtual endoscopy applications impose a couple of requirements on a visualization system which narrows the list of applicable rendering techniques:

- Being able to move the viewpoint into the dataset is the foremost requirement, and one that not all techniques easily fulfill.
- Since the viewpoint will always be very close to surrounding tissue, acceleration techniques for this special case should be available
- Rendering speed is essential, non-interactive framerates would rule out some of the most interesting applications of virtual endoscopy systems
- The strong perspective view inside the dataset requires an algorithm that can cope with real perspective rendering and does not impose further inaccuracy by approximating certain aspects of perspective projection
- Undersampling is almost always a problem and the limited resolution of the dataset becomes often obvious. A suitable algorithm should be able to deal with this issue as flexible as possible.
- Visualization of the endoscope and the attached tools requires an easy way to correctly visualize polygonal tools and their interactions with surrounding tissue. This requires, first of all, correct intersection with the volume dataset.

2.3.2 Techniques

The already mentioned problem of undersampling is less of an issue with iso-surface rendering, where the problem can be circumvented by generating triangles with the marching cubes algorithm [Lorensen and Cline, 1987a]. However, the number of triangles generated by this approach is usually large and may prevent rendering at interactive framerates, though a couple of acceleration techniques have been introduced [Bartz and Skalej, 1999, Hong et al., 1997, Mori et al., 1996, Vining et al., 1997, Lorensen et al., 1995].

Even worse, this approach does not allow for later changes of the iso-value, making it very inflexible. Also, rendering of multiple transparent objects requires presegmentation of the dataset and slows down rendering considerably again due to the necessity for sorting.

Adaptive Raycasters [Novins et al., 1990] can cope with the undersampling by adaptively oversampling the volume in certain regions at the cost of lower framerates. These performance issues still restrict the algorithm to iso-surfacing, with all of the problems mentioned before.

Figure 2.9: Direct Volume Renderings for virtual endoscopy applications provide a lot more insight into the dataset than a simple visualization of the iso-surface.

Splatting [Westover, 1990] can be very fast for certain datasets and enables filtered reconstruction of the voxels, but can also increase blurriness of the representation [Meißner et al., 2000].

The Shear-Warp algorithm [Lacroute and Levoy, 1994, Meißner et al., 2000] faces quality issues for large magnification factors due to the base-plane approach, which make it infeasible for this kind of application.

Slice-based approaches on graphics hardware [Cullip and Neumann, 1993a] mostly suffer from the limited amount of graphics memory available even on modern GPUs and inherent problems with perspective projection, resulting in visible sampling artifacts. Hardware based raycasting algorithms can partly solve this problem, but still suffer from the video memory limitation and various inflexibilities.

With none of the presented algorithms being primarily suitable for virtual endoscopy, most available systems either incorporate only iso-surface rendering to be able to offer interactive framerates, or employ huge multi-processing systems to handle the massive computational demand of a highly interactive DVR.

2.3.3 Applications

One of the first applications of virtual endoscopy was virtual colonoscopy [Vining et al., 1994a, Hong et al., 1995, Rubin et al., 1996, Hong et al., 1997, Laghi et al., 1999, Bartrolí et al.], which is a diagnostic tool to identify and locate polyps. Beyond diagnostics, the use of virtual colonoscopy is limited because of the highly mobile organ systems of the abdomen, which is changing the absolute position and shape of the colon significantly. Thus, once a polyp or anything unusual is found, an optical colonoscopy becomes necessary to estimate the danger and remove the polyp.

Virtual bronchoscopy is another important application, but unfortunately can not significantly improve the detection of tumors [Rogalla, 1999, Rogalla et al., 2000, Bartz et al., 2003]. This limits the possible value as a diagnostic tool [Mori et al., 1994, Vining et al., 1994b, Summers et al., 1996, Ferretti et al., 1996, Rodenwaldt et al., 1997], but it still is a valuable visualization tool for various purposes like resection or biopsy planning [Wegenkittl et al., 2000, Higgins et al., 2003, Bartz et al., 2003, Mayer et al., 2004].

Virtual ventriculoscopy examines the ventricular system of the brain, which is useful for diagnostic purpose as well as planning complex endoscopic surgery [Auer and Auer, 1998, Bartz et al., 1999a, 2001b, 2002] because of its ability to visualize risk structures like arterial blood vessels [Bartz et al., 2001b]. Combined with optical endoscopy [Bartz et al., 2002, Fischer et al., 2004], it can also be used for intra-operative navigation.

Examinations of the vascular systems like cerebral arteries [Bartz et al., 1999b, Beier et al., 1997], the Aorta [Davis et al., 1996] or the heart [Bartz, 2003, Bartz et al., 2001a] is another important application, where the main focus is on diagnosis and surgery planning.

A very specific application has recently been presented by Neubauer et al. [Neubauer et al., 2004], where virtual endoscopy is used to plan a complex endoscopic procedure to remove pituitary tumors.

2.3.4 Virtual Endoscopy Systems

As described above, various developed methods of virtual endoscopy have been applied to colonoscopy [Vining et al., 1994a, Hong et al., 1997, Laghi et al., 1999, Bartrolí et al.], bronchoscopy [Mori et al., 1994, Vining et al., 1994b, Ferretti et al., 1996, Rodenwaldt et al., 1997, Wegenkittl et al., 2000, Mayer et al., 2003, Higgins et al., 2003], ventriculoscopy [Auer and Auer, 1998, Bartz et al., 1999a, 2001b], and angioscopy [Davis et al., 1996, Beier et al., 1997, Gobbetti et al., 1998, Bartz et al., 1999b, 2001a].

In all these systems, a trade-off between graphics quality and rendering speed has to be found. In many cases, only surface models [Mori et al., 1996, Vining et al., 1997, Lorensen et al., 1995, Hong et al., 1997, Bartz and Skalej, 1999, Bartrolí et al., Nain et al., 2001] extracted with the marching cubes [Lorensen and Cline, 1987a] algorithm are rendered. However, despite the fact that this is fully hardware supported, the complexity of the generated geometry regularly exceeds the capabilities of even the latest graphics accelerators, thus requiring either high-end systems [Hong et al., 1997, Vining et al., 1997], algorithms to reduce the rendering complexity [Hong et al., 1997, Bartz and Skalej, 1999, Hietala and Oikarinen, 2000], or to relinquish interactive performance [Bartrolí et al., Beier et al., 1997].

On the other hand, volume rendering techniques can greatly increase image quality or rendering speed [Shadidi et al., 1996, Hong et al., 1995, Davis et al., 1996, You et al., 1997, Gobbetti et al., 1998, Serlie et al., 2001] - unfortunately, almost always one of these two is sacrificed. Even the use of high-end hardware or multi-processor setups did not lead to satisfying results.

Available systems for virtual endoscopy include:

- FreeFlight [Vining et al., 1997]: Developed at the University of Wake Forest, FreeFlight is one of the oldest systems and based on the OpenInventor API. It requires a surface representation which is generated using the marching cubes algorithm [Lorensen and Cline, 1987a], which is then used for endoscopic examination. Also, a texture-based volume renderer is incorporated, which unfortunately is limited to unshaded representations.

- EasyVision Endo3D: Developed by Philips Medical Systems, EasyVision Endo3D is based on an iso-surface raycaster that uses a low-resolution interaction rendering for interactive framerates.

- Syngo: Syngo is the overall platform for imaging workstations of Siemens Medical solutions. It uses the VolumePro technology [Pfister et al., 1999] combined with a software approach to deliver near-interactive framerates for iso-surface raycasting.

- VESA: Like Free-Flight, VESA is also based on a polygonal surface representation of a segmented organ. Performance of a few frames per second can be achieved for standard iso-surface renderings [Davis et al., 1996, Auer et al., 1997].

- VoxelView/Vitrea2: Based on texture-mapped direct volume rendering [Shadidi et al., 1996, Rubin et al., 1996], VoxelView offers the possibility to define camera paths and generate a video animation in a time-intensive offline process. Though Vitrea2 optimized the process of path generation, it is still an DVR offline-renderer that makes it impossible to change the path or camera angles on the fly.

- VICON: While employing sophisticated approaches for segmentation and path generation, the animation is still generated off-line [Hong et al., 1995]. Real-time visualization is restricted to iso-surface rendering using a polygonal representation calculated with the marching cubes algorithm.

- V3D-Viewer: Based on the VICON-system, the V3D-Viewer provides interactive iso-surface raycasting and the possibility to render semi-transparent surfaces. Unfortunately, once rendered surfaces become semi-transparent, the high framerates break down significantly.

- CRS4: Incorporating an texture-mapping-based approach [Cullip and Neumann, 1993a] using graphics hardware, this system provides rendering performance of a few frames per second for an unshaded DVR.

- VIVENDI: Also based on the VICON system, VIVENDI renders iso-surfaces requiring a polygonal representation of the volume calculated with the marching cubes algorithm. It introduces many enhancements that speed up rendering to achieve near interactive framerates.

- VirEn: Developed at the Vienna University of Technology, VirEn [Bartrolí et al., Wegenkittl et al., 2000, Bartrolí, 2001] also requires a polygonal representation generated with the marching cubes algorithm to provide interactive framerates for iso-surface renderings. Alternatively, direct volume rendering can be performed by utilizing the VolumePro system [Pfister et al., 1999]. Due to the limitation of this system to orthogonal representation, an algorithm was proposed that renders single slabs which are then warped to simulate perspective projection - unfortunately, to achieve sufficient quality, a high number of slabs is needed which in turn leads to non-interactive framerates.

- J-Vision: J-Vision from Tiani is a Java-based diagnostic workstation that features, among many others, a virtual endoscopy plug-in [Neubauer et al., 2004]. This plug-in allows for iso-surface rendering at interactive framerates. The iso-surface view can be enhanced with additional details about the density of the surface by taking more than one sample in proximity of the found surface.

- 3D-Slicer [Gering et al., 2001]: 3D-Slicer is a joined effort of the AI lab at MIT and the Surgical Planning Lab at Brigham's and Women's Hospital in Boston. Largely based on VTK, this system incorporates no additional acceleration techniques. A virtual endoscopy mode was just recently added, which again uses a surface model of a segmented organ to render iso-surfaces at interactive framerates.

2.3.5 Conclusion

Most existing techniques rely on a polygonal representation of segmented objects which is created with the marching cubes algorithm or a simple iso-surface ray-casting to achieve near-interactive framerates. Seeing the need for improvement of expressiveness, J-Vision incorporates an enhanced mode that supplies additional details about the properties of tissue and supports semi-transparent visualization of objects of interest.

A minority of systems allows for offline generated direct volume renderings on predefined paths, which results in expressive high-quality animations. Unfortunately, the lacking flexibility makes this approach useless for many of the interesting virtual endoscopy applications like intra-operative navigation.

So far, no system has been presented that allows for high-quality direct volume renderings at truly interactive framerates, which would be the next logical step in virtual endoscopy. Not only would this allow for easier evaluation of the density of surrounding tissue, but it would enable surgeons to better estimate the position of objects of interest that might not be visible in iso-surface renderings without the necessity of a pre-segmented dataset.

2.4 CPU vs. GPU based approaches

During the last couple of years, the role graphics cards play in modern computer systems has significantly changed. With the complexity and size of these chips already being higher than their general purpose counterpart and the flexibility increasing up to a level of a fully programmable chip with dedicated instructions, GPUs are no longer limited to calculating and rasterizing triangles at ever increasing speeds.

Instead, a large number of different applications for these chips have already been published, including simulations, physics frameworks, sound systems and even general purpose math libraries [Harris et al., 2002, Krüger and Westermann, 2003b]. This development lead to heavy discussion about what should actually be implemented on a GPU and what should. With this topic becoming more and more controversial, two distinct groups of people seem to emerge, with some others still waiting which side will be proved right in the end: The GPU enthusiasts, who want to try everything that can possibly be done in hardware on their graphics card, and the CPU programmers, who rather take the software approach because they do not want to risk features not being available on the GPU.

Compared to CPU-based approaches, the specific architecture of the graphics card requires different algorithms, and porting the same technique from the CPU to the GPU will not make sense in most cases. Good hardware based algorithms try to utilize the specific advantages a GPU has over a CPU in the best possible way, namely:

- A massively parallel architecture
- A separation into two distinct units (vertex and fragment shader) that can double the performance if the workload can be split accordingly
- Incredibly fast memory and memory interface
- Vector operations on 4 floats that are as fast as scalar operations
- Dedicated instructions for graphical tasks

More advantages may arise through the specific nature of a GPU-based algorithm. Since the environment is very different to that on a CPU, a lot of standard tasks of the GPU can be used to calculate necessary information in a very efficient way. Most of these advantages come from the use of implicit interpolation, texturing capabilities or the available buffers and their efficient implementation in graphics hardware (i.e. the hierarchical z-buffer). An algorithm like the raycaster presented in this thesis can take advantage of features like:

- Automatic calculation of ray positions by letting the hardware interpolate color values
- Built-In Fast Trilinear Interpolation of 3D-Textures

- Full floating point compositing at almost no cost
- Changing from orthogonal to perspective projection without additional effort
- Automatic calculation of intersections in the depth buffer

At the same time, these algorithms have to either circumvent or live with some of the disadvantages a GPU approach faces:

- Restriction of video memory
- No integer operations at this time
- Programmability still restricted in a number of ways, like limited loop count and limited conditional statements
- Readability of a GPU shader is still inferior to standard high-level languages
- Different vendors support different features and extensions, making it difficult to write an algorithm for every plattform
- Choice of API may be more crucial than on the CPU (OpenGL or DirectX? Assembler fragment programs or high-level shading language? And if so, which shading language?)
- Unstable drivers, half-implemented features etc...

That said, hardware approaches can often impress with amazing speed gains compared to software approaches, but at the same time require a very specific system with a certain graphics card and certain drivers and extensions available. The algorithm presented in this thesis is no exception, as there is at this time just one GPU that supports the required shader model 3 and the newest drivers are required to assure smooth execution.

However, with current development moving towards unified feature sets and the APIs becoming more and more complete, it should not be too long before GPU algorithms may run on every system regardless of the configuration.

For the algorithm presented in this thesis, the advantages of a GPU based approach outweigh the disadvantages, and the end result is a combination of speed and quality that would not have been possible to achieve otherwise. With the main disadvantages still concerning the programmability, readibility and the ease of use, hardware based algorithms just require a bit more work than their software counterparts. The only major disadvantage left is the limited video memory, which is adressed in chapter 4.2, but with the introduction of 512MB graphics cards and PCI-Express, allowing faster transfers to and from video memory, this is much less of an issue than it used to be.

2.5 GPU-based algorithms

In the field of hardware-based volume rendering, there are two distinct approaches for rendering datasets at highly interactive framerates. The first approach, as originally presented by Cullip and Neumann [Cullip and Neumann, 1993b] and further developed by Cabral et al. [Cabral et al., 1994a], is directly exploiting the GPUs texture mapping capabilities by creating some kind of (usually planar) sampling surface - either viewport aligned [Westermann and Ertl, 1998] with one 3D-texture, or object (axis) aligned [Rezk-Salama et al., 2000a] with a set of 2D-textures - and resampling the original data at this so-called proxy geometry. These two approaches are shown in Figure 2.10. Object-aligned techniques using a stack of 2d-slices are usually faster and easier to handle. However, since one separate stack has to be stored for every principal viewing direction, this tripples memory demands and leads to noticable switching when rotating the dataset. In comparison, viewport-aligned algorithms use one 3D-texture to store the data and generate the view-dependant geometry on-the-fly.

Both techniques are widely accepted now as a common way to render medium sized datasets in acceptable quality at interactive framerates and have been revisited, finetuned and extended many times, e.g. [Westermann and Ertl, 1998, Engel et al., 2001, Van Gelder and Kim, 1996, Meißner et al., 1999].

Though this approach is very similar to the way computer games make use of the GPU, which ensures that it runs at the highest possible speed, it has two serious drawbacks, which are all based on the fact that this is an object order approach: First, with standard texture-based slicing, everything that needs to be calculated for the final result, every texture fetch, gradient or lighting calculation, has to be done for every single fragment, no matter if it contributes to the final image or not. Advanced techniques like empty space skipping have been developed for texture-based approaches, but are very difficult to implement because of the unflexible nature of the algorithm. [Li et al., 2003, Li and Kaufman, 2003]. Second, implementing perspective projection (or even fly-through modes) and dealing with the resulting sampling artifacts is almost impossible.

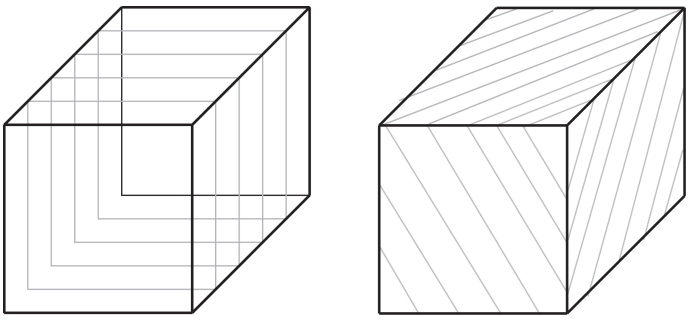

Figure 2.10: Slices in texture-based approaches can either be object-aligned (left) or viewport-aligned (right).

The first problem can be circumvented for the most part by extending the algorithm and has become less of an issue now, though all implementations are still not as efficient as in comparable raycasting approaches. But the second problem is still not solved satisfactorily and the lack of perspective projection limits the possible applications of this technique.

The second approach would be to implement a raycaster in the fragment shader of the GPU, as proposed by Krüger and Westermann [Krüger and Westermann, 2003a]. The basic idea here is to have two color images that represent the starting and ending positions of the ray in volume coordinates (i.e. texture coordinates for the lookup into the 3D-texture). These images can simply be generated with normal colored OpenGL geometry, so that all the interpolation work is done by the graphics card and smooth transitions of the vectors are achieved. By subtracting the starting position from the ending position, viewing vectors for every single screen pixel are retrieved and can be used to perform the raycasting. This approach is used as a basis for our raycasting environment, so it is discussed in greater detail in chapter 3.

Since this algorithm uses the graphics card in a very different way than most games do, there is often some additional effort required to find the most efficient solution for a certain task. Still, this approach is far more flexible, leaves more room for extensions and, most important, allows for perspective projection. Thus the decision for the raycasting approach is an obvious one when implementing a system that should be ready for virtual endoscopy applications. The various extensions that make for a complete full-fledged ray casting system for every possible kind of application will be presented in chapters 4 and 5.

3. Basic GPU Raycasting

As mentioned in the last chapter, the GPU raycasting algorithm ist built around the basic idea that normal geometry is rendered into a buffer with the the position of this geometry encoded in the color channel. OpenGL will interpolate the color values automatically, creating a correct position value for every single pixel. This way, it is possible to retrieve the position for a certain pixel later on with a single lookup into this image at the very same position.

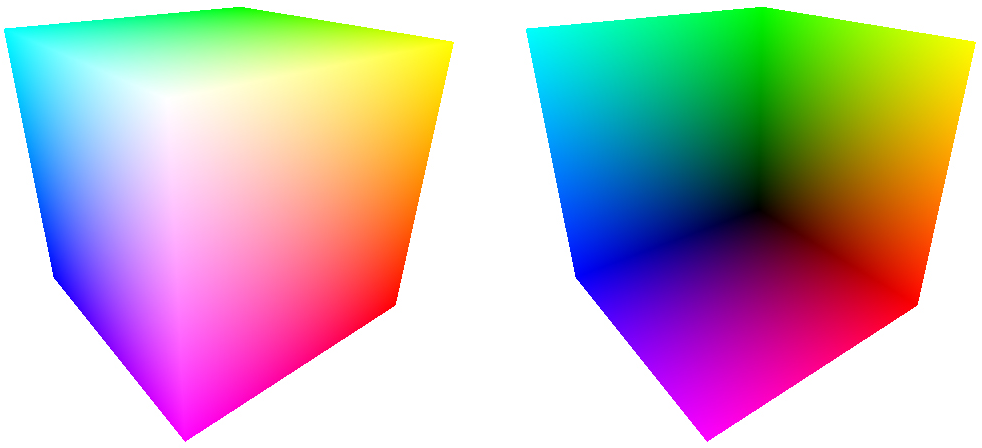

Figure 3.1: Rendering only the front or back faces of the color-coded bounding box retrieves starting and ending positions for the rays in volume coordinates for every single screen pixel.

3.1 Hardware-based Raycasting

For casting through a volume, a starting position as the ray enters the volume and an ending position when the ray leaves the volume are necessary. These two images can very easily be created by rendering a color-coded volume bounding box. Rendering only the front faces of this bounding box retrieves the starting positions of the rays at each pixel position while rendering the back faces in a second pass returns the respective ending positions. Both of these images can be seen in Figure 3.1.

By subtracting these two images, a 'direction image' or 'direction texture' is created, that holds the actual viewing vector for each pixel in volume coordinates. This way, a single lookup into this texture at a certain pixel position retrieves a viewing vector that just has to be multiplied with the position along that ray. This procedure is repeated until the ray has left the volume.

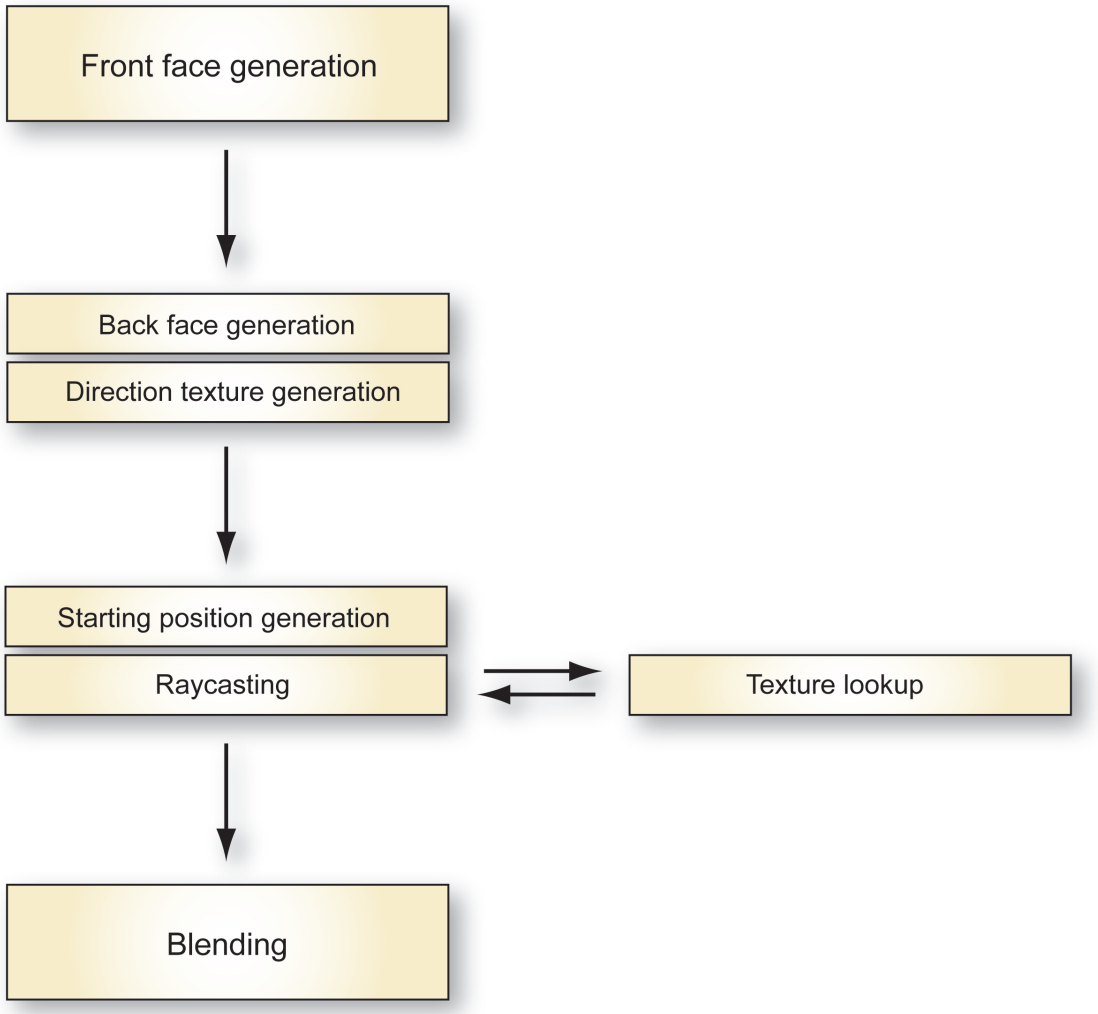

Figure 3.2: The rendering pipeline of the basic GPU raycasting algorithm.

All the values along that ray are composited, stored in a separate buffer and blended back to screen in the last pass. The final pixel color then is the equivalent of the integral along that ray, or at least a good estimation of it if the sampling rate was sufficient. The whole rendering pipeline is outlined in Figure 3.2.

Looking at the steps outlined above, it seems obvious that one should try to minimize the number of passes needed to generate the final image. The only image that is not needed for further processing is the backface image - containing the ending positions for the rays - because this information is implicitly present in the direction texture. Thus, it seems obvious to move backface rendering and direction texture generation into the same pass. This can be achieved with a simple fragment program that subtracts the value from the front face buffer at the same pixel position from the incoming back face color, immediately retrieving the viewing vector. For easier computation of the sampling position along the ray, the viewing vector should be normalized before it is written out to the buffer. Storing the initial length in the alpha channel makes it very easy to check whether the ray has already left the volume later on.

That said, the final raycasting algorithm comprises four passes:

1. front face generation: render the front faces of the color cube to a buffer.
2. direction texture generation: render the back faces of the color cube, subtract the front face color and store the normalized viewing vector together with its length in a separate direction texture.
3. raycasting: get the starting position from the front face image and cast along the viewing vector until the ray has left the volume.
4. blending: Blend the result back to the screen.

3.1.1 Front Face Generation

As mentioned before, the first pass has nothing else to do other than provide the starting positions for the rays. This can be very easily achieved by rendering only the front faces of the volume bounding box, where every corner vector is assigned its respective position as color value. Since the volume bounding box is always convex, there cannot be more than one front face for a particular pixel position, making any kind of depth test unnecessary.

The resulting image is a simple color cube, as shown in Figure 3.1, that is stored in a separate texture for later retrieval.

3.1.2 Direction Texture Generation

For direction texture generation, only the backfaces of the color cube are rendered. Again, there can only be one backface per pixel. Only this time, the result is not directly stored in a texture but rather given as input to a fragment program, which

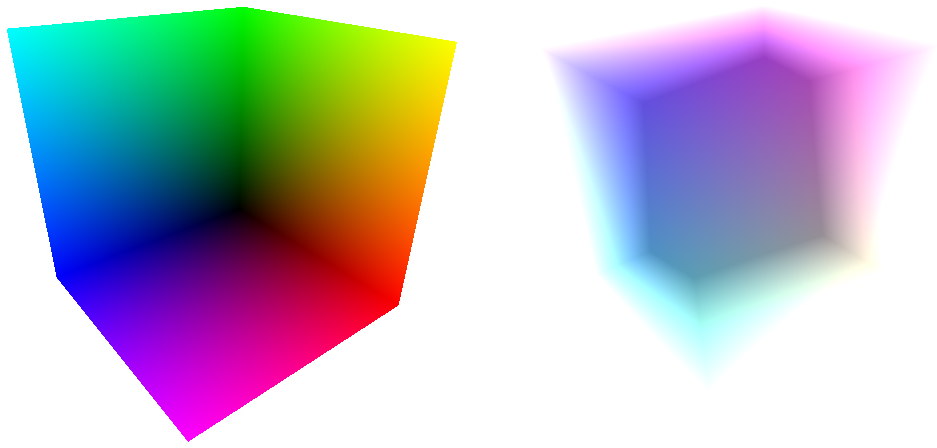

Figure 3.3: The back faces of the color cube (left) are never rendered out to a buffer, instead a direction texture is immediately created in the fragment program (right). Note that the right cube only appears to be smaller because of the low alpha values towards the border regions.

is responsible for generating the direction texture. The left image in Figure 3.3 shows the actual backface image of the color cube, that is never really written out to a buffer, but taken as intermediate step for calculation of the right image (the direction texture).

The fragment program gets the color value (i.e. position) of the backfaces and the current pixel position as input and makes a texture lookup at the same position into the front face image to retrieve the starting position. Subtracting these two values now retrieves the viewing vector. Normalizing this vector and retrieving the initial length can easily be acomplished in the fragment program, because there are dedicated instructions for both tasks.

Since the difference of starting and ending positions always result in a correct viewing vector for every possible viewing matrix, this scheme works for both orthogonal and perspective projection, which qualifies this approach for a wide variety of applications. All the setup for perspective projection and computation of the viewing vector, which takes quite some time in a software approach, is carried out by the graphics hardware in almost no time, because rendering of two color cubes imposes no challenge for a modern graphics processor.

With the generation of the direction texture, we have the perfect setup for the raycasting pass, which is still the computationally most expensive step of the algorithm.

3.1.3 Raycasting

For the actual raycasting to take place, the rays have to be started off by rendering some kind of geometry that will call the respective fragment program for every pixel. Rendering one quad filling the whole screen would be sufficient, but since the

geometry of the bounding box is that simple, there is no reason not to render the front faces of the color cube again, making sure that only those rays are started that have a valid starting position thus avoiding unnecessary checks.

The raycasting fragment program gets the color value (i.e. starting position of the ray) and the current pixel position as input and makes one texture lookup into the direction texture, retrieving the normalized viewing vector and the length.

All that is left to do now is to calculate the sample positions along the ray by multiplying the viewing vector with the respective sampling offsets and adding this vector to the starting position, which results in the absolute position within the volume. One lookup into the 3D volume texture retrieves the density value at this position, which is automatically trilinear interpolated by the graphics hardware.

Depending on the rendermode, this density value is compared to an iso-value or multiplied by the transfer function, usually stored as a 1D-texture. If shading is to be applied, another six lookups into the 3D texture have to be performed to calculate the gradient at the sampling position. Because video memory is precious on the graphics hardware, storing precomputed gradients is not an option anymore - even more since a gradient consisting of three floating point values takes six times the space of the density value (three floats with four bytes each have to be stored instead of one two-byte integer). The computed gradient serves as an estimation of the surface normal for the lighting calculations, which are again carried out by dedicated instructions on the GPU.

The final color contribution of the sample is summed up in a separate compositing buffer, and the next sample is taken until the ray has left the volume. To make use

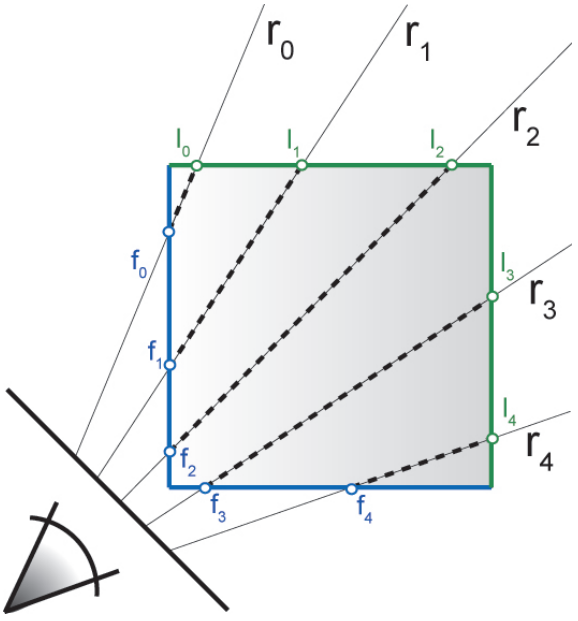

Figure 3.4: In the raycasting pass, the volume is sampled at regular intervals between the precomputed starting (f_0-f_4) and ending (l_0-l_4) positions.

of the advantage the image based approach offers, early ray termination should also be implemented, terminating the ray if the summed alpha value exceeds a certain threshold near 1.

In the case of iso-surface extraction, there is no compositing and the raycaster immediately terminates after the first successful lookup retrieving a value greater than the predefined threshold.

3.1.4 Blending

It should be noted that a separate blending pass is not a necessity, since all shader model 3 enabled graphics cards can perform floating point compositing in the screen buffer. Still, having a separate blending pass keeps the approach very flexible and allows for post-processing effects for future applications. Additionally, it gives the theoretical ability to blend together different volumes or even different parts of the volume that were rendered separately for a certain reason.

At the moment, the only feature taking advantage of this is the geometry intersection, which will be presented in chapter 4.3. However, the separate blending pass imposes no noticable performance hit and for this reason is performed even if geometry intersection is deactivated.

3.2 Implementation Details

So far we have introduced two ways of terminating a ray: The regular termination takes place once the ray leaves the volume, and is calculated by comparing the travelled distance to the length of the original viewing vector stored in the direction texture. However, to do this in the same pass the GPU has to be able to execute conditional breaks inside the loop, which requires a shader model 3 capable graphics card.

The introduced early ray termination can also only be efficiently implemented on such a GPU, since it requires one additional condition after every single sample, checking whether the accumulated alpha values have exceeded a certain threshold. By using the conditional registers introduced with the newest generation of graphics cards, these two checks can be carried out together, resulting in only one conditional break statement.

Implementing this with shader model 2 would require a separate ray termination pass, where we face a trade-off between two techniques: Having a termination pass after every sample requires 2 passes per sample, but provides the ability to exactly terminate the ray where necessary thus only calculating samples that are part of the final image. On the other hand, executing the termination pass only after a number of raycasting passes, which could themselves again calculate a number of samples, has less negative impact on performance but introduces the problem that the ray may be sampled outside the bounding box.

For a simple bounding box setup, this may be a neglectable disadvantage, because the volume outside the bounding box is empty anyway, leading only to a small performance hit. But when introducing advanced techniques like cache textures or geometry intersection, it often has to be made sure that rays are terminated correctly, because samples after the termination position may already be invalid or at least should not be part of the final image.

Thus, it makes sense to restrict the system requirements to shader model 3 enabled graphics cards, to account for all future enhancements of the algorithm and keep the pipeline as flexible as possible.

Another important point is that settings for the precision in the fragment program should be changed from ARB_precision_hint_fastest (the standard setting in ARB fragment programs, employing only 16-bit interpolations on nVidia hardware) to ARB_precision_hint_nicest (forcing the driver to full 32-bit precision on current nVidia cards) in order to get correct results. ARB_Precision_hint_fastest is preset because it represents a good trade-off between speed and quality. This is because current hardware is very much optimized to get the maximum possible performance out of regular applications, mostly games. ARB_Precision_hint_nicest will make sure that all values are interpolated with the maximum available precision, which is a necessity for the calculations that are carried out in the process of raycasting.

However, this precision may still not be enough: Color values are usually not used to store anything else than colors for screen rendering, where the precision that can be

observed by the human viewer is limited. Thus, this is an obvious target for driver optimizations and color values will normally be interpolated with less precision than for example texture coordinates, where precision defficiencies would be immediately detected. Thus, in addition to having to enable maximum precision, choosing color values for storage of the data does not seem like a good idea.

The algorithm presented in this thesis relies heavily on view vector precision, and the view vector itself is calculated from two interpolated color values. Even on highest quality settings, the interpolation is not sufficient for our purpose, so another way of calculating this vector is needed. For illustration purposes, all techniques in this thesis refer to the color value of the geometry, and thus all images were created by rendering intermediate results to the color buffer. The color buffer also provides anough precision to store the results at the end.

But when rendering intermediate geometry, the position inside the dataset is actually encoded in the texture coordinates, retrieving full interpolation precision. The change in the fragment program is trivial, because only the source of the input vector has to be changed from the color to the texture coordinates register.

4. Advanced Raycasting

The basic algorithm presented in the last chapter is simple, elegant and reasonably fast. However, there are a couple of shortcomings that limit the applicability for all kinds of applications. First, because of the lack of optimization incorporated, it is still rather slow when compared to slicing approaches, simply because these techniques make better use of the GPUs architecture and especially the triangle throughput. As mentioned earlier, the strength of image order approaches becomes primarily visible once advanced features like empty space skipping are incorporated.

Second, the dataset to be rendered can be only as big as the video memory and graphics driver permits. This not only restricts the applicability to datasets with an

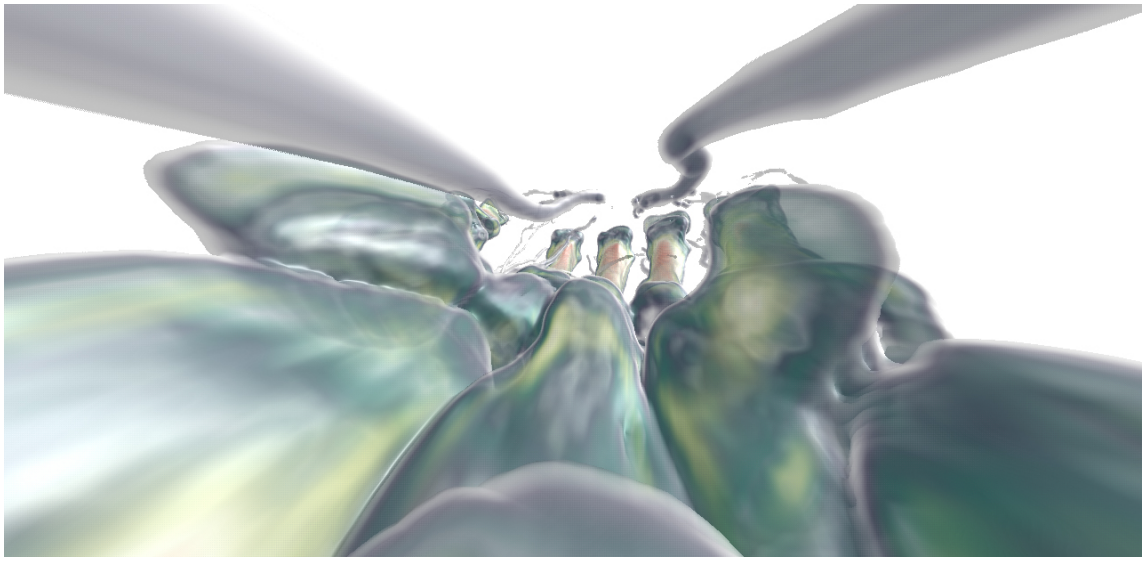

Figure 4.1: Combining the advanced raycasting techniques presented in this chapter allows for rendering of high-quality images from any angle within the volume.

overall size of less than about 400MB, but due to the state of the graphics drivers at the time of writing also to no more than 512 voxels resolution in any axis, regardless of overall size.

Third, the generation of rays on the border of the bounding cube makes sure that no empty space outside the volume has to be skipped, but on the other hand introduces additional sampling artifacts that resemble the outer shape of the bounding geometry, making the situation on undersampled datasets even worse.

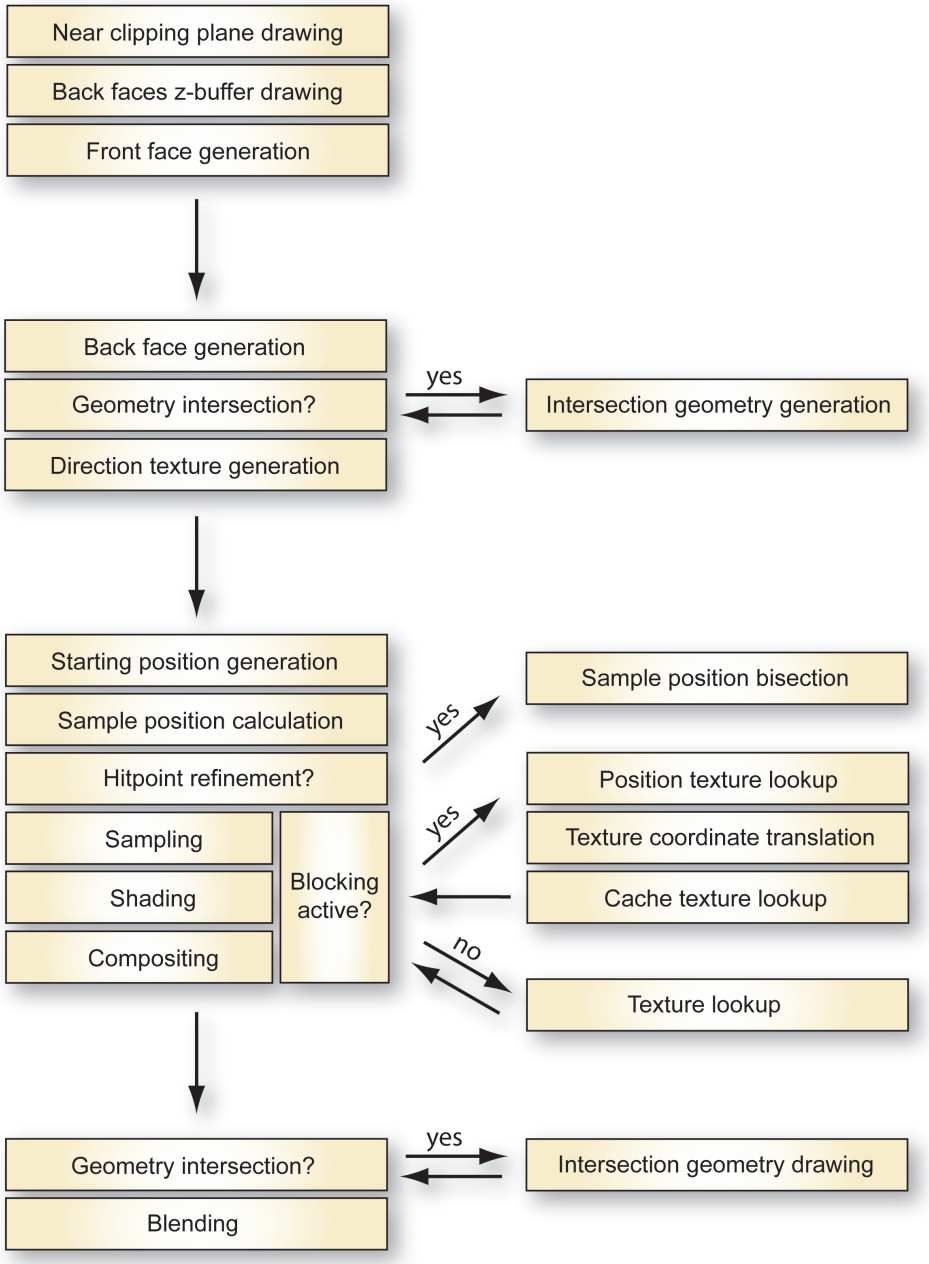

Figure 4.2: The enhanced rendering pipeling of the GPU raycaster. Throughout this chapter, the different parts of this pipeline will be explained.

And last, the generation of rays with normal OpenGL-geometry heavily relies on the fact that this geometry is always visible on the screen. Moving the viewport around - and especially into the volume - can cause serious problems as soon as parts of this geometry are clipped against the near clipping plane, resulting in visible holes in the image.

This chapter presents various techniques that help to deal with these shortcomings, making this hardware raycasting flexible and versatile enough for almost every kind of application. The enhanced rendering pipeline is presented in Figure 4.2, which will be explained in the course of this chapter.

4.1 Bounding Geometry Generation

When a visualization system is to be used in medical practice, overall performance is one of the crucial factors. It is the feeling of a natural, truly interactive 3D-model that sets a realtime visualization system apart from the old-fashioned stack of CT-slices and gives an unparalleled insight into the three dimensional structure of the object. Advanced shading techniques make for very impressive, almost artistical looking renderings of medical datasets. But the primary use of light and shadow in visualization renderings is that they give the human eye a hint of the property of the surface, and the position and orientation of objects in three dimensional space. This effect is dramatically stronger once an object starts to move and the human brain starts to process all the light reflections and shadow movements on the surface.

Furthermore, the result is probably visualized on a non-stereoscopic computer monitor, that lacks the most important hint the human eye needs when judging spatial properties of objects in real life. Thus, the visualization system has to make up for this and provide other important features of lifelike interaction in the best possible way, and nothing destroys the illusion of a manipulable object more than an image slideshow running at one or two frames per second.

Another important point is that a visualization system may not only be used for diagnostic purposes or in pre-operative planning - Virtual Endoscopy for example could also support the process of a surgery, delivering computer generated pictures of the area of interest while the endoscope is moving forward and allowing the surgeons to better estimate the consistency of the tissue they're about to hit or even

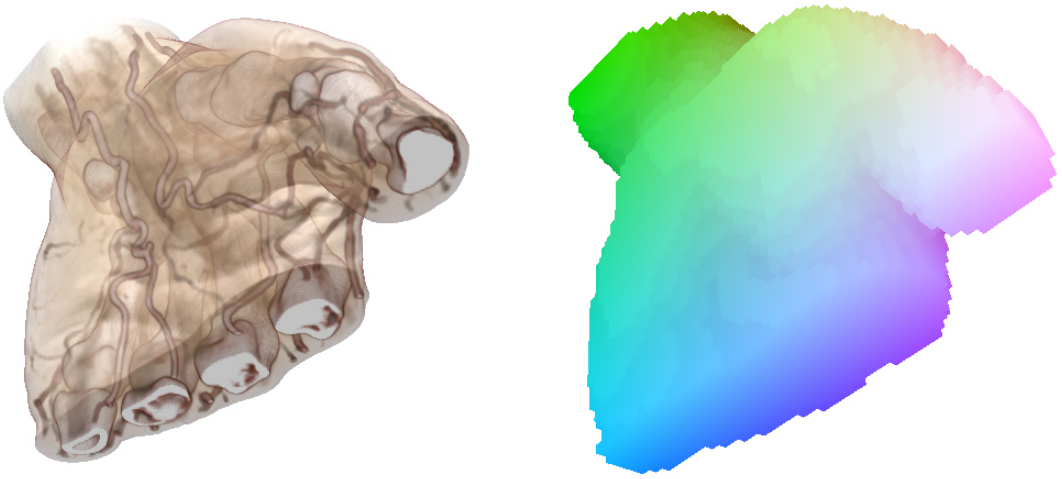

Figure 4.3: Replacing the simple bounding box in the last chapter with a more sophisticated bounding geometry implicitly skips all of the outer empty space at almost no cost.

see through it. If the system can not instantly react to every tiny movement the operator is making, it is probably only of limited use.

Thus, developing a good system for medical practice is all about achieving interactive framerates while not sacrificing image quality.

Various techniques have been published to speed up the process of raycasting, but not all of them are suitable for a GPU based approach. Early Ray Termination

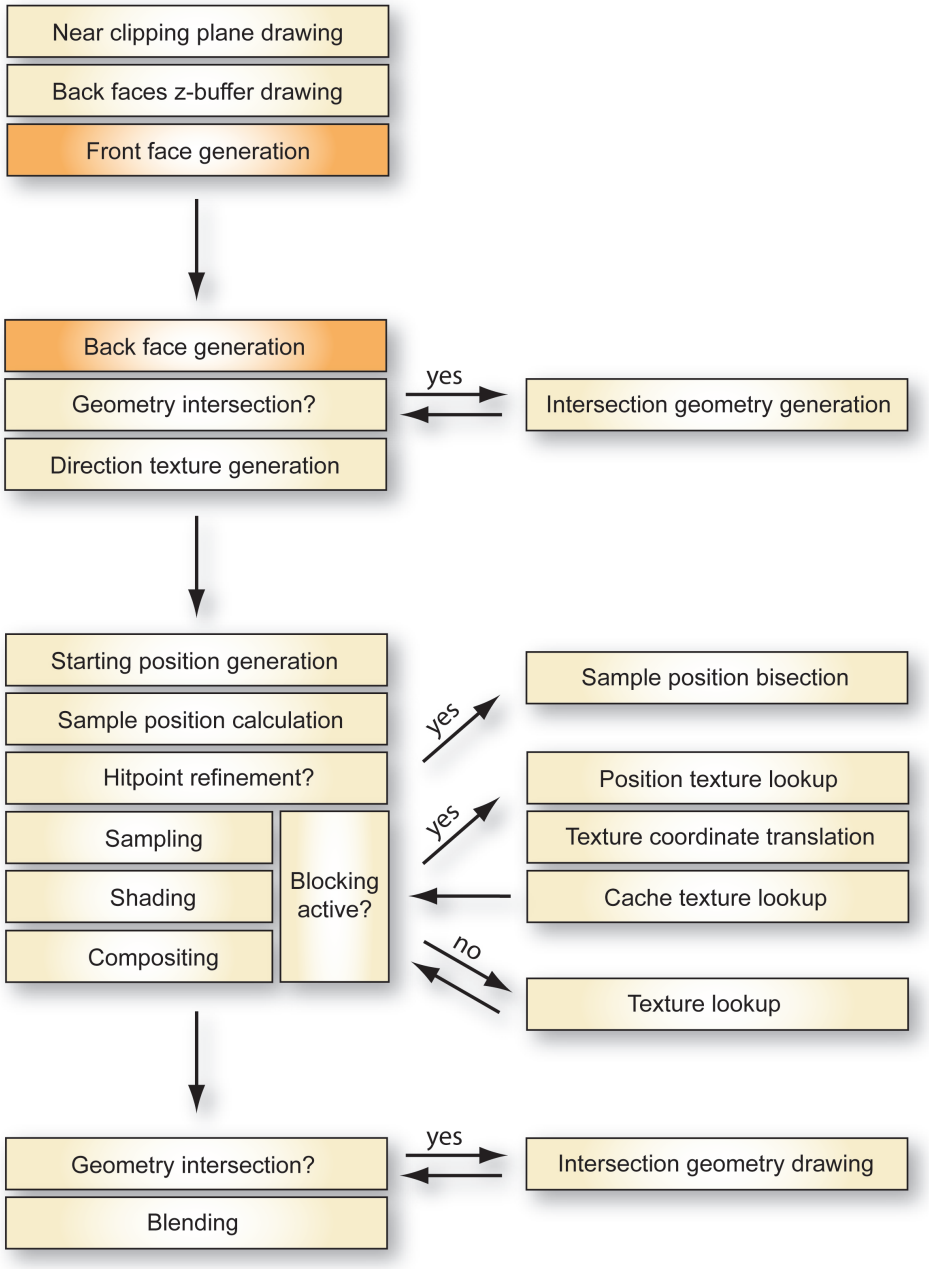

Figure 4.4: For incorporation of the complex bounding geometry, front and back face generation have to be modified.

has already been incorporated in the basic raycaster by constantly comparing the current alpha value to a threshold and exiting the raycasting pass once this threshold is reached. Empty Space Skipping is another suitable technique that can be implemented very efficiently on the GPU. Effectively, in a hardware based raycaster empty space skipping can be split up into two categories: Skipping outer empty space - which is what this section is all about -, and skipping empty blocks *between* two active blocks. The latter has to be performed in the raycasting program itself, incurring a bit of a performance hit because the fragment program gets more complicated and caching will not work as efficiently as before. Skipping of the empty space around the object, however, which is normally about 90-100% of the empty space present in the dataset, can be solved very efficiently on the GPU by modifying the basic raycasting approach a little bit and extending the idea of the colored bounding box. The necessary modifications to the pipeline are shown in Figure 4.4.

4.1.1 Considerations

In this section a blocked scheme for bounding geometry creation is introduced which significantly speeds up rendering time of the standard raycaster presented in chapter 3.

With the simple raycasting algorithm presented back then, only the pixel processing pipeline of the GPU is used, while the vertex pipeline is mostly lying idle. So whether the bounding box consists of twelve (as in our basic algorithm) or 100,000

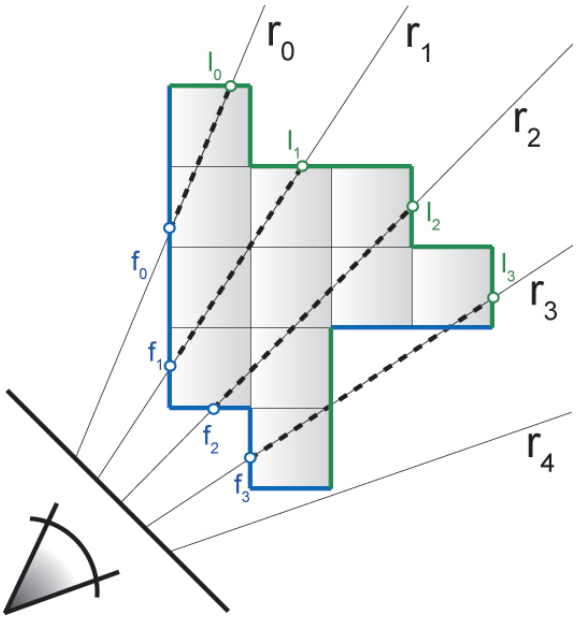

Figure 4.5: Front and back faces of our blocked bounding geometry, with grey boxes being active blocks, blue lines denoting front faces and green lines back faces. Note that the ray always starts at the first front face and ends at the last back face, even if there are inactive blocks inbetween as in the case of r_3.

triangles does not make much difference - even more since graphics cards nowadays are specially designed to handle the massive (and ever increasing) amount of geometry of current games without suffering huge performance hits. Having this in mind, increasing the complexity of the bounding box - making it a data dependent bounding geometry - seems like a good idea.

When modern graphics cards render a scene, they try to process vertices and pixels in parallel, in the best case leading to an equal distribution of the workload. Of course, if there's a small number of large triangles, the vertex processing engine will be partly idle, while only the pixel processing engine is busy - such a scene is said to be fill limited.

On the other hand, if there is a huge amount of visible triangles that consist of only one or two pixels, then the vertex engine will be the bottleneck, making the scene geometry limited. Both of this is of course undesirable, meaning that it is best to keep the average triangle size at a very constant rate that is - in the best case - close to the optimal size for the current generation of GPUs.

With todays graphics cards, this optimal pixel per triangle ratio is highly dependant on GPU brand and shader complexity, but usually somewhere between four and eight. Considering this, a blocking scheme with equally sized blocks (shown in Figure 4.5) seems to be a good idea, which decides for every block consisting of a number of voxels from the original dataset whether this block is of any interest or not. This decision is based on the transfer function or the current isovalue, depending on the rendermode. In the case of direct volume rendering, this means that if any voxel in this block is mapped to an opacity value greater than 0, the whole block has to be drawn.

4.1.2 Algorithm Overview

Culling against the iso-value or the current transfer function has to be performed whenever one of these parameters changes. To handle this efficiently, the minimum and maximum value for each block is stored and compared to the transfer function, meaning that if opacity is always zero between those two values, the block can be safely discarded.

It is important to note that one border voxel outside the current block has to be tested as well, because the filtering could cause an interpolated value inside the block to be greater than the threshold even if none of the voxel values inside the block is.

With this blocking scheme enabled, the border between empty and non-empty blocks defines a surface that can be rendered as normal OpenGL geometry with the position encoded in the color-channel, resulting in a bounding geometry of the volume. However, the separation into front and back faces is a little bit more complicated now, because the possibly non-convex bounding geometry does not guarantee for exactly *one* front and back face anymore. Thus, one needs to retrieve the *first* front face to start the rays and the *last* back face to stop them. This can be solved efficiently with a simple depth test of the OpenGL geometry.

As shown in Figure 4.5 in the case of r_3, this scheme does not necessarily skip all inactive blocks - another lookup in the fragment shader is necessary to determine whether there are empty blocks *between* the first front and last back face. However, this check is easy and does not slow down the fragment program noticably, and the implicit empty space skipping via the bounding geometry has no performance hit at all.

4.1.3 Implementation

As mentioned in the previous section, the border between empty and non-empty blocks defines the bounding geometry that is used multiple times in the process of raycasting. Considering this, it should be stored as efficiently as possible and in a way that it is always available for rendering. Using the Vertex Buffer Object Extension (VBO), the array of vertices and color values can be forced to be kept in the on-board video memory, avoiding unnecessary transfers of static geometry data.

The amount of memory available for these Vertex Buffer Objects is set in the BIOS under AGP Aperture Size, meaning the size within the main memory that is reserved for direct transfers to the graphics card. Unfortunately, only roughly half the memory specified there can actually be used (with the rest being reserved for caching), which means that an Aperture Size of 128MB is necessary for the algorithm to be able to handle very large data sets. Since there is no way to change the value in software later on, it has to be made sure that enough aperture memory is available on the system, forcing the user to reset the value manually before installing the application.

Coming to the actual algorithm, the first task is to render the front faces of the bounding geometry with depth test set to GL_LESS. This will make sure that in the end, the nearest front fact will be visible in the image, starting the rays from the nearest possible intersection with the bounding geometry.

Rendering the same scene with backfaces enabled and depth test set to GL_GREATER retrieves the farthest backfaces (i.e. the ending points for the viewing rays). Since the bounding geometry can be assumed to be closed, all screen pixels that have a valid front face color must also have a respective backface. This makes sure that the subtraction of color values always produces a valid viewing vector.

4.2 Rendering of Large Datasets

During the last couple of years, many existing algorithms have been reimplemented - and sometimes reinvented - on the GPU. With the GPU being much more than just a different processor, completely different ways of looking at the problem and thus new sets of algorithms to exploit its full potential are required.

Video memory has always been the weak side of GPU based algorithms, and this is mostly due to the fact that transferring data from the main memory more than offsets the performance gain that such a technique can possibly achieve. Thus, using the main memory even for parts of the storage required is not an option.

Figure 4.6: Rendering this 512x512x1112 dataset is possible even on a 256MB graphics card, as long as the active blocks fit into the memory.

Fortunately, there are two ongoing developments that put this disadvantage into perspective again:

1. Video memory has been doubled with almost every graphics card generation for quite a while now, and though current generation GPUs have already almost as much onboard memory available as main memory present in the system,

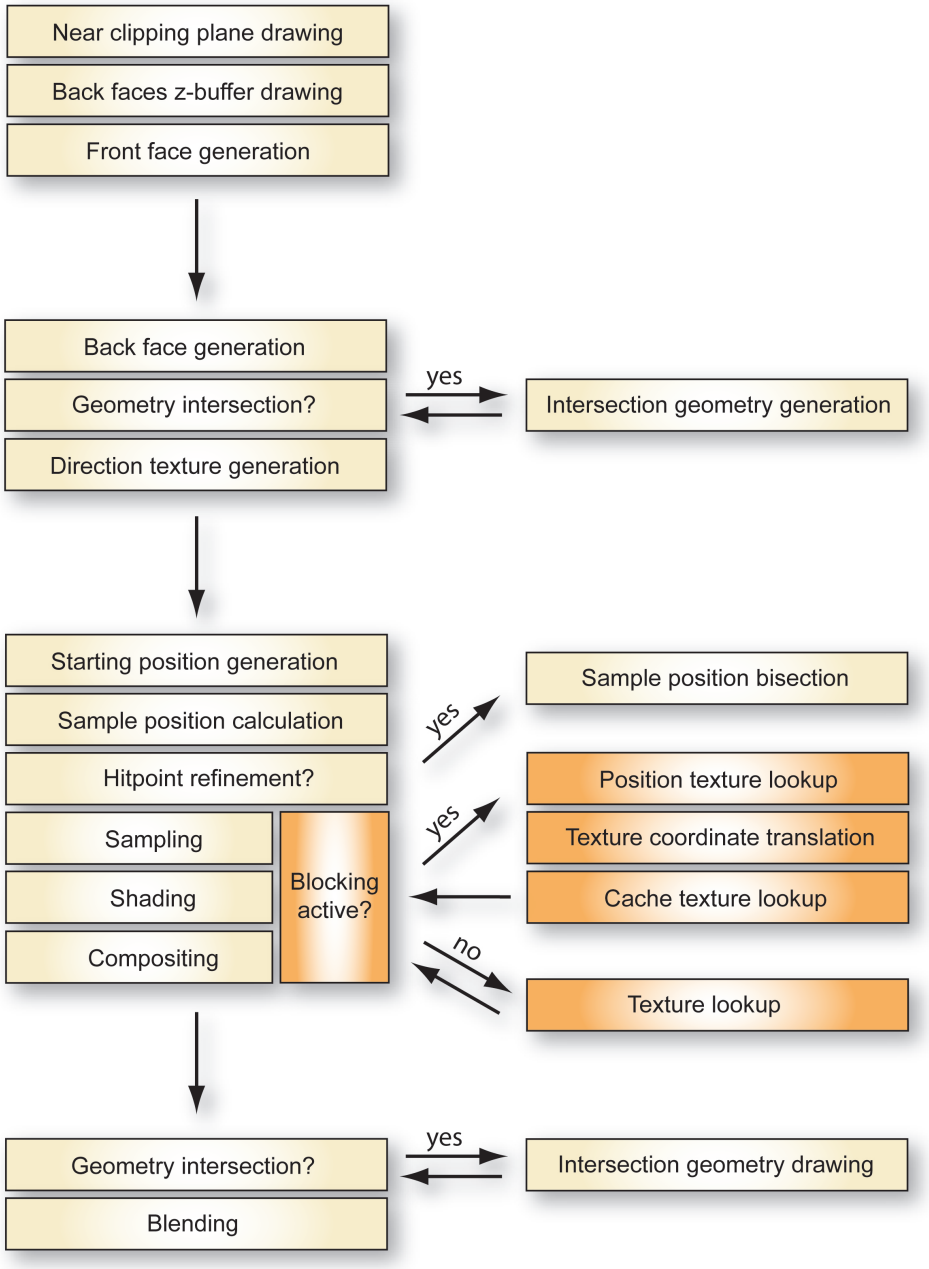

Figure 4.7: The blocking scheme requires a modification of the fragment program, where the simple texture lookup is replaced by a more complex series of position lookup, coordinate translation and cache texture lookup.

this trend does not seem to stop. Even more interesting, graphics cards have begun to boost the memory chip market with their ongoing demands for more memory on smaller chips at higher speeds. As of now, the memory technology built into high-end graphics cards is one or two generations ahead of 'normal' system memory specs.

2. With the introduction of PCI-Express, data transfer to (and especially from) the graphics card has been speeded up significantly. Unfortunately, we did not have the opportunity to test our algorithm on one of these systems, adapt it and experiment with the possible speed gains, but the first figures that were published throughout the web looked very promising. Furthermore, generation 2 PCI-Express is planned to be introduced next year, leading to another doubling of actual transfer rate.

Whether this technology makes outsourcing of additional data into main memory feasible remains to be seen - but with just roughly that kind of architecture we have today, there will always be a speed penalty incurred when fetching data from main memory.

That said, some kind of compression technique is needed that helps in storing the dataset more efficiently in the video memory while not slowing down data access too much. One approach to this is presented in the next section, which uses the blocking approach already introduced in the previous chapter to achieve smarter data storage. The altered pipeline can be seen in Figure 4.7.

4.2.1 Considerations

As mentioned in the introduction, video memory size keeps increasing with almost every generation of graphics card. Right now, 512MB cards are starting to hit the market, making it theoretically possible to store a 1024x512x512 dataset in 16-bit. The question whether 16-bit is necessary or not to achieve sufficient quality is unfortunately answered by a restriction of the texturing abilities of current GPUs. While 8 bit would still be a rather acurate representation of the original 12-bit-data in a CT-dataset, interpolation precision also falls back to integer when storing 8-bit-textures. This leaves hardly any other choice than storing the data in 16-bit-textures, leaving most of the available precision unused for now.

While it was not mentioned as one of the prime advantages of hardware-based approaches yet, the ability to composite at full floating point precision without slowing down calculation considerably is definitely one of the most useful features of this technique, which comes from the simple fact that GPUs are tailored towards floating-point computations much more than CPUs. Because of this, most software approaches have to use integer compositing to avoid the speed penalty involved with float compositing.

Figure 4.8 shows the difference between integer and floating point composition, and it can be clearly seen that only floating point compositing offers enough precision

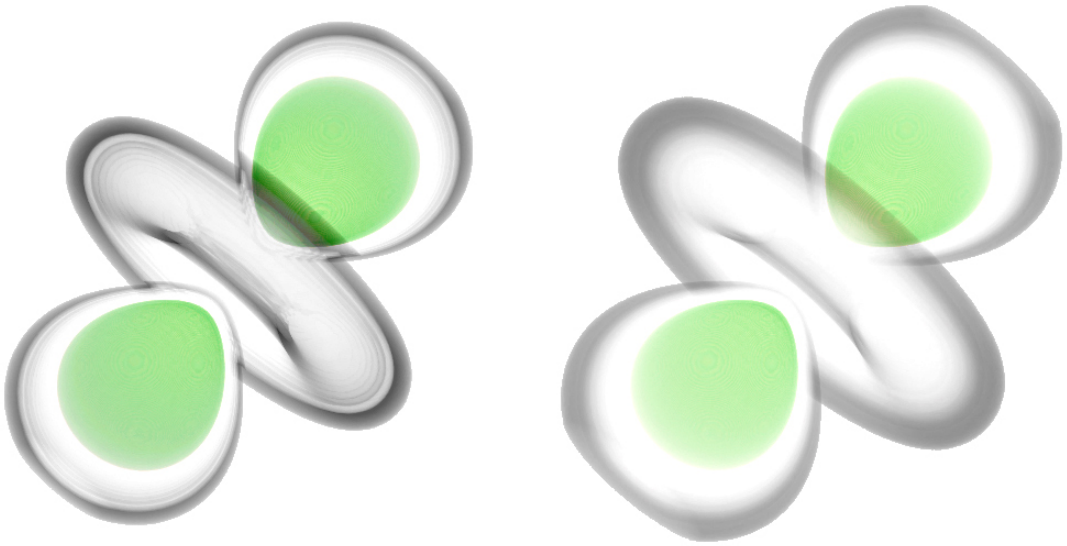

Figure 4.8: This comparison shows the difference between integer (left) and float compositing (right).

to render certain images at adaequate detail and without visible precision artifacts. When trying to visualize thin transparent structures (like the outer parts of this hydrogen), using integer compositing not only leads to precision artifacts noticable as visible banding in the left picture, but also to the loss of information where the precision is insufficient to store the low alpha values. Note that in the left picture, outer parts of the hydrogen seem to be missing and suddenly reappear when other objects are visible behing the current one, leading to an inconsistent representation of the volume.

Since applicability to all kinds of datasets and transfer functions is a requirement for a fully versatile raycaster, no compromise should be made regarding the precision of the stored data.

Having this in mind, there is still only 512MB of video memory available, and unfortunately not even all of this video memory can be reserved for the volume data - the geometry information and the textures needed in the process of rendering take up some space as well. In reality, only roughly 800x512x512 voxels can be stored on the graphics card, which is not sufficient for all kinds of applications.

4.2.2 Algorithm Overview

Considering the way the bounding geometry was created in chapter 3, this concept can be extended to data storage as well: Inactive blocks do not contain important information, and thus besides not being rendered they should not be kept in memory in the first place. The complete algorithm for caching and coordinate translation is described in [Hadwiger et al., 2005].

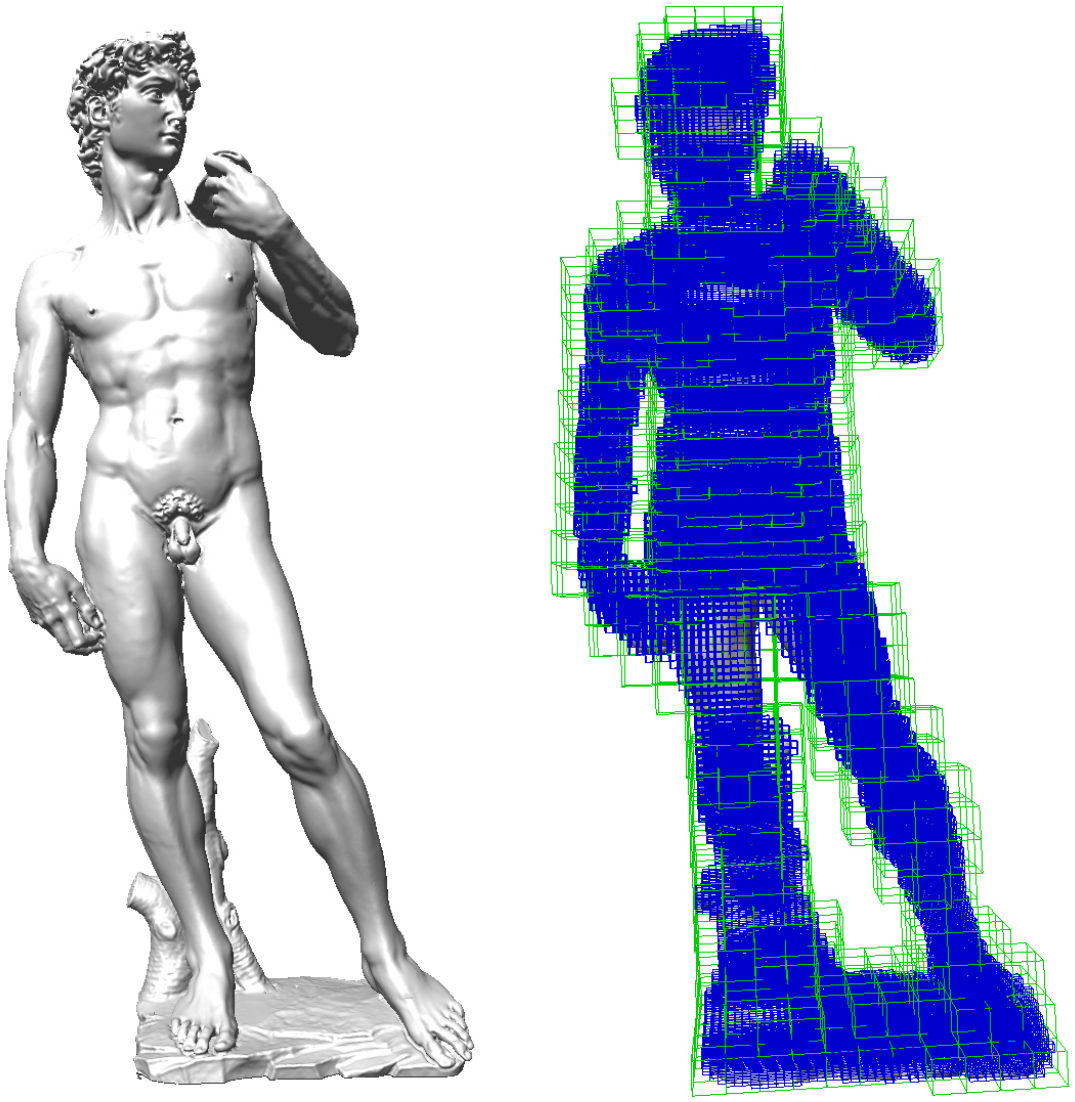

Figure 4.9: Rendering of Michelangelo's David (1536x576x352) with our 2-level blocking scheme - cache blocks are marked green and bounding blocks blue.

Storing the dataset in a 3D-texture is convenient for our algorithm, so the 3D-texture should be preserved, but the blocks in this texture do not have to have the same order as the original dataset, nor does the texture have to have the same size. Storing only the active blocks in a new 3D cache texture would be one possible solution - unfortunately, in order to preserve correct trilinear filtering, blocks have to be stored with 1 extra border voxel. Otherwise a sample could be interpolated between voxels from different blocks, which of course would result in a wrong value.

The key to this cache texture is a separate position lookup texture, that has exactly one entry for every block of the original dataset. This entry stores the position the block has in the cache texture.

Figure 4.10 illustrates what this two-way blocking scheme looks like on a regular dataset - in this particular example, only 30.5% of the cache blocks are active, leading to a memory consumption of only 36.6% of the original volume size. Even better, only 7.2% of the bounding blocks are active, meaning that 92.8% of the dataset can be implicitly skipped via the bounding geometry.

4.2.3 Implementation

As stated before, blocks in the cache texture have to be stored with one additional border voxel. This means that for every 4^3 block of data from the original dataset, a 6^3 block has to be reserved in the cache texture. By adding an offset of 0.5 to the coordinate texture, only one additional voxel has to be stored in each direction to ensure correct interpolation, thus resulting in a 5^3 block. Still, a 5^3 block needs roughly double the space of a 4^3 block - obviously, with a block size of four more space would be lost than saved.

This suggests that the optimum for the bounding geometry (which is usually around 4^3) and the one for the cache texture are of a different magnitude. That said, a two-way-blocking scheme looks like the best idea, with larger blocks (e.g. 32^3) for caching and smaller structures (4^3) for the bounding geometry. For the sake of efficiency (and simplicity), the small block size should be a factor of the large block size.

To accommodate for the different texture setup, the fragment program has to be slightly modified: Every lookup into the density texture has to be replaced with an access to the position lookup texture, a recalculation of the texture coordinates and finally one access to the cache texture with the modified coordinates. Original volume and lookup texture are both accessed with texture coordinates between zero and one, so the access to the lookup texture can be performed with the absolute

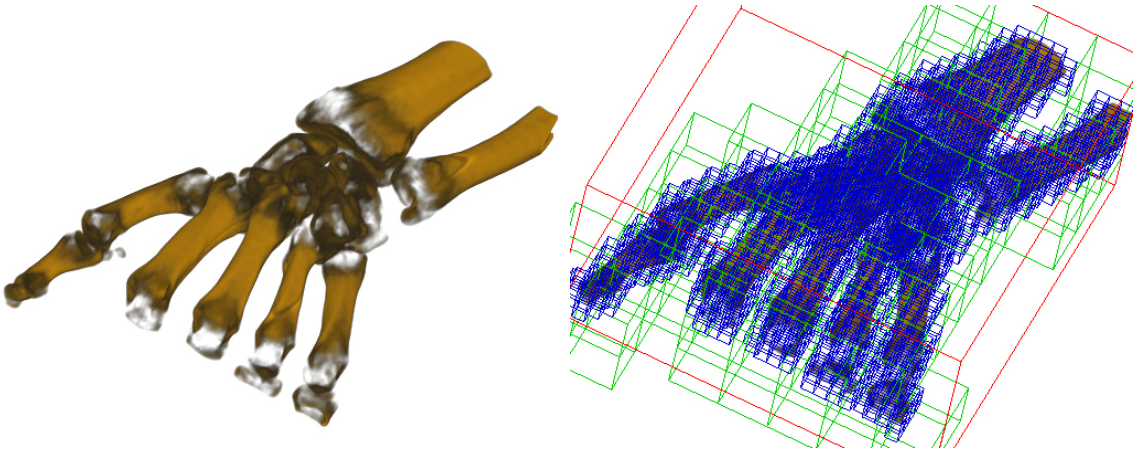

Figure 4.10: Rendering of a human hand with the caching scheme enabled. In this example, only 30.5% of the cache blocks are active and thus stored, leading to a memory consumption of only 36.6% of the original volume size.

position in the volume, just as the normal density lookup. Since the entries in the lookup texture point to specific coordinates, filtering has to be set to nearest neighbour, ensuring that for all voxels of the current block the same pointer to the cache texture block is returned.

With the block coordinates retrieved from the lookup texture, the second lookup in the cache texture is performed. This requires coordinate translation from lookup into cache texture, which basically means converting the offset inside the block into the cache coordinate system, always considering the absolute dimensions of the cache texture. This translation can be done with a couple of simple instructions in the fragment program, incurring only a minimal performance hit for these instructions and the additional texture lookup.

Again, culling against the iso-value or the current transfer function has to be performed whenever one of these parameters changes, so both layers should be recalculated at the same time. The most obvious approach is to first test each block of the bounding geometry, and later on mark all caching blocks as active that contain at least one active bounding geometry block.

4.3 Geometry Intersection

Being able to intersect the rendered volume with normal OpenGL geometry allows for a number of interesting applications, like 3D-Pointers that correctly blend into the scene, a 3D-grid that provides additional information about the position in the dataset or arbitrary meshes that could cut away part of the dataset for easier navigation.

In the case of virtual endoscopy applications this is an even more important feature, because of the need to visualize the endoscope and the tools attached to it. Obviously, a voxelized representation would be inappropriate for a number of reasons: First, the extreme perspective distortion of the endoscopy lens makes single voxels appear huge on screen when moving the viewpoint through narrow structures or tiny vessels, and this resolution would clearly be insufficient for medical tools. Second, changing position and especially orientation of the tools - something that happens probably every frame - would require some computational effort, since the voxel representation would always need to be re-rasterized. And third, the question of how to render the tools in different modes - like DVR and iso-surface rendering - and where they fit in in terms of density and transfer function range would still need to be answered. The final decisive argument is that common representations of these tools are almost always in the form of triangle meshes.

4.3.1 Considerations

Considering the complexity of raycasting, unnecessary parts of the final image should not be rendered at all, which is the way that most optimization strategies work. Intersecting the volume with geometry will introduce another level of occlusion, and it should again be taken care that parts of the volume that will not be visible because of this geometry will not be rendered at all.

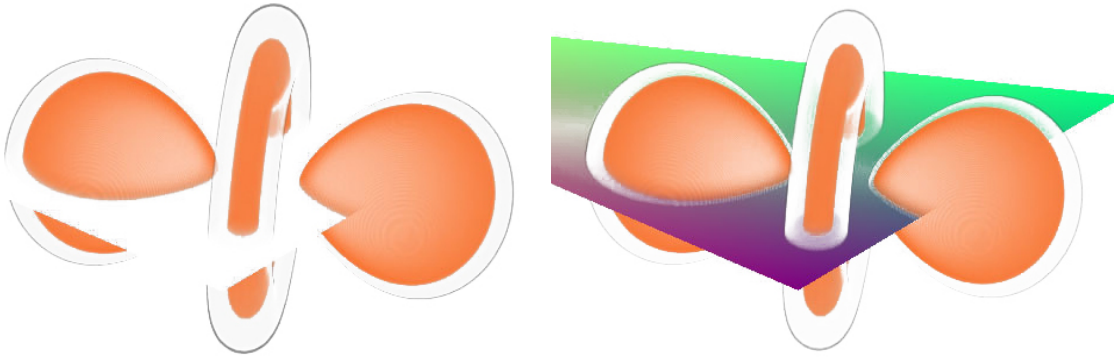

Figure 4.11: Modifying the ending geometry avoids rendering unnecessary parts of the volume (left picture). Blending the clipped volume with the geometry then gives the correct result.

So before thinking about adding geometry, there should be a way to arbitrarily clip away the parts that are cut off by the geometry and thus will not be visible. Fortunately, the bounding geometry introduced in the first section of this chapter offers a convenient way to do so. The differentiation between front and back faces (i.e. starting end ending points of the rays) becomes crucial again now, so it is important to know *which* of these has to be modified.

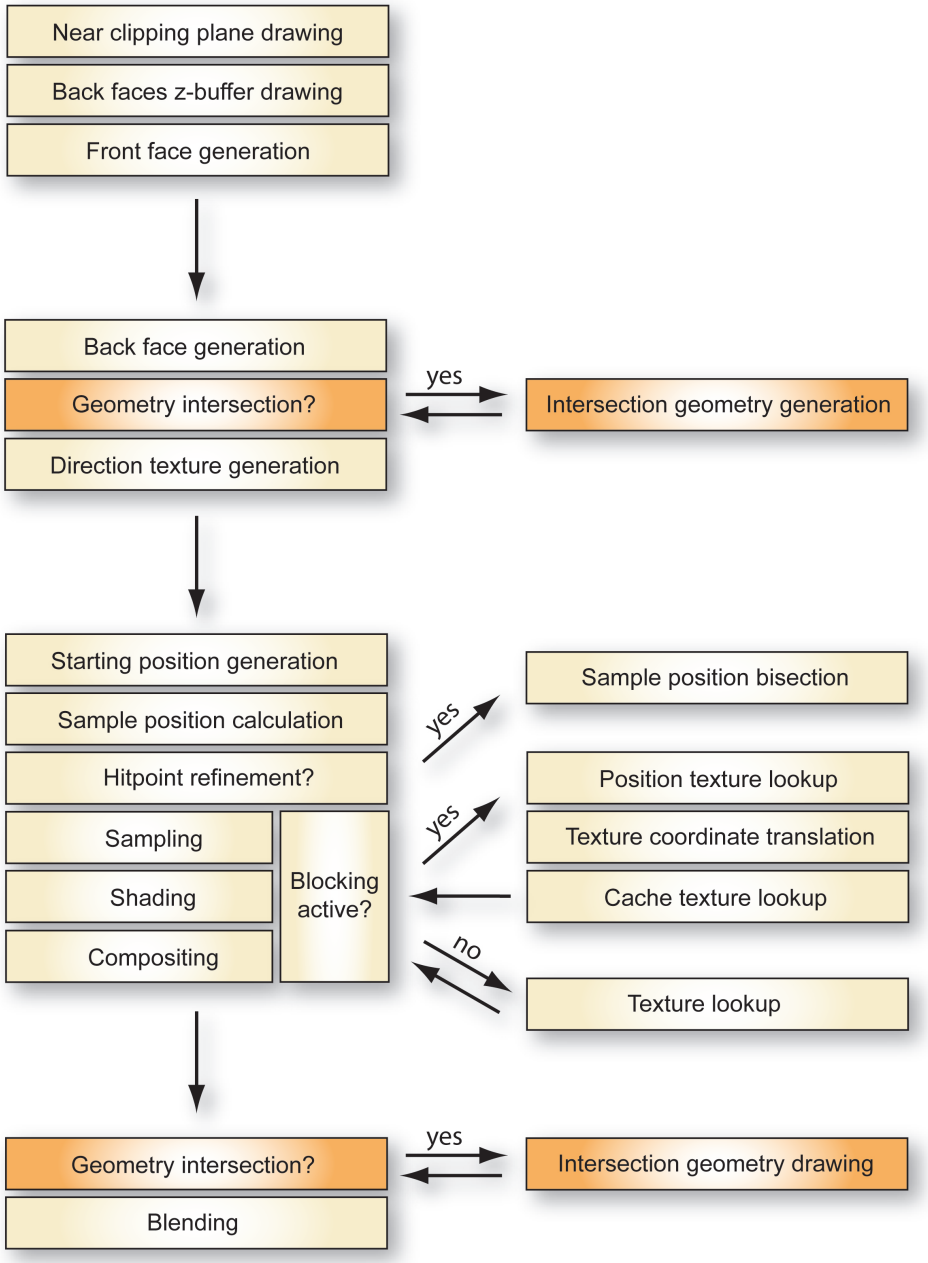

Figure 4.12: Geometry intersection changes the pipeline at two different locations: The backface generation has to be altered and the geometry has to be drawn to the back buffer before blending.

Modifying the starting points would result in clipping parts of the dataset from the viewer's side, which could be useful for 'opening up' the dataset if one wants to look at the inside without modifying the transfer function accordingly. Changing the ending points would result in clipping parts on the back side, which would be the same as putting a completely opaque object - like OpenGL geometry - there.

4.3.2 Algorithm Overview

Clipping the volume from either side requires modification of the starting or ending position of the rays. In the case of hitting an object, the ending geometry has to be modified, thus terminating the ray once it intersects this object. This behaviour can be achieved by changing the calculation of the bounding geometry (see Figure 4.12) - this way, the raycasting pass does not need to be altered. All that has to be done is that whenever the intersection geometry is nearer to the viewpoint than the backfaces of the bounding geometry (i.e. the ending positions of the rays), the ending position has to be replaced with the position of the intersection (see Figure 4.13).

After that, the viewing vectors are calculated just like before, yielding a correct direction and length until the intersection or bounding geometry is hit. The result of this will be an image where the volume looks like it has been clipped against an invisible wall (see Figure 4.11). All that is left to do now is to draw the intersection geometry into the screen buffer *before* the volume is drawn and then blend the clipped volume onto the screen accordingly, resulting in the correct image.

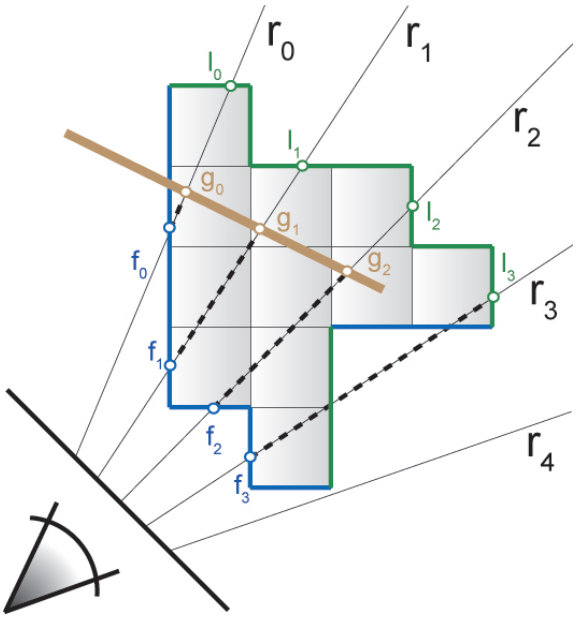

Figure 4.13: Adding intersection geometry to the scene modifies the ending position of the rays. Whenever the intersection with the geometry (g_0-g_2) is nearer to the viewpoint than the intersection with the back faces of the original bounding geometry (l_0-l_3), the ray has to be terminated at the position of the intersection geometry.

4.3.3 Implementation

Implementing this is straightforward: After rendering the front or back faces only the direction of the depth test needs to be changed and the clipping geometry must be rendered with the position inside the dataset encoded in the color channel (like the bounding geometry). In the case of the back faces, this means that our initial algorithm retrieved the *last* backface, so the depth test was set to GL_GREATER, which ensures that only pixels with a z-value greater than the current value are drawn. Reversing this test to GL_LESS now makes sure that the bounding geometry is modified only where the clipping geometry is nearer to the viewpoint - all other parts will be discarded. Color-coding the clipping geometry with its position in the dataset ensures that whenever a value is modified, the correct ending positions for the ray will be written into the texture.

It is important to note that with this extension to the initial algorithm, the ending geometry could easily be *in front of* the starting geometry (because the clipping geometry could even be in front of the front faces), yielding a negative direction vector. This makes it neccessary to check the direction vector before casting.

To achieve this, the current view vector in volume coordinates needs to be calculated, which can be done by transforming the initial view vector (0,0,-1) with the inverse transpose of the modelview matrix. This view vector now has to be compared to the direction vector, to check whether they point in the same direction or not. Taking the dot product of the two vectors will do just this - if the result is negative, the direction vector points away from the view vector and the ray should be terminated immediately.

4.4 Fly-Through Applications

So far, perspective projection is implemented as an implicit feature of the basic algorithm. Though this is not always immediately apparent on a rendered picture, the perspective greatly enhances the appearance of the image and makes interaction with the objects more lifelike, supporting the end user in estimating correct positions and spacial differences.

However, for some applications it might be interesting to move the viewpoint into the volume and explore the dataset in a fly-through mode like in Figure 4.14, espe-

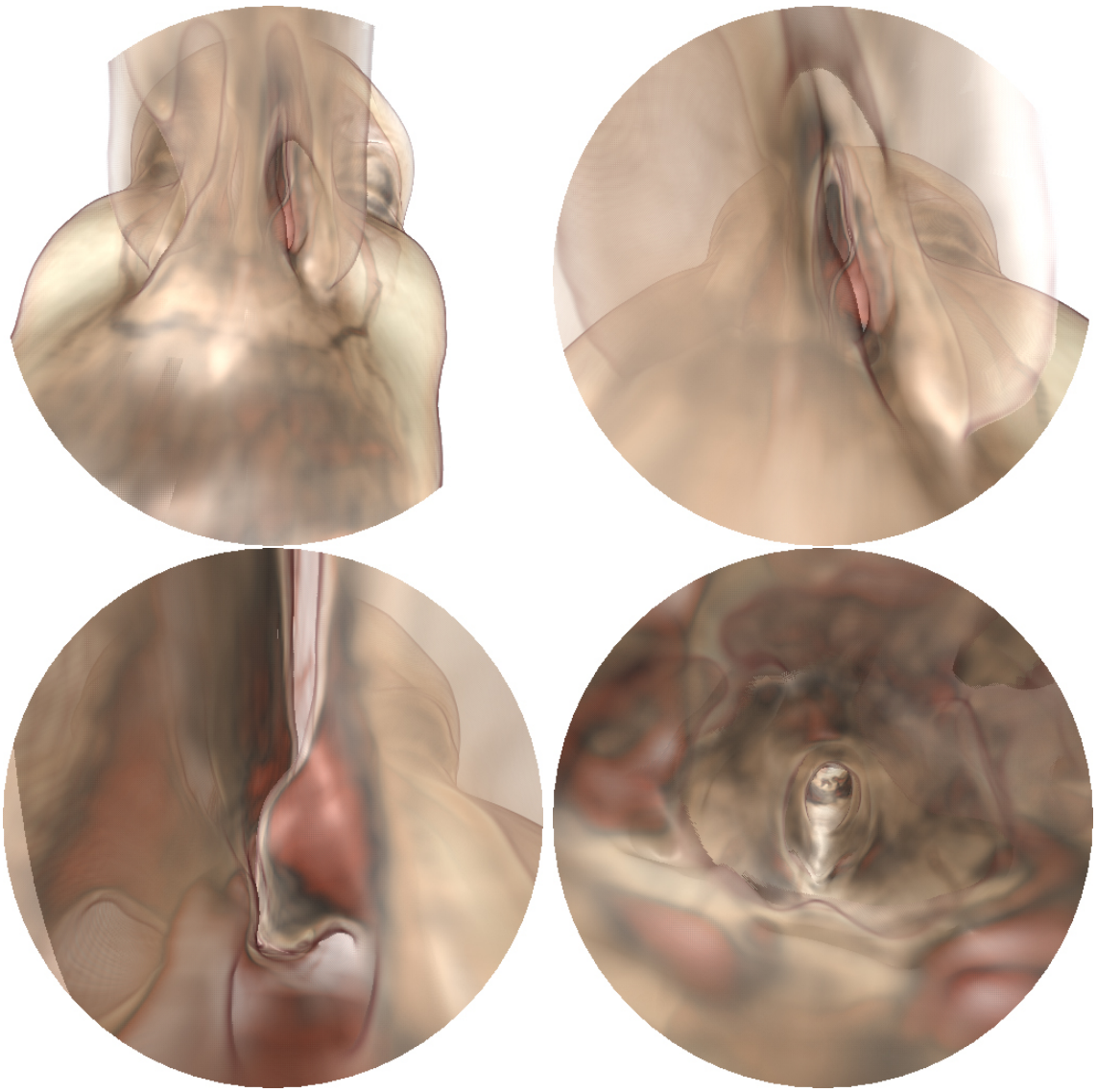

Figure 4.14: Moving the viewpoint inside the dataset is especially important for Virtual Endoscopy Applications. This sequence shows the process of flying through the CT scan of a human head, entering at the nose and moving further into the throat.

cially for virtual endoscopy applications that were a specific aim of this raycasting framework. This can be achieved with a modification of the front face generation pass, which is outlined in Figure 4.15.

4.4.1 Considerations

Considering the way the bounding geometry algorithm worked (see chapter 4.1), there is no problem moving the viewpoint as long as the camera does not touch the

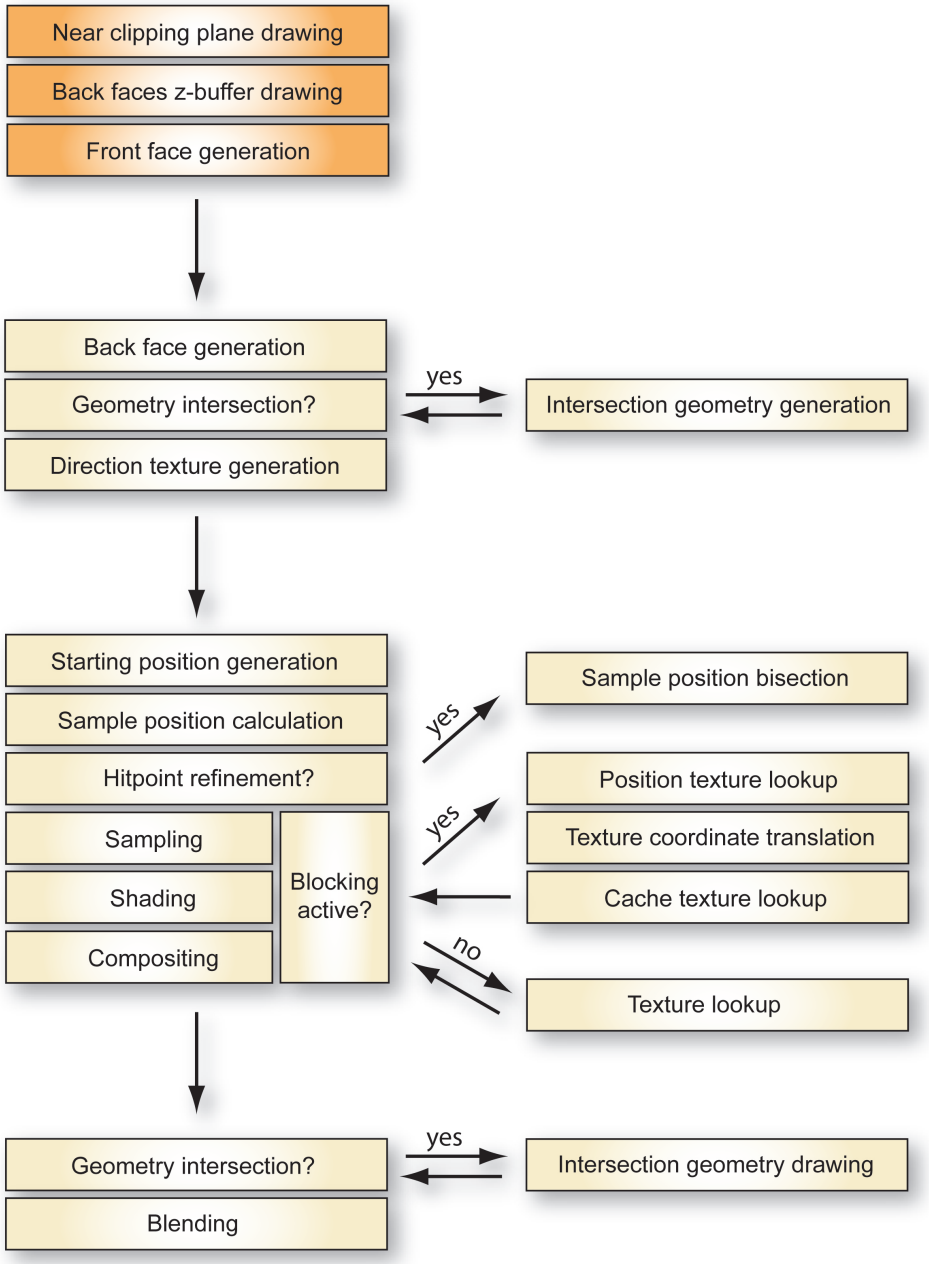

Figure 4.15: Moving the viewpoint into the dataset can be achieved by a modification of the front face generation, splitting this pass into three distinct steps.

geometry. But as soon as the near clipping plane intersects the bounding geometry, this intersection will generate holes in the geometry, resulting in rays not being started where they should. Figure 4.16 illustrates this behaviour.

What needs to be done is that whenever such an intersecion happens, the ray must be started on the intersection point with the near clipping plane. In practice this means that all holes have to be filled with color values of the position of the near clipping plane at this pixel.

4.4.2 Algorithm Overview

A simple way to do this is to draw the near clipping plane first (again with the colors encoding the absolute position in the dataset) and the front faces afterwards, ensuring that whenever there are no front faces to start from, the position of the near clipping plane will be taken. Unfortunately, this approach can only detect holes where no front faces are drawn at all (i.e. where the background color would shine through). Since the non-convex bounding geometry leads to the possible case of multiple front faces, a hole in the first front face would result in the next front face in the line of sight to shine through. Just drawing the front faces after the near clipping plane would then result in rays simply being started from the next front face, skipping the current active block.

This behaviour comes from the fact that so far the depth buffer did all the work and automatically retrieved the first front face. Now it has to be made sure that only

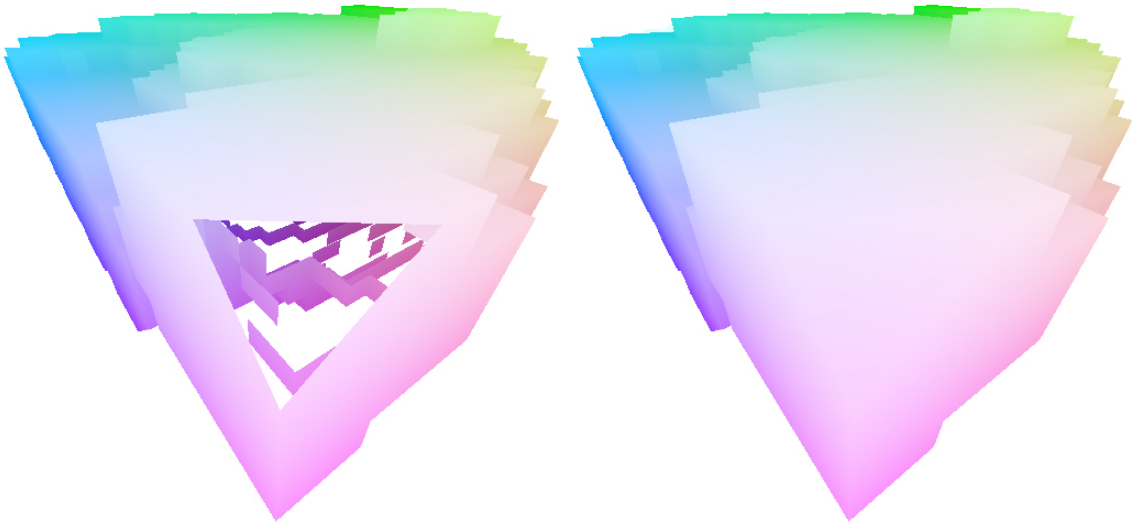

Figure 4.16: Moving the viewpoint into the dataset causes problems as soon as the near clipping plane intersects the bounding geometry. As can be seen in the left picture, the intersection causes visible holes where no rays are started, resulting in the same regions of the final image to be empty. The holes have to be filled with the respective position of the near clipping plane (right picture), resulting in a correct starting image.

the first front faces are drawn, even if the drawing fails because the geometry lies behind the near clipping plane.

Figure 4.17 illustrates the three considerations that need to be taken into account when determining the correct starting position:

1. Set starting position to the first front face in the line of sight
2. Set starting position to the near clipping plane if no front face exists
3. Set starting position to the near clipping plane anyway, if the current block is active

The third point is the crucial one here: When the starting position on the near clipping plane is *inside* an active block, then the ray has to be started at the near clipping plane anyway, even if there is another front face in the line of sight. But checking for each ray whether the starting position is inside an active block would be a tiring task that would slow down rendering considerably, so there needs to be a better criterion.

This criterion can be found with the following observation: A ray starts exactly then inside an active block, if it hits the first back face before the first front face, because since the ray is inside the geometry, it first has to leave it to be able to reenter. This test can be performed much more efficiently, and leads to the following test for determining of the correct starting position:

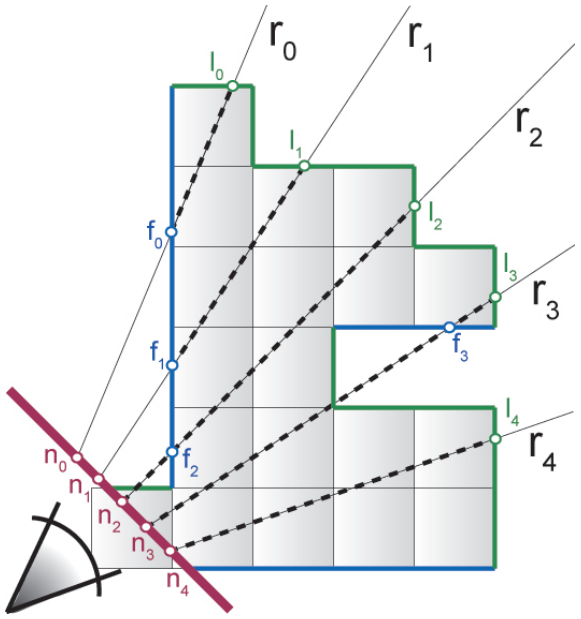

Figure 4.17: When moving the viewpoint into the dataset, the near clipping plane (red) can easily intersect the bounding geometry, leading to rays not being started. In this case, the ray must be started on the respective position of the near clipping plane (n_0-n_4) instead from the bounding geometry.

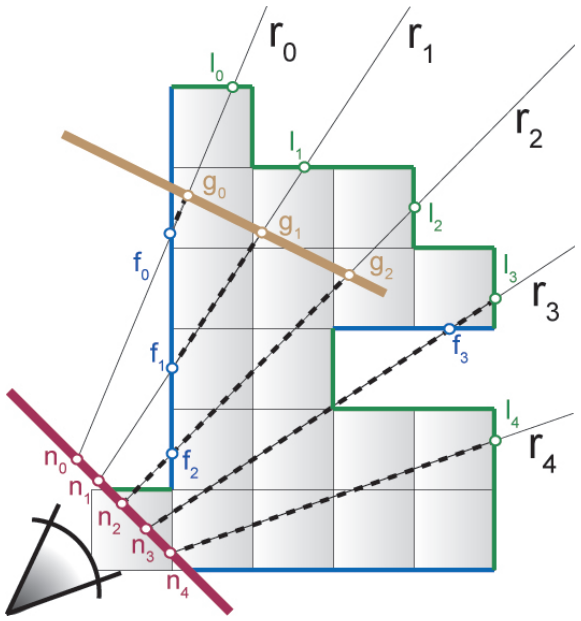

Figure 4.18: Adding intersection geometry to the scene works exactly the way it did before. Whenever the intersection with this geometry is closer to the viewpoint than the last back face of the bounding geometry, the position of the intersection geometry will be written to the buffer thus terminating the ray.

1. Set starting position to the near clipping plane
2. Calculate position of first backface in line of sight
3. Calculate position of first frontface in line of sight
4. If the first frontface is nearer than the first backface, start the ray there

Testing for the ending geometry and possible intersection geometry is performed like before, leading to the overall scheme for determining the viewing vector that is outlined in Figure 4.18.

4.4.3 Implementation

As mentioned above, the first task is to calculate the starting positions at the near clipping plane. To do this, all four bounding vectors of the current viewport have to be transformed to the volume coordinate system and assigned their respective colors. Then a simple quad is drawn with these four vertices, and since OpenGL interpolates the color values, a starting position for each pixel is automatically generated. The left picture in Figure 4.19 shows what such an image typically looks like, which is a simple color gradient that almost looks like a constant color due to the small difference in volume coordinates of the near clipping plane.

Rendering only the front faces of the bounding geometry with depth test set to GL_LESS will find the *first* front face in the line of sight whereas faces behind the

current viewpoint will not be rendered and thus automatically discarded. Still, as already observed, the rays should only be started at the next front face if there is no back face of the bounding geometry in front of it.

This behaviour can be achieved by simply rendering the backfaces of the bounding geometry to the depth buffer first with depth test set to GL_LESS. Since the position values of the backfaces are not needed here, color buffer writes are disabled. Rendering the front faces afterwards with GL_LESS still enabled makes sure that only those front faces are rendered that lie before the first backface, and in all other cases the color value of the near clipping plane (which was rendered in the first pass) is kept. Summarizing this, the generation of the starting positions is split into three passes:

1. Enable color buffer, disable depth buffer, draw color-coded near-clipping plane. This will start all rays at the near clipping plane where no other starting point can be determined.

2. Disable color buffer, enable depth buffer. Draw backfaces of bounding geometry with depth-test set to GL_LESS. This will set the depth buffer to the first backface, making sure that nothing behind that will be rendered anymore.

3. Enable color buffer, enable depth buffer. Draw frontfaces of bounding geometry with depth-test set to GL_LESS. This will draw all front-faces between the near

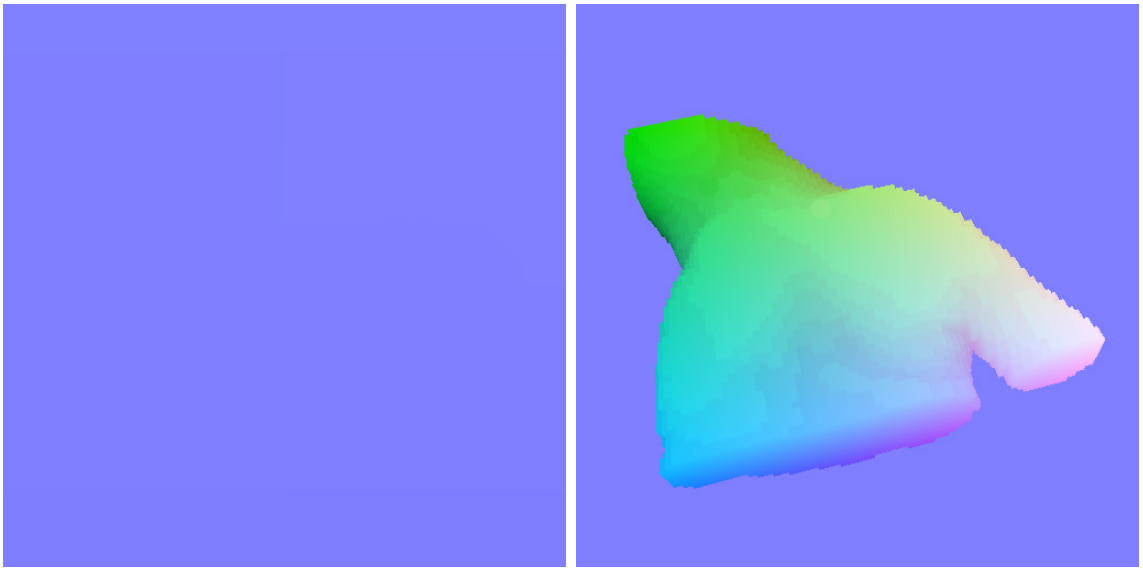

Figure 4.19: Drawing the near clipping plane with volume coordinates encoded in colors results in a simple color gradient, barely visible in the left picture because of the small difference of coordinates within the volume. The right picture shows the starting geometry rendered *after* the near clipping plane, resulting in all rays being started on either of the two surfaces.

clipping plane and the first backface. If there is no such frontface, the color of the near clipping plane will be taken as starting position.

All this affects, of course, just the front face generation pass, which is split into three separate passes itself now. Backface generation and raycasting are not affected by this change. Even better, the starting geometry can *always* be generated this way, whether the viewpoint is inside the volume or not, because drawing of the near clipping plane introduces only minimal overhead. If the viewpoint is outside the dataset, every valid starting position is overwritten with a front face anyway, always starting the rays at the bounding geometry like they should.

5. Quality Improvements

The biggest problem that any raycasting algorithm faces is the appearence of sampling artifacts due to undersampling. Since the process of raycasting is only an approximation of the true integral along the viewing ray, there is almost always a difference between the ideal integrated color value and the composited color consisting of a number of discrete samples. These differences diminish once the sampling distance is set to a small enough value, but usually this will result in a huge impact on performance.

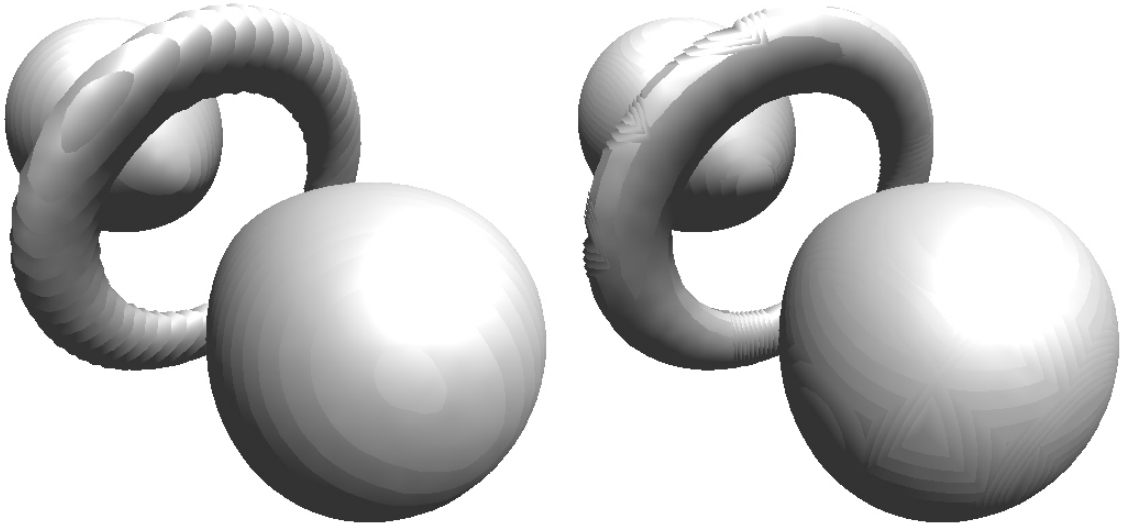

Figure 5.1: Rendering iso-surfaces at low sampling rates produces sampling artifacts that resemble the outer shape of the starting geometry. While the simple geometry (left) produces artifacts at regular intervals, the situation gets drastically worse with the complex bounding geometry (right), where the angle of the bounding geometry may lead to artifacts that completely destroy the three dimensional appearance.

Unfortunately, the situation is even a bit worse with the algorithm presented in this thesis: Normal texture based or software raycasting approaches may introduce regular sampling artifacts that may give the image a 'sliced' look. But since the rays are started exactly at the bounding geometry in our approach, sampling artifacts will also be aligned with the geometry and all the breaks in the geometry will make the artifacts even more visible.

Figure 5.1 illustrates this issue, where the left image is rendered using the bounding box of the volume as starting point for the rays and the right image uses a complex bounding geometry. When moving the viewpoint into the dataset for virtual endoscopy, things get even worse: One voxel may easily take up a huge part of the screen now, making the artifacts even more annoying because they can completely ruin the three dimensional effect the shading is trying to achieve.

Sampling everything with a higher sampling rate would be a possible solution, but that only changes the result in areas where features were partly or fully missed before, probably leaving huge parts of the image unchanged. Thus, increasing the sampling rate for all parts of the image does not solve the whole problem.

An obvious approach would be to introduce an adaptive sampling rate dependent on the gradient or the distance from the viewer. Unfortunately, this is an approach that might be a good idea in software, but the parallel architecture of GPUs is heavily dependent on efficient caching and highly parallelizable tasks. Having different sampling rates for different rays would definitely prevent both strategies from working efficiently, thus destroying the main performance advantage of the GPU.

So different techniques have to be developed that again take advantage of the nature of the algorithm and do not add too many conditional statements to our fragment program. Two techniques that solve these problems are presented in this chapter, one is specially designed for iso-surface extraction and the other one is particularly useful for direct volume rendering.

5.1 Hitpoint Refinement

Hitpoint refinement tries to improve the image quality in iso-surface renderings. The idea is based on the assumption that the current sampling rate is sufficient for detecting all features in the dataset and that the ray will not miss an object that should be rendered.

The reason why sampling artifacts look particularly bad in iso-surface renderings is that this rendering mode calculates the whole shading for the pixel depending on just one sample and thus one gradient. To define a smooth iso-surface, this gradient should be calculated always as close to the surface as possible. Unfortunately, with a sampling distance of 0.5, the ray might hit the object 0.01 units after the intersection point, but also perhaps even 0.49 units. This results in completely different gradients, which is why these huge 'steps' are visible in undersampled renderings.

The idea is now to cast the ray with the normal sampling distance, and once the iso-surface is hit to use this result as a first estimation of the real hitpoint and refine it succesively. To do this, bisection steps back and forth are performed until a sufficient precision is reached. This step has to be inserted right after the calculation of the sample positions, as shown in Figure 5.3.

5.1.1 Considerations

As mentioned in the introduction, iterative changes to the sampling rate should be avoided as well as implementations relying on too many conditional statements. This

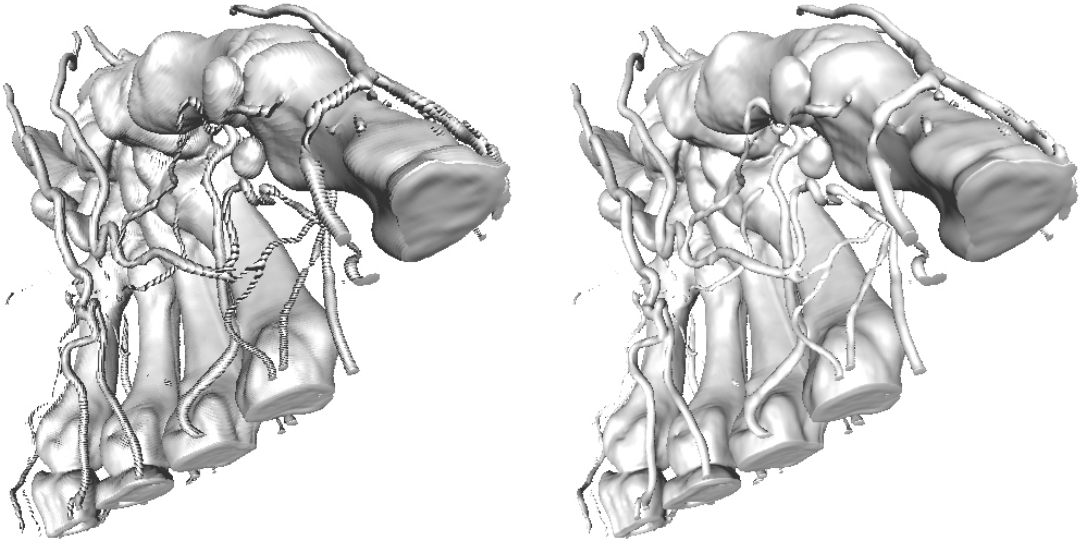

Figure 5.2: Iso-surface extraction, especially on datasets with small features compared to the sampling distance, is prone to producing sampling artifacts that destroy the three dimensional appearance of the image especially in high frequency areas, as shown in the left picture. Turning hitpoint refinement on eliminates those sampling artifacts without any impact on performance (right picture).

is especially important in the case of iso-surface raycasting, where framerates are usually very high and a tiny change in the algorithm might very well cut performance in half if the graphics pipeline is used inefficiently.

For this reason, a simple bisection scheme seems perfectly suitable. First, it can be done in a few additional steps after the initial surface detection. Second, the sampling rate is constantly changed but since it is always cut in half with every further bisection step, there is again no out-of-order change in the iteration variable.

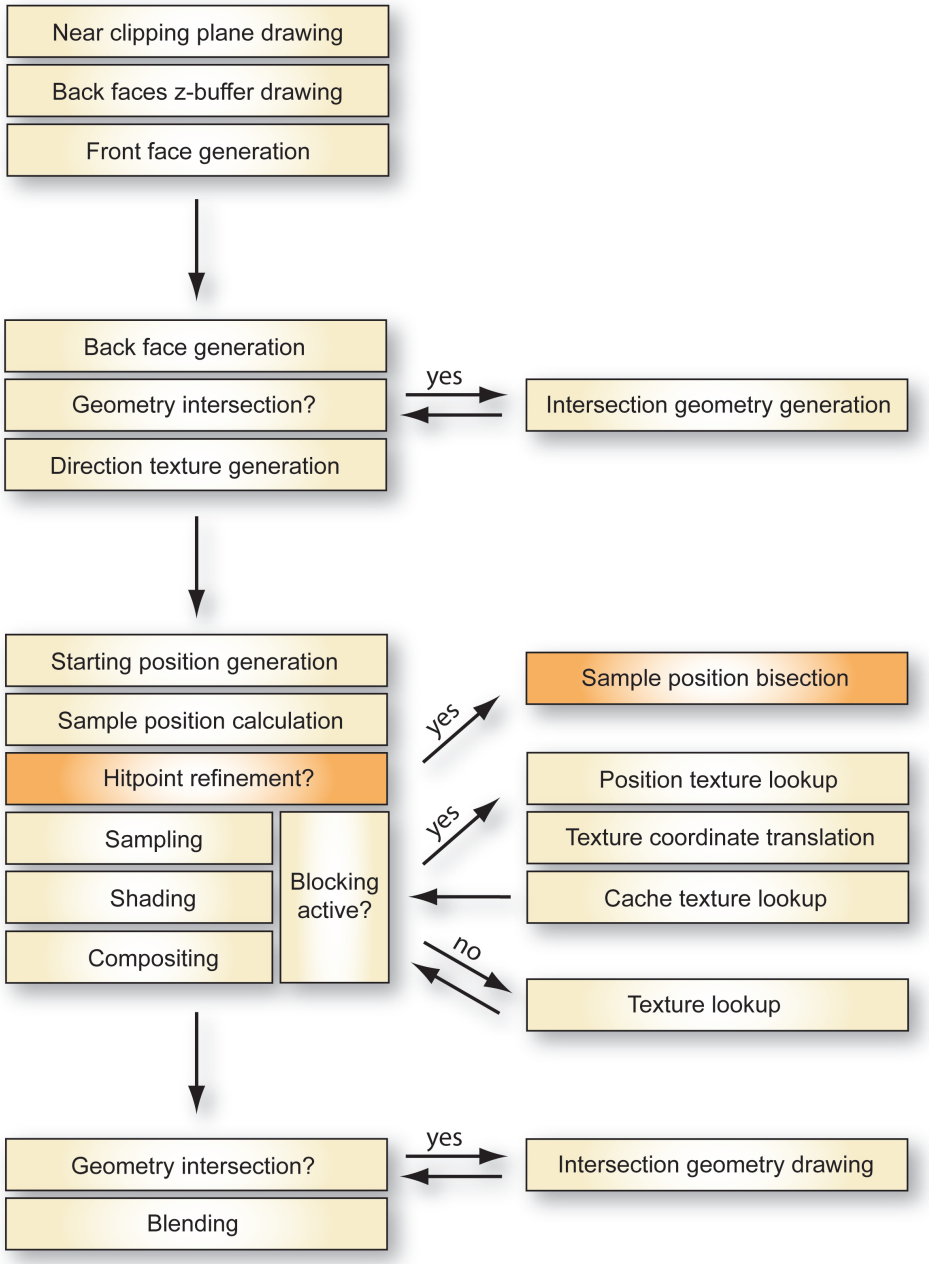

Figure 5.3: Hitpoint refinement is carried out in the fragment shader, where an additional step is added after the sample position calculation.

Third, since every bisection step effectively doubles the precision of the final hitpoint, performing only five or six steps should be more than enough. If, for example, the basic raycasting is performed with a sampling distance of 0.5, the possible error after six bisection steps would be less than 0.01, which should be sufficient for our purpose.

5.1.2 Algorithm Overview

Starting from the first estimation of the intersection (i.e. the first sample point along the ray where the density is greater than the iso threshold), the algorithm goes one halfstep back (half the previous sampling distance) and checks the density value at the new position. The next step will again be half the previous stepsize (making this one fourth of the original sampling distance), and will depend on the density on this new position: If it is still above the threshold, the next halfstep will also be taken backwards, otherwise forwards. If this bisection is repeated six times, we have an intersection that is 64 times more precise than our original estimation, which should be sufficient for most applications, no matter how low the original sampling distance is.

As shown in Figure 5.4, hitpoint refinement dramatically increases the image quality of iso-surface renderings - even more when the sampling distance is very low. Because of this, it is easily possible to increase the sampling distance to 400 or 500%, as long as no important features are completely missed by the ray. As can be seen in Figure 5.5, the sampling distance can even be five times the voxel distance for

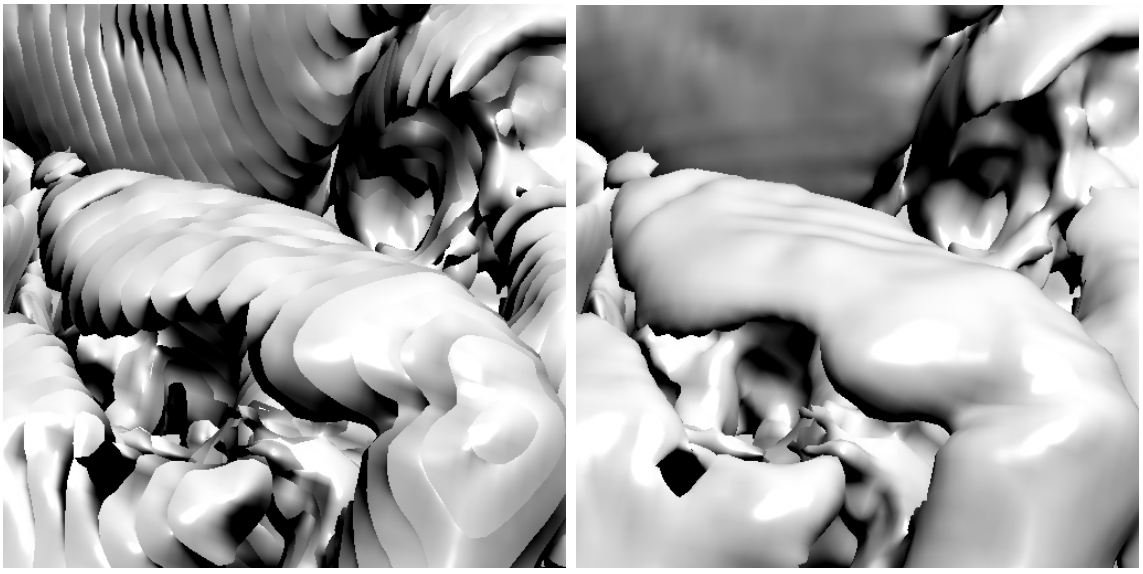

Figure 5.4: Moving the viewpoint within the dataset, sampling artifacts become even more apparent as the size of the voxel footprint onscreen increases. This behaviour is even worse when moving around, since the artifacts start to move with respect to the ray starting positions, resulting in constant flickering. Enabling hitpoint refinement results in a huge improvement in image quality without any impact on performance.

certain datasets without any visible difference. This makes it extremely useful for interaction renderings while the user is moving or rotating the dataset, because even *if* tiny details were missing because they are skipped by the ray, the user would hardly notice while moving around. As soon as the mouse is released, the sampling distance should be reduced again to ensure that all the details are rendered correctly.

5.1.3 Implementation

Considering the above algorithm, there is obviously one condition that has to be checked in each bisection step: If the new sample is still higher than the isovalue, the next step is still performed backwards, otherwise the next step will go forwards. At the same time the sampling distance must be cut in half, which means that overall the current sampling distance has to be multiplied with either 0.5 or -0.5.

Fortunately, there is one limited 'conditional' statement available in the fragment program which does not slow down execution because no branching is performed: Depending on whether a decision variable is below zero or not, one of two values can be assigned to a variable. By subtracting the isovalue from the result of the volume

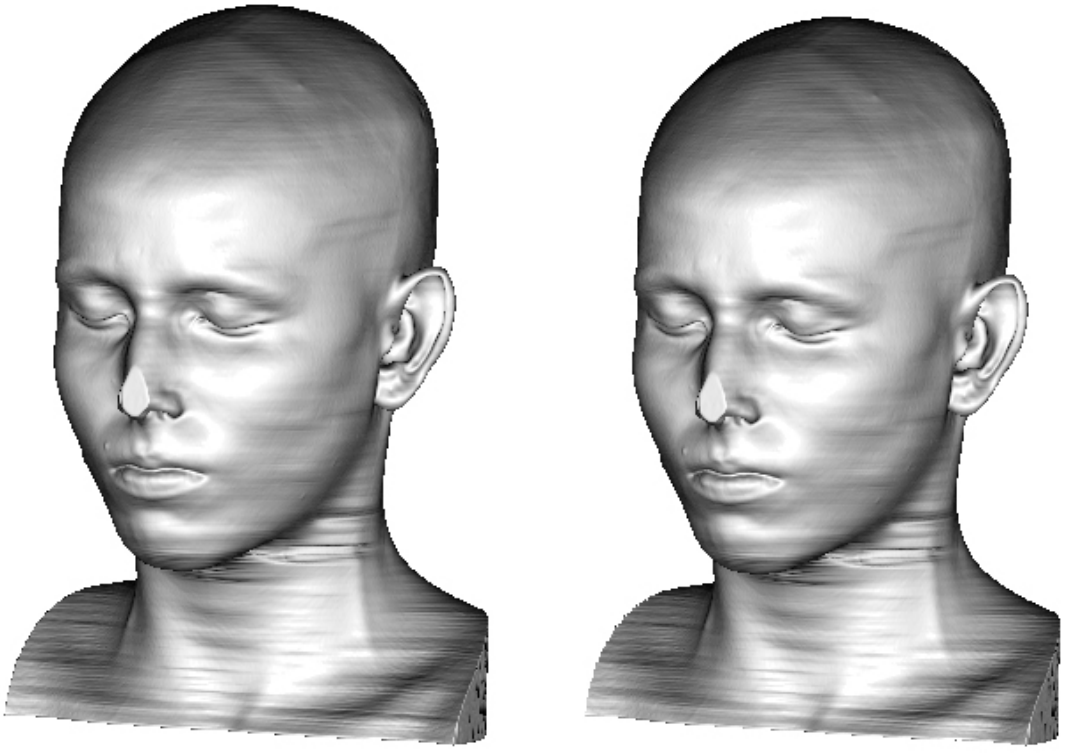

Figure 5.5: This comparison of two rendered images of a 512x512x333 dataset shows the benefits hitpoint refinement can have on intermediate renderings. The results for sampling distance 1.0 (left, about 24fps) and sampling distance 5.0 (right, about 66fps) are almost identical when hitpoint refinement is turned on.

texture lookup (i.e. the current density), this decision variable can easily be created and either -0.5 or 0.5 can be assigned to a multiplier of the sampling distance.

In our implementation, we're always performing six bisection steps, which was enough even for visualizing tiny structures in virtual endoscopy mode. Interestingly enough, performance slightly increases on a current generation GeForece 6 when performing these six steps after the the first hitpoint determination. This was surprising and there is still no completely sound explanation for that - apart from the assumption that it is probably due to some pipelining issue - but this has been reproduced in numerous tests on various systems. However, even with slightly different implementations there should not be a noticable performance loss if the six steps are just unrolled at the end of the fragment program.

5.2 Interleaved Sampling

Especially when moving the viewpoint into the dataset for exploration of small structures like blood vessels, we regularly face the problem of visible sampling artifacts. One possible solution to this would be to increase the sampling rate, but this would slow down overall performance unneccessarily, because the higher sampling rate is not necessary in regions that are farther away from the viewer. Adapting the sampling rate according to the distance from the viewpoint might be a solution to this - unfortunately, the highly parallelized architecture of the GPU relies heavily on very similar tasks being executed in every single step, and constantly changing a lookup parameter in a loop trashes the caching scheme. Another alternative would be the use of pre-integration, but in contrast to the simplification to a linear approximation interleaved sampling delivers a result that is always true to the original data.

Interleaved sampling is a straightforward technique that significantly enhances the visual quality of the final image without a noticeable impact on performance. Keller and Heidrich proposed interleaved sampling as a solution to bridge the gap between regular and irregular sampling patterns [Keller and Heidrich, 2001].

5.2.1 Considerations

The idea is based on the observation that regular sampling patterns are easy and fast to compute, but prone to produce sampling artifacts, while irregular sampling patterns achieve much better results at the cost of higher computational demands. The solution is to use an irregular sampling pattern that covers *multiple pixels*, so that two adjacent pixels will never have the same pattern. Though this approach was basically meant to improve on the results of multisampling, the authors suggest using interleaved sampling for volume rendering as well.

Since supersampling is not an option in computationally expensive algorithms like raycasting, the sampling positions are only interleaved in z-direction (the view direction of the camera). This means choosing a small offset in z-direction that is different for adjacent pixels, but will repeat after a number of pixels. This can be achieved by rendering for example four smaller images with different offsets into a buffer and later combining them back to one large image. In our case, we modify the rendering pipeline by simpling adding a small offset in z-direction when calculating the sample position (see Figure 5.6).

5.2.2 Algorithm

The easiest way to do this would be to have some kind of modulo function of the screen coordinate. This modulo function should deliver a dithering pattern that does not degrade image quality too much but at the same time covers as much different sample positions between this sample and the next one as possible, thus minimizing the chance of missing a certain feature of an object.

However, choosing this pattern is not an easy task. Depending on the structure of the object and the transfer function involved, interleaved sampling may solve a

couple of problems and get rid of most of the sampling artifacts, but at the same time not always deliver the most visually appealing result.

Consider the case of two very thin structures that map to completely different colors in the transfer function and are roughly viewport-aligned. Without interleaved sampling, only one of these colors may be visible, because the highest intensities of the other color may be missed by the ray samples. Turning interleaved sampling on, a strong dithering pattern consisting of these two colors will be visible, depending

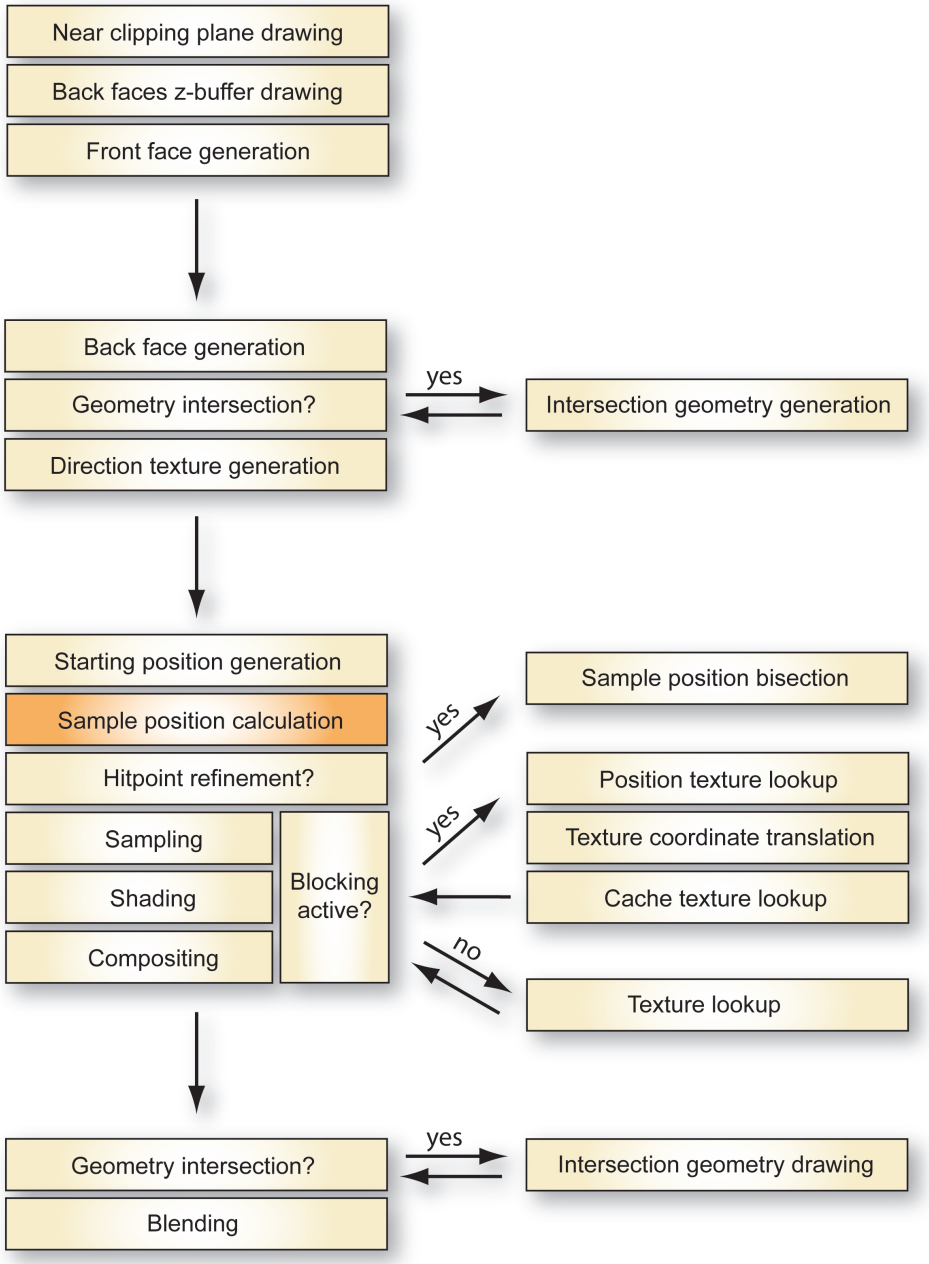

Figure 5.6: Interleaved sampling requires only a small modification of the sample position calculation, where a small offset in z-direction is added.

Figure 5.7: Interleaved sampling provides an easy way of getting rid of annoying artifacts, even when designing transfer functions with high peaks of semi-transparent surfaces which are usually prone to produce sampling artifacts.

on which of the two structures was hit by the ray with a particular offset, as shown in Figure 5.8.

Though the first result is definitely more appealing, the second picture is closer to the correct rendering (which would be achieved by choosing an indefinitely small sampling distance, or at least one that makes sure every tissue is adequately represented) because it shows both colors that are present at these sampling positions. Interleaved sampling can be taken then as an indication that given the current dataset and transfer function, the present sampling rate is not sufficient and important features might me missed.

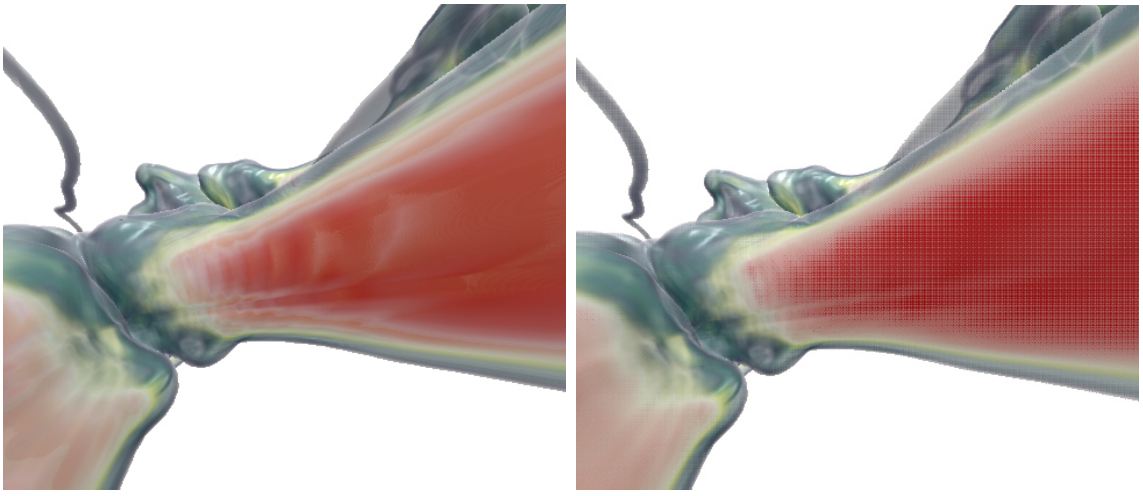

Figure 5.8: Turning interleaved sampling on does not always produce a more appealing result, but is a good indicator of undersampling in the other cases. In this example, the thin white tissue over the red bone is hardly visible in the left picture, except for a few sampling artifacts. Turning interleaved sampling on in the right picture suggests that the sampling rate should be increased.

On the other hand, introducing only a very simple pattern with an offset of 0.5 at every other pixel may not be the optimal solution for determining possibly missed objects, but even this simple approach is able to get rid of most of the sampling artifacts while introducing only minor dithering artifacts due to the simple pattern.

Whether interleaved sampling is suitable for iso-surface raycasting is probably a matter of taste: The dithering will definitely introduce some new information about the thickness of the tissue in areas where the offset will make the difference whether the ray hits an object or not. On the other hand, the difference between hitting this object and not hitting anything thus returning the background color is a significant one, resulting in a strong visible dithering pattern which looks a bit odd compared to the smooth shading that has been achieved with hitpoint refinement presented in the last section.

Introducing the simple pattern with an offset of 0.5 at every other pixel makes the image a bit more appealing, but at the same time introduces two visible layers at the boundaries of very thin tissue. Thus, the applicability of this technique for iso-surface extraction is highly dependent on the kind of application, the importance of image quality vs. expressiveness and whether the visualization of thin structures is necessary at all.

5.2.3 Implementation

As stated before, a good way of interleaving the ray offsets would be to calculate some kind of modulo function of the pixel position. Unfortunately, in the fragment shader this can only be done by using the FRC command, which returns the fraction of a float value.

Thus, calculating a repeating offset this way requires a simple function of the x and y coordinate, which retrieves a float value where the fraction can be taken as an offset. A couple of easy computations at the beginning of the fragment program can achieve this by adding the coordinates with different coefficient, so there's hardly any performance hit.

Of course, the offset has to be scaled to the current sampling distance, meaning that it would not make any sense if the offset would be farther away than one sample distance. This also means that increasing the sampling rate will result in lesser impact of the interleaved sampling, also leading to a reduction of the dithering artifacts.

This makes interleaved sampling especially useful for interaction renderings, where the dithering patterns are less visible because of the constant movement, and will disappear as soon as the mouse is released and the image is rendered with a higher sampling rate. This is even more important because the sampling artifacts are of course unwanted in still renderings, but impose an even bigger problem when moving around, because the constant flickering of sampling artifacts destroys the illusion of closed surfaces.

5.3 Iso-surface shaded DVR

Choosing the right rendermode for a particular application is crucial for being able to view the *important* details at sufficient framerates. This thesis already showed that using iso-surface rendering in order to achieve sufficient rendering speed is not necessary anymore. However, once iso-surface rendering drops out because of the limited expressiveness, there is still the major decision between shaded and unshaded DVR.

Both of these techniques require some background knowledge from the user as well as a little bit of experience in order to produce appealing images. Even worse, designing suitable transfer functions is a bit different in both cases, making it infeasible to 'test' both rendering modes before deciding which one is more meaningful.

Generally, shaded DVR offers more insight into the structure of a surface and delivers images that look more natural. It has been established as the de-facto standard in volume visualization, and every new algorithm is compared to previous ones by the speed of a shaded DVR rendering. It should be noted, however, that certainly a part of shaded DVRs popularity is due to the fact that the rendered images look much more impressive than unshaded DVR and are a big eye-catcher on every title page.

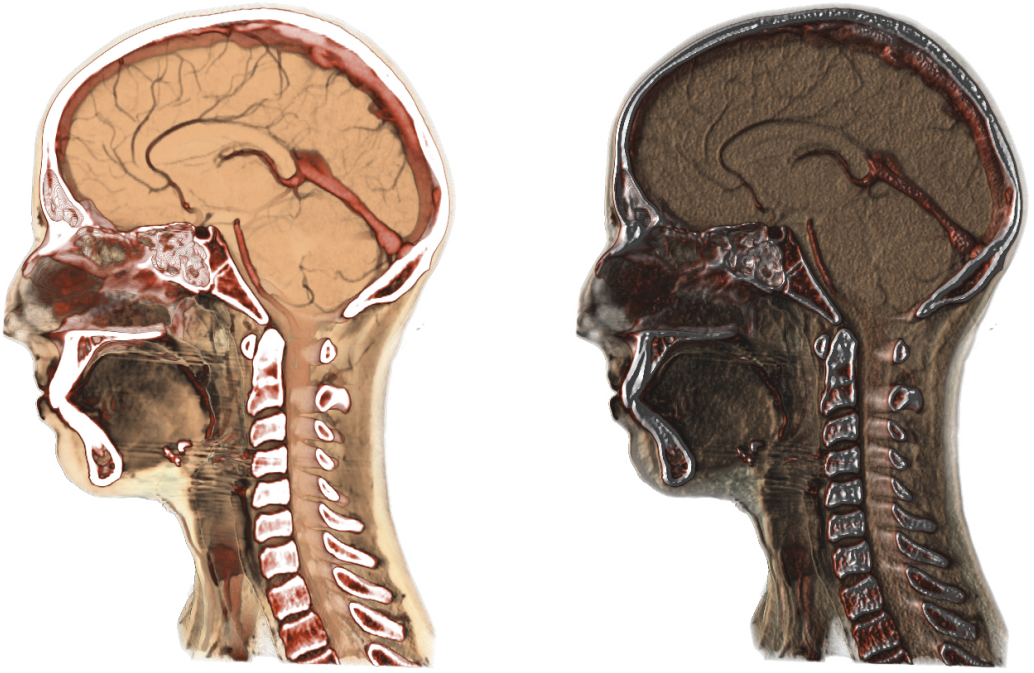

Figure 5.9: Comparison of unshaded (left) vs. shaded DVR (right) reveals the problems shaded DVR faces when visualizing semi-transparent tissue. Visibility of blood vessels in the brain significantly suffers from the noise introduced by the shading, making a segmentation via the transfer function extremely hard. Also, the unshaded DVR looks noticably sharper, making it easier to distinguish object boundaries.

This is also due to the fact that most of these images use very steep ramps in the transfer functions which leads to visualization of one (sometimes perhaps two) iso-surfaces in the final image. Since this is missing the important point of direct volume rendering - the visualization of different semi-transparent tissue - the question is still which rendering mode is best for typical DVR images with large semi-transparent regions.

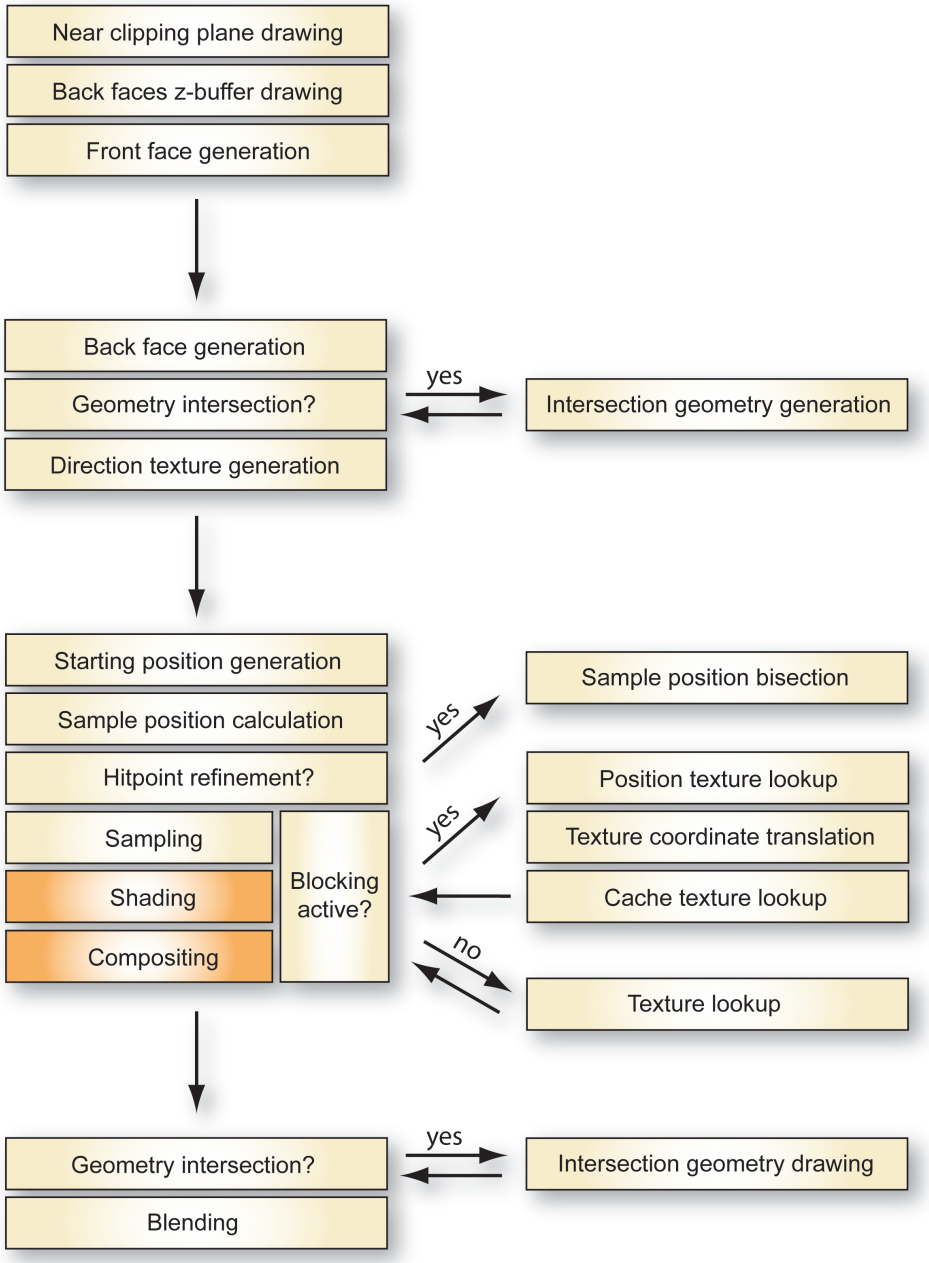

Figure 5.10: Iso-surface shaded DVR is carried out in the fragment program, modifying only shading and compositing of the sample values along the ray.

With an unshaded DVR, special surface properties are hard to see, but on the other hand the assigned color is not altered (and in some cases even covered) by light and shadow, so that the user can always properly estimate the boundaries of an object of interest. Furthermore, the plethora of lighting information in shaded DVRs leads to a noticable blurring and thus a significant degradation of overall image quality, which may not be welcome in medical applications where accuracy of the representation is one of the primary goals. Figure 5.9 compares shaded and unshaded DVR, where the defficiencies of shaded DVR especially in parts of the image with semi-transparent tissue become obvious. The instability of the gradient in homogeneous regions results in permanent changes of the lighting conditions, leading to introduction of further noise that covers almost everything else in these areas. Figure 5.11 is another example for this behaviour, where the masses of lighting information renders all other inner structures almost invisible.

This behaviour comes from the fact that per definition of a shaded DVR, all samples along the ray are shaded and only then composited to the final color. This means that the algorithm actually simulates hundreds of transparent slices one after the other, all correctly shaded and even highlighted. Of course this is far more lighting information than the human eye can perceive, just registering blurred highlights and shadows instead.

One reason for this is that this situation hardly ever occurs in nature, and we're more used to seeing transparent objects that have highlights only on the *surface*. And even if we were faced with transparent objects of diverse densities more often it would still be highly unlikely that the human eye could ever distinguish hundreds of layers of lighting information to form a model of separate surface properties.

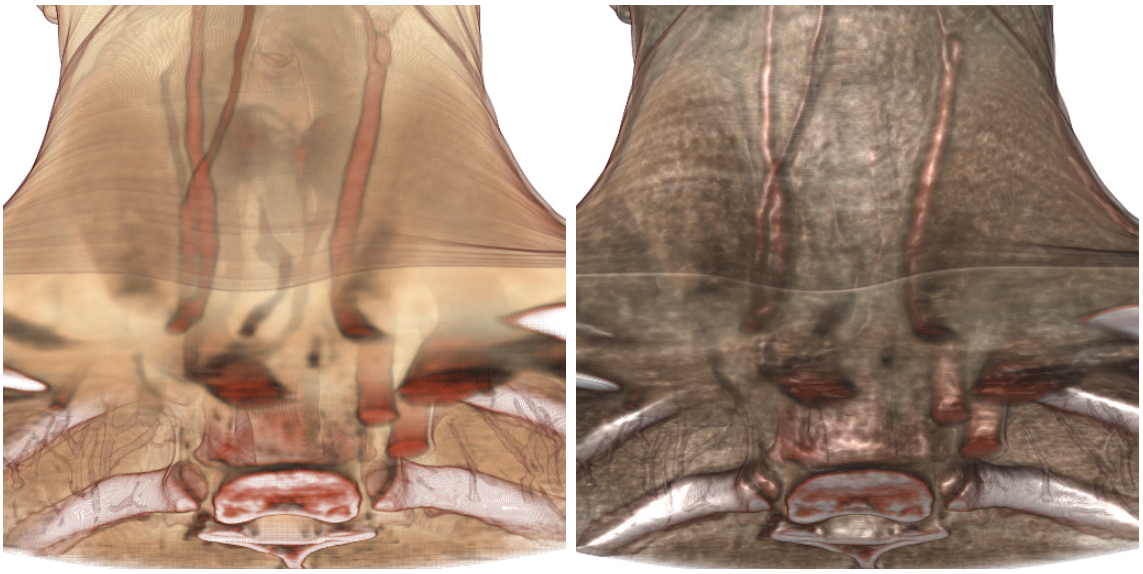

Figure 5.11: Rendering the human head from the outside again highlights the difficulties shaded DVR has with semi-transparent surfaces. Inner structures are almost completely lost due to the introduced noise.

With all the above said, it seems logical to introduce a rendermode that shades only a particular surface and uses unshaded DVR for the rest of the volume, leaving the eye only *one* layer of lighting information to decypher. Additionally, this saves a lot of computational power, since a lighting calculation is a costly process that involves six additional texture lookups to start with (if central difference is used). The rendermode introduced in this section will always presume that the shaded surface *surrounds* the tissue of interest, only allowing shading on the outside. The necessary modifications to the pipeline are shown in Figure 5.10. Extensions to this algorithm could very well allow more shaded surfaces inside the tissue, which could make sense as long as they can be clearly distinguished by a human observer.

Figure 5.12 highlights the differences between the three modes when rendering a CT scan of a human head with identical settings and transfer functions. The unshaded image lacks any information about the surface properties of the head thus not giving the impression of a three dimensional model. On the other hand, the shaded DVR generates a believable 3D-model with plenty of lighting and shading information, but hides a lot of useful information of the dataset by introducing significant noise. Iso-surface shaded DVR, however, combines the two advantages and produces a believable, three dimensional rendering of the human head with all the detail present in the unshaded representation.

5.3.1 Considerations

Knowing that a rendermode should be implemented that supports one shaded surface followed by an area of unshaded DVR, we can revert to two of the already implemented techniques that do just that, using all of the knowledge gained so far.

Figure 5.12: Comparing all three rendermodes shows the weaknesses of unshaded and shaded DVR compared to ISO-surface shaded DVR.

Through hitpoint refinement the estimation of the first iso-surface, and thus the shading, will be far more precise than a normal shaded DVR would be. Interleaved sampling will make sure that no feature is missed by simply 'stepping over it'.

By taking the refined position as starting position for the DVR raycaster, another advantage of this rendering mode becomes obvious: Since the starting positions of the DVR precisely resemble the shape of the object, and other density values inside the object often follow roughly the same shape, the amount of sampling artefaces is greatly reduced, as can be observed in Figure 5.13. Still, interleaved sampling can be turned on in the DVR pass as well to get rid of as many artifacts as possible.

5.3.2 Algorithm

The beginning of the algorithm is identical to the isosurface raycaster presented before. First, an offset for the interleaved sampling is calculated from the screen coordinates. This offset is taken as starting position from which samples are taken on regular intervals along the viewing ray until the density value exceeds a certain threshold (i.e. the iso-value). A sometimes annoying sideeffect of this technique is the fact that tissue intersecting with the near clipping plane will always result in strong dithering patterns visible at the very front, since it is very likely that the amount of tissue in front of the near clipping plane gets thin enough sooner or later to be completely skipped by some rays. This case is of course very frequent in virtual endoscopy applications, where the user is constantly moving through tissue thus intersecting the current isosurface. To avoid this behaviour, the first sample

Figure 5.13: Almost all of the sampling artifacts in the left picture (which was rendered without interleaved sampling for illustration purposes) have disappeared in the iso-surface shaded mode, which was rendered using the same sampling distance and resolution.

should be taken at position 0 anyway, and the offset for interleaved sampling should only be added if this first sample is still below the threshold.

The exact intersection point is calculated with hitpoint refinement as outlined in chapter 5.1. From this intersection point, the interleaved sampling offset is again added to the current position and given as starting position to the DVR raycaster.

The raycaster will finally continue until the ray has left the volume or another criterion for early ray termination has been met. Finally, the shaded iso-surface and the composited color from the DVR will be blended together depending on an adjustable parameter. This makes sure that the user can change the transparency of the isosurface to a level where it will not occlude important information and at the same time give a good impression of the structure of the outermost tissue.

5.3.3 Implementation

The implementation of this rendering mode does not diverge too much from the implementation of the two modes it incorporates. Whether these two modes are executed one after another or in a single fragment program should not make too much difference and is primarily a matter of taste. Having two separate passes makes the approach more flexible and allows for future additions to the algorithm, but of course has the overhead of a second rendering pass with complete geometry setup. Having only one fragment program, on the other hand, seems simpler in the additional setup but results in a far more complex fragment program that is prone to driver errors and cache-inefficiencies. Having experimented with the latter approach for some time, it exposed more errors in the graphics card drivers than any of the other algorithms presented in this thesis.

6. Results

This chapter will present an overview of the results we were able to achieve and can be taken as a quick reference to be able to compare our algorithm to similar approaches. All figures were taken on a Pentium 4 3.2 Ghz with 2048MB of system memory and a GeForce 6800 GT with 256MB of video RAM.

Figure 6.1: This iso-shaded DVR of a human hand is rendered at more than 50 frames per second.

6.1 Performance

Image resolution for all renderings was 512x512 and sampling distance was set to the voxel distance.

6.1.1 Bounding Geometry

This section compares rendering performance with and without the bounding geometry presented in chapter 4.1. Depending on how much outer empty space can be skipped, enabling the bounding geometry can achieve significant performance improvements.

Dataset	BG disabled [fps]	BG enabled [fps]
Hand (256x256x128) ISO	34.04	92.01
Hand (256x256x128) DVR	27.08	64.53
Head (512x512x333) ISO	4.98	11.18
Head (512x512x333) DVR	3.52	8.67
David (1536x576x352) ISO*	1.89	7.52
David (1536x576x352) DVR*	1.54	5.67

Table 6.1: Comparison of rendering speed with and without the complex bounding geometry. *Michelangelo's David was rendered with the blocking scheme enabled due to its size. BG = bounding geometry.

For most datasets, enabling the bounding geometry results in a performance increase of a factor two to three when visualizing the complete volume (i.e. no significant parts of the volume are clipped via iso-value or transfer function). In the case of large datasets like Michelangelo's David that are more sparsely populated, the comparatively small block size can lead to even greater speed-ups.

Dataset high ISO*	BG disabled [fps]	BG enabled [fps]
Hand (256x256x128) ISO	7.51	156.03
Head (512x512x333) ISO	2.96	165.95

Table 6.2: Comparison of rendering speed for sparsely populated volumes. *The threshold was set to a very high iso-value in order to render only the innermost structures.

Changing the threshold to an iso-value that will clip large parts of the volume highlights the problem of a standard raycaster: The need to cast through all of the empty space that does not contribute to the final image will result in unacceptable low framerates. On the other hand, enabling the bounding geometry will result in casting only through the small visible parts of the volume, resulting in a speedup up to a factor of 50.

6.1.2 Rendering Modes

This section compares performance in different rendering modes, mainly showing the huge performance hit shaded DVR imposes.

As can be seen in this comparison, shaded DVR slows down rendering by more than 60% compared to unshaded DVR due to the computational demand for gradient and lighting calculations. In comparison, iso-shaded DVR imposes only a minor speed penalty of about 15%.

Dataset	ISO [fps]	Unshaded [fps]	Shaded [fps]	Iso-shaded [fps]
Hand (256x256x128)	92.01	64.53	24.12	56.72
Head (512x512x333)	11.18	8.67	3.53	7.16

Table 6.3: Comparison of different rendering modes.

6.1.3 Hitpoint Refinement

Hitpoint refinement does not have a noticable impact on final rendering performance. As noticed in chapter 5.1, enabling hitpoint refinement even leads to a small increase in rendering performance - a fact that has been reproduced in numerous test on different systems. This may very well be a pipelining issue and may change with further iterations of the graphics driver, but even then the noticable performance hit should be comparable to interleaved sampling.

Dataset	HR disabled [fps]	HR enabled [fps]
Hand (256x256x128) ISO	89.13	92.01
Head (512x512x333) ISO	10.54	11.18

Table 6.4: Comparison of hitpoint refinement speed penalty.

6.1.4 Interleaved Sampling

As mentioned in chapter 5.2, interleaved sampling has almost no impact on rendering performance, resulting in a neglectable drop in framerates while at the same time providing far superior image quality.

Dataset	IS disabled [fps]	IS enabled [fps]
Hand (256x256x128) DVR	11.18	11.13
Head (512x512x333) DVR	8.67	8.54
Hand (256x256x128) iso-DVR	56.72	55.48
Head (512x512x333) iso-DVR	7.16	7.03

Table 6.5: Comparison of interleaved sampling speed penalty.

For normal direct volume rendering, the performance drop is usually somewhere around 1%. Iso-surface shaded DVR uses interleaved sampling for the surface detection as well as for the DVR, resulting in a performance hit of about 2%.

6.2 Quality Improvements

6.2.1 Hitpoint Refinement

Figure 6.2: Undersampling the dataset in first-hit renderings can lead to sampling artifacts (left) that can be eliminated by turning hitpoint refinement on (right).

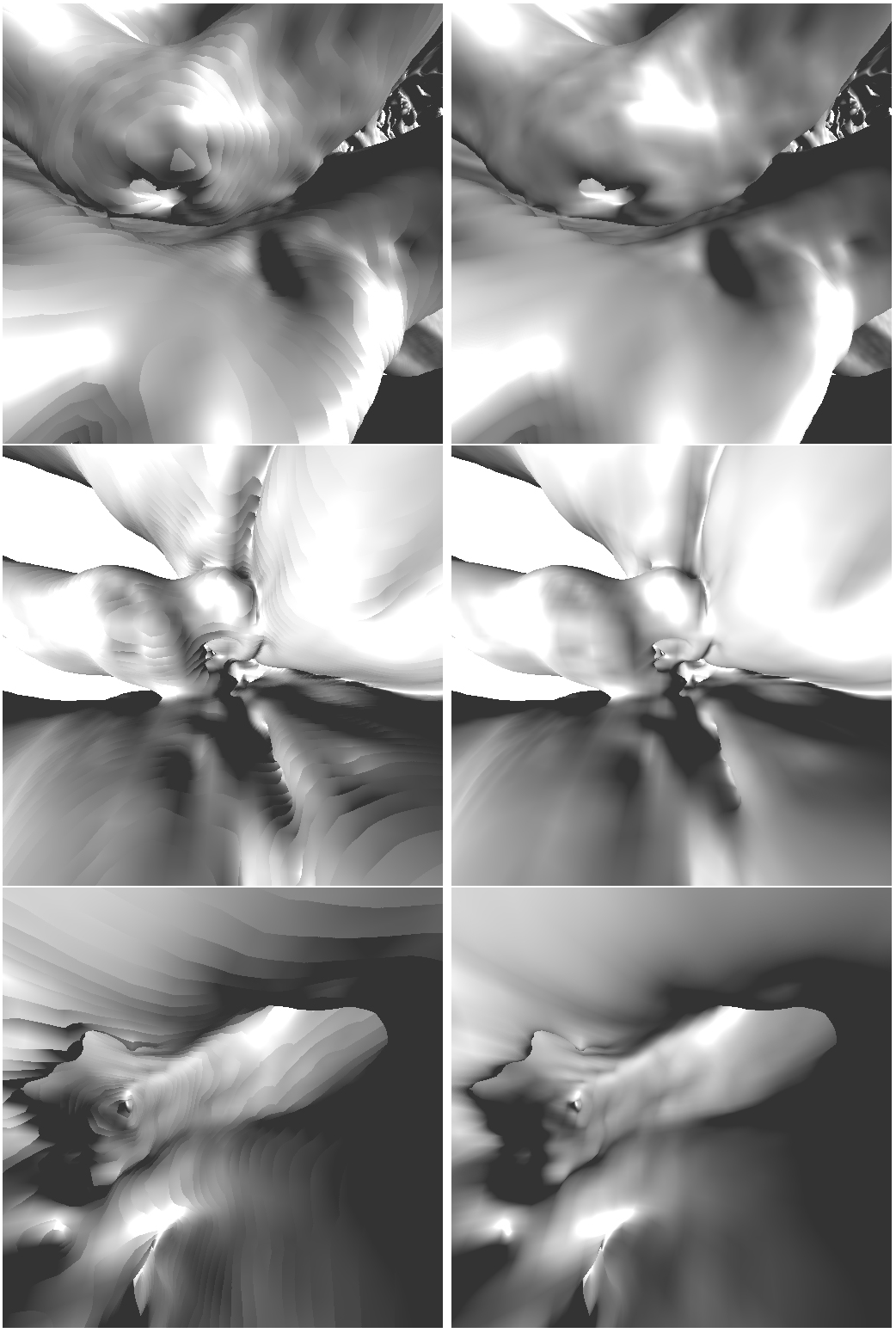

Figure 6.3: More examples where hitpoint refinement substantially increases image quality.

6.2.2 Interleaved Sampling

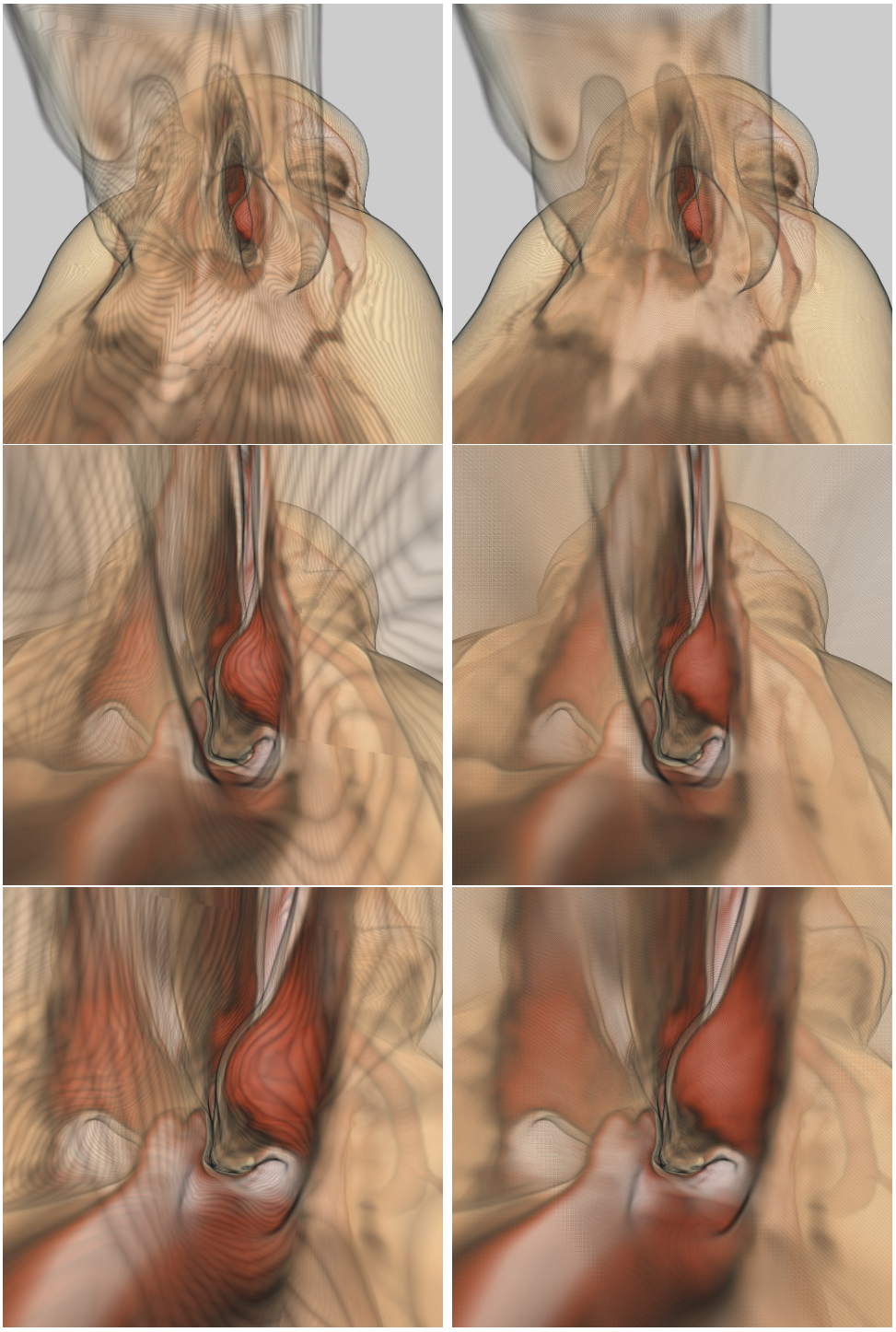

Figure 6.4: Undersampling the dataset in DVR can lead to sampling artifacts (left) that can be eliminated by turning interleaved sampling on (right).

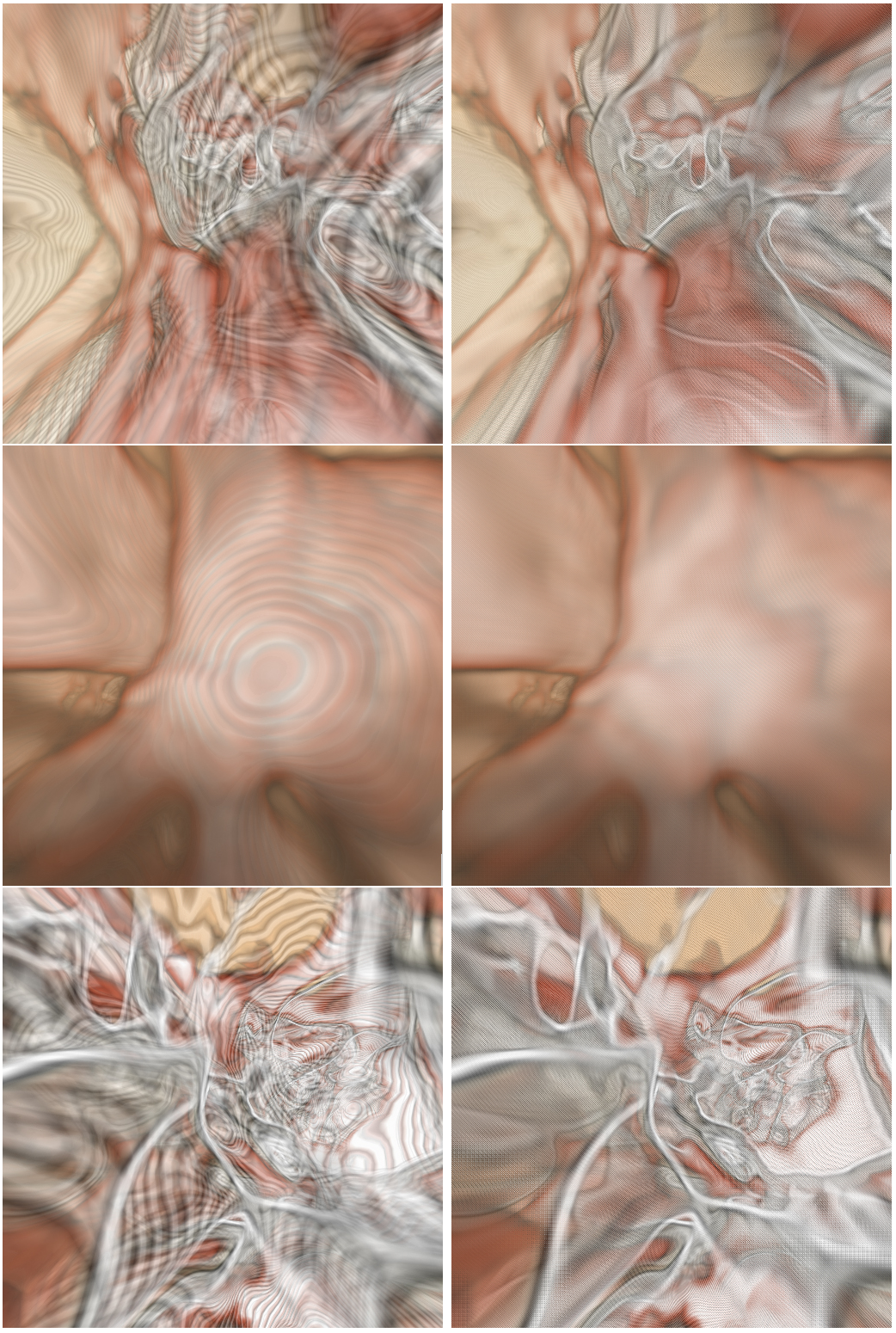

Figure 6.5: Especially for very demanding transfer functions, interleaved sampling dramatically increases image quality.

6.3 Medical Applications

As stated before, though the rendering pipeline was kept as versatile as possible, the raycaster was designed especially for virtual endoscopy applications. Thus, cooperation with hospitals and surgeons was essential in developing a system that would fit their needs in the best possible way and would incorporate all the features that are missing in other systems available at this time.

This cooperation lead to a constant exchange of pre-builds of the raycaster on the one side, and impressive images of the use in daily clinical practice on the other side, that was heavily influencing further development and provided most useful insights into the work process that this system should be integrated into.

Some of the images that were the result of this cooperation will be presented in this section, highlighting the improved visibility of important features that would be difficult to detect otherwise.

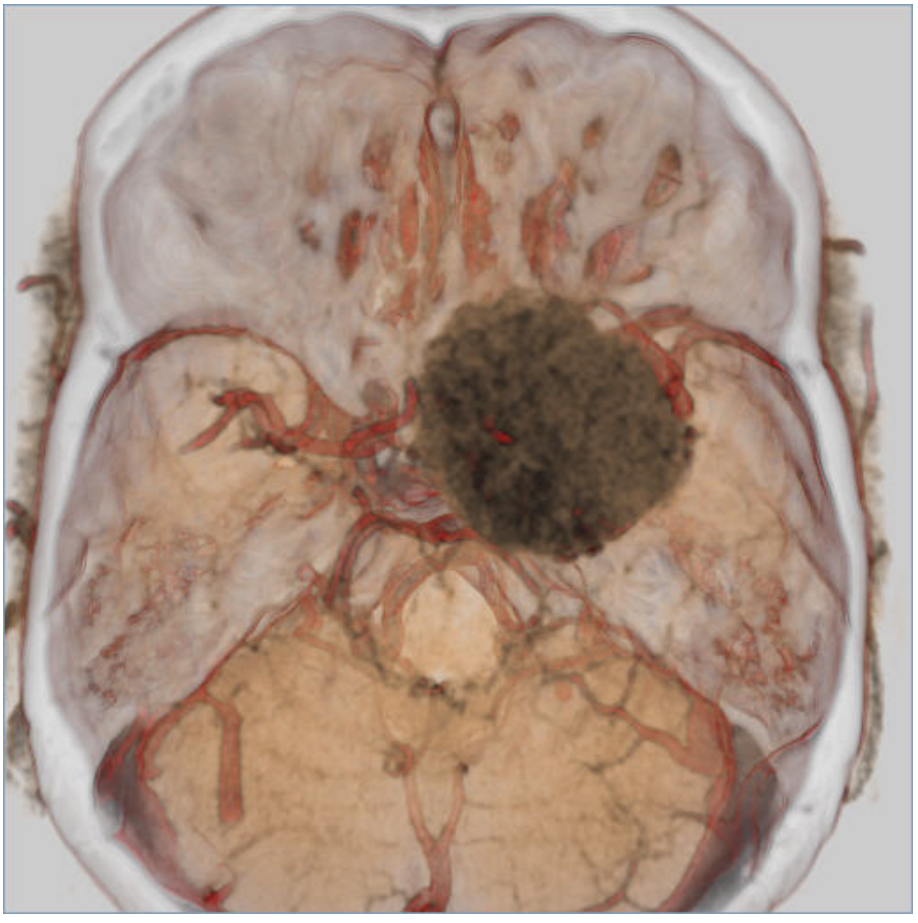

Figure 6.6: Using the raycaster in pre-operative planning gives additional insight on the position and structure of objects of interest. Image courtesy of S. Wolfsberger, Department of Neurosurgery, Medical University Vienna.

The difference of our raycaster compared to the enhanced iso-surfaces visualization available in J-Visions virtual endoscopy plug-in [Neubauer et al., 2004] is shown in Figure 6.6. In the top left picture, simple color-mapping is applied, giving an indication about the rigidity of the visible iso-surface. The top right picture shows the second rendering mode, double iso-surfacing, which identifies a second iso-surface after the first one and combines them into one image. At last, in the bottom left a limited volume rendering is applied, actually performing a couple of DVR steps after the current iso-surface.

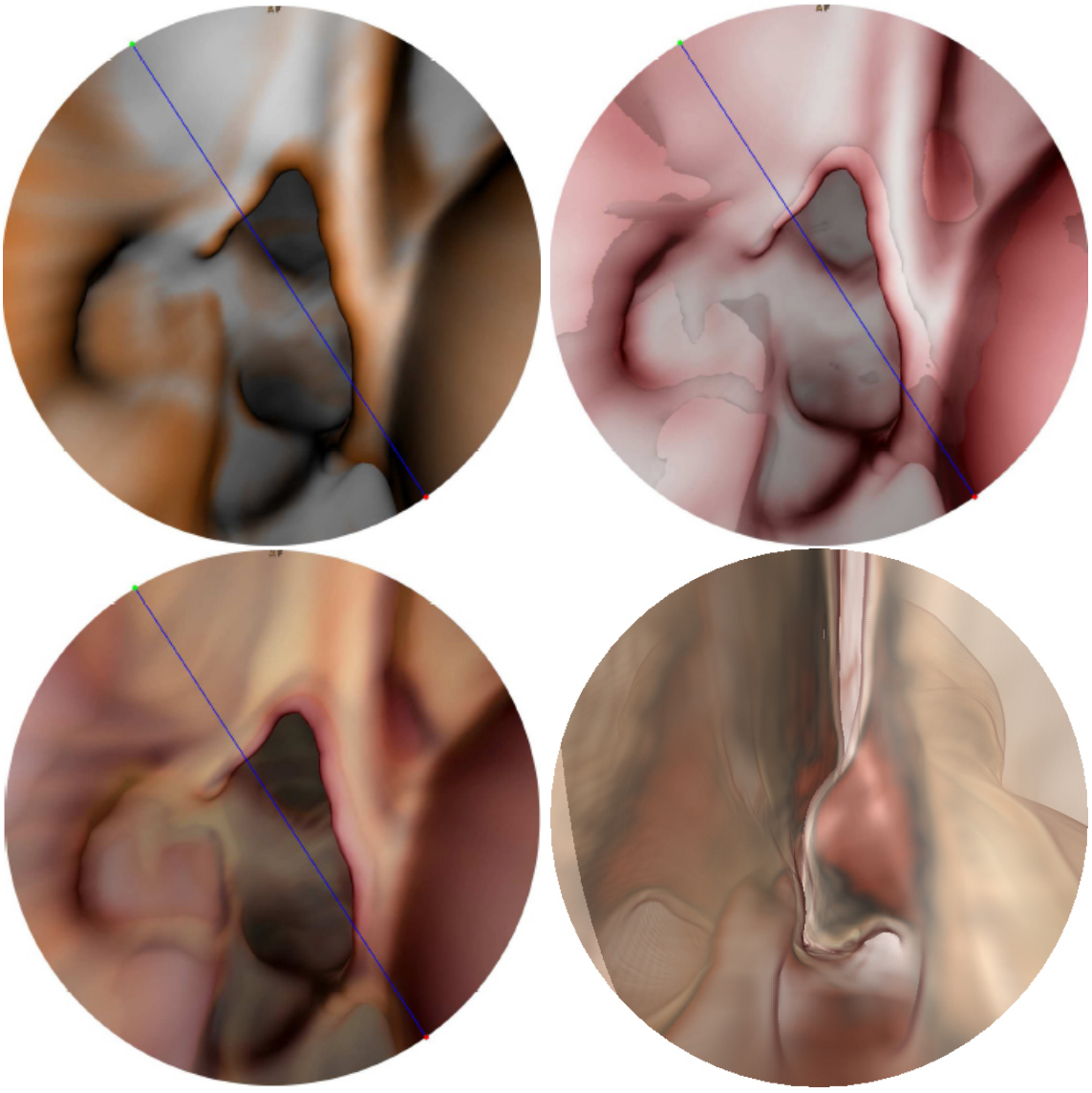

Figure 6.7: Comparison of different rendermodes from STEPS [Neubauer et al., 2004] (top left: color-mapping, top right: double iso-surface, bottom left: volume rendering) and our raycaster (bottom right).

Especially the last mode represents a significant improvement over standard iso-surface visualizations, but can not be compared to a full DVR due to its limited range. Furthermore, though especially optimized, the software implementation does not allow for higher framerates, requiring a reduction of resolution to achieve near-interactive rendering speed.

As shown in the bottom right picture of Figure 6.6, employing a real DVR introduces another level of expressiveness by visualizing different layers of transparent surfaces, making navigation within the dataset a lot easier.

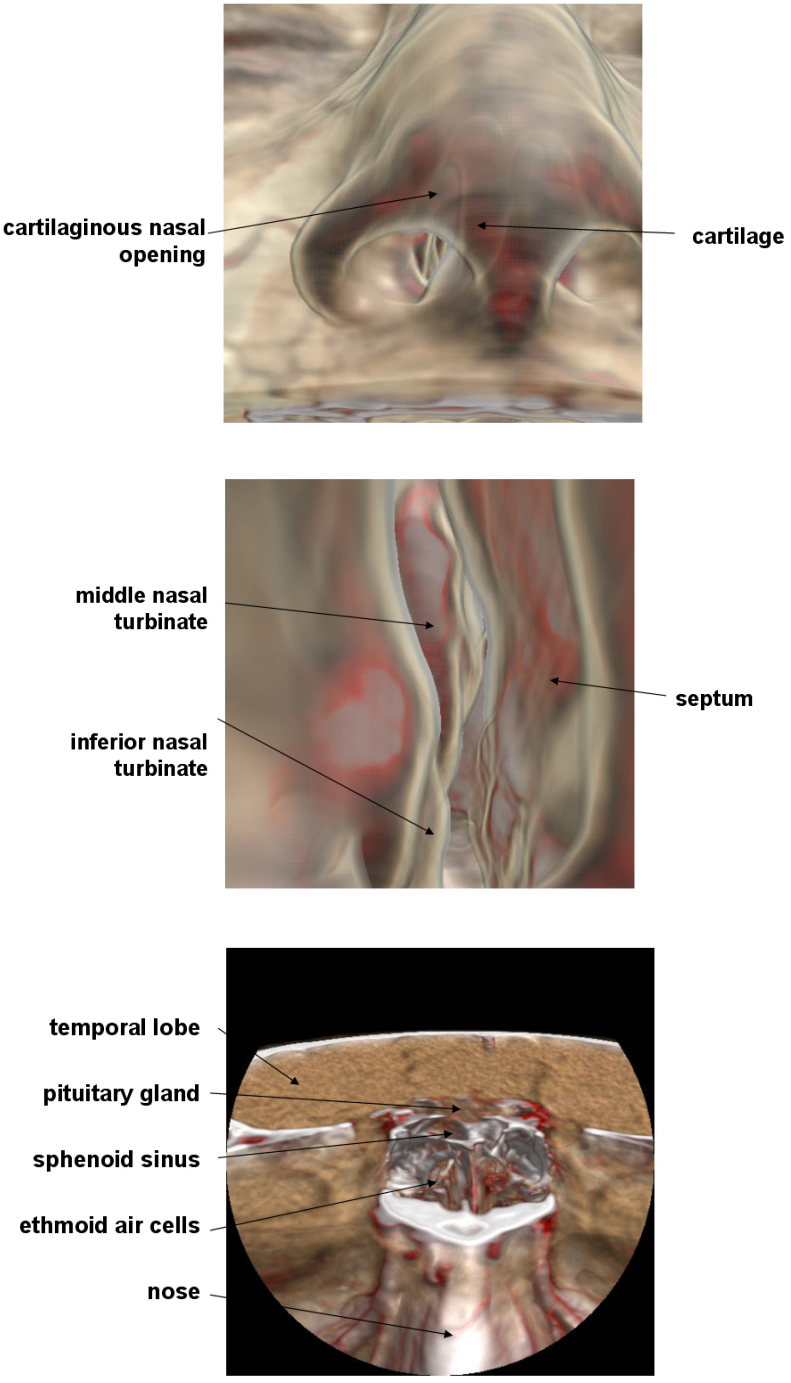

Figure 6.8: Besides diagnosis and pre-operative planning, teaching purposes are an important application of virtual endoscopy systems. Images courtesy of S. Wolfsberger, Department of Neurosurgery, Medical University Vienna.

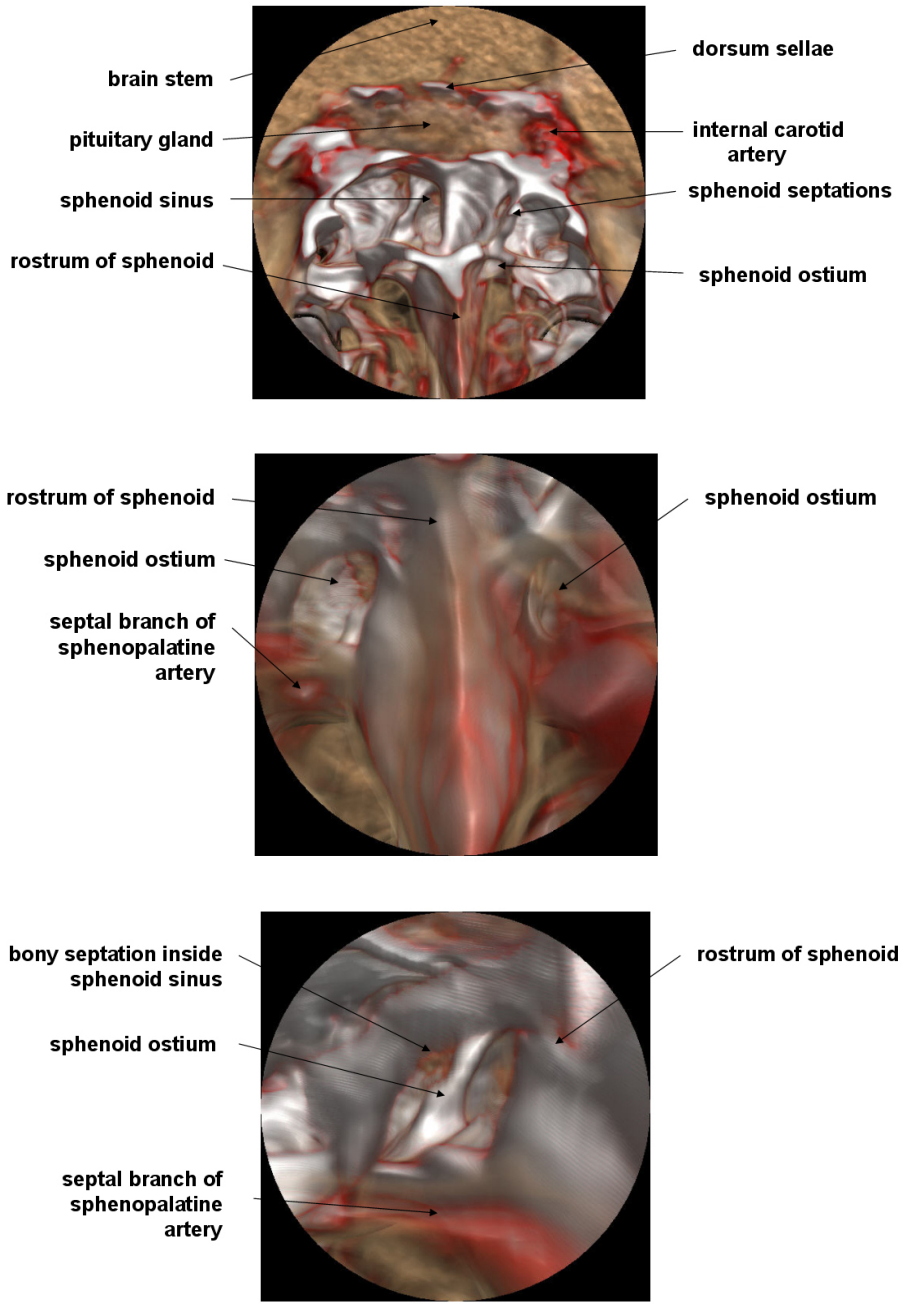

Figure 6.9: Interactive DVR allows for easier identification of certain structures and objects of interest. Images courtesy of S. Wolfsberger, Department of Neurosurgery, Medical University Vienna.

7. Conclusions and Future Work

This chapter concludes the work presented in this thesis and gives an outlook into possible future work, including extensions to existing techniques, incorporation of new features and restructuring of the existing pipeline to allow for even more flexible handling of different applications.

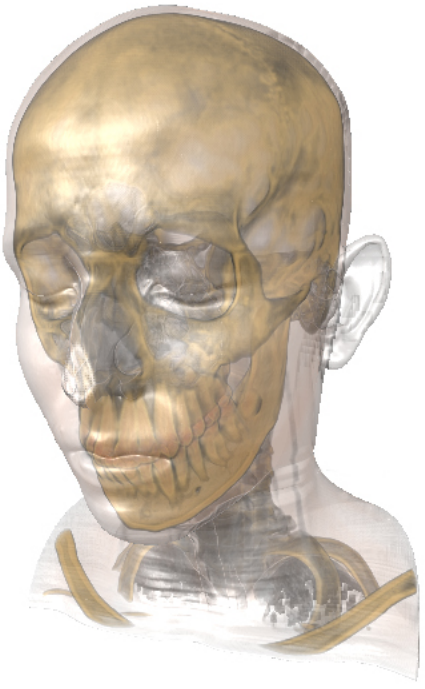

Figure 7.1: Visualization of different objects - like skin and bones - could be enhanced with the support for segmented datasets.

7.1 Conclusions

This thesis tried to demonstrate that interactive perspective DVR is already possible to achieve and thus speed concerns should not prevent the use of DVR renderings in virtual endoscopy anymore. Additionally, several enhancements and additions to the basic algorithm were presented that overall make for a full-fledged raycasting environment capable of meeting a wide variety of visualization demands.

Specific algorithms were introduced that speed up rendering time to a level where no compromise needs to be made regarding rendermode or transfer function, keeping framerates constantly high for every possible combination of different datasets and visualization strategies. Furthermore, a number of specialized techniques were presented that deal with additional issues that arise in the process of rendering volumes in the context of different applications, like memory management for large datasets and the possibility to correctly intersect the volume with OpenGL-geometry to allow rendering of 3D-pointers, grids and endoscopic tools - a necessity for a real-time virtual endoscopy system. Also, arbitrary clipping can be applied to the volume at any time to cut away parts of the dataset that are of no interest to the user.

At last, various quality problems were adressed and dealt with, mostly concerning the visibility of sampling artifacts in the presence of tiny structures or high-frequency transfer functions. Two techniques were presented here that significantly improve rendering quality without a noticable speed penalty, one for iso-surface extraction and one for direct volume rendering. Also, a new rendermode was presented that combines the advantages of shaded and unshaded DVR and incorporates all extensions that were presented in this thesis. This mode is especially useful for virtual endoscopy, where one distinct iso-surface needs to be extracted together with a DVR of the area behind this tissue, which enables the user to visualize areas of interest - like a tumor that has to be removed - behind the iso-surface without the need for pre-segmentation of the dataset.

Overall, this makes for a complete raycasting framework that can seamlessly switch between orthogonal and perspective projection and enter fly-through mode at anytime to explore the dataset from the inside while always delivering high-quality full-resolution renderings from every viewpoint at interactive framerates. However, there are still some limitations that can render this approach inferior to other techniques for specific demands. In some cases, we have already tried to deal with these limitations and will continue to do so, to make the hardware raycasting approach as competitive as possible. Other limitations like the partially unflexible nature of the graphics hardware will not likely disappear in the near future, since it will not be easily possible to build a chip with all the advantages of a highly parallelized architecture but none of the disadvantages.

That said, the GPU will never be the platform of choice for algorithms that are not suitable for highly parallel environments, like a completely adaptive raycaster, and systems that require similar visualization strategies should better revert to CPU-based approaches. Still, a lot of future work may go into building algorithms that combine the advantages of both worlds and can still be executed on a GPU, like a

semi-adaptive raycaster that may be able to change sampling rates without completely destroying the caching strategy. This approach would be similar to what we have done for the memory management, where the employed strategy loosens the memory limitations while still saving enough of the hardware 'bonus' to make this approach more than competitive. Still, the work on the memory management to be able to render even bigger datasets - which will be described in greater detail below - will definitely be continued as long as the speed-up gained from the hardware algorithm still outweighs the additional effort of additional data transfer.

To conclude all that was said before, the main point of this thesis was not to show that a GPU is a better platform for graphical algorithms in general, but that it is already flexible enough even for complex visualization systems and can be a very powerful environment once certain algorithm criteria are met.

7.2 Future Work

Our future work will mostly focus on removing the last limitations of the memory management, making it possible to render any kind of dataset regardless of the size. Other topics include the extension of the new rendermode, allowing for multiple iso-surfaces that are independently colored and shaded, full support for segmented datasets and further incorporation of deferred shading into the pipeline, making it even more flexible.

7.2.1 Memory Management

The biggest limitation of the blocked memory management so far is the fact that it still limits the overall size of the dataset. Employing this scheme, the available video memory only has to be sufficient for storing all the *active* blocks in the dataset anymore, which is a huge improvement. However, this also means that the user could still be restricted when designing a transfer function for a very large dataset, because as soon as this transfer function would result in more active blocks than fit into the memory, some blocks would be randomly selected and removed from rendering.

Of course this is no desirable behaviour, and though this combination of a huge dataset with a transfer function activating most of its blocks will be a rare one, it still possibly limits the applicability of the framework. Thus, the last logical step would be rendering multiple parts of the volume one at a time and then blending these parts together at the very end. This will of course result in heavy traffic across the graphics bus, which is why it seems like a perfect task for a PCI-Express equipped system. The actual performance gains of this technology for an application like this remains still to be seen, but it certainly will be faster than the AGP equivalent.

7.2.2 Surface Shaded DVR

The new rendermode that was introduced in this thesis was specially taylored to the needs of virtual endoscopy applications, where the properties and structure of the first iso-surface and the coloring of certain tissue behind this surface are or primary interest. However, comparing normal outer renderings with this new mode to traditional shaded DVs suggests that a couple of other interesting applications could be found. Though the improved quality of the image compared to a shaded DVR is clearly visible, there are certain circumstances where the surface shaded DVR lacks shading of inner structures like blood vessels or bones that would contribute to the clarity and applicability of the image.

Thus, the possibility of adding more than one shaded surface could add another level of detail to the image, resulting in even more convincing renderings while not sacrificing image quality. How many of these surfaces can clearly be distinguished by the human eye remains to be seen, but the choice of adding additional layers should probably be left to the user.

7.2.3 Deferred Shading

Virtually all of the presented algorithms in this thesis are highly suitable for incorporation into a deferred shading pipeline, because all of them rely on precomputed images giving them additional information about the calculations on the current screen pixel. Completely separating the shading process from the rest of the algorithm would only be another logical step that would unify all possible rendermodes into one scheme.

The beauty of this approach lies in its extensibility, since from then on every surface-based rendermode could be incorporated by simply changing the fragment program that calculates the final shading after the first iso-surface extraction. The possibility of multiple combinations of where to use which fragment program could then lead to a framework in which there are no fixed rendermodes, but arbitrary combinations of shaders for every single step of the algorithm.

7.2.4 Segmentation

At last, full support for segmentation is planned to be incorporated into the ray-caster. This means that segmented objects can be assigned different iso-values and transfer functions and can easily be hidden to improve the visibility of other objects in the dataset.

Segmentation is of primary interest once important features cannot easily be detected using only a specific transfer function. This is mostly due to the fact that the iso-value of a region of interest is commonly represented in other tissue throughout the dataset as well.

Regardless of that, many datasets in medical practice are pre-segmented anyway, and simply ignoring this information while trying to reproduce the exact same result by modifying the transfer function might not be such a good idea.

Acknowledgements

This work has been carried out as part of the basic research on medical visualization at the VRVis Research Center for Virtual Reality and Visualization in Vienna (http://www.vrvis.at) which is funded in part by the KPlus program of the Austrian government. The medical data sets are courtesy of Tiani Medgraph and the Department of Neurosurgery at the Medical University Vienna.

I would like to express my sincere gratitude and appreciation to all those who made this diploma thesis possible: Foremost to my advisor Markus Hadwiger, who has supported and encouraged me for the last two years and has been a steady source of knowledge and insight into the areas of visualization and computer graphics as well as a person that it is a real pleasure to work with, to Katja Bühler, who accepted the topic of this thesis at a time when the outcome and use of these techniques was still unclear and questioned by many and has been most forgiving about various missed deadlines due to sudden indispositions at the most inappropriate times, and, last but not least, to 'the Master' Eduard Gröller not only for his supervision of this thesis, but for being a spiritual guide in technical as well as non-technical issues throughout the second part of my studies. Thy bidding will be done.

I would further like to thank Stefan Wolfsberger from the Department of Neurosurgery, Medical University Vienna, for the fruitful collaboration that provided the necessary feedback from clinical practice and for giving me the opportunity to join the workshop for neuro-surgery to get a deeper insight into clinical routine in general and the foremost problems with endoscopic procedures in particular, and to André Neubeuer for his support in integrating the raycaster into J-Vision.

I thank the VRVis for providing me a work place and a computer and the whole staff for the pleasant working atmosphere throughout my time there. At last, I want to thank Christoph Berger, who was a tremendous help in the first months of my research and one of the most likeable persons I have ever met before he lost his life in a tragic accident. More than one year has passed since then, but the loss of him as a colleague and friend is still inconceivable for all of us.

Bibliography

ATI. ATI web page. http://www.ati.com/.

D. Auer and L. Auer. Virtual endoscopy - a new tool for teaching and training in neuroimaging. In *International Journal of Neuroradiology*, pages 4:3–14, 1998.

L. Auer, D. Auer, and J. Knoplioch. Virtual endoscopy for planning and simulation of minimally invasive neurosurgery. In *First Joint Conference, Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery, volume LNCS 1205*, pages 315–318, 1997.

A. V. Bartrolí. Visualization techniques for virtual endoscopy. In *PhD thesis, Technical University of Vienna*, 2001.

A. V. Bartrolí, A. König, and E. Gröller. Viren: Virtual endoscopy system. machine graphics & vision. In *8(3):469-487*, page 1999.

A. V. Bartrolí, R.Wegenkittl, A. König, E. Gröller, and E. Sorantin. Virtual colon flattening. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 127–136, 2001a.

A. V. Bartrolí, R. Wegenkittl, A. König, and E. Gröller. Nonlinear virtual colon unfolding. In *Proc. of IEEE Visualization*, 2001b.

A. V. Bartrolí, R. Wegenkittl, A. König, and E. Gröller. Perspective projection through parallely projected slabs for virtual endoscopy. In *Proc. of Spring Conference on Computer Graphics*, pages 287–295, 2001c.

D. Bartz. Prototyping a virtual colonoscopy system. In *Master's thesis, Dept. of Computer Science, University of Erlangen-Nürnberg*, 1996.

D. Bartz. Extraction and visualization of coronary vascular structures. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, 2003.

D. Bartz. Virtual endoscopy in research and clinical practice. In *Computer Graphics Forum*, page 24(1), 2005.

D. Bartz, . Gürvit, D. Freudenstein, H. Schiffbauer, and J. Hoffmann. Integration von navigation, optischer und virtueller endoskopie in der neuro- sowie mund-, kiefer- und gesichtschirurgie. In *Jahrestagung der Deutschen Gesellschaft für Computer-und Roboterassistierte Chirurgie e.V. (CURAC)*, 2002.

D. Bartz, . Gürvit, M. Lanzendörfer, A. Kopp, A. Küttner, and W. Straßer. Virtual endoscopy for cardio vascular exploration. In *Proc. of Computer Assisted Radiology and Surgery*, pages 960–964, 2001a.

D. Bartz, D. Mayer, J. Fischer, S. Ley, A. del Río, S. Thust, C. Heussel, H. Kauczor, and W. Straßer. Hybrid segmentation and exploration of the human lungs. In *Proc. of IEEE Visualization*, 2003.

D. Bartz and M. Meißner. Voxels versus Polygons: A Comparative Approach for Volume Graphics. In *Volume Graphics*, 1999.

D. Bartz and M. Skalej. Vivendi - a virtual ventricle endoscopy system for virtual medicine. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 155–166,324, 1999.

D. Bartz, M. Skalej, D. Welte, W. Straßer, and F. Duffner. A virtual endoscopy system for the planning of endoscopic interventions in the ventricle system of the human brain. In *Proc. of BiOS'99: Biomedical Diagnostics, Guidance and Surgical Assist Systems, volume 3514*, page 91.100, 1999a.

D. Bartz, W. Straßer, . Gürvit, D. Freudenstein, and M. Skalej. Interactive and multi-modal visualization for neuroendoscopic interventions. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 157–164, 2001b.

D. Bartz, W. Straßer, M. Skalej, and D. Welte. Interactive exploration of extra- and intracranial blood vessels. In *Proc. of IEEE Visualization*, pages 389–392,547, 1999b.

J. Beier, T. Diebold, H. Vehse, G. Biamino, E. Fleck, and R. Felix. Virtual endoscopy in the assessment of implanted aortic stents. In *Proc. of Computer Assisted Radiology*, pages 183–188, 1997.

I. Bitter, M. Sat, M. Bender, K. McDonnell, A. Kaufman, and M. W. CEASAR. A smooth, accurate and robust centerline extraction algorithm. In *Proc. of IEEE Visualization*, pages 45–52, 2000.

K. Bjorke. High-quality filtering. In *GPU Gems*, pages 391–415. Addison Wesley Professional, 2004.

J. Blinn. Jim blinn's corner: Image compositing—theory. *IEEE Computer Graphics and Applications*, 14(5):83–87, 1994.

A. Bode, F. Dammann, E. Pelikan, M. Heuschmid, E. Schwaderer, M. Schaich, and C. Claussen. Analyse von artefakten bei der virtuellen endoskopischen darstellung auf basis von spiral-ct-daten. In *RöFö: Fortschritte auf dem Gebiet der Röntgenstrahlen und der neuen bildgebenden Verfahren*, pages 173:245–252, 2001.

J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *Proceedings of SIGGRAPH 2003*, pages 917–924, 2003.

U. Bordoloi and H.-W. Shen. Space efficient fast isosurface extraction for large datasets. In *Proceedings of IEEE Visualization 2003*, pages 201–208, 2003.

M. Brady, K. Jung, H. Nguyen, and T. Nguyen. Two-Phase Perspective Ray Casting for Interactive Volume Navigation. In *Proc. IEEE Visualization*, 1997.

S. Bruckner. Efficient volume visualization of large medical datasets. In *Master's thesis, Computer Science Department, Technical University of Vienna*, 2003.

B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 91–98, 1994a.

B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proc. of Symposium on Volume Visualization*, pages 91–98, 1994b.

S. Chen. Quicktime vr . an image-based approach to virtual environment navigation. In *Proc. of ACM SIGGRAPH*, pages 29–38, 1995.

Y.-J. Chiang, C. Silva, and W. Schroeder. Interactive out-of-core isosurface extraction. In *Proceedings of IEEE Visualization '98*, pages 167–174, 1998.

P. Cignoni, P. Marino, E. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.

C. Consortium. Eu cophit project: Computer- optimised pulmonary delivery in humans of inhaled therapies. In *http://www.cophit.co.uk*.

R. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings of IEEE Visualization '93*, pages 261–267, 1993.

B. Csébfalvi and E. Gröller. Interactive volume rendering based on a bubble model. In *Proceedings of Graphics Interface 2001*, pages 209–216, 2001.

B. Csébfalvi, L. Mroz, H. Hauser, A. König, and E. Gröller. Fast visualization of object contours by non-photorealistic volume rendering. In *Proceedings of Eurographics 2001*, pages 452–460, 2001.

T. Cullip and U. Neumann. Accelerating volume reconstruction with 3d texture hardware. In *Technical Report TR93-027, University of North Carolina at Chapel Hill*, 1993a.

T. Cullip and U. Neumann. Accelerating volume reconstruction with 3D texture mapping hardware. Technical Report TR93-027, Department of Computer Science, University of North Carolina, Chapel Hill, 1993b.

F. Dachille and A. Kaufman. GI-Cube: An Architecture for Volumetric Global Illumination and Rendering. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 119–128, 2000.

F. Dachille, K. Kreeger, B. Chen, I. Bittner, and A. Kaufman. High-quality volume rendering using texture mapping hardware. In *Proceedings of Eurographics/SIGGRAPH Graphics Hardware Workshop 1998*, 1998.

F. Dachille, K. Kreeger, M. Wax, A. Kaufman, and Z.Liang. Interactive navigation for pc-based virtual colonoscopy. In *Proc. of SPIE Medical Imaging*, 2001.

J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Workshop on Volume Visualization*, pages 91–98, 1992.

C. Davis, M. Ladds, B. Romanowski, S. Wildermuth, J. Knoplioch, and J. Debatin. Human aorta: Preliminary results with virtual endoscopy based on three-dimensional mr imaging data sets. In *Radiology*, pages 199:37–40, 1996.

D. Dey, D. Gobbi, P. Slomka, K. Surry, and T. Peters. Automatic fusion of free-hand endoscopic brain images to three-dimensional surfaces: Creating stereoscopic panoramas. In *IEEE Transactions on Medical Imaging*, pages 21(1):23–30, 2002.

M. Doggett. An Array Based Design for Real-Time Volume Rendering. In *Proc. 10th Eurographics Workshop on Graphics Hardware*, pages 93–101, 1995.

M. Doggett and G. Hellestrand. A Hardware Architecture for Video Rate Smooth Shading of Volume Data. *Computers and Graphics*, 19(5):695–704, 1995.

R. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *Proceedings of SIGGRAPH '88*, pages 65–74, 1988.

F. Duffner, W. Dauber, M. Skalej, and E. Grote. A new endoscopic tool for the crw stereotactic system. In *Stereotactic and Functional Neurosurgery, volume 67(3-4)*, pages 213–217, 1994.

D. Ebert, C. Morris, P. Rheingans, and T. Yoo. Designing effective transfer functions for volume rendering from photographic volumes. *IEEE Transactions on Visualization and Computer Graphics*, 8(2):183–197, 2002.

D. Ebert, F. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing & Modeling: A Procedural Approach.* Third Edition, Morgan Kaufman Publishers, 2003.

D. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume models. In *Proceedings of IEEE Visualization 2000*, pages 195–202, 2000.

T. Elvins. A Survey of Algorithms for Volume Visualization. *Computer Graphics*, 26(3):194–201, 1992.

K. Engel. Interactive high-quality volume rendering with flexible consumer graphics hardware. In *Eurographics State-of-the-Art-Report*, 2002.

K. Engel and T. Ertl. Texture-based volume visualization for multiple users on the world wide web. In *Virtual Environments '99. Proc. of the Eurographics Workshop in Vienna, Austria*, pages 115–124, 1999.

K. Engel, M. Hadwiger, J. Kniss, and C. Rezk-Salama. *High-Quality Volume Graphics on Consumer PC Hardware*. Course Notes for Course #42 at SIGGRAPH 2002, ACM SIGGRAPH, 2002.

K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of Graphics Hardware 2001*, pages 9–16, 2001.

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

S. Fang, T. Biddlecome, and M. Tuceryan. Image-Based Transfer Function Design for Data Exploration in Volume Visualization. In *Proc. IEEE Visualization*, 1998.

R. Fernando and M. Kilgard. *The Cg Tutorial - The Definitive Guide to Programmable Real-Time Graphics*. Addison Wesley, 2003.

G. Ferretti, D. Vining, J. Knoplioch, and M. Coulomb. Tracheobronchial tree: Three-dimensional spiral ct with bronchoscopic perspective. In *Journal of Computer Assisted Tomography*, pages 20(5):777–781, 1996.

J. Fischer, D. Bartz, and W. Straßer. Occlusion handling for medical augmented reality using a volumetric phantom model. In *Proc. of ACM Symposium on Virtual Reality Software and Technology*, 2004.

G. and W. Straßer. A compact volume rendering accelerator. In A. Kaufman and W. Krueger, editors, *IEEE Symposium on Volume Visualization*, pages 67–74. ACM SIGGRAPH, Oct. 1994. ISBN 0-89791-741-3.

T. Galyean. Guided navigation of virtual environments. In *Proc. of ACM Symposium on Interactive 3D Graphics*, pages 103–104, 1995.

D. Gering, A. Nabavi, R. Kikinis, N. Hata, L. O'Donnell, E. Grimson, F. Jolesz, P. Black, and W. Well. An integrated visualization system for surgical planning and guidance using image fusion and open mr. In *Journal on Magnetic Resonance Imaging*, pages 13:967–975, 2001.

E. Gobbetti, P. Pili, A. Zorcolo, and M. Tuveri. Interactive virtual angioscopy. In *Proc. of IEEE Visualization*, pages 435–438, 1998.

A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of SIGGRAPH '98*, pages 447–452, 1998.

GPGPU. General Purpose Computations on GPUs (GPGPU) web page. http://www.gpgpu.org/.

S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. VOTS: VOlume doTS as a point-based representation of volumetric data. In *Proceedings of Eurographics 2004*, 2004.

T. Günther, C. Poliwoda, C. Reinhard, J. Hesser, R. Männer, H.-P. Meinzer, and H.-J. Baur. VIRIM - A Massively Parallel Processor for Real-Time Volume Visualization in Medicine. In *Proc. 9th Eurographics Workshop on Graphics Hardware*, pages 103–108, 1994.

S. Guthe and W. Strasser. Real-time decompression and visualization of animated volume data. In *Proceedings of IEEE Visualization 2001*, pages 349–356, 2001.

S. Guthe, M. Wand, J. Gonser, and W. Strasser. Interactive rendering of large volume data sets. In *Proceedings of IEEE Visualization 2002*, pages 53–60, 2002.

M. Hadwiger, C. Berger, and H. Hauser. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of IEEE Visualization 2003*, pages 301–308, 2003a.

M. Hadwiger, H. Hauser, and T. Möller. Quality issues of hardware-accelerated high-quality filtering on PC graphics hardware. In *Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) 2003*, pages 213–220, 2003b.

M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. In *Proceedings of Eurographics 2005*, page to appear, 2005.

M. Hadwiger, T. Theußl, H. Hauser, and E. Gröller. Hardware-accelerated high-quality filtering on PC hardware. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2001*, pages 105–112, 2001.

M. Hadwiger, T. Theußl, H. Hauser, and E. Gröller. Mip-mapping with procedural and texture-based magnification. In *SIGGRAPH 2003 Sketches and Applications*, 2003c.

M. Hadwiger, I. Viola, T. Theußl, and H. Hauser. Fast and flexible high-quality texture filtering with tiled high-resolution filters. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2002*, pages 155–162, 2002.

M. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. In *Proceedings of Graphics Hardware 2002*, pages 109–118, 2002.

P. Hastreiter, C. Rezk-Salama, G. Greiner, and T. Ertl. Interactive Direct Volume Rendering of the Inner Ear for the Planning of Neurosurgery. In *Bildverarbeitung in der Medizin: Algorithmen, Systeme, Anwendungen*. Springer, 1999a.

P. Hastreiter, C. Rezk-Salama, B. Tomandl, K. Eberhardt, and T. Ertl. Comparing the Quality of Interactive Volume Rendering Methods in Neuroradiology. In *Proc. Rapid Prototyping in Medicine and Computer-Assisted Surgery (CAS)*, 1999b.

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

H. Hauser, L. Mroz, G.-I. Bischi, and E. Gröller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, 2001.

T. He, L. Hong, A. Kaufman, and H. Pfister. Generation of Transfer Functions with Stochastic Search Techniques. In *Proc. IEEE Visualization*, 1996.

H. Hege, T. Höllerer, and D. Stalling. Volume Rendering, Mathematical Foundations and Algorithmic Aspects. Technical Report TR93-7, Konrad-Zuse-Zentrum für Informationstech., Berlin, 1993.

W. Heidrich and H.-P. Seidel. Realistic, hardware-accelerated shading and lighting. In *Proceedings of SIGGRAPH '99*, pages 171–178, 1999.

R. Hietala and J. Oikarinen. A visibility determination algorithm for interactive virtual endoscopy. In *Proc. of IEEE Visualization*, pages 29–36, 2000.

W. Higgins, J. Helferty, and D. Padfield. Integrated bronchoscopic video tracking and 3d ct registration for virtual bronchoscopy. In *Proc. of SPIE Medical Imaging, volume 5031*, pages 80–89, 2003.

K. Hillesland, S. Molinov, and R. Grzeszczuk. Nonlinear optimization framework for image-based modeling on programmable graphics hardware. In *Proceedings of SIGGRAPH 2003*, pages 925–934, 2003.

J. Hladuvka. *Derivatives and Eigensystems for Volume-Data Analysis and Visualization*. PhD thesis, Vienna University of Technology, 2001.

J. Hladuvka, A. König, and E. Gröller. Curvature-based transfer functions for direct volume rendering. In *Proceedings of Spring Conference on Computer Graphics 2000*, pages 58–65, 2000.

J. Hladuvka, A. König, and E. Gröller. Salient representation of volume data. In *Proceedings of the Joint Eurographics/IEEE TVCG Symposium on Visualization 2001*, pages 203–211, 2001.

L. Hong, A. Kaufman, Y. Wei, A. Viswambharan, M. Wax, and Z. Liang. 3d virtual colonoscopy. In *Proc. of IEEE Symposium on Biomedical Visualization*, pages 26–32, 1995.

L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He. Virtual voyage: Interactive navigation in the human colon. In *Proc. of ACM SIGGRAPH*, pages 27–34, 1997.

J. Huang, K. Mueller, N. Shareef, and R. Crawfis. FastSplats: Optimized splatting on rectilinear grids. In *Proceedings of IEEE Visualization 2000*, pages 219–226, 2000.

V. Interrante. Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution. In *Proceedings of SIGGRAPH '97*, pages 109–116, 1997.

T. Itoh and K. Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1996.

T. Jankun-Kelly and K.-L. Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, 2001.

J. Kajiya and B. Von Herzen. Ray Tracing Volume Densities. In *Proc. SIGGRAPH*, 1984.

A. Kaufman. Voxels as a Computational Representation of Geometry. In *The Computational Representation of Geometry. SIGGRAPH '94 Course Notes*, 1994.

A. Keller and W. Heidrich. Interleaved sampling. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 269–276, 2001.

G. Kindlmann. *Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering.* Master's thesis, Cornell University, 1999.

G. Kindlmann and J. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of IEEE Volume Visualization '98*, pages 79–86, 1998a.

G. Kindlmann and J. Durkin. Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering. In *IEEE Symposium on Volume Visualization*, 1998b.

G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization 2003*, pages 513–520, 2003.

J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization 2001*, pages 255–262, 2001.

J. Kniss, G. Kindlmann, and C. Hansen. Multi-dimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002a.

J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. In *Proceedings of IEEE Visualization 2002*, pages 109–116, 2002b.

J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):150–162, 2003.

G. Knittel. A scalable architecture for volume rendering. *Computers and Graphics*, 19(5):653–665, Sept.–Oct. 1995. ISSN 0097-8493. URL http://www.elsevier.com/cgi-bin/cas/tree/store/cag/cas_sub/browse/browse.cgi?year=1995&;volume=19&issue=5&aid=9500044.

G. Knittel. TriangleCaster: extensions to 3D-texturing units for accelerated volume rendering. In *Proceedings 1999 Eurographics/SIGGRAPH workshop on Graphics hardware*, pages 25–34, 1999.

L. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of SIGGRAPH 2001*, pages 57–66, 2001.

M. Kraus and T. Ertl. Adaptive texture maps. In *Proceedings of Graphics Hardware 2002*, pages 7–15, 2002.

J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *Proceedings of IEEE Visualization 2003*, pages 287–292, 2003a.

J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. In *Proceedings of SIGGRAPH 2003*, pages 908–916, 2003b.

P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH '94*, pages 451–458, 1994.

A. Laghi, P. Pavone, V. Panebianco, I. Carbone, and L. Francone. Volume-rendered virtual colonoscopy: Preliminary clinical experience. In *Proc. of Computer Assisted Radiology and Surgery*, pages 171–175, 1999.

S. Lakare, M. Wan, M. Sato, and A. Kaufman. 3d digital cleansing using segmentation rays. In *Proc. of IEEE Visualization*, pages 37–44, 2000.

E. LaMar, B. Hamann, and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of IEEE Visualization '99*, pages 355–361, 1999.

A. Lastra, S. Molnar, M. Olano, and Y. Wang. Real-time programmable shading. In *ACM Symposium on Interactive 3D Graphics*, pages 59–ff., 1995.

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

M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9 (3):245–261, 1990.

W. Li and A. Kaufman. Real-time volume rendering for virtual colonoscopy. In *Proc. of Volume Graphics*, 2001.

W. Li and A. Kaufman. Accelerating volume rendering with texture hulls. In *Proceedings of IEEE VolVis 2002*, pages 115–122, 2002.

W. Li and A. Kaufman. Texture partitioning and packing for accelerating texture-based volume rendering. In *Proceedings of Graphics Interface 2003*, pages 81–88, 2003.

W. Li, K. Mueller, and A. Kaufman. Empty space skipping and occlusion clipping for texture-based volume rendering. In *Proceedings of IEEE Visualization 2003*, pages 317–324, 2003.

B. Lichtenbelt, R. Crane, and S. Naqvi. *Introduction to Volume Rendering*. Prentice-Hall, New Jersey, 1998.

Y. Livnat and C. Hansen. View dependent isosurface extraction. In *Proceedings of IEEE Visualization '98*, 1998.

Y. Livnat, H.-W. Shen, and C. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of SIGGRAPH '87*, pages 163–169, 1987a.

W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. of ACM SIGGRAPH*, pages 163–169, 1987b.

W. Lorensen, F. Jolesz, and R. Kikinis. The exploration of cross-sectional data with a virtual endoscope. In *R. Satava and K. Morgan, editors, Interactive Technology and New Medical Paradigms for Health Care*, pages 221–230, 1995.

A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. In *Proceedings of IEEE Visualization 2002*, pages 211–218, 2002.

E. Lum and K.-L. Ma. Hardware-accelerated parallel non-photorealistic volume rendering. In *Proceedings of the International Symposium on Non-Photorealistic Animation and Rendering (NPAR) 2002*, 2002.

S. Marschner and R. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of IEEE Visualization '94*, pages 100–107, 1994.

N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

D. Mayer, D. Bartz, J. Fischer, S. Ley, A. del Río, S. Thust, H. Kauczor, W. Straßer, and C. Heussel. Hybrid segmentation and virtual bronchoscopy based on ct images. In *Academic Radiology*, pages 11(5):551–565, 2004.

D. Mayer, D. Bartz, S. Ley, S. Thust, C. Heussel, H. Kauczor, and W. Straßer. Segmentation and virtual exploration of tracheo-bronchial trees. In *Proc. of Computer Assisted Radiology and Surgery*, 2003.

T. McReynolds, D. Blythe, B. Grantham, and S. Nelson. Advanced graphics programming techniques using OpenGL. In *SIGGRAPH 2000 course notes*, 2000.

M. Meißner, U. Hoffmann, and W. Straßer. Enabling classification and shading for 3D texture mapping based volume rendering. In *Proceedings of IEEE Visualization '99*, pages 207–214, 1999.

M. Meißner, J. Huang, D. Bartz, K. Müller, and R. Crawfis. A practical evaluation of four popular volume rendering algorithms. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 81–90, 2000.

M. Meißner, U. Kanus, and W. Straßer. VIZARD II: A PCI-Card for Real-Time Volume Rendering. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 61–67, 1998.

D. Mitchell and A. Netravali. Reconstruction filters in computer graphics. In *Proceedings of SIGGRAPH '88*, pages 221–228, 1988.

D. Moore and J. Warren. Mesh Displacement: An Improved Contouring Method for Trivariate Data. Technical Report TR-91-166, Rice University. Dept. of Computer Science, 1991.

K. Mori, J. Hasegawa, J. Toriwaki, H. Anno, and K. Katada. A fast rendering method using the tree structure of objects in virtualized bronchus endoscope system. In *Proc. of Visualization in Biomedical Computing, volume LNCS 1131*, pages 33–42, 1996.

K. Mori, J. Hasegawa, J. Toriwaki, S. Yokoi, H. Anno, and K. Katada. A method to extract pipe structured components in three dimensional medical images and simulation of bronchus endoscope images. In *Proc. of 3D Image Conference*, pages 269–274, 1994.

K. Mueller, T. Möller, and R. Crawfis. Splatting Without the Blur. In *Proc. IEEE Visualization*, 1999.

K. Mueller and R. Yagel. Fast Perspective Volume Rendering with Splatting by Using a Ray-Driven Approach. In *Proc. IEEE Visualization*, 1996.

H. Müller, N. Shareef, J. Huang, and R. Crawfis. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):115–134, 1999.

Z. Nagy, J. Schneider, and R. Westermann. Interactive volume illustration. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2002*, 2002.

D. Nain, S. Haker, R. Kikinis, and W. Grimson. An interactive virtual endoscopy tool. In *Proc. of Workshop on Interactive Medical Image Visualization and Analysis*, 2001.

A. Neubauer, S. Wolfsberger, M. Forster, L. Mroz, R. Wegenkittl, and K. Bühler. Steps - an application for simulation of transsphenoidal endonasal pituitary surgery. In *Proc. of IEEE Visualization*, 2004.

L. Neumann, B. Csébfalvi, A. König, and E. Gröller. Gradient estimation in volume data using 4d linear regression. In *Proceedings of Eurographics 2000*, pages 351–357, 2000.

K. Novins. *Towards Accurate and Efficient Volume Rendering*. PhD thesis, Cornell University, 1994.

K. Novins, F. Sillion, and D. Greenberg. An efficient method for volume rendering using perspective projection. In *Proc. of ACM SIGGRAPH*, pages 95,102, 1990.

NVIDIA. NVIDIA web page. http://www.nvidia.com/.

N. L. of Medicine". The Visible Human Project. available online at `http://www.nlm.nih.gov/research/visible/`, 1996.

M. Ogata, T. Ohkami, H. C. Lauer, and H. Pfister. A real-time volume rendering architecture using an adaptive resampling scheme for parallel and perspective projections. In *IEEE Symposium on Volume Visualization*, pages 31–38, 1998.

OpenGL. OpenGL web page. http://www.opengl.org/.

R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. EM-Cube: An architecture for low-cost real-time volume rendering. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 131–138, 1997.

S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings of IEEE Visualization '98*, pages 233–238, 1998.

V. Pekar, R. Wiemker, and D. Hempel. Fast detection of meaningful isosurfaces for volume data visualization. In *Proceedings of IEEE Visualization 2001*, pages 223–230, 2001.

H. Pfister, C. Bajaj, W. Schroeder, and G. Kindlmann. The Transfer Function Bake-Off. In *Proc. IEEE Visualization*, 2000a.

H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. In *Proc. of ACM SIGGRAPH*, pages 251–260, 1999.

H. Pfister and A. Kaufman. Cube-4 – A Scalable Architecture for Real-Time Volume Rendering. In *IEEE Symposium on Volume Visualization*, 1996a.

H. Pfister and A. Kaufman. Cube-4 - A scalable architecture for real-time volume rendering. In *IEEE Symposium on Volume Visualization*, pages 47–54. IEEE, Oct. 1996b. ISBN 0-89791-741-3.

H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, and R. Machiraju. The transfer function bake-off. In *Proceedings of IEEE Visualization 2000*, pages 523–526, 2000b.

M. Pharr. Fast filter width estimates with texture maps. In *GPU Gems*, pages 417–424. Addison Wesley Professional, 2004.

T. Purcell, I. Buck, W. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *Proceedings of SIGGRAPH 2002*, pages 703–712, 2002.

H. Ray, H. Pfister, D. Silver, and T. Cook. Ray Casting Architectures for Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), March 1999.

H. Ray and D. Silver. A Memory Efficient Architecture for Real-Time Parallel and Perspective Direct Volume Rendering. Technical Report CAIP-TR-237, Rutgers-State University of New Jersey, 1999.

C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In *Proceedings of Graphics Hardware 2000*, pages 109–118, 2000a.

C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization. In *Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 2000b.

C. Rezk-Salama, P. Hastreiter, K. Eberhardt, B. Tomandl, and T. Ertl. Interactive Direct Volume Rendering of Dural Arteriovenous Fistulae in MR-CISS Data. In *Proc. Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 1999a.

C. Rezk-Salama, P. Hastreiter, G. Greiner, and T. Ertl. Non-linear registration of pre- and intraoperative volume data based on piecewise linear transformations. In *Proc. Vision, Modelling, and Visualization (VMV)*, 1999b.

C. Rezk-Salama, P. Hastreiter, J. Scherer, and G. Greiner. Automatic adjustment of transfer functions for 3D volume visualization. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2000*, pages 357–364, 2000c.

C. Rezk-Salama, P. Hastreiter, C. Teitzel, and T. Ertl. Interactive exploration of volume line integral convolution based on 3d–texture mapping. In *Proc. IEEE Visualization*, 1999c.

J. Rodenwaldt, L. Kopka, R. Roedel, A. Margas, and E. Grabbe. 3d virtual endoscopy of the upper airways: Optimization of the scan parameters in a cadaver phantom and clinical assessment. In *Journal of Computer Assisted Tomography*, pages 21(3):405–411, 1997.

P. Rogalla. Virtual endoscopy: An application snapshot. In *Medica Mundi*, pages 43(1):17–23, 1999.

P. Rogalla, A. Nischwitz, A. Heitema, R. Kaschke, and B. Hamm. Virtual endoscopy of the nose and the paranasal sinus. In *European Radiology*, pages 16:787–789, 1998.

P. Rogalla, J. T. van Scheltinga, and B. Hamm. Virtual endoscopy and related 3d techniques. In *Springer-Verlag, Heidelberg*, 2000.

R. Rost. *OpenGL Shading Language*. Addison Wesley, 2004.

S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of VisSym 2003*, pages 231–238, 2003.

S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of IEEE Visualization 2000*, pages 109–116, 2000.

G. Rubin, C. Beaulieu, V. Argiro, H. Ringl, A. Norbash, J. Feller, M. Dake, R. Jeffrey, and S. Napel. Perspective volume rendering of ct and mr images: Application for endoscopic imaging. In *Radiology, volume 199*, pages 321–330, 1996.

P. Sabella. A Rendering Algorithm for Visualizing 3D Scalar Fields. In *Proc. SIGGRAPH*, 1988.

T. Saito and J. Toriwaki. New algorithms for euclidian distance transformation of an n-dimensional digitized picture with applications. In *Pattern Recognition*, pages 27(11):1551–1565, 1994.

Y. Sato, C.-F. Westin, and A. Bhalerao. Tissue Classification Based On 3D Local Intensity Structures for Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 6, April 2000a.

Y. Sato, C.-F. Westin, A. Bhalerao, S. Nakajima, N. Shiraga, S. Tamura, and R. Kikinis. Tissue classification based on 3D local intensity structures for volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 6 (2):160–180, 2000b.

J. Schimpf. 3Dlabs OpenGL 2.0 white papers. http://www.3dlabs.com/support/developer/ogl2/.

J. Schneider and R. Westermann. Compression domain volume rendering. In *Proceedings of IEEE Visualization 2003*, pages 293–300, 2003.

W. Schroeder, K. Martin, and B. Lorensen. The visualization toolkit. In *Prentice Hall, Upper Saddle River, NJ, 2nd edition*, 1998.

I. Serlie, F. Vos, R. van Gelder, J. Stoker, R. Truyen, F. Gerritsen, Y. Nio, and F. Post. Improved visualization in virtual colonoscopy using image-based rendering. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 137–146, 2001.

SGI. SGI web page. http://www.sgi.com/.

R. Shadidi, V. Argiro, S. Napel, L. Gray, H. McAdams, G. Rubin, C. Beaulieu, R. Jeffrey, and A. Johnson. Assessment of several virtual endoscopy techniques using computed tomography and perspective volume rendering. In *Proc. of Visualization in Biomedical Computing, volume LNCS 1131*, pages 521–528, 1996.

C. Sigg, M. Hadwiger, M. Gross, and K. Bühler. Real-time high-quality rendering of isosurfaces. Technical Report TR-VRVis-2004-015, VRVis Research Center, 2004.

C. Stein, B. Becker, and N. Max. Sorting and hardware assisted rendering for volume visualization. In *Proceedings of IEEE Symposium on Volume Visualization*, 1994.

R. Summers, D. Feng, S. Holland, M. Sneller, , and J. Shelhamer. Virtual bronchoscopy: Segmentation method for real-time display. In *Radiology*, pages 200:857–862, 1996.

J. Sweeney and K. Müller. Shear-warp deluxe: The shear-warp algorithm revisited. In *Data Visualization (Proc. of Symposium on Visualization)*, pages 95–104, 2002.

S. Tenginakai, J. Lee, and R. Machiraju. Iso-surface detection with model-independent statistical signatures. In *Proceedings of IEEE Visualization 2001*, pages 231–238, 2001.

T. Theußl. *Sampling and Reconstruction in Volume Visualization.* Diplomarbeit, Vienna University of Technology, 1999.

T. Theußl, H. Hauser, and E. Gröller. Mastering windows: Improving reconstruction. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 101–108, 2000.

U. Tiede, T. Schiemann, and K.-H. Höhne. High quality rendering of attributed volume data. In *Proceedings of IEEE Visualization '98*, pages 255–262, 1998.

B. Tomandl, P. Hastreiter, K. Eberhardt, H. Greess, U. Nissen, C. Rezk-Salama, and W. Huk. Virtual labyrinthoscopy: Visualization of the inner ear with interactive direct volume rendering. *Radiographics*, 20(2), March 2000.

K. Turkowski. Filters for common resampling tasks. In A. Glassner, editor, *Graphics Gems I*, pages 147–165. Academic Press, 1990.

K. Udupa and G. Herman. *3D Imaging in Medicine*. CRC Press, 1999. ISBN 084933179X.

C. Upson and M. Keeler. V-BUFFER: Visible Volume Rendering. In *Proc. SIGGRAPH*, 1988.

A. Van Gelder and K. Kim. Direct volume rendering with shading via three-dimensional textures. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 23–ff., 1996.

A. Vilanova, R. Wegenkittl, A. König, and E. Gröller. Mastering perspective projection through parallelly projected slabs for virtual endoscopy. Technical Report TR-186-2-00-12, Institute of Computer Graphics, Vienna University of Technology, 2000.

D. Vining, R. Shifrin, E. Grishaw, K. Liu, and R. Choplin. Virtual colonoscopy (abstract). In *Radiology, volume 193(P)*, page 446, 1994a.

D. Vining, R. Shifrin, E. Haponik, K. Liu, and R. Choplin. Virtual bronchoscopy (abstract). In *Radiology, volume 193(P)*, page 261, 1994b.

D. Vining, D. Stelts, D. Ahn, P. Hemler, Y. Ge, G. Hunt, C. Siege, D. McCorquodale, M. Sarojak, and G. Ferretti. Freeflight: A virtual endoscopy system. In *First Joint Conference, Computer Vision, Virtual Reality and Robotics in Medicine and Medical Robotics and Computer-Assisted Surgery, volume LNCS 1205*, pages 413–416, 1997.

I. Viola. *Applications of Hardware-Accelerated Filtering in Computer Graphics*. Diplomarbeit, Vienna University of Technology, 2002.

M. Wan, F. Dachille, and A. Kaufman. Distance field based skeletons for virtual navigation. In *Proc. of IEEE Visualization*, pages 239–246, 2001.

M. Wan, A. Kaufman, and S. Bryson. High-performance presence-accelerated ray casting. In *Proc. of IEEE Visualization*, pages 379–388, 1999.

R. Wegenkittl, A. V. Bartrolí, B. Hegedüs, D. Wagner, M. Freund, and E. Gröller. Mastering interactive virtual bronchioscopy on a low-end pc. In *Proc. of IEEE Visualization*, pages 461–465, 2000.

M. Weiler and T. Ertl. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In *Proceedings of IEEE Visualization 2001*, pages 199–206, 2001.

M. Weiler, M. Kraus, M. Merz, and T. Ertl. Hardware-based ray casting for tetrahedral meshes. In *Proceedings of IEEE Visualization 2003*, pages 333–340, 2003.

M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, and T. Ertl. Level-of-detail volume rendering via 3D textures. In *Proceedings of IEEE VolVis 2000*, pages 7–13, 2000.

D. Weiskopf, K. Engel, and T. Ertl. Volume clipping via per-fragment operations in texture-based volume visualization. In *Proceedings of IEEE Visualization 2002*, pages 93–100, 2002.

D. Weiskopf, K. Engel, and T. Ertl. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):298–312, 2003.

R. Westermann. The rendering of unstructured grids revisited. In *Proceedings of Joint Eurographics/IEEE TCVG Symposium on Visualization 2001*, pages 65–74, 2001.

R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proceedings of SIGGRAPH '98*, pages 169–178, 1998.

R. Westermann and B. Sevenich. Accelerated volume ray-casting using texture mapping. In *Proceedings of IEEE Visualization 2001*, pages 271–278, 2001.

L. Westover. Interactive Volume Rendering. In *Chapel Hill Volume Visualization Workshop*, 1989.

L. Westover. Footprint Evaluation for Volume Rendering. In *Proc. SIGGRAPH*, 1990.

L. Westover. *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, UNC Chapel Hill, 1991.

Wildcat. Wildcat Graphics web page. http://www.intense3d.com/.

J. Wilhelms and V. G. A. A Coherent Projection Approach for Direct Volume Rendering. In *Proc. SIGGRAPH*, 1991.

O. Wilson, A. V. Gelder, and J. Wilhelms. Direct Volume Rendering via 3D-textures. Technical Report UCSC-CRL-94-19, Univ. of California, Santa Cruz, 1994.

C. Wittenbrink, T. Malzbender, and M. Goss. Opacity-weighted color interpolation for volume sampling. In *Proceedings of IEEE Symposium on Volume Visualization*, pages 135–142, 1998a.

C. Wittenbrink, M. T., and G. M. Opacity-Weighted Color Interpolation for Volume Rendering. In *IEEE Symposium on Volume Visualization*, 1998b.

R. Yagel, D. Stredney, G.Wiet, P. Schmalbrock, L. Rosenberg, D. Sessanna, and Y. Kurzion. Building a virtual environment for endoscopic sinus surgery simulation. In *Computers & Graphics*, pages 20(6):813–823, 1996.

S. You, L. Hong, M. Wan, K. Junyapreasert, A. Kaufman, S. Muraki, Y. Zhou, M. Wax, and Z. Liang. Interactive volume rendering for virtual colonoscopy. In *Proc. of IEEE Visualization*, pages 343–346, 1997.

K. Zuiderveld, A. Koning, and M. Viergever. Acceleration of ray-casting using 3D distance transforms. In *Visualization in Biomedical Computing*, pages 324–335, 1992.