UNIVERSITY OF CALGARY

Three-Dimensional Medical Image Visualization Techniques on Modern Graphics Processors

by

Eric Penner

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

December, 2009

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "Three-Dimensional Medical Image Visualization Techniques on Modern Graphics Processors" submitted by Eric Penner in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE.

Supervisor, Dr. Sheelagh Carpendale
Department of Computer Science

Supervisor, Dr. J. Ross Mitchell
Departments of Clinical
Neurosciences and Radiology

Dr. Jon Rokne
Department of Computer Science

Dr. Morley Hollenberg
Departments of Physiology and
Pharmacology, and Medicine

Date

Abstract

Three-dimensional medical image visualization is an important investigative and diagnostic tool in medicine. In addition to being a powerful diagnostic tool, it already plays a crucial role in several procedures including surgical simulation, image guided surgery and virtual endoscopy.

The enormous computations involved in interactively visualizing a medical dataset have always been a barrier to its full utilization in clinical practice. Thankfully, recent advancements in commodity programmable graphics hardware allow for not only interactive visualization, but superior image quality and shading options that are beginning to rival custom medical workstations and even sophisticated offline software techniques.

This thesis presents several new techniques that leverage the new features in modern commodity graphics hardware to improve the performance, quality and realism of real-time medical image visualization.

Acknowledgements

It is a pleasure to thank the many people who made this thesis possible.

It is difficult to overstate my gratitude to my supervisors, Dr. J. Ross Mitchell and Dr. Sheelagh Carpendale. Their enthusiasm, inspiration, understanding and support helped make this degree a very enjoyable experience. Throughout my research and thesis-writing they provided encouragement, sound advice, good teaching, good company, and lots of good ideas. I would have been lost without them.

I would also like to thank all my professors from whom I learned a great deal: Especially Dr. Faramarz Samavati and Dr. Richard Frayne, and Dr. Jim Little and Dr. Michiel van de Panne from the University of British Columbia.

I am indebted to my many student colleagues for providing a stimulating and fun environment in which to learn and grow. I am especially grateful to Daniel Adler, Sonny Chan, Thor Bjarnason, Sylvia Drabycz, Maxwell Sayles, Luke Olson and John Brosz.

I am very grateful to all of the administrative and technical support staff for all their support. Mark Simpson, Susan Lucas, Beverley Shevchenko and Lorraine Storey deserve special mention.

Lastly, and most importantly, I wish to thank my family for all their support and encouragement to follow my heart wherever it takes me.

To Yuko,
my love and
inspiration.

Table of Contents

List of Figures

Chapter 1

Introduction

In this thesis we present new methods that leverage the recent advances in the power, programmability and data capacity of commodity graphics processors to greatly accelerate 3D medical visualization algorithms while simultaneously increasing diagnostic image quality and realism. In this chapter we introduce the reader to the medical imaging field, discuss how changes in computer hardware have had a large impact on medical visualization, and provide the motivation for our research. We then discuss the specific goals of this thesis and provide an overview of how the thesis is organized.

1.1 Medical Imaging

Medical imaging has become an integral part of modern medicine. In a broad sense, medical imaging refers to a set of techniques that non-invasively produce images of internal parts of the body. Understanding how the human body is built, how it functions and how it goes wrong has been greatly assisted by imaging technology that allows physicians to see beneath the skin without the need to cut it open. It is now used not only in diagnostic procedures throughout the medical field, but also in the planning and evaluation of surgery and radiotherapy.

Medical imaging relies not only on the physics of the interaction between energy and matter but also on the technology used to acquire, process and display huge quantities of biological data. Not that long ago, medical imaging referred primarily to x-rays which were transmitted through the body and processed on film much like a photograph. In the last three decades however, medical imaging has advanced rapidly, assisted by huge advances in digital technology which continues to become less expensive, more compact and computationally faster.

Over the last three decades, imaging modalities such as computed tomography (CT), magnetic resonance (MR), positron emission tomography (PET) and ultrasound have been developed to acquire not only projective images but also cross-sectional images of the human body. Unlike projective images which project an entire 3D volume into one 2D image, a cross-sectional image obtains an *unoccluded* view of one 2D slice of the body. Furthermore, by acquiring many 2D cross sections, full 3D images can now be routinely acquired in all the imaging modalities mentioned above.

A subset of medical imaging which seeks to make use of the vast quantity of 3D data is 3D medical visualization. Medical visualization is the process of generating images from raw medical data to gain insight into its qualitative and quantitative features. While clinicians can gain some insight from visualizing 3D data as a series of 2D slices, a 3D reconstruction of the data provides a much richer and highly comprehensive view of the structures contained in the image, due to our implicit ability to perceive 3D shapes (see Figure 1.1).

1.2 Motivation for Real-time Visualization

The largest hindrance of 3D medical visualization tools in the past has been their practicality. Due to the vast computational requirements of processing a 3D data-set, only one image could be generated every few seconds or minutes using offline rendering algorithms. Creating just one illustrative image often requires iterating between adjusting complex rendering parameters and navigating to an area of interest, resulting in a long, complicated process. This lengthy process often outweighed its usefulness, especially with many patients awaiting timely diagnosis or treatment. The accuracy of the render-

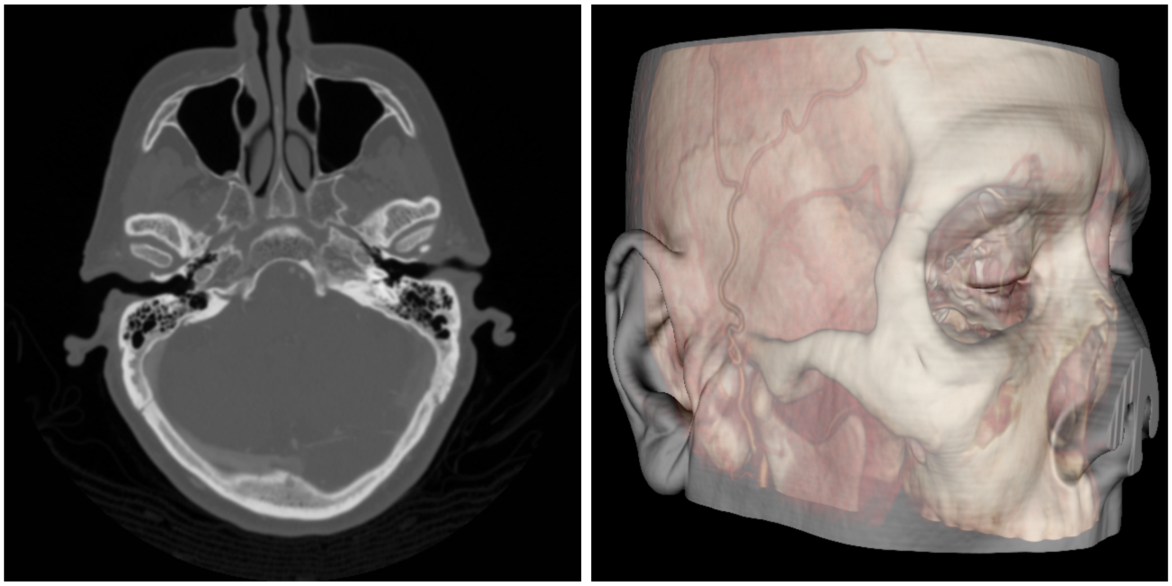


Figure 1.1: Left: A 2D slice from a CT scan. Right: The same scan rendered using our software.

ing could often be reduced in many ways to allow for increased responsiveness, but this decreased accuracy could lead to misdiagnosis or other errors.

In order to improve the interactivity of 3D medical visualization, expensive custom workstations have been designed with dedicated hardware that allow interactive real-time 3D visualization. While these workstations were the first plausible way to enable many new techniques such as surgical simulation, image guided surgery and virtual endoscopy, they come at a great cost to hospitals and are extremely inflexible compared to standard desktop software.

In the last few years, a major shift has occurred in consumer processor design. Processors have reached practical limits in clock speed, and have moved towards parallel multi-core processors to realize further speed gains. This shift has been fueled primarily by graphics processors which were initially designed to solve the "embarrassingly parallel" task of rendering millions of polygons in video games. As graphics processors became more flexible it became apparent that they could be used not only to render polygons, but also to perform a variety of “data-parallel” processing tasks, and that the algorithms designed for such an architecture can serve a variety of high-performance computing tasks.

1.3 Thesis Goals

The goal of this thesis is to demonstrate the potential of using modern commodity graphics hardware to greatly reduce the execution time and increase the quality of 3D medical visualization. We present new methods that leverage the recent advances in the power, programmability and data capacity of commodity graphics processing units to greatly accelerate 3D medical visualization algorithms while simultaneously increasing diagnostic image quality and realism.

There are four interrelated constraints that must be considered when designing a medical visualization application:

1. Diagnostic Quality - Volume rendering artifacts can not obscure important details
2. Interactivity - High frame rates must be maintained with minimal down-sampling
3. Realism - Accurate lighting can provide extra details and depth cues
4. Architecture - Architectural support for new features, multiple operating systems and multiple rendering API’s

1.3.1 Diagnostic Quality

Three dimensional medical image visualization usually involves editing an *iso-surface* or a *transfer function*. An iso-surface specifies a surface of interest corresponding to a scalar value in the data, whereas a transfer function maps all scalar values returned by the scanner into optical properties such as color and opacity. By editing the iso-surface or transfer function, different anatomical structures can be visualized and their spatial relationships examined without having to continually extract coarse polygonal surfaces.

The amount of information in a CT image is directly related to the radiation dose administered to acquire the image. If a visualization tool obscures or otherwise does not display all of the information in the scan, or obscures it with rendering artifacts, then the amount of radiation administered could be reduced simply by increasing the visualization quality of a lower-resolution scan. An artifact that obscures details of even two voxels in diameter effectively means that only one eighth of the radiation administered was utilized in the visualization. Thus, reduction of visual artifacts is crucial before a rendering system can be used in medical settings.

Unfortunately, volume rendering has a history of being very prone to distracting rendering artifacts, often termed “wood-grain” artifacts. These artifacts arise from three separate causes: Precision Artifacts, Interpolation Artifacts and Under-Sampling Artifacts. This thesis presents a volume rendering pipeline that runs on commodity graphics processing units (GPUs) that strongly reduces these distracting artifacts while maintaining high frame rates.

Precision artifacts usually arise due to the use of lower precision integer calculations which are used in place of floating point calculations to increase speed. Since modern graphics processors now support 32-bit floating point calculations throughout the entire pipeline, they are an excellent candidate to reduce this source of artifacts.

Interpolation artifacts occur when a sample is needed from the volume that falls in between voxel centers. Tri-linear interpolation, which is used almost exclusively in volume rendering, will interpret the value to be the weighted average of the 8 neighboring voxels. Tri-Cubic interpolation, which uses a weighted 4x4x4 grid of voxels, is almost universally preferred but almost never used since this is very computationally expensive. In this thesis we apply a method that uses the GPU’s built in interpolation hardware to build higher order interpolation kernels that are much less expensive and thus feasible to use in a real-time application. The result is the reduction of interpolation artifacts and a better ability to detect subtle structural details.

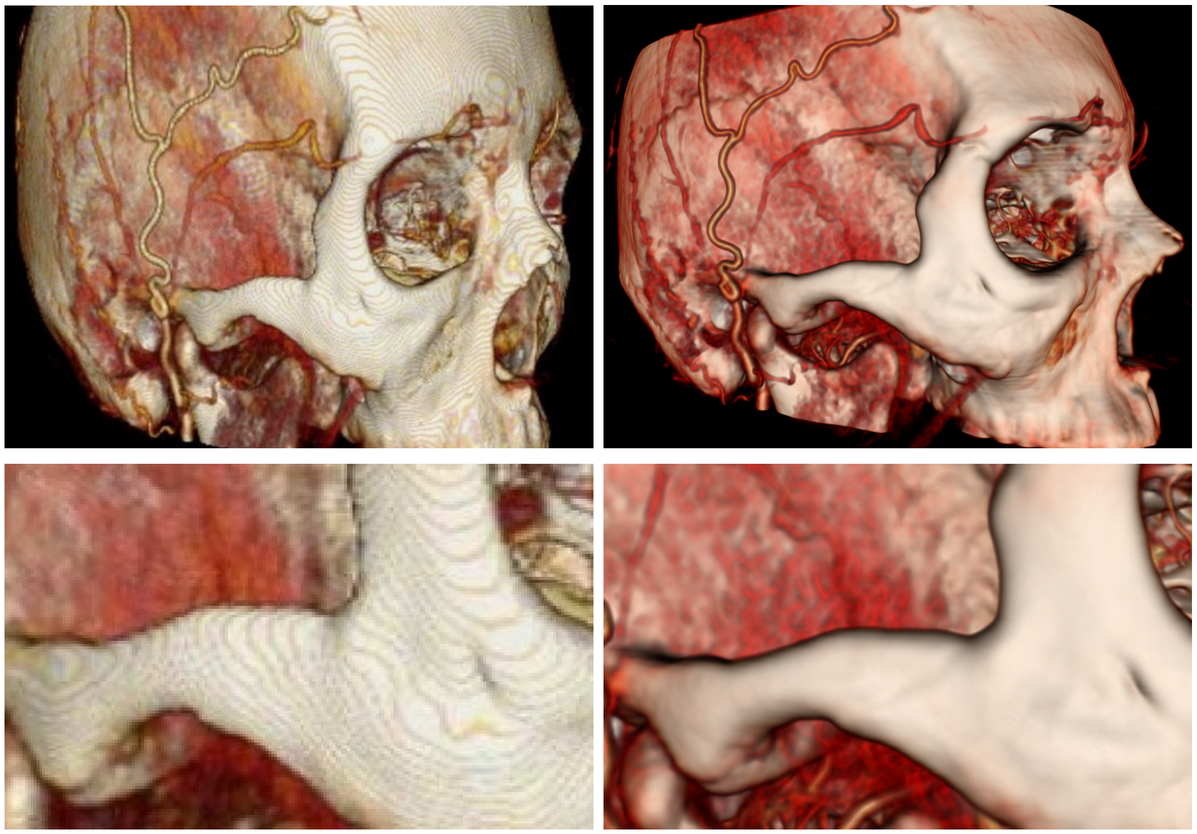


Figure 1.2: Head-to-head comparison of renderings with and without artifacts. Left: Image generated with the common open source software package VTK. Right: An image produced by our renderer using similar rendering parameters.

Finally, under-sampling artifacts are the biggest source of wood-grain artifacts in volume rendering. While an adequate sampling rate is usually dictated by the highest frequency of a signal, volume rendering becomes more complicated due to the addition of the transfer function that is applied during volume sampling. The two step sampling process has the effect of multiplying the frequency of the transfer function and frequency of the volume data, making adequate sampling very difficult to achieve. We have come up with a new pre-integrated sampling algorithm that uses a native GPU data-structure called a mip-map to alleviate transfer function under-sampling. Compared to other approaches that use large lookup tables or extra samples, our approach uses neither extra memory nor processing overhead.

1.3.2 Interactivity

Achieving interactive frame rates while maintaining high quality is one of the most challenging tasks in volume rendering. An interactive system can provide immediate feedback to the user which allows for a much better understanding of the spatial relationships of features of interest. It also becomes possible to fine tune the transfer function and visualization in ways which are simply not possible with off-line rendering techniques. This thesis presents a new method for accelerating volume rendering on the GPU which traces large groups of rays in parallel to skip empty space. This results in an order of magnitude speed up for most medical datasets, and performs much better than other approaches in difficult situations such as when skipping internal empty space.

1.3.3 Realism

The strongest influence on our perception of natural objects is the interaction of light with an object. Lighting variations can reveal subtle contours while complex shadows can reveal the distances between objects. The illumination models most often used for volume rendering are only capable of representing local illumination phenomena. However, it has been demonstrated [43, 47, 105] that subtle lighting effects from neighboring structures support spatial comprehension and even improve perception of shapes compared to direct lighting. A sophisticated illumination model should therefore include not only local illumination effects but also include global effects that depend on neighboring structures such as complex soft shadows. This thesis presents a new method for pre-computing the global illumination effect known as ambient occlusion. We first show how this complex interaction can be pre-computed for volume datasets, and then use the discrete nature of a volume dataset to take the pre-computation time from days using monte-carlo ray-tracing down to seconds using a statistical approximation.

1.3.4 Architecture

Finally, GPU volume rendering systems suffer from many difficult architectural challenges. The GPU must be programmed separately from the CPU using shader programs that aren't intended for such a complex task. As shader complexity grows, a volume rendering pipeline becomes more and more difficult to maintain. Simple design strategies such as code reuse and object oriented design are difficult to apply when developing a large library of GPU programs. Different rendering API's such as Direct3D and OpenGL also offer different feature sets and depending on the year released one will often offer a better feature set than the other. The work presented in this thesis leverages a custom written API abstraction library that allows for cross platform development on any graphics API and across operating systems. It also leverages a shader management system that allows for code reuse and object oriented design of the volume rendering pipeline.

1.4 Overview

Chapter 2 provides a more in depth overview of medical image processing, analysis and acquisition methods that are relevant to this thesis. Chapter 3 introduces modern graphics processors and the relatively new field of general purpose computing on graphics processors. Since this thesis focuses on a few separate areas within volume rendering, general related research is presented first in Chapter 4, while specific research related to our two contribution areas are presented together with our contributions in chapters 5 and 6. Chapter 5 discusses the importance of optimizing volume rendering and our hierarchical frustum-casting technique for accelerating GPU ray-casting on the GPU. Chapter 6 discusses the importance of shadows and realistic lighting techniques such as ambient occlusion. We then present our technique which is a new algorithm to approximate ambient occlusion and soft shadows very efficiently on the GPU. Discussion of our contributions and future work is also presented together with our contributions in chapters 5 and 6, but we provide a brief review and reach conclusions in chapter 7. Our publication to Volume Graphics 2008, which was the basis for chapter 6, is provided in Appendix A.

Chapter 2

Medical Imaging

This chapter provides a more in depth overview of medical image processing, analysis and acquisition methods that are relevant to this thesis. This chapter is not intended to provide a review of the state of the art, but rather to put our research in context given the highly interdisciplinary nature of our lab, and to provide a broad overview of the entire medical imaging field for the interested reader without a medical imaging background. Medical imaging in general is first introduced, followed by a description of the current medical imaging modalities. An overview is then given of the three pillars of medical imaging - registration, segmentation and visualization - and the importance of real-time applications to solve these problems in medical practice.

2.1 Introduction to Medical Imaging

Medical imaging is most often perceived to designate the set of techniques that allow clinicians to non invasively 'see inside' the human body. The first and most well know medical imaging technique, the planar x-ray, has largely served this purpose for just over a century. In the last few decades, however, there has been an explosion of new and exciting imaging techniques that can provide an array of vital information about the structural, chemical and electrical properties of the human body.

Technically speaking, medical imaging techniques can be seen as solutions of mathematical inverse problems. This simply means that the cause (properties of living tissue) must be inferred from the effect (the observed measurements from the imaging device). Problems of this nature often have many valid solutions, meaning that while they may provide crucial information, there will always be certain level of uncertainty involved, thus it is often argued that a human interpretation of the results will always be necessary.

While the body is inherently 3D, the images captured by x-rays have always been 2D, resulting in ambiguity. The solution to this problem came in the late 70's and early 80's when advances in engineering and computing brought new 3D imaging technologies such as computed tomography (CT), magnetic resonance imaging (MRI), positron emission tomography (PET) and 3D ultrasound.

While the technology behind each of the above approaches is different, what they have in common is the ability to capture *cross-sectional* images, rather than *planar-projections*. While a cross-section is also 2D, each pixel - or *picture element* - in the image represents a single position in the body, rather than an entire line through the body like in a projective image. Thus, by capturing many 2D cross-sections, an entire 3D volume can be captured. The pixels from each slice become voxels - or *volume elements* - of the resulting volume image, and have a one-to-one relationship with locations in the body.

While new imaging technologies can provide vast quantities of new information about a patient, interpreting this information is not as straight forward as it might seem. Each imaging modality takes advantage of different physical properties to create an image, and in its raw form this data may be hard to decipher. In the case of ultrasound, the image consists of echoing sonic pressure waves, while in projection radiography (x-ray), the measurement is the strength of attenuated x-rays having travelled through materials with different atomic densities. In each case, what is recorded by the imaging device and what is required by the clinician are not completely aligned.

Computer software can greatly assist in transforming the data from its raw form into a form that is more useful to a human observer, thus aligning disparate requirements of physicians with the product of imaging acquisition methods. Due to this, physical image acquisition and computer-based image processing are often intricately linked under the umbrella of medical imaging.

2.2 Imaging Modalities

In medical imaging, any of the various types of equipment or probes used to acquire images of the body, such as radiography, ultrasound and magnetic resonance imaging are called modalities. Each modality is based on different physical phenomena and thus capture different types of information. The information captured, while physically mutually exclusive, may or may not provide complementary information about the patient. Thus while one modality usually suffices to diagnose a given problem, it is not uncommon for multiple modalities to be used in a single case.

2.2.1 Computed Tomography

While the mathematical concepts and theories behind computed tomography (CT) date back to the late 1800s and early 1900s, the mechanical and computational knowledge required to build a CT device postponed its invention. The first commercially viable CT scanner was invented by Sir Godfrey Hounsfield in 1967.

The primary concept behind CT is the Radon transform, which describes how an image can be represented as a series of rotated projections through that image and vice-versa. The easiest way to visualize how a CT slice is reconstructed is through the convolution back projection algorithm (see Figure 2.1) which illustrates what the Radon transform describes mathematically.

Causes of artifacts in CT include the scatter, or complete absorption of x-rays. Since the Radon transform assumes rays travel in straight lines through the body, completely attenuation or scattering of rays can cause ‘streaks’ through the image during reconstruction. A common cause of these artifacts include amalgam fillings [97] (see Figure 2.2).

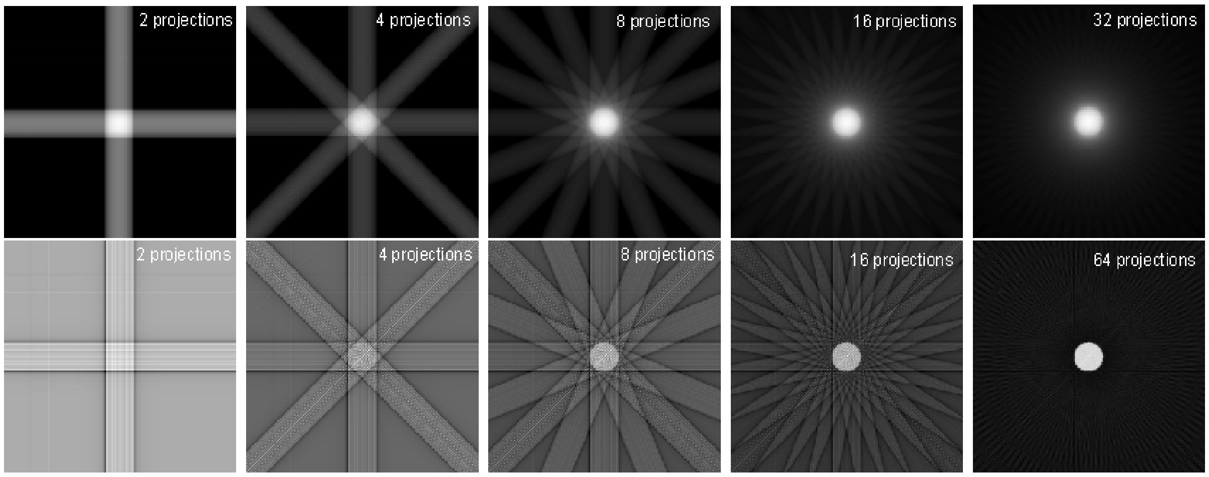


Figure 2.1: Illustration of the convolution back projection algorithm. Each projection of the object is projected back onto image at the angle with which it was acquired. As the projections are summed together the structure of the object is acquired (a white circle in this case). With a sufficient number of projections the entire slice can be reconstructed. Back projection alone produces a blurred image (top row). Thankfully this blur is uniform and thus sharpening filters can be used restore the original image (bottom row).

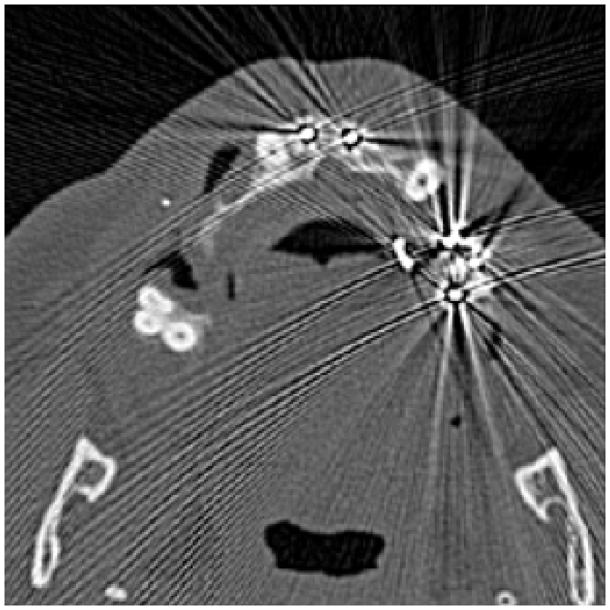


Figure 2.2: Amalgam fillings causing 'streak' artifacts in CT.

Advances in CT technology have focused on the often contradictory goals of decreasing acquisition time, increasing image resolution, and decreasing the dose of radiation to the patient. A modern CT scanner consists of an x-ray tube mounted on a gantry that revolves it around the patient (see figure 2.3). A modern CT scanner will cost between \$500,000 and \$2,000,000.

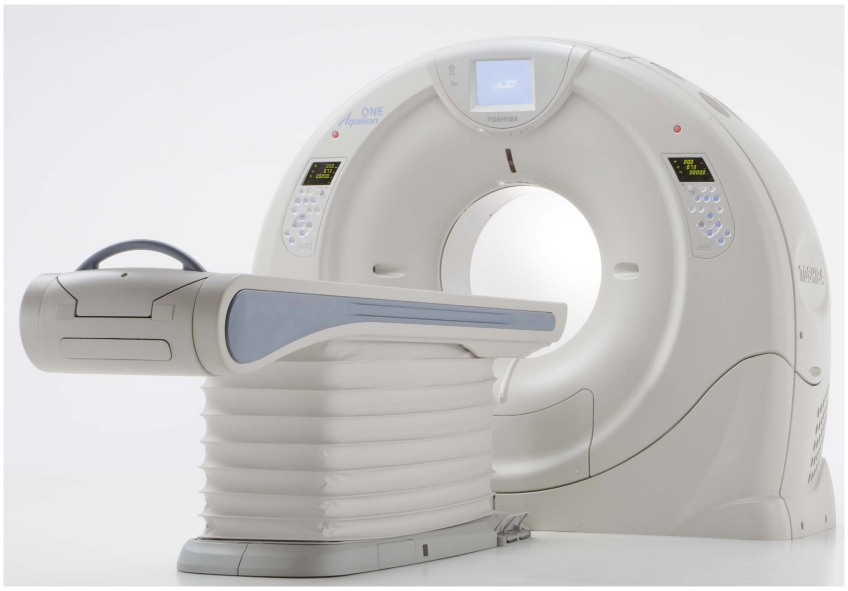


Figure 2.3: A modern CT scanner. The Aquilion One by Toshiba acquires 320 slices in one rotation around the patient, enabling imaging of an entire organ in fractions of a second.

While the first CT scanner used a single beam detector which needed to switch between translating back and forth and rotating in small increments (taking minutes per slice), later generations used multiple moving detectors and eventually full rings of stationary detectors to greatly acceleration the acquisition process and reduce the artifacts resulting from disparities between detectors.

Through the 1990s, scanners moved from stationary ring detectors to multi-slice detectors capable acquiring two, four and then 16 slices simultaneously. Increasing the number of slices acquired in one pass translated into shorter examination times. Reduced times allowed processing of images with fewer artifacts caused by movement. CT scanners have continued down this path until today, with the most recent being the 320 slice AquilionOne by Toshiba. These extremely high resolution scanners can capture an entire organ in very high detail with one rotation around the patient, taking only a fraction of a second per scan while reducing radiation dose drastically. All the while, vast improvements in computer technology have reduced the computation time from 2.5 hours per slice down to near real-time reconstruction of an entire scan.

The result of the CT acquisition process is a set of images that represent cross-sections through the body. Similar to an X-ray, each voxel represents the material's opacity to X-rays. At the energy levels use for CT, a material's opacity to X-rays depends mostly on the density of electrons in the material. Unlike standard X-rays, which can only portray relative differences in attenuation, a CT computes a unique attenuation at every voxel. In order to standardize this measurement, the Houndsfield Unit (HU) was created. The houndsfield unit is calibrated by setting the attenuation of water to 0 and air to -1000.

Another advantage of computed tomography is its high numerical precision. While a normal grey-scale or colour image is represented with 8-bits of precision, allowing for 256 unique values, a standard CT will returns results with 12-bits of precision (usually stored in a 16-bit integer). This high precision can allow for detection of very subtle differences in attenuation. Since the entire range of CT values cannot be displayed on a standard monitor at the same time, a typical application will allow the clinician to 'window and level' the the display. The 'window' refers to the range of values displayed, and the 'level' refers to the midpoint of the range of values. Values outside of the current window are saturated to black or white.

2.2.2 Magnetic Resonance Imaging

Magnetic resonance imaging (MRI) is based on measuring the resonating frequency (or spin) of atoms under magnetic fields. Like computed tomography, MRI is able to capture

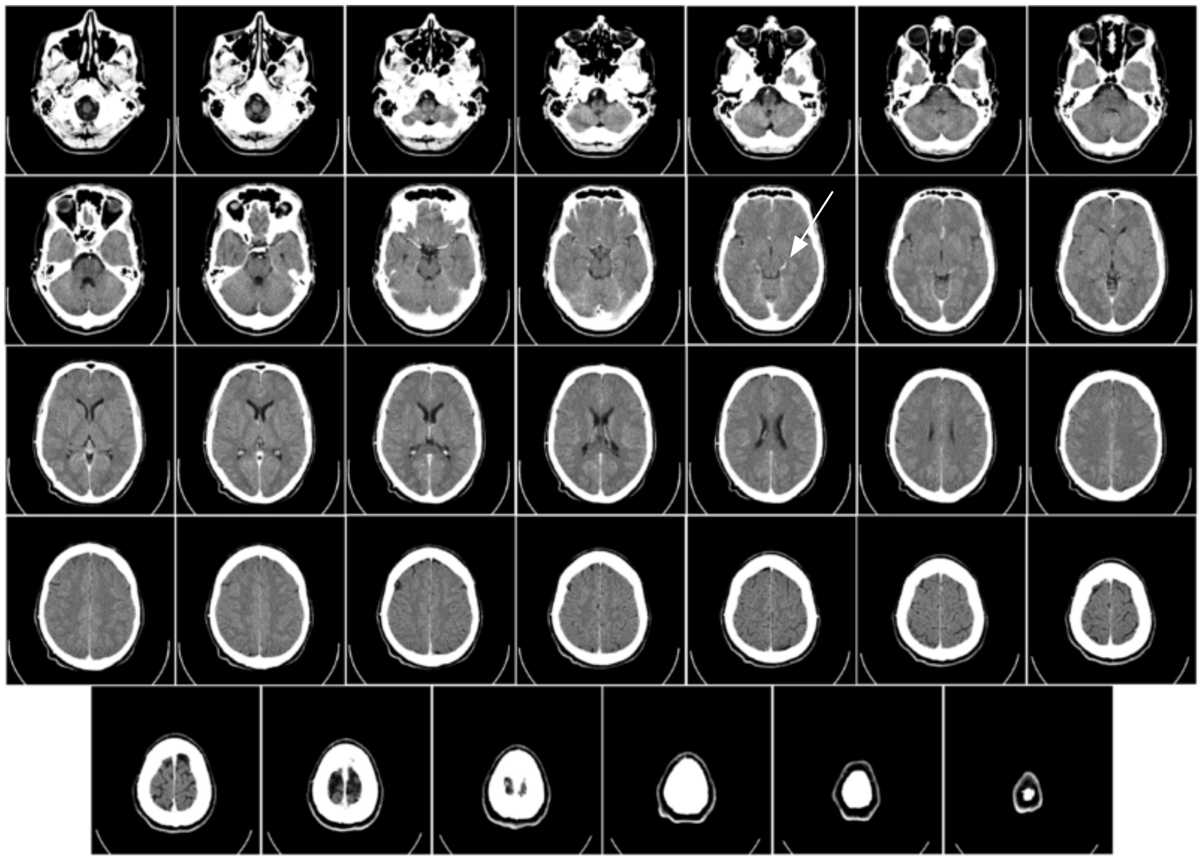


Figure 2.4: A CT angiogram of the brain. Each image corresponds to a cross section through the brain. The blood vessels are brighter due to an injected contrast agent. One enhanced blood vessel is denoted by the white arrow in the second row.

cross-sectional images of the body. Unlike computed tomography, it uses no ionizing radiation. Image contrast in MRI is determined due to a number of factors including the biochemical environment of water molecules, the movement and diffusion of fluid and the density of water molecules in tissue. Consequently, MRI provides much greater contrast between the soft tissues of the body than computed tomography. This makes MRI especially useful in neurological, oncological (cancer) and musculoskeletal imaging. Although MRI provides a lot of soft tissue contrast to the human observer, image intensity in MRI is very relative and can even vary for the same material within the same image. This makes it more difficult to make quantitative measurements based on image intensity.

A detailed discussion the physics of MRI acquisition is well beyond the scope of this

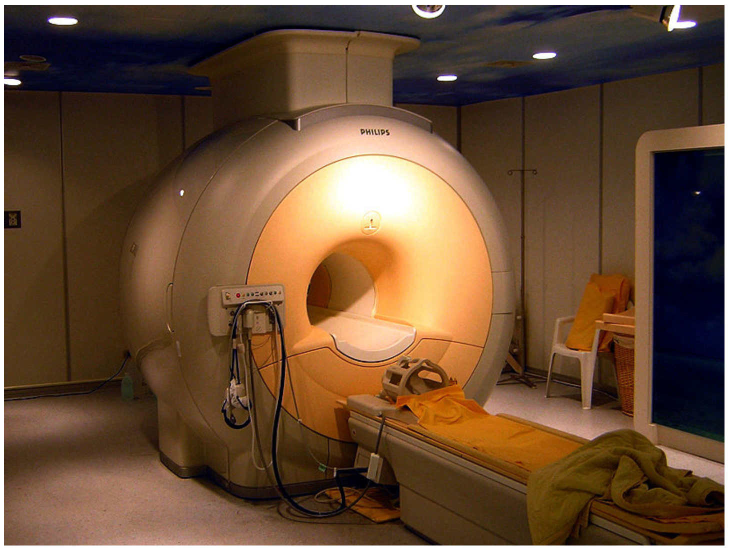


Figure 2.5: A Philips 3-Tesla MRI scanner

work, however the interested reader can find a very good introduction in [97] or [42]. Very generally, an MRI imaging device consists of a large magnet, radio transmitters which deliver pulses of radio waves (vibrating magnetic fields), a radio receiver antenna, and electronics that decode an RF signal (see figure 2.5). The basic principle behind all magnetic resonance imaging is a property that nuclei have in a magnetic field. Electromagnetic pulses can cause the nuclei to absorb energy from the pulse and then radiate this energy back out. The key to MRI is the resonance equation, which shows that the resonance frequency is proportional to the magnetic field it is experiencing [42]. By applying a magnetic field gradient, whereby the intensity of the magnetic field drops off linearly along a certain axis, the resonance frequency of atoms will depend on their position along the axis.

As a very simple example, if we picture two physically separate tissues that experience the same magnetic field, we would only see one “peak” in the magnetic resonance spectrum. By applying a magnetic gradient such that the two tissues had different magnetic field strengths, each tissue would resonate at a different frequency and we could identity which parts of the signal come from which tissue. This is an over simplification

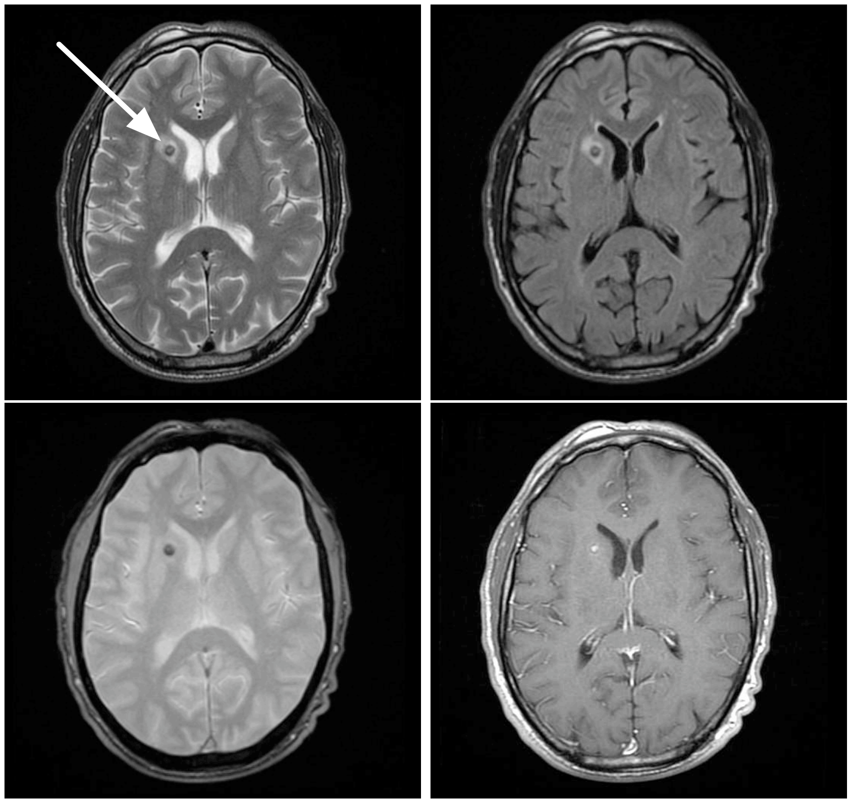


Figure 2.6: Comparison of different types of MRI contrast of a patient with a small brain lesion (indicated by white arrow). In a T2-weighted scan (top left), water and fluid-containing tissues are bright. In T2-FLAIR (top right) free water such as the cerebrospinal fluid is made dark. T2* (bottom left) enhances contrast for certain tissues such as venous blood. In T1 with a gadolinium injection (bottom right), the contrast of white and dark matter are reversed, but blood vessels are enhanced by the gadolinium. The lesion is also enhanced by the gadolinium in this case.

however, and real MRI acquisition involves complex applications of carefully chosen radio frequency pulses, magnetic gradient fields, and radio frequency measurements.

There are two primary characteristics of the response signal which are typically called *T1* and *T2*. Both T1 and T2 refer to exponential decay times of atomic spins back to a steady state. The interested reader can refer to [42, 97] for more detail. These two characteristics together with a large variety of pulse sequences gives MRI great versatility. Depending on the pulse sequence and T1 or T2 weighting different types of tissue will be emphasized. For example white matter will be brighter than grey matter in a T1

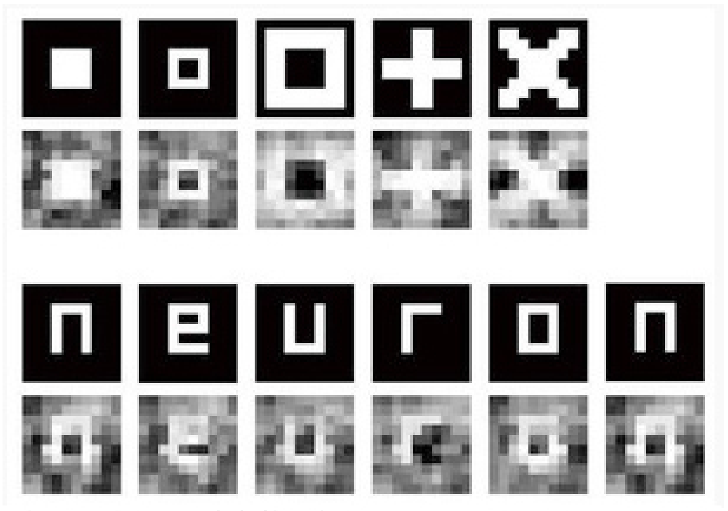


Figure 2.7: Very coarse images reconstructed from the brain using fMRI while being shown the corresponding letters [74]. The researchers used learned correlations between activated brain regions while showing the patient a set of random training images.

weighted image, but this is reversed for T2 (see Figure 2.6).

Like in computed tomography, a contrast agent can be injected to perform MRI angiography. By lowering the resolution and performing many scans over time, varying rates of oxidization can be detected across the brain, allowing for functional measurements of brain activity (fMRI). Creating imaging methods using MRI is still a very active area of research. Very recently, researchers have even managed to reconstruct rudimentary images of what a patient is looking at. After being shown a set of training images, the researchers were able to generate very low resolution images of letters the patient was shown purely from analyzing the patient's brain using custom MRI pulse sequences and machine learning [74] (see Figure 2.7).

While an MRI scan is free of ionizing radiation, making it theoretically harmless to the patient, MRI scanners still have safety concerns. Due to the extremely strong magnetic field required for and MRI (often up to 60,000 times the earth's own magnetic field), there are several safety issues that must be addressed in MRI facilities. Missile-effect accidents, where ferromagnetic objects are attracted to the center of the magnet, have resulted several injuries and even deaths. In order to help reduce the risks of projectile accidents, ferromagnetic objects and devices are typically prohibited in proximity to the MRI scanner, with non-ferromagnetic versions of many tools and devices are typically retained by the scanning facility. Patients undergoing MRI examinations are required to remove all metallic objects, often by changing into a gown or scrubs. The magnetic field and the associated risk of missile-effect accidents remains a permanent hazard, as superconductive MRI magnets retain their magnetic field, even when not in use. In fact, starting up a super conducting magnet after it has been shut down or *quenched* costs tens of thousands of dollars since they require liquid helium to cool the magnet back to superconducting temperatures.

2.2.3 PET and SPECT

Both positron emission tomography (PET) and single photon emission computed tomography (SPECT) both involve the injection of special compounds into the bloodstream. These compounds are known as *radiotracers*. Radiotracers are formed by incorporating a *radionuclide* into a normal compound such as glucose, water or ammonia, or into molecules that bind to sites of interest. Radionuclides are radioactive isotopes with short half lives such as carbon-11 (20 min), nitrogen-13 (10 min), oxygen-15 (2 min), and fluorine-18 (110 min). As this radioactive decay occurs, gamma rays and/or subatomic particles are emitted. This decay is then detected either directly (in SPECT) or indirectly (in PET). Due to the short half lives of most radioisotopes, the radiotracers must be produced using a cyclotron and radiochemistry laboratory that are in close proximity to the PET/SPECT imaging facility. However, the half life of fluorine-18 is long enough such that it can be still be manufactured at an offsite location. PET and SPECT technology can be used to trace the biologic pathway of any compound in living humans, provided it can be *radiolabeled* with a radionuclide. Thus the specific processes that can be probed with PET/SPECT are virtually limitless, and radiotracers for new target molecules and processes are being synthesized all the time [97].

To detect the emitted gamma rays, SPECT imaging utilizes what is called a *gamma camera* to detect gamma rays directly. A gamma camera is simply a lead block with very small parallel holes which permit gamma rays to travel only in one direction before hitting a sensitive film. Unlike x-ray imaging, where all x-rays originate from one location, gamma rays may originate anywhere in the body, which is why they must be focused in one direction by the gamma camera. This type of projection of rays is known as an orthographic projection. After the gamma camera is in place, 3D SPECT imaging can be performed in the same way as computed tomography. The gamma camera is simply rotated around the patient and a 3D tomography can be computed from the sequence of images.

PET captures special powerful gamma rays that are emitted in exact opposite directions. Positrons, which are emitted during radioactive decay, undergo a process known as *positron annihilation* when they encounter an electron. This annihilation results in two photons (gamma rays) being shot out in opposite directions. The amount of energy in these photons is dictated by Einstein's famous equation $e = mc^2$ [97] , and thus the PET detectors can easily determine which gamma rays were the result of annihilation. The PET imaging device captures these photons by a ring of detectors placed around the patient. The detectors use extremely accurate timing information to pair the photons and determine the *line of response*. By collecting large numbers of these lines, an image can be built similar to how computed tomography is computed. However, due to the limited number of events that are collected (in the tens of thousands, compared with billions in CT) PET typically produces a much lower resolution and less accurate image than CT or MRI.

2.3 Medical Image Processing and Analysis

As mentioned above, medical imaging and image processing an analysis have become intricately linked, to the point where it is difficult to discuss one without the other. Development of very sophisticated reconstruction algorithms were a prerequisite to developing almost all imaging techniques after the planar x-ray image. Today, medical imaging is undergoing a similar explosion of data as other IT industries. As the technology improves, imaging devices are creating exponentially more information, and as advanced imaging devices become inexpensive and more readily available, more and more patients are taking advantage of new imaging procedures.

This explosion of information is radically changing how physicians deal with analysis of these data. Where it used to be practical to analyze a 3D scan by inspection of all the 2D slices, it is now becoming common practice to analyze scans using many computer assisted processes. The most common questions that clinicians wish to gain answers to by looking at an image or set of images include:

- What is the extent and size of a particular structure or process?
- How are structures related in 3D space, including distances and relationships between neighboring structures?
- How does data from one image correlate to data in an other image, be it from the same patient/modality or a different patient/modality?

Attempts to address this set of questions has led to three interrelated groups of algorithms in medical image processing and analysis: segmentation, registration, and visualization, each of which we will discuss briefly.

2.3.1 Registration

Image registration is the process of transforming different sets of data into one coordinate system. Registration is necessary in order to be able to compare or integrate the data obtained from different measurements. In medical imaging this occurs when images are taken at different times, in different patients, or using different imaging modalities. After successful image registration, correlated anatomical locations from different images will be aligned.

The information gathered over time, from different patients, or different modalities are often complementary. For a single patient, a physician might want to align the functional information from a PET scan with the structural detail provided from a CT scan, or the structural changes in one CT scan to an earlier CT. For multiple patients, a physician might want to collect a statistical map of how often and where certain cancers are found in the brain. Image registration permits automatic or semi-automatic fusion of any number of scans into a common space, which is a much more powerful diagnostic tool than any single image alone.

Image registration in a single patient can sometimes be accomplished through the use of *fiducial markers* which are rigidly attached to the patient, or manually chosen anatomical markers (such as the tip of the nose, for example). In this case, registration becomes a fairly simple process of rigidly aligning a set of known point correspondences.

However, when fiducial markers are not present, registration becomes a much more algorithmically challenging task. Within a single modality, automated feature detection can be used followed by rigid or non-rigid alignment of matched features. In different modalities, however, features in one image may not map to the same feature in the other, making comparison even more complicated.

The gold standard for multi-modal alignment uses an error metric that measures the alignment of two images called mutual-information. Two overlapping images have higher

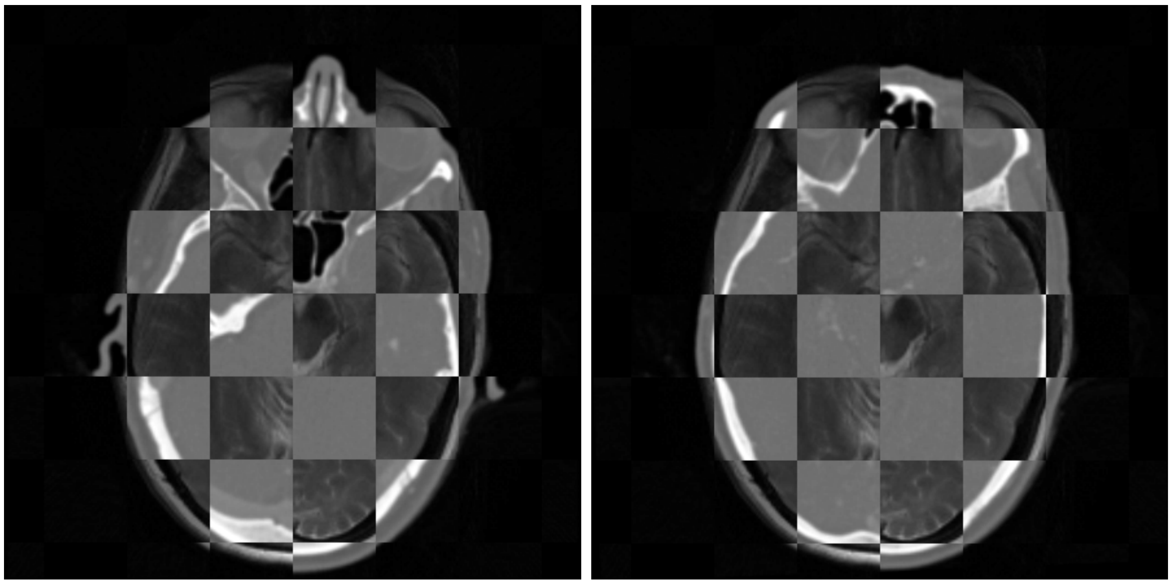


Figure 2.8: Illustration of registering two complementary modalities (CT and MRI). The checkerboard pattern is used to show the misalignment (left) and alignment after registration (right) [15].

mutual information when they are poorly aligned, as sets of image values are very poorly correlated in the two images. To align the images a numerical optimization algorithm is used that iteratively guesses a good transformation, evaluates an error metric (in this case mutual information) and guesses again using knowledge of the error from previous guesses. Assuming a good initial starting point, the algorithm will usually converge on a good alignment. Due to the uncertainty involved, a physician will usually be involved just to verify that the registration was successful.

2.3.2 Segmentation

In medical image processing, segmentation refers to the process of partitioning an image into multiple segments - sets of pixels/voxels. The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Segmentation of organs and/or tissues is required to determine important characteristics such as the size, shape or volume of an anatomical structure.

Image segmentation may be performed manually, automatically or semi-automatically. Manual image segmentation requires a knowledgeable physician to provide a contour for each slice of the image that contains the organ or tissue. In addition to being very time consuming, manual segmentations have a high degree of variability between physicians, thus a diagnosis based on size or volume measurements from these segmentations becomes problematic.

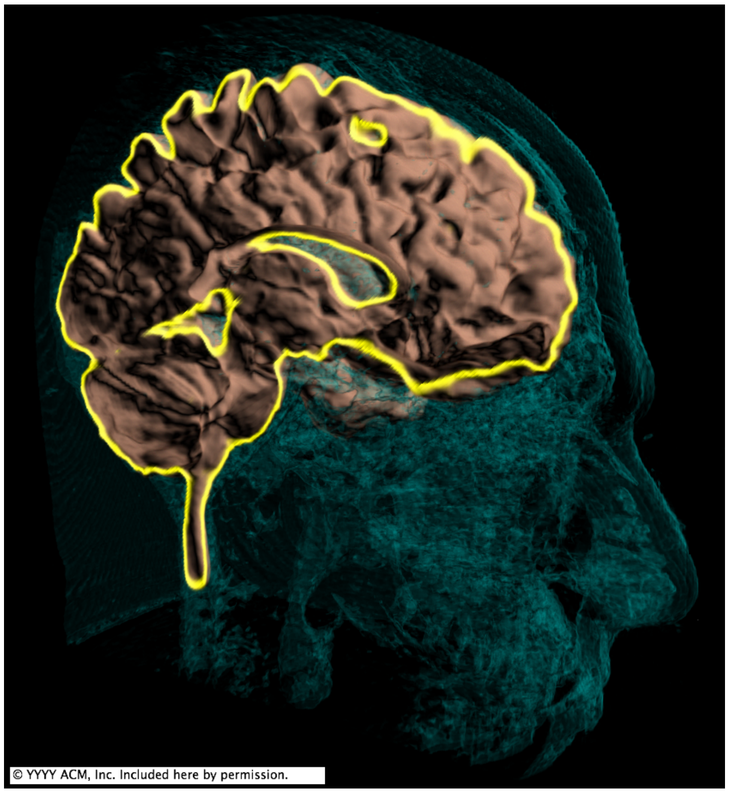


Figure 2.9: Interactive segmentation of the brain being performed inside an interactive 3D volume rendering visualization [61].

Automatic or semi-automatic segmentation utilize automated region growing algorithms that can be constrained based on curvature and various thresholds entered by the physician. Unfortunately automatic segmentation is often very difficult due to poorly defined boundaries, limited image resolution and high levels of image noise. To improve automatic methods, often algorithms are customized for particular organs or tissues and use a combination of automated and manual control.

2.3.3 Visualization

Visualization refers to the use of images, diagrams and animations to communicate a message. Medical visualization is a specialized type of *scientific visualization* which attempts to transform, select or represent data from simulations or experiments to allow for exploration, analysis and improved understanding of the data. In the context of medical imaging, usually visualizations are designed to illustrate raw acquired data in a form more familiar or clear to the physician.

As a concrete example, consider the data acquired in a modality called *diffusion tensor imaging* (DTI). In DTI, a specialized set or MRI pulse sequences are used to acquire not only one value per voxel, but 6 or more values representing the *diffusion tensor* of the voxel (more values can also be used better represent “off-diagonal” information). The diffusion tensor more or less represents the direction and magnitude of diffusion of water in the brain. If mentally visualizing a 3D image from slices is difficult for a normal scan, DTI makes it nearly impossible without assistance. A few scientific visualizations of DTI include color coding the image based on gradient flow direction, and even simulating the flow of water through the brain and generating 3D ‘tracts’ which represent the neural pathways in the brain (see figure 2.10).

Another example of medical visualization which is the focus of this thesis is volume visualization. While it is still common clinical practice to review patient scans as two dimensional planar images in the format that they were acquired, this is becoming more and more impractical for many applications. If one thinks of following a curving vessel through 3D space or planning the trajectory of a surgical intervention, it becomes clear how much useful information can be gleaned from having a simple 3D view of the data. Figure 2.10 illustrates how a correctly designed 3D visualization can immediately identify

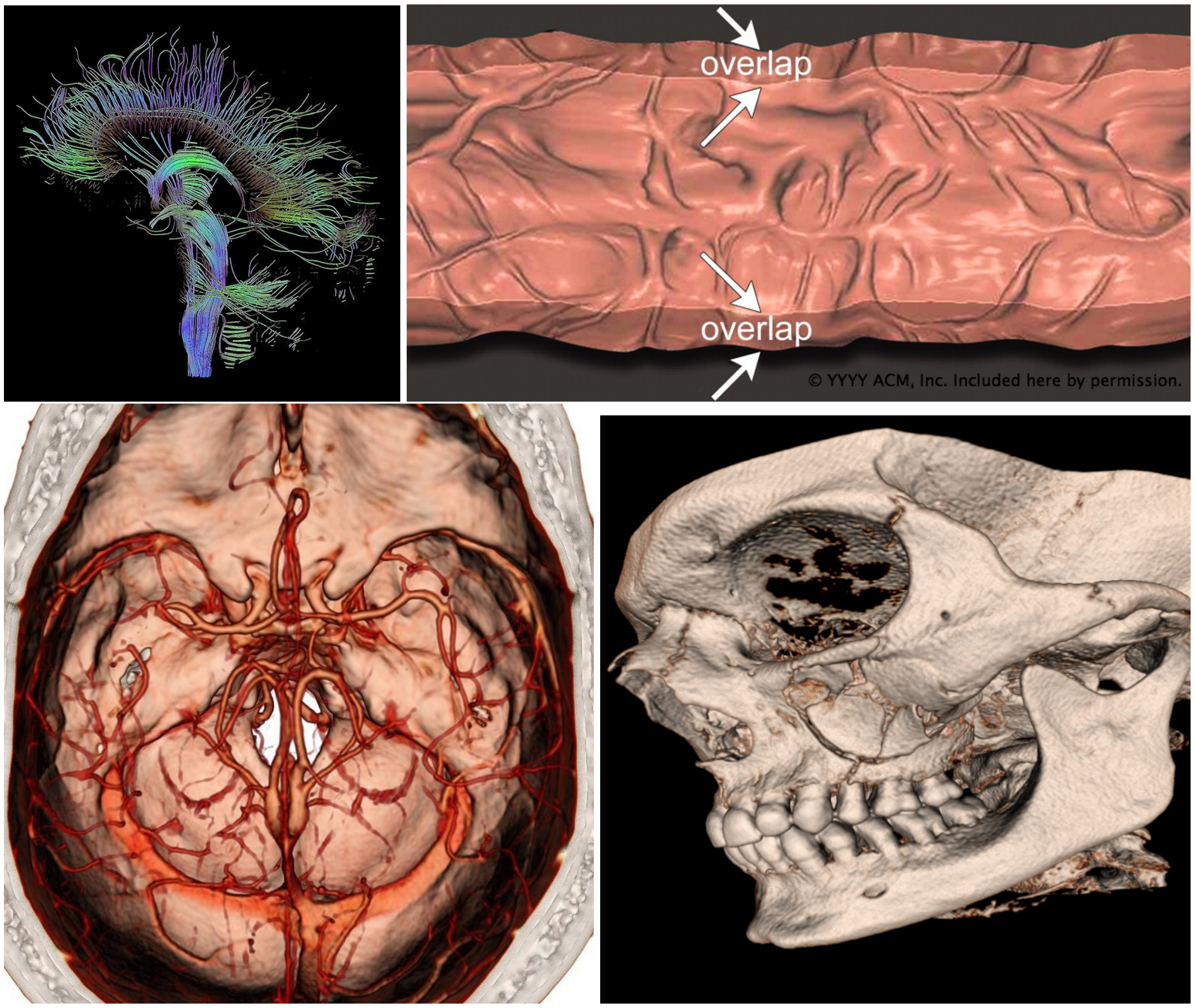


Figure 2.10: Visualizations of medical image volumes. Neural tracts are simulated from diffusion tensor images (top left) [45]; An ‘unravelled’ view of the colon used for quickly finding polyps in virtual CT colonoscopy (top right) [40]; Interactive volume visualization from our software of a CT angiogram to diagnosis ischemic stroke (bottom left) and to plan reconstructive surgery (bottom right).

the location of an ischemic stroke, or provide crucial structural details of broken bones before reconstructive surgery. However, while a 3D view can be more intuitive, it also has the drawback of occluding itself and hiding potentially important information.

Thus, an effective 3D visualization needs to provide both a detailed unobstructed view of the object in question together with its context in 3D space, while at the same time being easy to navigate. The correct way to achieve these disparate goals is often situation dependent. One illustrative example is in virtual colonoscopy, where the physician wants to find lumps on the colon called *polyps*. While a normal 3D view can mimic a real colonoscopy, polyps can be found much faster by 'unrolling' the colon wall into a 2D sheet that can all be viewed like a piece of paper (see Figure 2.10).

2.3.4 Real-time Applications

While many of techniques above are useful for illustrative purposes and in the lab, applying them to real patient data in the field comes with a large set of additional constraints. The state-of-the-art algorithms from each of the above categories can take minutes, hours, or even days to complete. While those timelines might be acceptable for creating a few carefully chosen illustrations, in the real world physicians need to make many complicated diagnoses daily, and don't have the luxury of setting up a complicated simulations or visualizations for each patient unless it can be performed quickly and painlessly. While the most advanced algorithms might produce provably better results, they will often stay in the realm of research labs unless they can be made efficient and practical in a clinical setting. The phrase 'Time is brain' used by stroke physicians illustrates how urgent a timely diagnosis can be.

In seeking to make medical imaging processing analysis practical for use in the field, a large amount of recent research has focused on optimizing medical imaging algorithms to run in real-time or near real-time. While large compromises in the quality of results used to be acceptable to provide real-time performance, the vast improvements in computing power and algorithmic advances are blurring the line between high quality offline approaches and lower quality real-time approaches.

Chapter 3

Graphics Processing Units

Since the contributions of this thesis lie in utilizing the power of modern graphics processors, this chapter serves to introduce a reader with a traditional computer science background to the relatively new field of general purpose computing on the GPU (or GPGPU). The software developed for this thesis makes exclusive use of GPU computing so a discussion of the added complexity of GPU programming is warranted before we delve into the specifics of our research.

3.1 Introduction

Over the course of the last few years there has been an explosion in the computer games industry that has fueled unprecedented growth and advancement of graphics processing hardware. While commodity CPU's have traditionally realized speed gains by increasing clock speeds and decreasing transistor sizes, GPUs have been able to take advantage of the "embarrassingly parallel" nature of computer graphics problems to vastly increase raw processing power. As reported by ATI, the Radeon HD 3870 includes 640 stream processors consisting of over 1.3 billion transistors and running at peak performance of over 1 TFlop at a memory bandwidth of 86.4GB/s. Although CPUs are now starting to take advantage of multi-core processing as well, this speed is still an order of magnitude faster than the peek performance of 80 GFlops and 8.5 GB/s on an Intel Xeon 7000 Quad-Core CPU.

Semiconductor capability, driven by advances in transistor design and fabrication technology, increase at the same rate for both CPUs and GPUs. So why has graphics hardware performance increased so much more rapidly than CPU performance in recent years? The difference is primarily attributed to major architectural differences in the two platforms. CPUs are optimized for high performance on sequential algorithms, with a large proportion transistors dedicated to out-of-order execution and branch prediction. The highly parallel nature of graphics problems has allowed GPUs to utilize a much higher percentage of transistors on arithmetic throughput. Additionally, GPUs can make use of higher transistor counts very effectively by simply adding processing cores. Sequential performance on CPUs does not scale in the same manner. This has resulted in a rate of growth that is significantly faster than the often-quoted Moore's Law (see Figure 3.1).

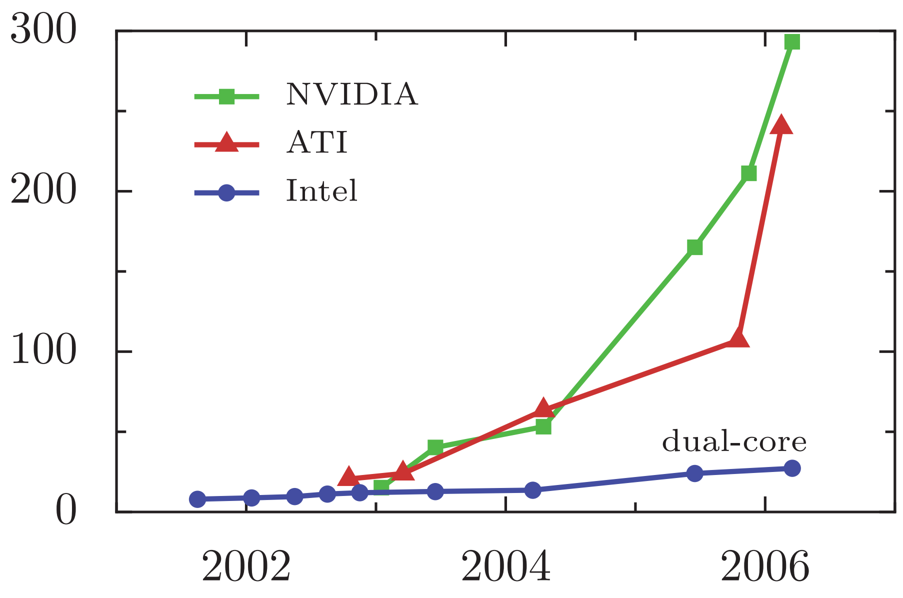


Figure 3.1: The floating point performance of NVIDIA and ATI GPUs compared to Intel CPUs leading up to 2007 [78]. This trend continues to accelerate.

With graphics processing units quickly becoming the dominant computing power in desktop computers, it is only natural to try to make use of this power for other computational problems. Since virtually every computer sold today comes with a graphics processor, the GPU's ubiquity and low cost are making it a very compelling alternative to super computers, clusters and expensive workstations.

3.2 General Purpose Computing on the GPU

Only a few years ago, graphics processors consisted of custom hardware that could only perform one task; their job involved simply drawing, or *rasterizing* triangles onto a 2D array of memory called the *framebuffer*. As time progressed, graphics accelerators were outfitted with custom hardware to perform more complicated graphics operations such as transform and lighting (T&L). It wasn't long before programmable elements were added to allow for custom lighting calculations on a per-vertex and per-pixel level. Each pixel could be processed by a short program that could use textures as input and each geometry vertex could likewise be processed by a short program before it was projected onto the screen. Along with programability came support for much higher precision calculations. While fixed-function pipelines were limited to 8-bit calculations, programmable hardware used 16-bit and eventually 32-bit calculations throughout the entire pipeline.

The first programmable GPUs had extremely limited instruction sets and programs were limited to only a few instructions in length. Basic flow control mechanisms such as branching and looping were also unsupported, meaning exactly the same instructions had to be executed on every pixel or vertex in an object. By 2003 many of these limitations had been removed and several high-level C-like languages were introduced. Microsoft's High-Level Shading Language (HLSL) [72] , the OpenGL Shading Language (GLSL) [87] and NVIDIA's C for graphics (Cg) [68] were all proposed as languages that support a familiar C-like syntax as well as native vector and matrix calculations. All of these languages are still in use today.

With each generation of GPUs adding new levels of programability, researchers and developers have become interested in harnessing this power and flexibility for general purpose computing. This effort has become collectively known as GPGPU or "General Purpose computing on the GPU". A vibrant developer community has already emerged [1] and there is already very promising research in the literature. Nonetheless, the GPU remains a specialized piece of hardware and many applications exist that will likely never be suitable for execution on the GPU. To understand when and where the GPU can be utilized, we first have to understand the underlying architecture and programming model. From there we can describe how an algorithm can be structured for effective execution on a GPU.

3.3 The GPU Pipeline

Modern graphics hardware is designed to maximize performance of real-time rendering applications. Graphics applications require very high performance for very repetitive and independent pixel and vertex operations. To exploit the parallel nature of these operations, graphics hardware is traditionally structured as a pipeline which is divided into several stages (Figure 3.2). Each graphics primitive is processed by each stage in the pipeline, and each stage is implemented using additional hardware. Since each vertex and pixel can be calculated independently, each stage can be implemented using several processing cores. The more cores there are, the faster a given 3D object can be transformed and shaded. Each programmable stage inputs and outputs a limited number of four-component floating-point vectors. The vertex stage takes fixed size vertices as input and output while the pixel processor takes interpolated vertices from the rasterizer, and outputs a number of color values. The recent addition of the geometry stage allows for entire primitives to be taken as input and a variable number of primitives to be emitted as output.

[1] http://www.gpgpu.org

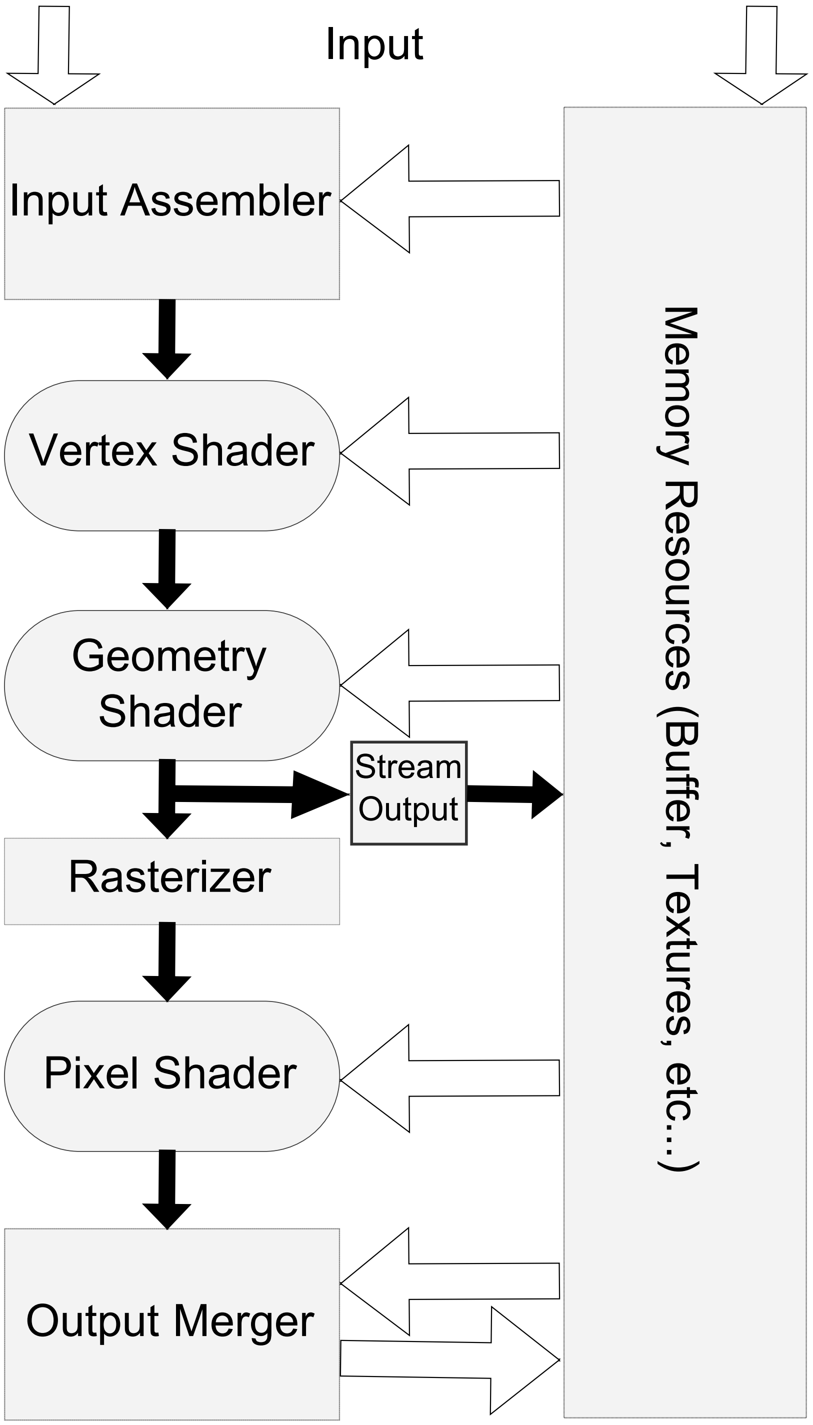


Figure 3.2: The modern graphics pipeline. The vertex, fragment and geometry stages are programmable. [8]

Traditionally each stage of the graphics pipeline used separate processors which had different capabilities. Pixel processors could read from textures while vertex processors could not, and there were several times more pixel processors than vertex processors to account for the ratio of pixels to vertices in the pipeline. The recent trend, however, has been towards a pool of unified *stream processors* that have the same capabilities and can be used by any stage of the pipeline. This insures that no stage in the pipeline will become a bottleneck, as more processors can simply be allocated to that stage on the fly.

3.4 The GPU Programming Model

With all the recent advances in GPU technology, it seems only natural for it to be used outside of the narrow scope for which it was initially intended. Unfortunately, since GPUs were developed with only graphics applications in mind, the programming model is very different from that of traditional CPUs. Therefore, utilizing the GPU is not simply a matter of learning a new programming language and porting over application code. Geometry primitives, texture mapping and rendering passes must be understood and applied in the correct way to invoke a parallel computation on the GPU. Even the most trivial of programs must at a minimum initialize the graphics pipeline, create textures and vertices, invoke the renderer, and read back the results from the GPU.

While graphics APIs such as Direct3D and OpenGL require the use of graphics terminology, the GPGPU community often uses a more abstract concept of *stream programming* when designing parallel algorithms. Stream programming is an inherently parallel programming model which describes problems in terms of *streams* and computational *kernels* (Program 1). To specify a program using a stream programming language, the programmer must find *data-parallel* calculations — identical calculations that are applied to many independent pieces of data. The data elements are grouped together and referred to as a stream while the calculations applied to each element are referred to as the computational kernel. A complex application can be built by chaining together several streaming operations. By specifying a program in this way unnecessary serial dependencies in the code are removed. Since the streaming kernel may be applied independently to each data element, each streaming operation may be shared by an arbitrary number of processors without any additional effort by the programmer.

Program 1 Conversion of a typical serial for-loop into SIMD and then a stream and computational kernel for a fictional stream processor.

Sequential Paradigm

```
for(int i = 0; i < 100 * 4; i++)
    result[i] = source0[i] + source1[i];
```

Single Instruction Multiple Data (SIMD)

```
for(int el = 0; el < 100; el++) // for each vector
    vector_sum(result[el], source0[el], source1[el]);
```

Stream Programming Paradigm

```
streamElements 100
streamElementFormat 4 numbers
elementKernel "@arg0+@arg1"
result = kernel(source0, source1)
```

We can illustrate how these concepts map directly to GPU programming by looking at a simple example. A typical GPGPU program will use the pixel processor to execute a streaming operation. A program will usually be structured as follows [78]:

1. The programmer determines a data-parallel piece of the application. The computational kernel is specified as a fragment program and the input and output streams are stored in GPU memory using textures.

2. To invoke the kernel, a the range of output stream elements and a domain of input elements must be specified. This is accomplished by passing geometry vertices to

the GPU. Typically a quadrilateral (quad) is oriented on the frame buffer to line up with a 2D rectangular region. The output range is specified using the vertex positions and the input domain is specified using texture coordinates.

3. The GPU rasterizer interpolates the quad's vertices (including texture coordinates) and generates a fragment for every pixel in the quad.

4. The fragment program (kernel) is executed for every fragment. The fragment program can read from abitrary memory locations using texture reads, but can only output to a fixed location in memory (the pixel's location in the frambuffer).

Program 2 Example GLSL fragment program for the stream programming kernel from program 1. The input and output streams are specified using using geometry.

```
uniform sampler2D source0;
uniform sampler2D source1;

void main(void)
{
    vec4 arg0 = texture2D( source0, gl_TexCoord[0].xy );
    vec4 arg1 = texture2D( source1, gl_TexCoord[0].xy );
    gl_FragColor = arg0 + arg1;
}
```

To mitigate the need for graphics experts, several libraries have been proposed to abstract the stream programming features of GPUs into a more generic stream programming library. Sh [71] uses ad-hoc polymorphism (operator overloading) to specify streams and kernels within C++ and is being commercialized by RapidMind [81]. The BrookGPU programming language [12] provides C extensions to specify streams and kernels. Many GPGPU algorithms are also characterized in terms of stream programming. Recognizing the demand for general purpose computing on GPUs, NVIDIA has released

Cuda [77] and ATI has released FireStream [4]. Unfortunately, these libraries only work on their respective platforms, and have not been widely adopted for this reason. More recently, an open standard called OpenCL has been introduced which will be supported by both ATI and NVIDIA. Microsoft will also be releasing DirectCompute as part of the Direct3D 11 API.

Chapter 4

Volume Rendering

This chapter introduces, and formalizes for later chapters, the basic background and concepts of volume rendering. We leave more in depth literature reviews of ray-casting optimization and volume lighting techniques for chapters 5 and 6 respectively, where we present our contributions in these areas. For the interested reader, *Real-Time Volume Graphics* [25, 32] by Hadwiger et al. also provides an excellent in depth introduction to volume rendering. In terms of notation, this chapter follows the conventions used in a few seminal papers in the area [26, 69, 70].

4.1 Introduction

In traditional computer graphics, 3D objects are represented using surface descriptions such as polygonal meshes, NURBS (non-uniform rational B-Splines), or subdivision surfaces. Using this style of modeling, light transport is usually only evaluated on points of the surface, and thus lacks the ability to account for light interactions that takes place in the interior of an object. Volume rendering, on the other hand, describes a range of techniques used to generate images directly from 3D scalar datasets such as those acquired in medical imaging. The most common approaches to volume rendering involve extracting 2D surfaces embedded within the dataset, or treating the entire volume as material that interacts with light. The former is generally referred to as *iso-surface* rendering, while the latter is known as *direct volume rendering*.

More formally, *Iso-surface rendering* is concerned with finding the distribution of a desired value within a volume data set. This set consists of all points that satisfy the

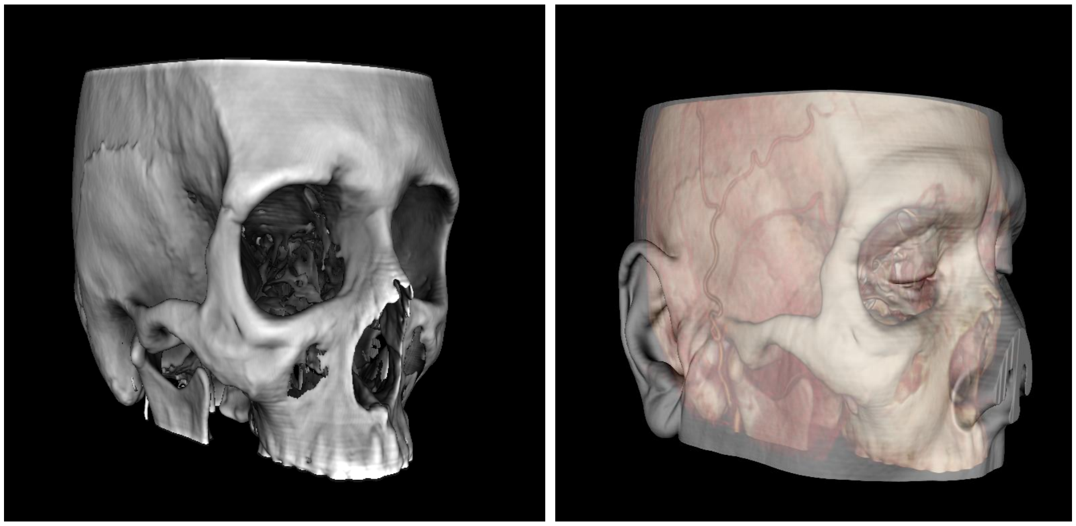


Figure 4.1: Left: A single iso-surface rendering. Right: A DVR rendering of the same dataset.

trivariate function $f(x, y, z) = c$ where c is the *iso-value*. Direct volume rendering (DVR) involves trying to capture the interaction of light and participating media. Light may be *absorbed*, *scattered*, or *emitted* by gaseous or solid material that participates in the light's propagation through the volume.

Since volume rendering is based on the concepts of light transport and image processing, it is worthwhile to first discuss the theoretical background in these areas before delving into the recent developments in later chapters.

4.2 Volume Rendering Pipeline

Volume visualization has the goal of visually extracting information from a 3D scalar field. This can be written as a mapping from a three dimensional space to a single value:

$$\phi : \mathbb{R}^3 \rightarrow \mathbb{R} \tag{4.1}$$

Since the 3D scalar field originates from discrete measurements on a 3D grid, it leads to the question of how the function ϕ can be evaluated for any point in the 3D domain. This refers to the problem of data *reconstruction*, or *interpolation*. Assuming such a reconstruction is available, the scalar value is then mapped to a set of optical properties that describe how light interacts with that point in space. This mapping is called *classification* and is usually implemented through the use of a *transfer function*. Mathematically a transfer function simply maps a scalar value provided by ϕ to a colour and opacity:

$$c = q_c(\phi(p)) \tag{4.2}$$

$$\alpha = q_\alpha(\phi(p)) \tag{4.3}$$

Assuming this second mapping also exists, we are faced with the problem of *lighting* the classified material. Lighting is performed using what is known as the *volume rendering integral*. The volume rendering integral and all of its stages are known as the volume rendering pipeline. We will discuss each of these stages in turn.

4.3 Reconstruction

Reconstruction (or interpolation) refers to the processing of creating plausible data for points where samples were not aquired. Since volume rendering involves sampling along rays that travel in arbitrary directions through the volume, most sampling locations are not directly located on an image voxel. The problem is to decide how to use the neighboring image values to determine what the value might have been at the sample location.

The most common interpolation methods include nearest neighbour, tri-linear, spline-based, and in theory the sine cardinal (sinc) filter. According to signal and information theory, sinc interpolation produces an image that is most true to the original. However, it

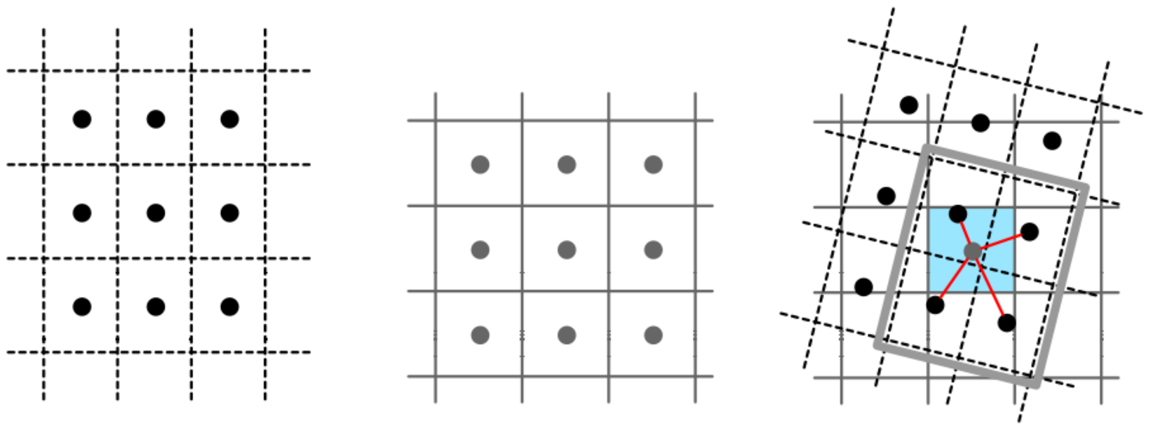


Figure 4.2: Left: A grid of voxels in the volume. Center: A grid samples from rays traversing through the volume. Right: Interpolation is needed when the two grids fail to light up perfectly. Multiple image values are used to reconstruct a value for the sample location. [15]

is extremely computationally expensive compared to other interpolation methods unless it is performed in the fourier domain.

The form of interpolation most frequently used in volume rendering applications is tri-linear interpolation, because it gives acceptable results and often has native support in graphics hardware. Cubic and other B-spline interpolation kernels are much closer to the sinc interpolation kernel, but are much more expensive to compute when compared to linear interpolation. Interpolation is the aspect of volume rendering that is often given the least consideration. However, it is important because the choice of interpolation can produce very noticeable artifacts as shown in Figure 4.3. For the interested reader, [69] provides an extensive examination of the pros and cons of each style of interpolation when used for volume rendering.

Since all graphics processors include custom hardware to perform bilinear and trilinear sampling, it makes sense to make use of this sampling hardware whenever possible. For our implementation we adopted a technique proposed by [94] to efficiently design higher

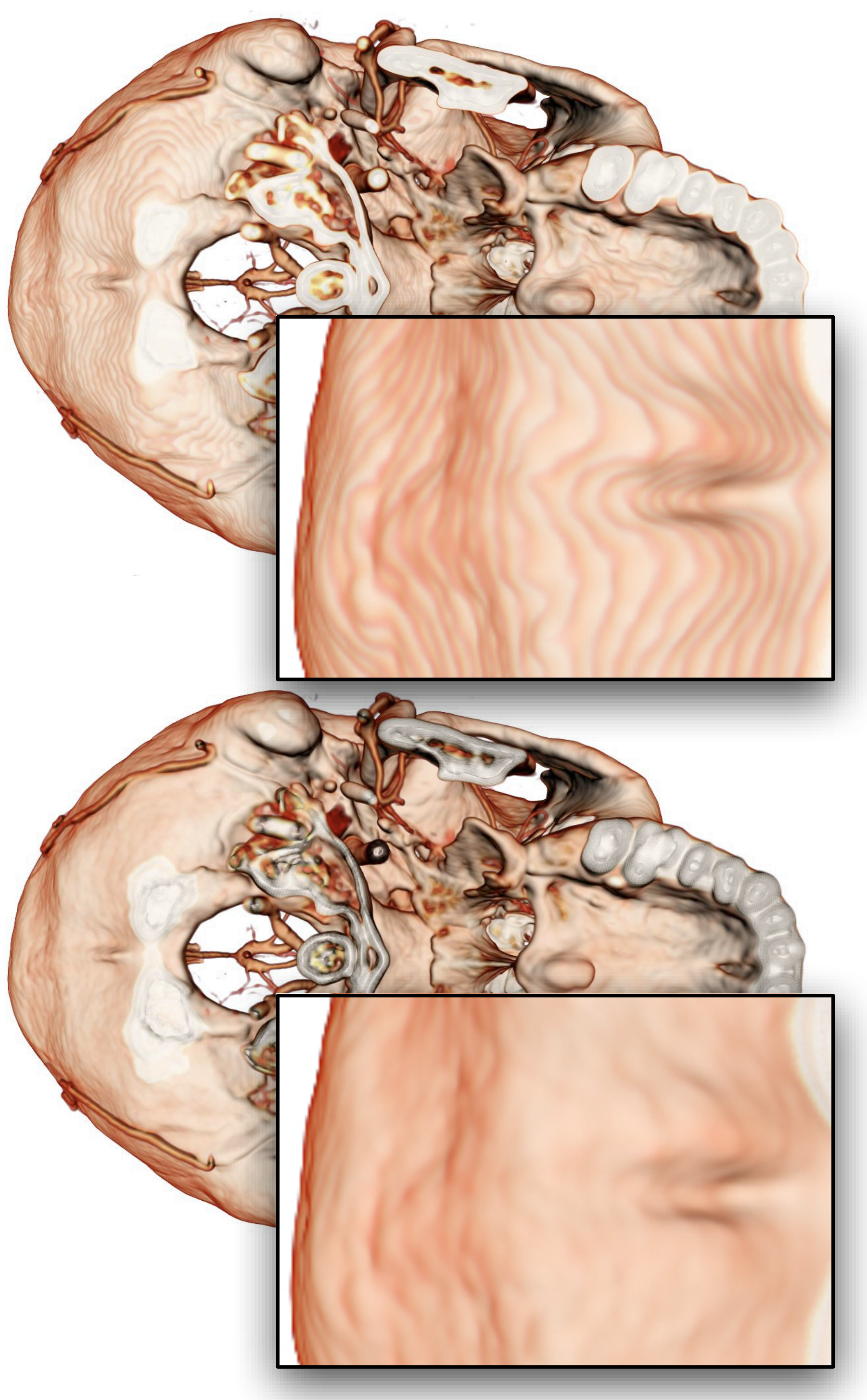


Figure 4.3: The importance of interpolation. Top: Tri-linear interpolation supported in hardware. Bottom: Optimized cubic B-spline interpolation used in our software. We use the hardware's native tri-linear samples to build a cubic B-spline sample efficiently.

order filters. While a naive cubic interpolation filter would take 64 samples (4x4x4), an optimized filter only takes 8 samples. The key is to choose the positions of the samples based on the ratio of the desired sample weights and let the tri-linear sampling hardware fetch 8 samples at a time.

4.4 Classification

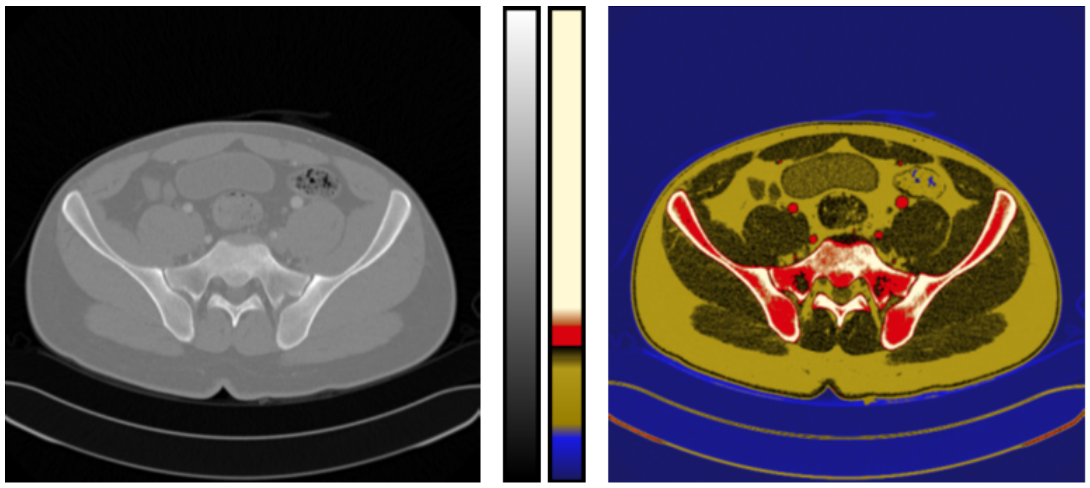


Figure 4.4: Left: Slice of a CT scan. Center: Illustration of a transfer function that maps scalar values to colour/opacity values. Right: Left image with the transfer function applied. The transfer function is completely opaque in this 2D example for clarity, however in 3D DVR rendering usually most values are at least partially transparent.

In order to see important details embedded in a volume, it is necessary to make undesired areas invisible and desired areas visible or highlighted. This can be achieved in a number of ways, but at the voxel level this is usually achieved through a transfer function. A transfer function classifies each voxel and assigns it a specific color and opacity value (see Figure 4.4. Mathematically it is simply a function that takes a scalar value as input and returns a colour and opacity as output.

$$\begin{aligned} c &= q_c(s) \\ \alpha &= q_\alpha(s) \end{aligned} \tag{4.4}$$

This mapping is illustrated in figure 4.4 where air is blue, fat and soft tissues are beige, contrast agent (blood) is red and bone is white. This transfer function is completely opaque, but usually it would be at least partially transparent in a real DVR rendering. Although application of a transfer function seems trivial, when combined with interpolation it can cause severe artifacts in volume rendering. Combining a transfer function with volume interpolation has the result of multiplying the Shannon-nyquist sampling rates of the volume and range of transfer function between two volume samples. Figure 4.5 illustrates the worst case.

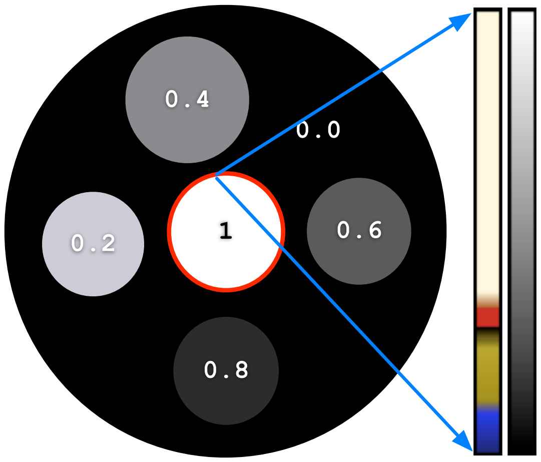


Figure 4.5: An illustration of transfer function under-sampling. When image interpolation is enabled, the thin red bar above contains all values between 0 and 1. Depending on very small variations in the sampling location the transfer function can return any value. Almost no sampling rate can solve this problem when using naive transfer function classification.

Thankfully, a technique called *pre-integrated sampling* [26] allows us to separate the transfer function integration from the rendering integral. Pre-integrated sampling uses a 2D lookup table of all possible integrals from one sample to another within the transfer function. When looking into the transfer function, both a front and back sample are specified in the lookup.

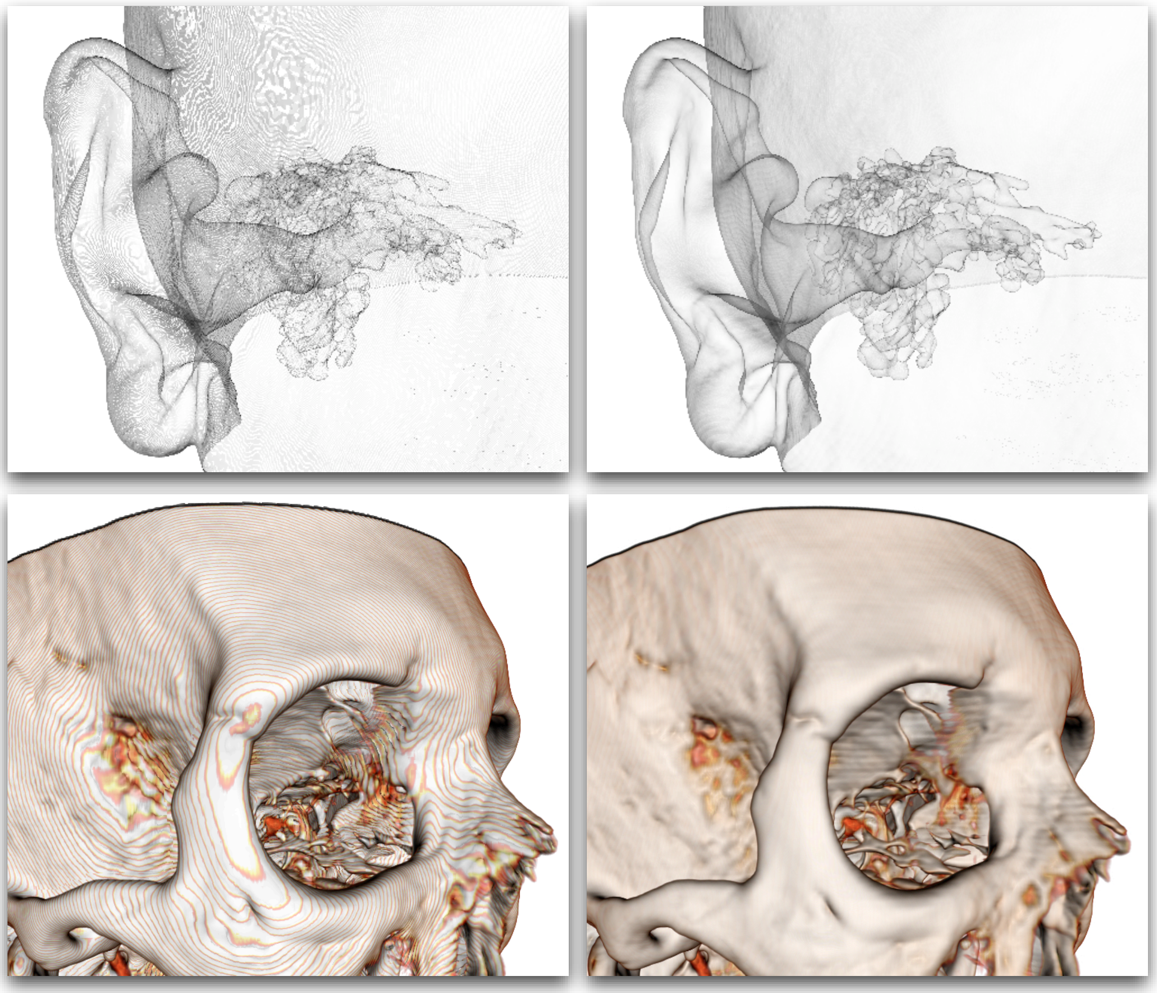


Figure 4.6: Comparison with and without pre-integrated transfer in our software. Images on the right use the same number of samples but achieve much higher quality with pre-integrated transfer functions.

4.5 Volume Rendering Integral

We have now described how at any point in the volume we can both find a scalar value, and determine the optical properties for that value. To perform proper lighting, however, we need to consult the volume rendering integral. Integrals are often used as physically correct rendering *formulas*, as they are continuous. Unfortunately, rendering integrals to not specify the *algorithm* to calculate the formula, and a naive algorithm based on the integral alone is often prohibitively expensive. Most algorithms therefore attempt to approximate the rendering integral rather than compute it exhaustively.

In volume rendering, the most pronounced effects of light interaction that must be accounted for are listed below [25]:

Emission Matter releases energy in the form of light, increasing light energy.

Absorption Matter absorbs incoming light and turn it in to heat, reducing light energy.

Scattering Light can be scattered by participating media, changing the direction of light propagation. *In-scattering* adds additional energy to a ray while *out-scattering* removes energy.

An an optimal model for computing light transport should include the effects of emission, absorption and multiple scattering. Because a complete solution becomes very computationally intensive, it is very common to neglect the effects of multiple scattering and use a more tractable emission-absorption model with single in-scattering of light towards the eye as depicted in Figure 4.7). For the interested reader, a comprehensive discussion of light transport models can be found in [70]. The volume rendering formula for emission and absorption can be solved by integrating along the direction of light from a starting

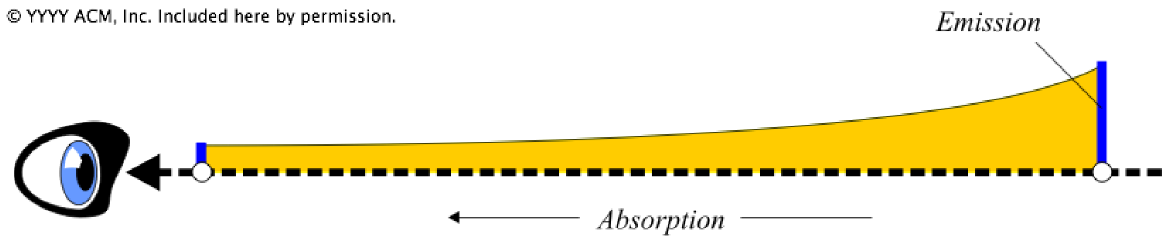


Figure 4.7: Illustration of a simple emission absorption lighting model. Light decays exponentially by absorption but additional light can be added to the ray by emission or single scattering (in-scattering) from an external light. [25]

point behind the volume to an endpoint in front of the volume.

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(s) e^{-\int_{s_0}^{D} \kappa(t)dt} ds \tag{4.5}$$

In this form we integrate from the starting point $s = s_0$ to the endpoint $s = D$. I_0 represents the background radiance entering from behind the volume at the initial position S_0 and $I(D)$ is the light energy leaving the volume at $s = D$ in the direction of the camera. The first term represents the background light energy (radiance) contribution while the second term represents the emissive contribution $q(s)$ from every point on the ray attenuated by the absorption $\kappa(t)$ along the remaining distance to the camera. The effect of single scattering or in-scattering — the reflectance of light via a local illumination model — can be incorporated without added complexity by simply treating local illumination as part of the emissive term.

It is often more intuitive to think of this integral in terms of transparency rather than attenuation. We can define the transparency of the material between s_1 and s_2 as

$$T(s_1, s_2) = e^{\tau(s_1,s_2)} = e^{-\int_{s_0}^{D} \kappa(t)dt} \tag{4.6}$$

While absorption ranges from 0 to ∞, transparency T ranges from 0 to 1. With this definition of transparency we can redefine the volume rendering integral as follows

$$I(D) = I_0 T(s_0, D) + \int_{s_0}^{D} q(s)T(s, D)ds \tag{4.7}$$

Direct volume rendering seeks to efficiently compute this integral for every ray in the image. The most common way to approximate this integral is with a Riemann sum over n equidistant positions along the ray defined as $s_0, s_1, ..., s_n$. If we define discrete approximations of transparency T and color c for the ith interval as

$$\begin{aligned} T_i &= T(s_{i-1}, s_i) \\ c_i &= \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)ds \end{aligned} \tag{4.8}$$

The radiance at the endpoint D can then be recursively defined as

$$I(D) = I(s_n) = I(s_{n-1})T_n + c_n = (I(s_{n-2})T_{n-1} + c_{n-1})T_n + c_n = ... \tag{4.9}$$

This recursion terminates when it reaches I_0 which is simply the background color. This can be rewritten as a summation as follows

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} T_j \tag{4.10}$$

We can describe this summation as summing the contribution of all colors along the ray after first weighting them by the accumulated transparency up to that point. Transparency is often interchanged with opacity or α which is defined as

$$\alpha_i = 1 - T_i \tag{4.11}$$

4.6 Rendering Techniques

Now that we have discussed the theory behind volume rendering. We can discuss some of the basic algorithms that approximate the volume rendering integral and apply the volume rendering pipeline.

4.6.1 Compositing

With the exception of fourier volume rendering [67], all volume rendering algorithms involve some form of iterative compositing scheme. Compositing is used to integrate the volume rendering equation in either back-to-front or front-to-back order. Usually the algorithm chosen for rendering will imply which compositing order is needed. In some cases, such as half-angle-slicing [51], both back-to-front and front-to-back are needed.

Front-to-back compositing evaluates the volume rendering integral in discrete steps using the operator:

$$\begin{aligned} C_{i+1} &= (1 - A_i) \cdot \alpha_i \cdot c_i + C_i \\ A_{i+1} &= (1 - A_i) \cdot \alpha_i \quad + A_i \end{aligned} \tag{4.12}$$

Where, respectively, C, A denote the color and opacity value of the ray, c, α denote the color and opacity value given by applying the transfer function, and i denotes the sample index. Back-to-front compositing uses the familiar alpha blending operator:

$$C_{i+1} = \alpha_i \cdot c_i + (1 - \alpha_i) \cdot C_i \tag{4.13}$$

Both discrete approximation methods can be viewed as discretizing the volume into slabs (or shells). Each slab emits light and absorbs light but light emitted in each slab is not attenuated by the slab itself. An important addition to these volume integration schemes is opacity correction. The opacity specified by a transfer function must assume a given sampling interval Δx. If the true sampling interval $\Delta\tilde{x}$ is different, the corrected opacity $\tilde{\alpha}$ must be adjusted in the exponent:

$$\tilde{\alpha} = 1 - (1 - \alpha)^{\frac{\Delta\tilde{x}}{\Delta x}} \tag{4.14}$$

This is especially important if a variable sampling rate is used during rendering or if the transfer function opacity is specified in a real world unit such as millimeters but the data has an independant sampling density.

In addition to these physically based compositing operators, medical imaging often makes use of additional illustrative operators. Examples include X-ray, which simulates X-ray attenuation, and maximum intensity projection (MIP), which chooses the value with the highest intensity.

4.6.2 Texture Slicing

Texture slicing is one of the first algorithms designed for volume rendering. It involves drawing large 2D polygons which slice through the volume. Slices can be drawn in front-to-back or back-to-front order using the compositing operators from the last section. Since basic texture mapping of polygons was one of the first hardware accelerated features, it is also the first approach used to achieve real-time frame-rates.

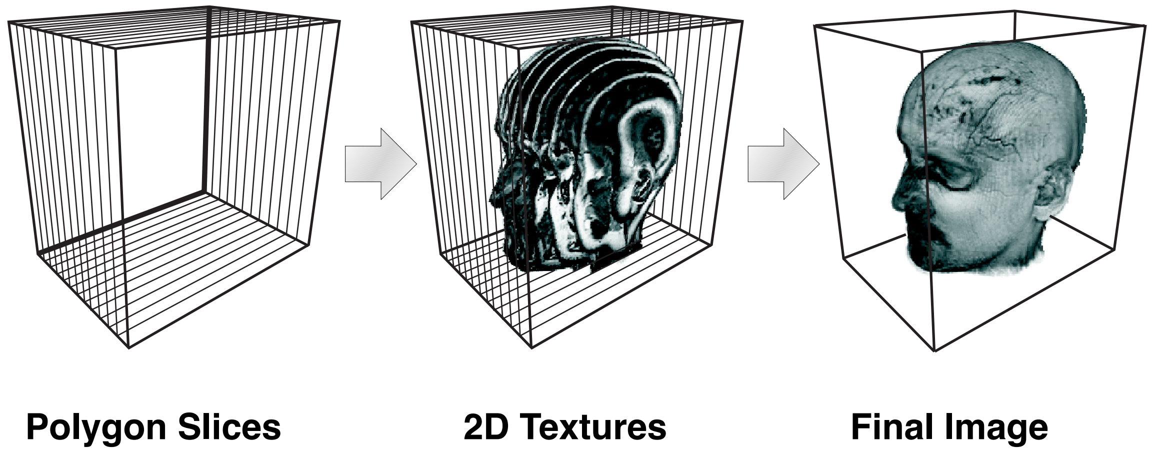


Figure 4.8: Slice based volume rendering using 2D textures. This form of rendering breaks down when the planes becomes perpendicular with the viewing direction. To support all views three different volumes are used. [25]

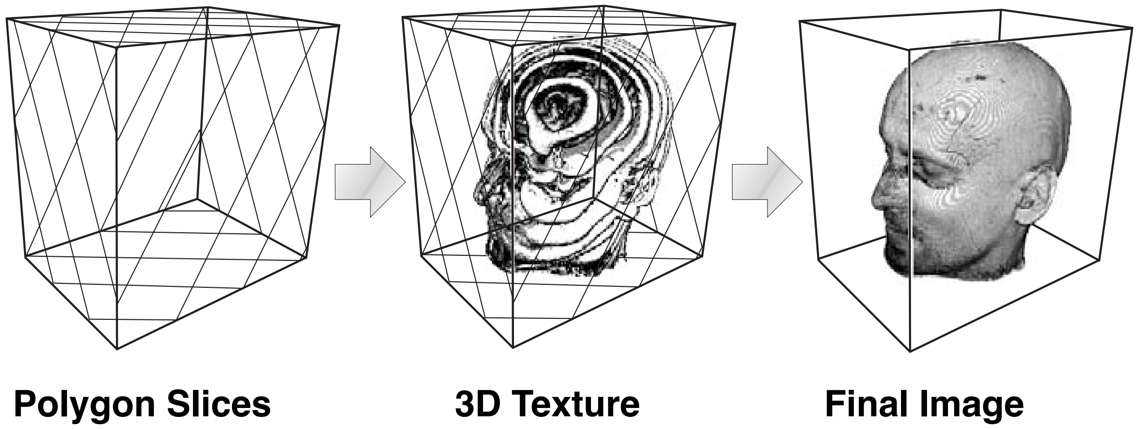


Figure 4.9: Slice based rendering using view aligned slices allows the volume to be rendered from any angle. 3D Texture mapping must be supported for view aligned slicing. [25]

The first approach to texture slicing involved storing a volume in many 2D textures and rendering the slices in sorted order (see Figure 4.8). Since the slices were aligned with the volume, three copies of the texture were needed depending on the viewing angle; visible popping occurred when switching between these volumes.

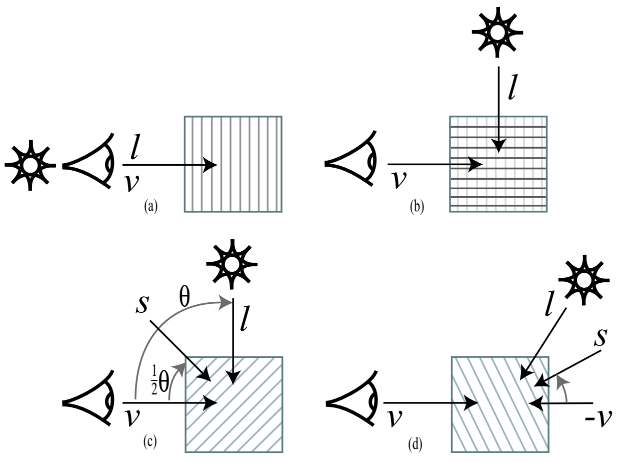


Figure 4.10: Half-Angle slicing accumulates lighting information at the same time rendering the volume. The half-angle between the light vector l and view vector v always provides a valid plane s shared by the two views. Rendering switches between back–to-front and front-to-back rendering depending on the lighting direction. [51]

Once 3D texture mapping became available, the next logical progression was to use clipped view aligned polygons to render the volume (see Figure 4.9). Now only one copy of the volume was needed, and smooth transitions between all views were provided by hardware trilinear texture sampling. One initial drawback of this technique was variable performance depending on the viewing angle. Early graphics hardware stored textures linearly in memory, so performance would suffer when not viewing the volume along the z axis. New hardware has solved this problem by storing textures in a swizzled memory lay-out in which neighboring pixels are store together in memory.

More advanced illumination techniques have also been implemented using slice-based volume rendering. Half-angle slicing [51] accumulates lighting information using slicing in tandem with rendering (Figure 4.10). By using the half-angle between the light and view vectors, lighting can be accumulated using the same slice planes. Shadows as well as approximate sub-surface scattering can be supported with half-angle slicing.

4.6.3 Ray-Casting

The other approach to volume rendering treats each pixel as a ray which is computed independently, rather than iterating all rays together in image space slices (see Figure 4.11). Ray-casting has always been a popular CPU algorithm for high-quality volume rendering, but has only become popular on the GPU recently due to the added flexibility of recent GPUs. Basic GPU ray-casting involves using a cube to represent the volume and performing the ray-casting in several passes (see Figure 4.12). Basic optimizations to GPU ray-casting involved using the CPU to generate proxy geometry that encloses the visible parts of the volume better than a simple cube (see Figure 4.12).

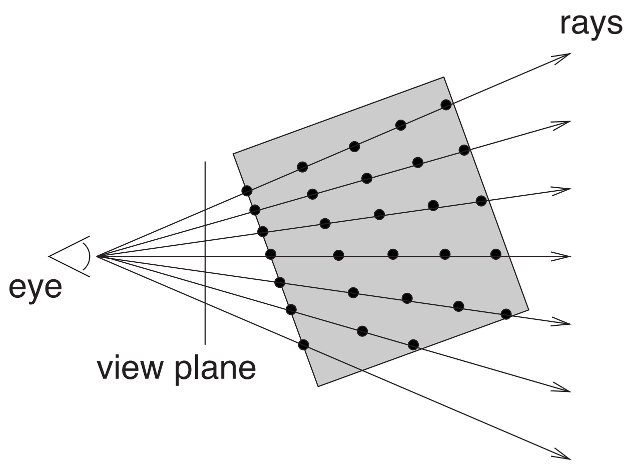


Figure 4.11: Ray-casting renders the volume using a dedicated ray for each pixel. Samples need not lie on the same plane, so equidistant sampling is possible.

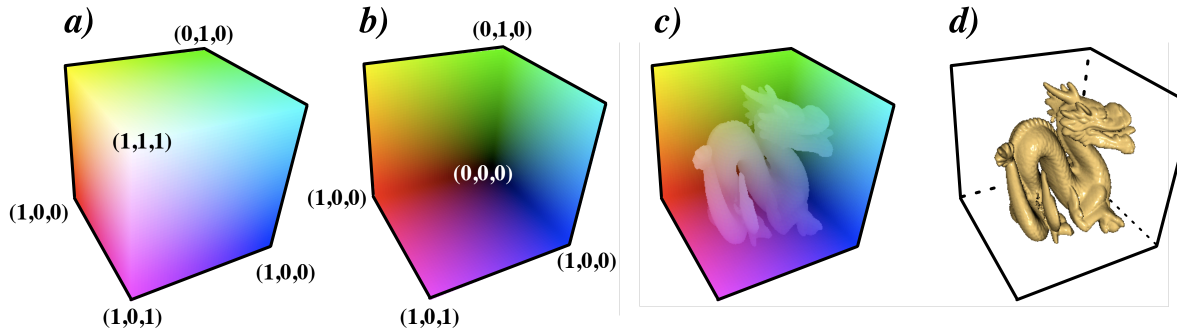


Figure 4.12: GPU iso-surface ray-casting. Ray entry and exit points are stored by rendering a cube which represents the volume. The front (a) and back (b) faces are rendered and stored in textures. In a subsequent pass a ray is cast from the front face to back face until the iso-surface is found (c). Lighting is performed in a fourth pass (d).

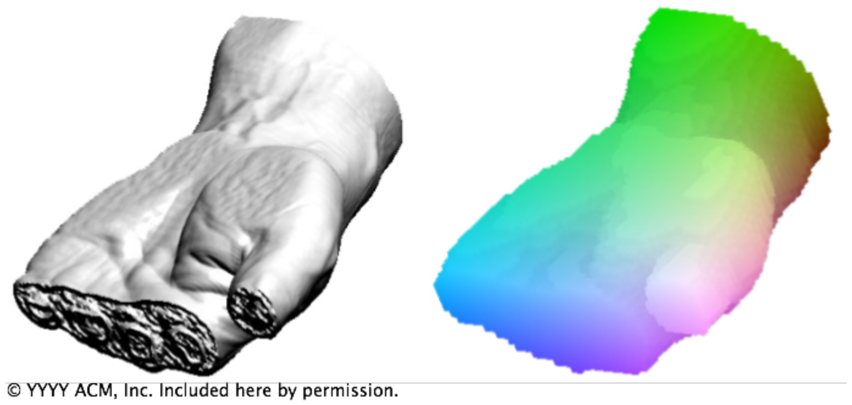


Figure 4.13: Image from [89] of data specific proxy geometry used to optimize GPU ray-casting. The volume is broken into cubes which are tested for visibility on the CPU and then uploaded to the GPU each time the transfer function or iso-surface is changed.

Chapter 5

Acceleration Using Hierarchical Frustum Casting

This chapter introduces one of the main contributions of this thesis, which is a novel method for accelerating ray-casting using a technique we call hierarchical frustum casting. This chapter also reviews additional relevant literature. We first discuss general approaches to optimization, followed by architecture specific optimizations and then ray-casting specific optimizations. We then turn to our technique and discuss the reasoning behind and the implementation details of our approach.

5.1 Introduction

Since the interaction of light needs to be evaluated at all the positions in the 3D volume, direct volume rendering quickly becomes a very computationally intensive task. To understand how many operations are required let's look at an example. With a typical screen resolution of 1024^2, a reasonable volume size of 512^3 and a frame rate of $30FPS$, a brute force ray-caster would need to perform over 30 billion compositing operations per second $(30 \cdot 1024^2 \cdot 512 \cdot 2)$. Even a very basic compositing step requires many lookups into memory and many calculations. Just to compute the gradient, for example, we would need 6 tri-linear memory lookups (using central differencing). This alone would require over 1.5 trillion memory accesses per second! ($30 \cdot 1024^2 \cdot 512 \cdot 2 \cdot 6 \cdot 8$).

With such high computational costs, building an interactive volume renderer would be an insurmountable task without serious optimizations. As such, much work has been focused on improving the performance of both off-line and interactive volume rendering methods. We first look at how one should approach such an optimization problem, as well as at the GPU specific constraints that must be considered. We then turn to our optimization approach which utilizes multi-pass coherent ray-casting to achieve drastic performance increases without requiring complex CPU/GPU interaction.

5.2 Approaches to Optimization

Algorithmic optimizations usually fall into two overlapping groups. Architectural optimizations and input specific optimizations. Architectural optimizations seek to maximize the performance of an algorithm given the constraints of a given hardware architecture, while input specific algorithms make key assumptions about the input data and then exploit these assumptions for performance gains. A sophisticated system will be optimized for the given architecture and have several input specific algorithms to choose from that can be chosen based on the input. A given input specific algorithm may even perform worse on certain types of input due to the extra overhead incurred to exploit a given assumption, but this can be mitigated by applying a detection step and choosing which algorithm is appropriate for the input. A classic example of this is Brent's method [10] for root finding, where three different root finding methods are applied depending on the output of a simple analysis of the input function.

In a worst case dataset for volume rendering, every voxel will have a distinguishable contribution in the final image. This sets the worst case performance for all volume rendering algorithms, and currently only architectural optimizations can improve this worse case. Thankfully, common datasets are far from this worst-case, having many data specific properties that can be exploited for performance gains. The most common data specific property is spatial coherence (also called data locality). Architecturally speaking, cache coherence and branching coherence are very important.

5.2.1 Spatial Coherence

Spatial coherence refers to neighboring voxels tending to have similar scalar values which results in large regions with similar appearance and large regions of empty space. In medical data, as little as 1% and rarely more than 60% of voxels contain visible data [88]. When only translucent or opaque iso-surfaces are rendered rarely more than 5% percent of voxels are visible before rays become fully saturated or exit the volume. Spatial coherence is usually exploited through use of hierarchical 3D data structures such as a kd-trees, octrees, macro cells, bounding volume hierarchies (BVHs), or the 3D distance transform. Fast traversal algorithms similar to bresenham's line drawing algorithm [11] can be used for volumes [1], octrees [63] and distance fields [17] to quickly skip empty space and find surface intersections.

In volume rendering it is undesirable to have large preprocessing steps each time the iso-surface or transfer function is changed and thus variants of the above algorithms that store min-max bounds [21] or Lipschitz bounds [95] can be used which are independent of transfer function changes. Lipschitz bounds store the maximum image gradient in a cell, while min-max bounds store the minimum and maximum image values. Unfortunately when using a distance transform one must still recalculate the distance field after the transfer function changes, but there are linear time algorithms [27] and methods to compute them quickly on the GPU [20, 85].

5.2.2 Cache Coherence

Cache coherence refers to how an algorithm accesses memory. Generally speaking, algorithms which access contiguous memory locations will perform much better than algorithms that thrash back and forth between distant memory cache blocks. GPU architectures have evolved to take advantage of the similar memory access patterns of neighboring pixels. Neighboring pixels are computed in lock step with each other using the same instruction pointer, rather than executing one pixel's calculations before starting the next pixel. When this group of neighboring pixels accesses memory, all the accesses are performed at the same time and likely access the same location, resulting in much better cache coherence. CPU ray tracing algorithms have in the past had poor cache coherence since neighboring rays tend to traverse the same voxels or tree nodes, but at different times.

CPU ray tracing cache coherence has been addressed recently with research into coherent ray tracing methods [102]. Coherent ray-tracing increases cache coherence while taking even greater advantage of spatial coherence. This is achieved by tracing large groups of neighboring rays (called packets) at the same time, rather than waiting for one ray to finish before moving on to the next. The result achieved is very similar to what is natively supported in GPU architectures in terms of cache coherency. In addition to cache coherence, work can often be amortized across the entire packet. For example, empty space skipping can be performed for the entire packet, instead of for every individual ray. Reshetov et al. [83] use frustum tracing and SIMD optimized frustum-box culling to cull kd-tree nodes for large multi-resolution groups of rays. Wald et al. [103] use coherent grid traversal (CGT) to drastically accelerate ray tracing (see figure 5.1). Coherent grid traversal has been extended to coherent octree traversal (COT) by Knoll et al. [52] for CPU based first-hit iso-surface rendering and extended for particle ray tracing by Gribble et al. [44].

5.2.3 Branching Coherence

Branching coherence is also very important for GPU methods and refers to how neighboring pixels behave in loops and branches. While the GPU supports four-wide SIMD instructions like on x86 and Cell architectures, there is also an inter-pixel 'SIMD width' in which neighboring pixel threads share the same instruction pointer. This width ranges

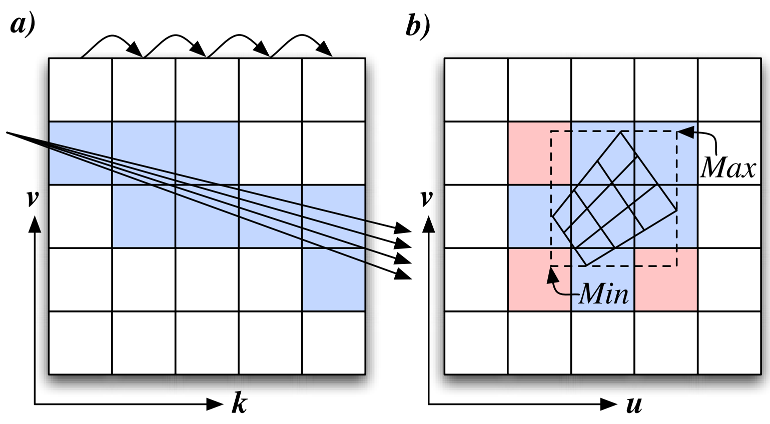


Figure 5.1: Illustration of coherent grid traversal (CGT) on the CPU. Neighboring rays (a) usually traverse mostly the same cells (colored blue)); computating these rays individually results in incoherent memory access and redundant instructions. CGT picks the major ray axis k of the packet and creates an axis aligned bounding frustum (b) in the u,v plane. This frustum encloses all rays in the packet. Traversal of the frustum involves traversing two rays (min, max). Using SIMD instructions one traversal step takes only one SIMD add and one SIMD float-to-int conversion. A 2D loop then checks the cells bounded by the min/max cells at each step. Some unneeded cells are checked due to the loose bounding box approximation (colored in red).

from 16 on the latest models (NVIDIA 8000 series and ATI 2000 series) to much higher on older models. While this arrangement tends to provide improved cache coherence and improves arithmetic throughput by dedicating more transistors to number crunching operations, it also trades off single pixel branching performance. Essentially, this results in arithmetic units sitting idle unless there are enough pixels executing the same instruction.

As an illustrative example, consider a simple if-then-else block that contains 10 instructions in the if-block and 10 instructions in the else-block. If all neighboring pixels execute only the if-block, then the else block can be skipped resulting in 100% efficiency. However, if only one pixel in the group requires the else block, this will result in both the if-block and the else-block being executed for the entire group of pixels, resulting in only 50% utilization of the arithmetic units. If each of these if/else-blocks contain a nested if-then-else block then efficiency will drop to 25%. One can see how the large throughput of the GPU can quickly be consumed by inefficient branching operations. For this reason, Horn et al. [41] found branching coherence to be the main bottleneck for their single pass kd-tree GPU ray tracer. As pixels diverged down the KD-tree, all the neighboring pixels needed to follow all the paths taken by neighboring rays, drastically reducing performance.

5.3 GPU Volume Rendering

While advanced coherent approaches are being utilized on the CPU to increase performance, GPU based approaches have typically relied on brute force approaches that rely solely on the increased arithmetic throughput of the GPU. More advanced ray tracing approaches that utilize hierarchical data structures have so far suffered from major branch performance issues resulting in performance similar to, or only slightly faster than, CPU approaches.

While ray-casting is becoming the standard approach for both CPU and GPU approaches, the GPU has had a much more diverse history. The first hardware accelerated methods for volume rendering were proposed by Cullip and Neumann [76] and Cabral et al. [14]. They represent the volume as a 3D texture or a stack of 2D textures, and resample onto viewport aligned planes called *proxy geometry*. Successive planes are rendered using back-to-front compositing. Texture based approaches have been refined several times in recent years [26, 51].

Although a very flexible and useful algorithm, it is also hard to optimize, often resulting in consistent worst-case performance even when most of the volume is empty or not visible. Although much more difficult to optimize than ray-casting, a few optimizations have been proposed [49, 88].

To avoid the extra costs, GPU ray-casting approaches have also been presented.

Krueger and Westermann [55] propose a method to accelerate volume rendering based on early ray termination and space-skipping. Their algorithm uses alternating ray-casting and empty space skipping passes using a min-max macro-cell grid. Roettger et al. [84] propose a ray-casting system which avoids low quality 8 bit frame buffer blending and a method for artifact free volume clipping. Hadwiger et al. [34] and Scharsach [89] propose a combined approach to empty space skipping using object order bounding geometry and adaptive sampling, as well as advanced deferred shading [94].

5.4 Coherent GPU Frustum-Casting

We now turn to our optimization approach which utilizes multi-pass coherent frustum-casting to achieve drastic performance gains. Our technique is advantageous in that it requires no CPU/GPU communication, and is based purely on ray-casting. The extra steps required can even be implemented as custom compositing operations, making it possible to implement our method within an existing ray-casting system using the outputs from empty space skipping passes as inputs to ray-casting.

As mentioned above, most existing acceleration techniques rely on a combination of CPU pre-preprocessing and CPU proxy-geometry extraction. While proxy geometry is rarely used on the CPU, the reason it is used frequently on the GPU is that GPUs have been designed to be very efficient at taking a group of unorganized triangles and rasterizing them in depth correct order onto the screen. By extracting a *shell* that encapsulates the visible data in a volume and drawing this using the conventional pipeline, you can effectively find the entry/exit point of each ray, quickly skipping past large empty parts of the volume.

Unfortunately, finding the proxy geometry shell has always been performed on the CPU since until recently GPUs could not generate geometry on the fly. This results in a more complex rendering architecture and adds the potential for pipeline *stalls* as the new geometry is uploaded to the GPU. It also doesn't allow for skipping any internal space in the volume as the depth buffer can only be use to find either the front or back of the shell.

Our goal was to achieve similar or better performance than proxy geometry solutions using a ray-casting only approach. Unfortunately, recent GPU ray-tracing results have shown less than optimal performance for standard kd-tree and octree traversal algorithms on the GPU, commonly citing poor branching performance [41]. In addition to using ray-casting, we also wished to take advantage of the recent work in coherent traversal, since it seems wasteful for tens or even hundreds of rays to traverse the same empty voxels. Again, unfortunately techniques like coherent-grid-traversal require a complex 2D loop to check all the cells within a packet frustum at each step(see figure 5.1). Although there haven't been any GPU implementations of these methods, it seems likely they would also exhibit poor branching performance without major modifications.

To alleviate complex branching issues mentioned above, we decided to pursue alternatives that fit more readily into the GPU's memory and execution model. We settled on a multi-pass technique that is implemented as a simple for-loop with no additional branches. Unlike traditional tree based traversal techniques, our technique doesn't require a stack, which is the primary source of complicated branching in ray traversals [41]. Additionally, we don't require complex looping to check the cells in our frustum traversal.

The multiple passes in our algorithm are used to move from coarse packets of rays to fine packets of rays and finally to individual rays that perform the final compositing. We will first describe a custom data structure we created to allow for easy casting of large packets of rays through empty space. We will then describe how our acceleration uses a new technique called coherent frustum casting to very quickly cast through empty space and converge on the visible parts of the volume very quickly.

5.4.1 The Overlapped Min-Max Octree

The thrust of our approach is to use a new data-structure that alleviates the problems in ray-tracing and coherent ray-tracing approaches and lets us reap the benefits of these approaches without the drawbacks. The new data-structure is a modified octree that makes casting large packets of rays much easier.

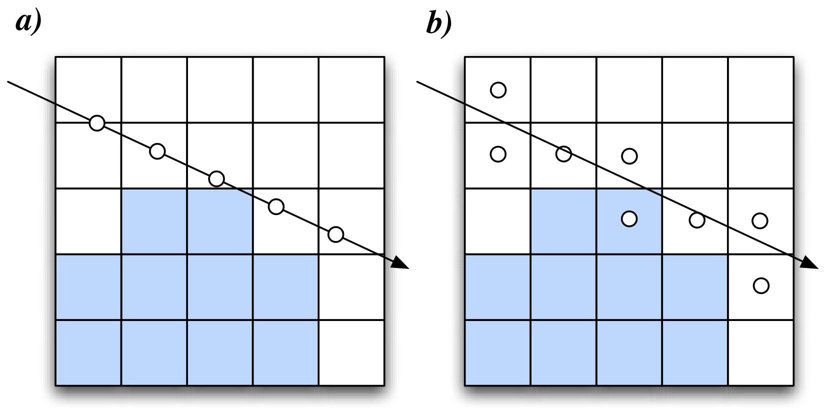


Figure 5.2: Illustration of ray-casting vs ray-traversal. Casting has no notion of the of the grid structure and simply samples at a fixed interval. This results in missing an occupied octree cell (a). Ray-traversal often uses a cell traversal algorithm like Bresenham's algorithm [11] until the ray reaches an occupied cell (b). Every grid cell touched by the ray is sampled, thus no occupied cells are missed.

We observed that while advanced ray-*tracing* algorithms tend to be slow on the GPU, ray-*casting* fits very nicely into the GPU's memory and execution model (see figure 5.2. Thus, our approach alters the octree data-structure to allow for *casting* of entire packets at once, rather than *tracing* of ray packets as in [83, 103]. Although definitions vary and there is often overlap in their definitions, ray-casting or ray-*marching* is usually defined as consecutive sampling along a ray at fixed intervals. Conversely, ray-traversal (which is usually part of ray-tracing) involves explicitly intersecting the ray with planes or cells, such that the step size changes at each step. Casting fits very nicely into the GPU execution model as it comes down to very repetitive texture sampling operations. However, acceleration approaches based on standard octrees require more complicated ray-traversals, such that no octree cells are missed.

In order to allow for casting of an entire packet of rays, we designed a new data-structure that we call the *overlapped min-max octree* or OMMO (see figure 5.3). At each cell in the OMMO, we store the minimum and maximum value contained within that cell as well as within a user defined border or *overlap* around the cell. The overlap is important, as it can give a guarantee that there is no visible data within a certain distance from a sample location, no matter where the sample is taken. The result is that an entire packet of rays can be cast using one representative ray in the center of the packet, so long as the width of the packet is lower than the overlap in the octree (see next section).

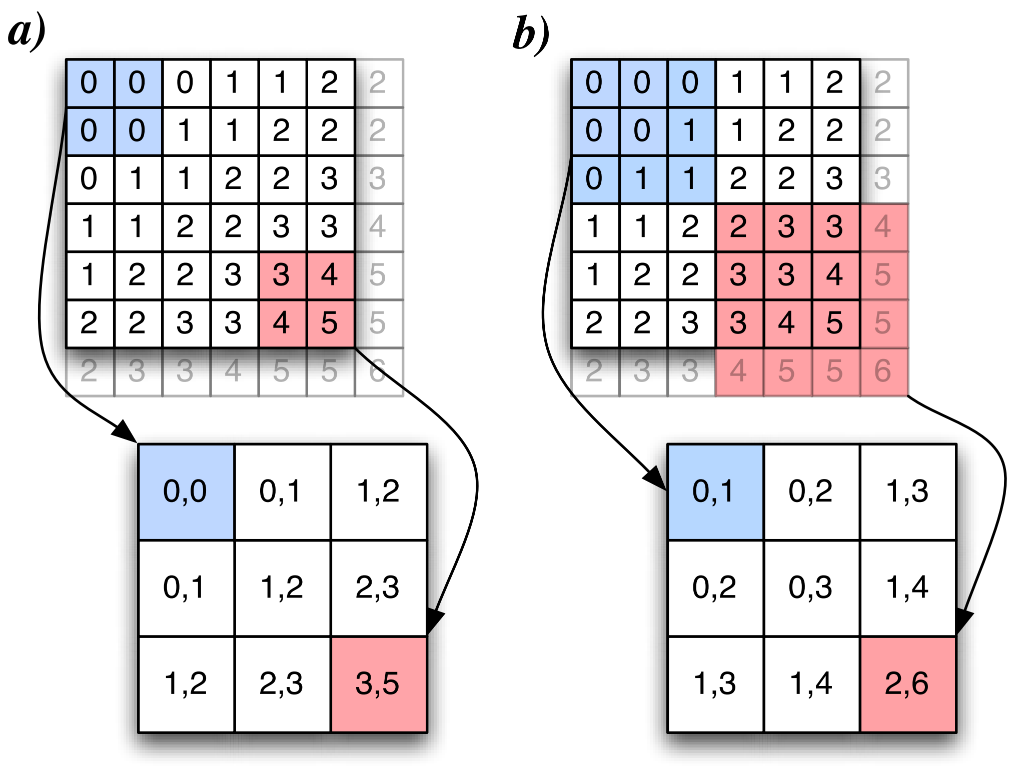


Figure 5.3: Calculation of the second level of the overlapped min-max octree (OMMO). (a) A normal min-max octree stores the minimum and maximum values within a given cell. (b) The overlapped min-max octree adds an overlap of specified size when computing the minimum and maximum values. Here an overlap of one cell is use, but we found two cells to be effective. The overlap is clamped (blue cell) at edges of the volume.

5.4.2 Frustum Casting

We will now describe how we utilize the OMMO to greatly accelerate skipping over empty regions of the volume. The idea behind our approach comes from adapting the sphere tracing algorithm [36]. Sphere tracing assumes there is a dense distance field describing the distance of the closest visible surface to that point. As demonstrated in figure 5.4, the number of samples required to converge on a surface is drastically reduced using sphere tracing.

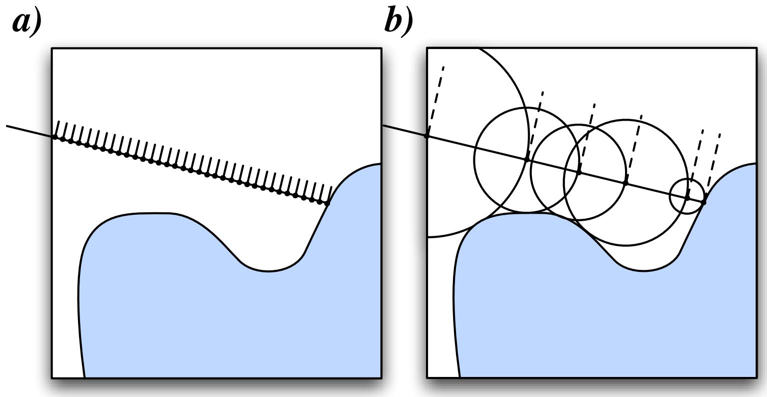


Figure 5.4: Illustration of sphere tracing compared to normal ray-casting. Traditional ray-casting uses many redundant samples of empty space while approaching a surface (a). Sphere tracing converges on the surface using only a few samples (b).

Unfortunately, there are many drawbacks to using a distance field in practice. The primary drawback is that the distance field must be recomputed every time the visible surface changes. In a medical visualization setting, the iso-surface or transfer function are often being altered. Although fast algorithms have been devised to calculate a distance field, none of them approach real-time for the size of a common medical dataset.

Our approach tries to keep the advantages of having a distance field while removing the requirement that it be updated each time the transfer function changes. We also address the ability to cast an entire group of rays at once. Although the overlapped min-max octree doesn't give us an accurate distance to the surface, it does provide a lower

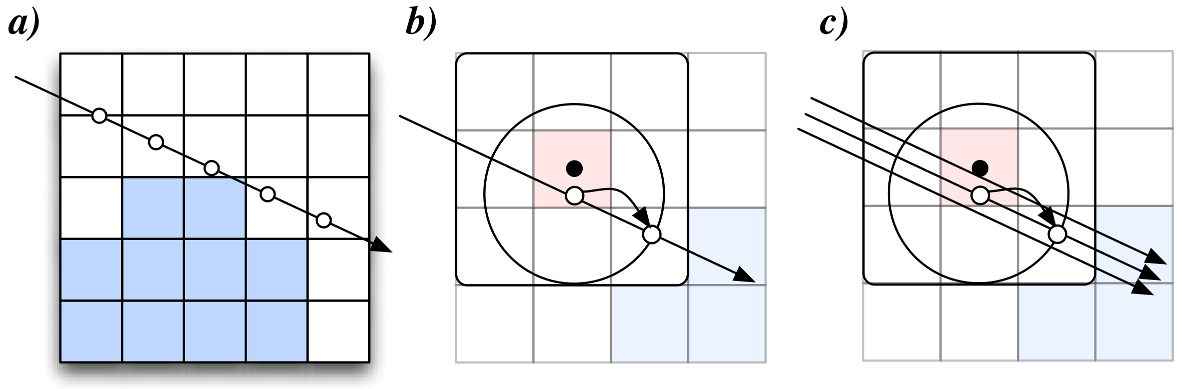


Figure 5.5: Illustration of frustum casting. Ray-Casting has no notion of the of the grid structure, resulting in a missed occupied cell (a). An overlapped min-max sample guarantees a minimum step distance from any sample taken within a min/max cell (b). We use this minimum distance to cast an entire frustum of up to 256 rays at once(c).

bound on the distance based on the amount of overlap and the sampling location within the octree. Figure 5.5 demonstrates an overview of how our algorithm uses samples from the min-max octree to guarantee that no visible cells are missed while casting, and then extends this to an entire packet of rays bounded by a frustum. We call this technique *frustum casting.*

While figure 5.5 demonstrates the idea behind frustum casting, we will now demonstrate how to calculate the exact distances and step sizes depending on the overlap, sampling location and frustum width of a packet of rays. Given the sampling location, we can very quickly calculate a lower bound on the distance to the surface. In the event that there is no visible material in the OMMO cell, we can find the distance to the closest border of the OMMO cell as follows:

$$R = min_{x,y,z}(w - abs(round(s) - s)) \tag{5.1}$$

Where w is the half-width of OMMO cell and s is the location where the sample was taken, assuming nearest neighbor sampling and that OMMO cells are centered on a grid at every integer. Given this radius R, of empty space around the sampling location, we can now determine a safe step distance for one ray (which is just R) or for many rays, given some information about the frustum bounding those rays.

When casting a large frustum of rays together, we wish to insure that none of the rays in the packet will hit any visible material during a large step. Given the radius R from above, this now becomes a simple problem of determining the step size that results in the corners of the frustum intersecting with the bounding sphere that we know is empty. Figure 5.6 illustrates this calculation for orthogonal and perspective projections.

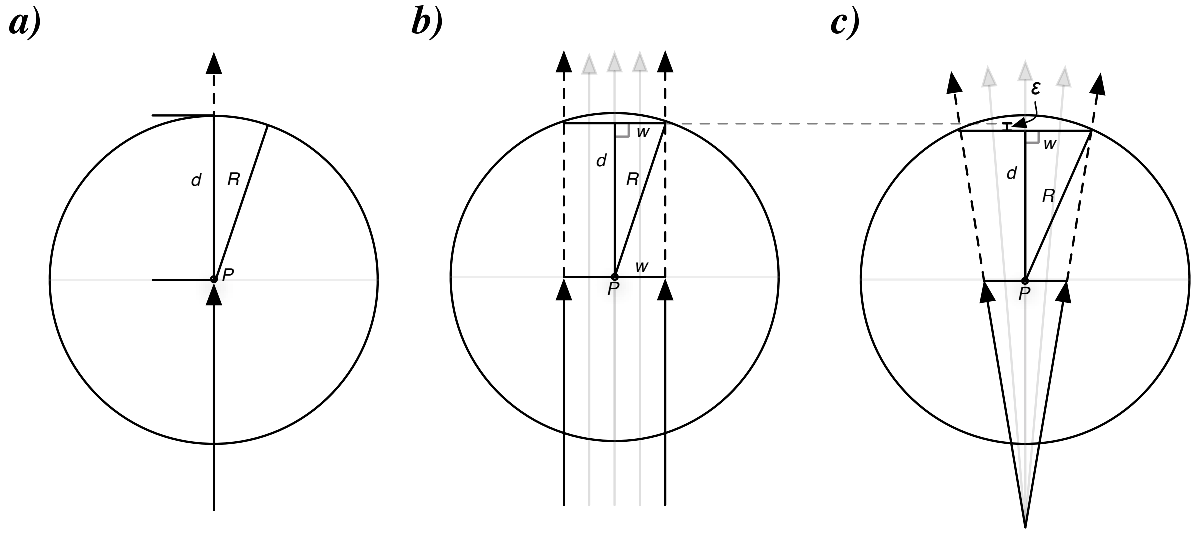


Figure 5.6: When casting only one ray (a), the step size d is equal to the radius R of empty space. In frustum casting a conservative step size must be used to insure no rays exit the sphere. For orthogonal projection (b), we calculate the step size based on the half-width w of the packet and R using Pythagorean theorem. For perspective projection (c) we would have to solve a quadratic equation to find d at every step, so we simply approximate the step size assuming the step size from orthogonal projection and subtracting a small epsilon ϵ.

5.4.3 Hierarchical Frustum Casting

Now that we have devised a method to cast large packets of rays (or frustums) through empty space, we describe how we also apply this technique in multiple passes to progress from large packets to small packets and finally into individual rays.

Multi-pass GPU algorithms are used for a few reasons in GPU programming. The two main reasons are as an alternative to costly branching operations, and to pass output of one *stage* of the algorithm to another stage. Stages are usually defined to break an algorithm into groups of repetitive operations with well defined inputs and outputs. In our case, each pass of the algorithm casts a specific size of ray packets, and the output is the distance at which those packets intersect visible material. Each successive pass uses smaller packets which start where the larger packet left off (Figure 5.7). These passes can be visualized as grey-scale images representing the depth reached by each packet (Figure 5.8).

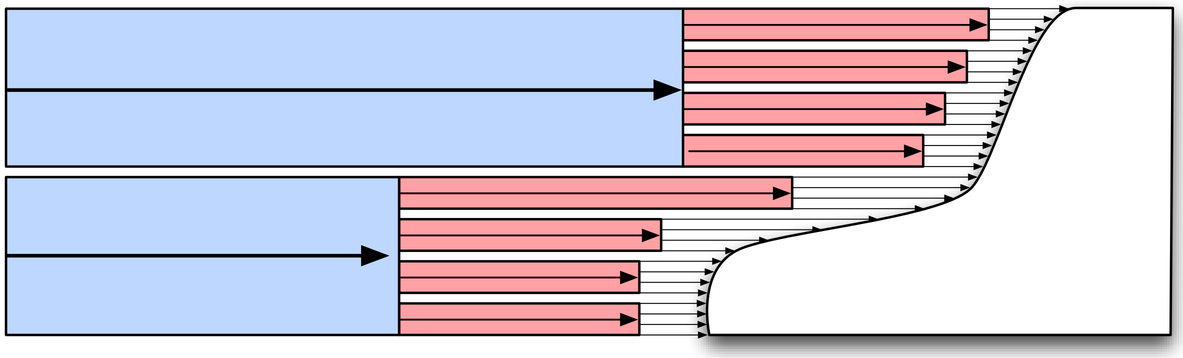


Figure 5.7: Illustration of multi-pass frustum casting with two frustum casting passes (blue and red) followed by ray-casting (a). The output from each pass are saved in a buffer as input to the next pass. In 3D, each blue and red frustum would represent 256 and 16 rays respectively.

As described above, we can now cast large packets of rays through the overlapped min-max grid by simply sampling and adjusting the step size based on some very simple calculations. By using multiple passes, we can also work from very large coarse packets, to finer packets and finally to individual rays. One remaining issue we have yet to discuss is how we choose which level of the octree to use. Each level in the octree will only facilitate up to a certain step size based on the overlap used in that level. Optimally we

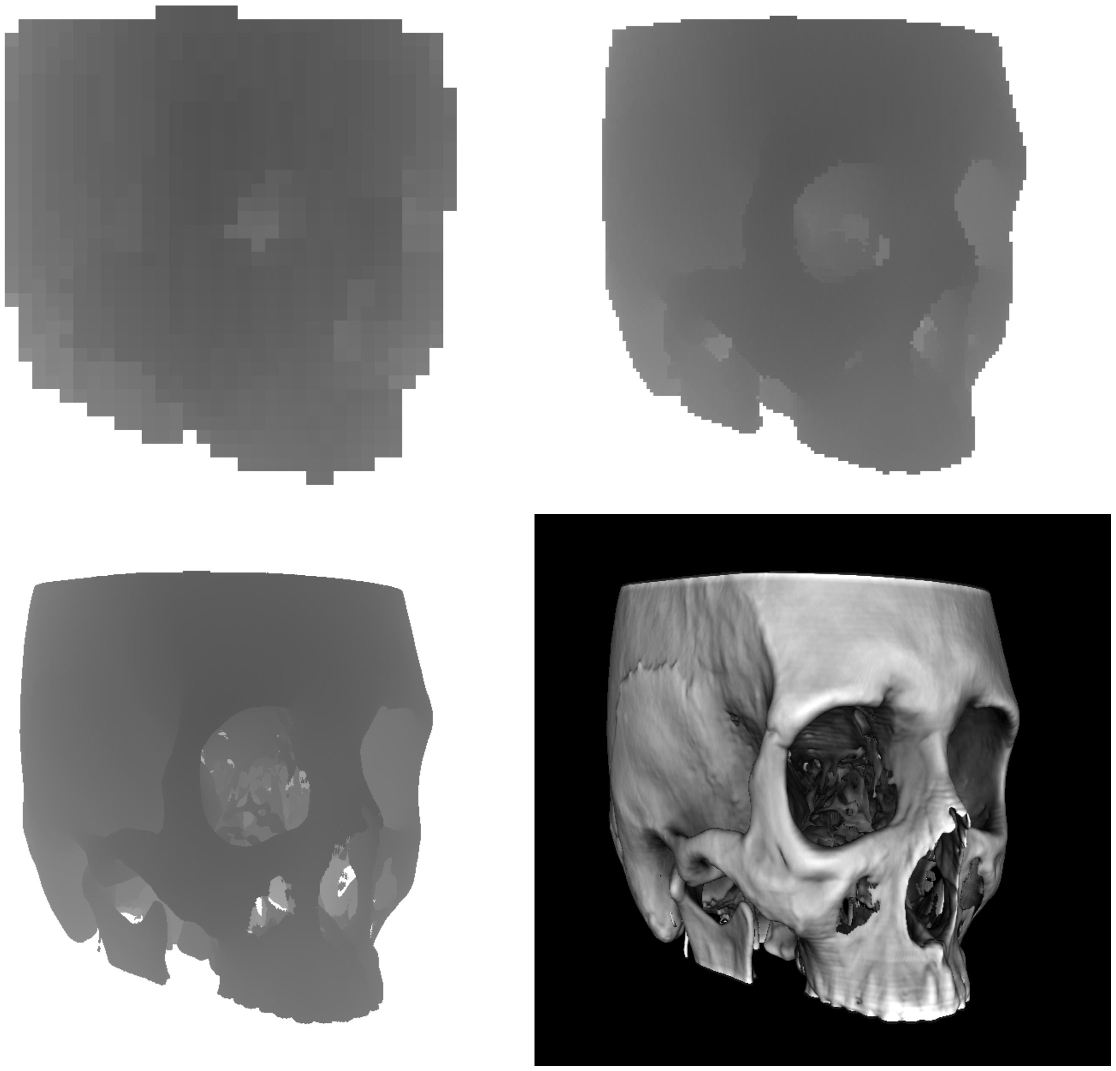


Figure 5.8: Each pass from figure 5.7 can be visualized as a grey scale image representing depth. The top-left and top-right images represent the first two passes of 256 and 16 ray packets respectively. The bottom-left represents the final depth of the iso-surface, and the bottom-right represents represents the final rendered image using a single iso-surface rendering with phong lighting and ambient occlusion.

would like to start at a higher level in the octree initially to facilitate larger step sizes, until we get closer to the surface (like in a standard octree traversal). While we could add extra logic to traverse up and down the tree, this would result in requiring a stack or at minimum extra branching operations.

To avoid complicating the casting procedure, we instead devised a new way of deciding which octree level should be used. Rather than traversing the tree and letting the octree level determine the step size, we reverse this and instead chose a reasonable step size and let that step size determine the octree level needed to achieve that step size. As we can

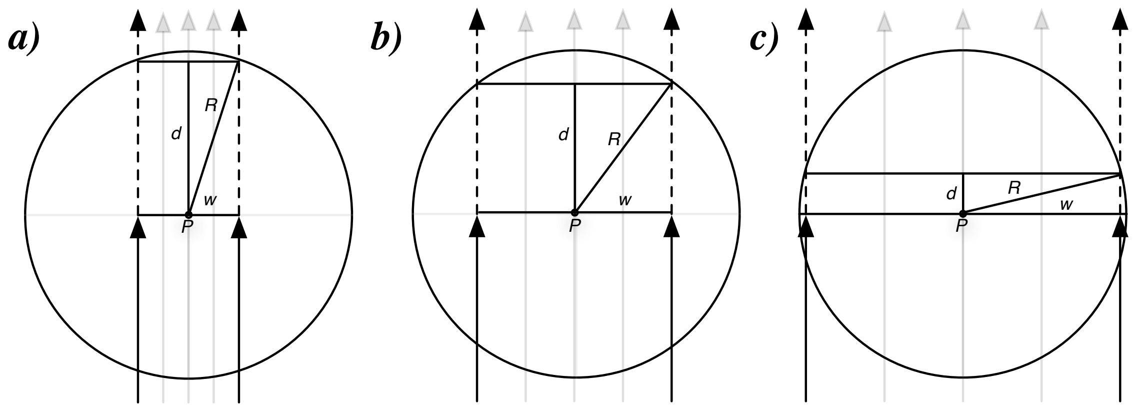


Figure 5.9: Illustration of how the step size varies according to the packet's size. Since this is a quadratic relationship, the step size decreases slowly at first (a,b) but very quickly as the packet size approaches to the sphere's radius (c).

see in figure 5.9, as the packet width increases relative to the size of OMMO cell, the maximum step size decreases until it finally reaches zero. Thus, we choose the octree level based on the packet size. Contrary to the intuition that we should try to maximize the step distance by choosing a higher octree level, we actually try at all times to choose the lowest octree level possible. The reason for this is, while a higher octree level may provide a larger potential step size, it also increases the chances that visible material will be found in the min-max range and that the entire packet will have to terminate and break into smaller packets. Thankfully, by grouping rays into very large packets, the large width of the packet will result in a choice of a high octree level. Thus, large packets will take large steps and smaller packets will take smaller but more accurate steps that get closer to the final surface.

Since the octree cells (and the amount of overlap) increase by powers of two at each level, the formula for choosing the octree level is quite simple. In fact, since the octree is stored in a mip-mapped 3D texture on the GPU, we use hardware accelerated texture sampling instructions to make the choice of octree level for us. Assuming we want a minimum step size of d and have an overlap percentage of O, we can calculate the octree level that provides this step size from figure 5.9 and the Pythagorian theorem as follows:

$$level = \lceil log_2(\frac{\sqrt{d^2 + w^2}}{O})\rceil \tag{5.2}$$

Conveniently, the mip-map level computed automatically by graphics hardware is a similar calculation:

$$level = round(log_2(max(dx, dy)) \tag{5.3}$$

where dx and dy are also usually computed automatically as the partial derivatives of the texture coordinates relative to screen space coordinates, but can also be specified explicitly. Recall from figure 5.9 that the minimum step size should be specified relative to the packet half-width. We therefore let:

$$d = \alpha w \tag{5.4}$$

Then, by simplifying and reducing to the form of equation 5.3 we get:

$$\begin{aligned} level &= \lceil log_2(\frac{\sqrt{\alpha^2 w^2 + w^2}}{O}) \rceil \\ &= \lceil log_2(\frac{\sqrt{(\alpha^2 + 1)w^2}}{O}) \rceil \\ &= round(log_2(w\frac{\sqrt{\alpha^2+1}}{O}) + 0.5) \\ &= round(log_2(\beta w)) \qquad\qquad (\text{ let } \beta = \sqrt{2}\frac{\sqrt{\alpha^2+1}}{O} \text{ }) \end{aligned} \tag{5.5}$$

Thus by letting $dx = dy = \beta w$ the hardware will compute the correct octree level to guarantee a step size of α times the packet width from the current location. Since α and O are constant for the duration of the ray-cast we can pre-calculate β as a constant for the pixel program. At this point we could determine the level of the octree, the amount of overlap and apply equation 5.1 to find the exact safe step distance, but all these extra instructions can only increase the step size by a maximum factor of two (assuming overlap equal to or greater than the cell size). Instead, we found it was much faster to let the hardware compute everything for us and simply use the minimum step size.

We then need only keep track of the frustum half-width of the packet (w). Conveniently, the packet width either stays the same in the case of an orthogonal projection, or increases linearly as the ray travels away from the origin of a perspective projection. To generalize this calculation, we calculate the initial frustum width from the distance between neighboring rays at the ray origin (eye, or front plane), and calculate the rate of change of the frustum width from the difference in the direction vectors of neighboring rays. This is very simple on the GPU using the derivative instructions, which compute the partial derivative of any value relative to the screen x and y coordinates.

Program 3 Shader code for casting a packet through the OMMO

Initialize Ray Packet (Frustum)

```
...
//Calculate half-width of frustum cast by this pixel
float rayHalfWidth     = length( ddx( rayOrigin.xyx ) )    * SQRT_2 / 2.0;
float rayHalfWidthRate = length( ddx( rayDirection.xyz ) ) * SQRT_2 / 2.0;

//Store width/rate in position/direction w coordinate
rayPos.w        = rayHalfWidth;
rayDirection.w  = rayHalfWidthRate;
...
```

Cast Ray Through OMMO

```
...
while ( !rayFinished )
{
    ...
    stepLength = rayPos.w * alpha;
    rayPos += rayDir * stepLength;
    dx = rayPos.w * beta;
    rayFinished |= IsVisible( texture3dGrad( OMMO, rayPos.xyz, dx, dx ) );
    ...
}
...
```

5.4.4 Accelerating the Min-Max Visibility Test

Until now we have assumed that given the minimum/maximum value and an arbitrary transfer function that we can easily test whether a given cell contains any visible voxels. While there are existing approaches to perform this test in the literature, they have been designed to work with proxy geometry solutions. As such we came up with a few extra alternatives to compute this test very quickly that trade-off time, accuracy and pre-computation time. We will briefly discuss these alternatives here. See figure 5.10.

The only approach we found in the literature involves creating a one-bit 2D lookup table indexed by the minimum and maximum values (figure 5.10c). Using this approach, every time the transfer function is changed the lookup table must be recomputed and uploaded to the GPU. The min/max visibility test then becomes a simple lookup into the table using the min/max as indices. Obviously, even though there is only one bit for each combination of min/max values, this results in a massive texture if an accurate range of 12-bit or 16-bit values is to be used. In addition, if ray-casting is to be used rather than proxy geometry, this entire texture must be uploaded to the GPU every time the transfer function is changed.

Beginning with this technique, the first thing we addressed was eliminating the large 2D lookup table, and subsequent CPU/GPU bottleneck of uploading a large texture to the GPU. As a first step, we noted that the min/max visibility test is equivalent to checking if the *sum* of the alpha values in the [min,max] range of the transfer function is non-zero. This is obvious as any non-transparent value will result in a non-zero sum, and indicates that there is non-transparent material in that range. To quickly find the sum of the [min,max] range, we used a common data structure called the *summed-area-table* (SAT), or *integral function.* The value in the summed area table is simply the sum of all values before it in the function itself:

$$SAT(x) = \sum_{x'=0}^{x} f(x') \tag{5.6}$$

Using the integral function, the sum or average over any range can be computed as:

$$Sum(x_1, x_2) = SAT(x_2) - SAT(x_1) \tag{5.7}$$

$$Ave(x_1, x_2) = \frac{SAT(x_2) - SAT(x_1)}{x_2 - x_1} \tag{5.8}$$

Thus, using the SAT, we can replace one lookup into a large 2D texture by two lookups into a very small 1D lookup texture (figure 5.10) . While this method worked very well, it introduced an extra texture lookup in the inner loop of the ray-caster. To even further reduce the computation time, we created an approximate approach that performed almost identically to the SAT but required only one lookup.

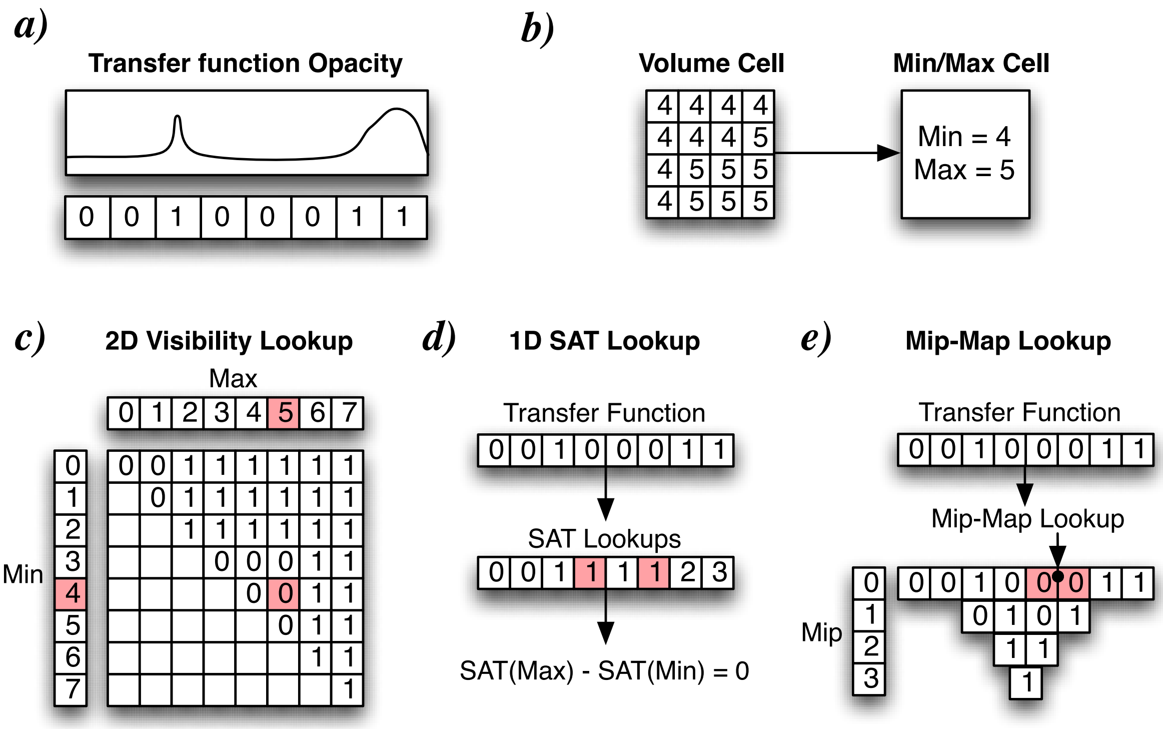


Figure 5.10: For a given transfer function(a), and a min/max cell(b), the min/max visibility test is usually performed with a one-bit 2D lookup table(c). To save memory, speed and pre-computation time, we have devised two new techniques. We use either two lookups into an SAT lookup table (d) or one bilinear lookup into a mip-mapped transfer function(e).

To reduce the visibility test to one memory lookup, we looked at the effect of mip-mapping the transfer function. The result of mip-mapping a texture is a hierarchy of images that are each half the size of the last. Mip-maps are usually used to reduce *aliasing* of texture lookups when a triangle becomes smaller and smaller on the screen. Furthermore, mip-mapping is built into the GPU pipeline and adds no extra cost when sampling. The mip-map is usually chosen automatically based on the local gradient (rate of change) of the texture coordinates in screen space. The mip-map is chosen such that neighboring pixels in the chosen mip-map will be large enough to enclose the texture region occupied by a screen pixel.

To reduce the visibility test to one bilinear lookup, we simply chose our sample carefully such that:

$$x = \frac{Max + Min}{2} \text{ (sample location)} \tag{5.9}$$

$$dx = Max - Min \text{ (texture coordinate gradient)} \tag{5.10}$$

$$mip = \lceil log_2(dx) \rceil \text{ (mip-map - computed by hardware)} \tag{5.11}$$

The result will be a linear blend of two values which enclose the entire [min,max] range of the transfer function. The ratio of the linear interpolation is irrelevant as we only need to know that every value in the range contributed to the final value. If any non-zero alpha is found in the range then the value of the lookup with be non-zero and we know that the space is occupied. It should be noted that although this sample encloses the $[min, max]$ range, it is also inaccurate by as much as 50% of the range, resulting in some false-positive visibility tests. Thankfully this inaccuracy grows as the range grows, so that there is only a large error when the range of image values is very large. In this case there is very likely be something visible anyway, and we found the overall effect of this error to be negligible with the mip-map lookup always outperforming both the 2D lookup and the SAT lookup. If a mip-map lookup is to be used exclusively, we can optimize this lookup even further by storing x and dx (average and difference) directly instead of storing the min and max values and computing x/dx.

One remaining issue that should be mentioned is numerical precision. Both the SAT test and the mip-map test require sufficient precision in order to work. Otherwise a small non-zero alpha value can be lost when it is blended with many other values. To address this, for both approaches we use 16bit precision to represent the 8bit transfer

Program 4 Final code for visibility test using a mip-mapped transfer function. This takes only one memory lookup and a few instructions, or simply one memory lookup if ave/diff are stored directly instead of min/max.

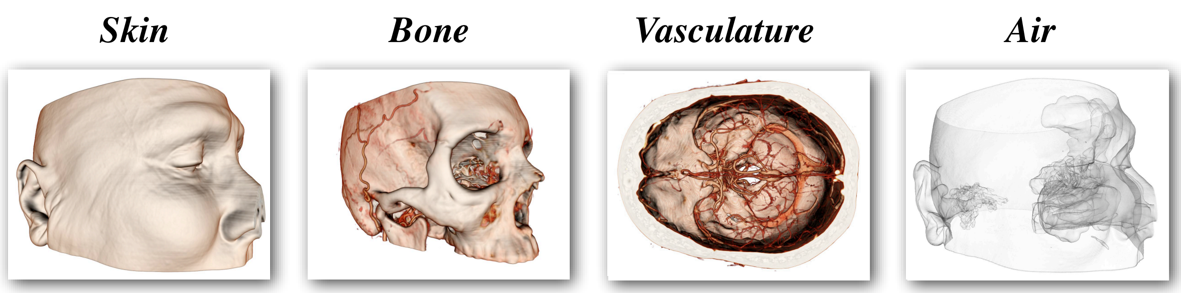
```
bool IsVisibleMinMax( float min, float max )
{
    float ave  = (min + max) * 0.5;
    float diff = max - min;
    return texture2dGrad( sampler, ave , diff, diff ) != 0.0;
}

bool IsVisibleAveDiff( float ave, float diff )
{
    return texture2dGrad( sampler, ave , diff, diff ) != 0.0;
}
```

function values. Another alternative is to pre-preprocess the alpha values into a binary representation of visible or invisible (1 or 0) since we don't care about the specific opacity when skipping empty space.

5.5 Results

While the final result of optimizations is improved performance for a given image, this is slightly complicated to measure, as different data-sets have different amounts of empty space and benefit differently from empty space skipping. Since *any* empty space skipping procedure is likely to increase performance substantially on most data-sets, we thought it would be informative to measure both the increase in performance, as well as the total time spent on the empty space skipping passes themselves. This is especially informative in difficult cases where little can be done to improve performance. In such situations space skipping optimizations can result in a performance *decrease* equal to the overhead incurred by the additional space skipping steps. In some prior work this time can account for as much as 30% of the frame time in algorithms while are already costly such as DVR.

We have chosen a few views to highlight both the best and worst case rendering performance. Table 5.11 shows the total rendering time with and without our optimizations as well as the total time spent in the empty space skipping passes. It is important to note that the empty space skipping passes never exceed 2ms (500fps), which we feel is a reasonable cost even if it results in no speed-up in a worst-case dataset. Figure 5.12 illustrates the results in a few graphs. As we expect, rendering performance is consistently improved by a factor of around 15 times with the exception of views with large amounts of internal empty space.



Empty Space Skipping Time (milliseconds)			
View	16x16 Packets	4x4 Packets	16x16 + 4x4 Packets
Skin	< 0.5ms (2000fps)	1.5ms (670fps)	< 1.0ms (1000fps)
Skull	< 0.5ms (2000fps)	1.6ms (630fps)	< 1.0ms (1000fps)
Vasculature	< 0.5ms (2000fps)	2.2ms (450fps)	< 1.0ms (1000fps)
Air	< 0.5ms (2000fps)	1.5ms (670fps)	< 1.0ms (1000fps)

Iso-Surface Rendering Time (milliseconds)			
View	Not Optimized	Optimized	Speedup
Skin	115.5ms (8.7fps)	7.5ms (133fps)	15.4x
Skull	130.9ms (7.7fps)	8.1ms (123fps)	16.2x
Vasculature	201.3ms (5.0fps)	11.1ms (90fps)	18.1x
Air	110.4ms (9.1fps)	7.5ms (133fps)	14.7x

DVR Rendering Time (milliseconds)			
View	Not Optimized	Optimized	Speedup
Skin	190.2ms (5.2fps)	12.6ms (79.3fps)	15.3x
Bone	220.0ms (4.5fps)	12.9ms (77.5fps)	17.2x
Vasculature	322.8ms (3.1fps)	34.0ms (29.4fps)	9.5x
Air	410.3ms (2.4fps)	186.7ms (5.5fps)	2.2x

Figure 5.11: Performance tables for our empty space skipping passes, iso-surface rendering and DVR rendering for the skin, bone, vasculature and air views. Empty space skipping time is especially noteworthy. Performance is generally improved by at least 1500% except in cases where there is significant internal empty space. It was difficult to measure GPU times under 1ms, thus we have provided upper bounds only when this is the case. These tests were all performed on a GeForce 8800 card.

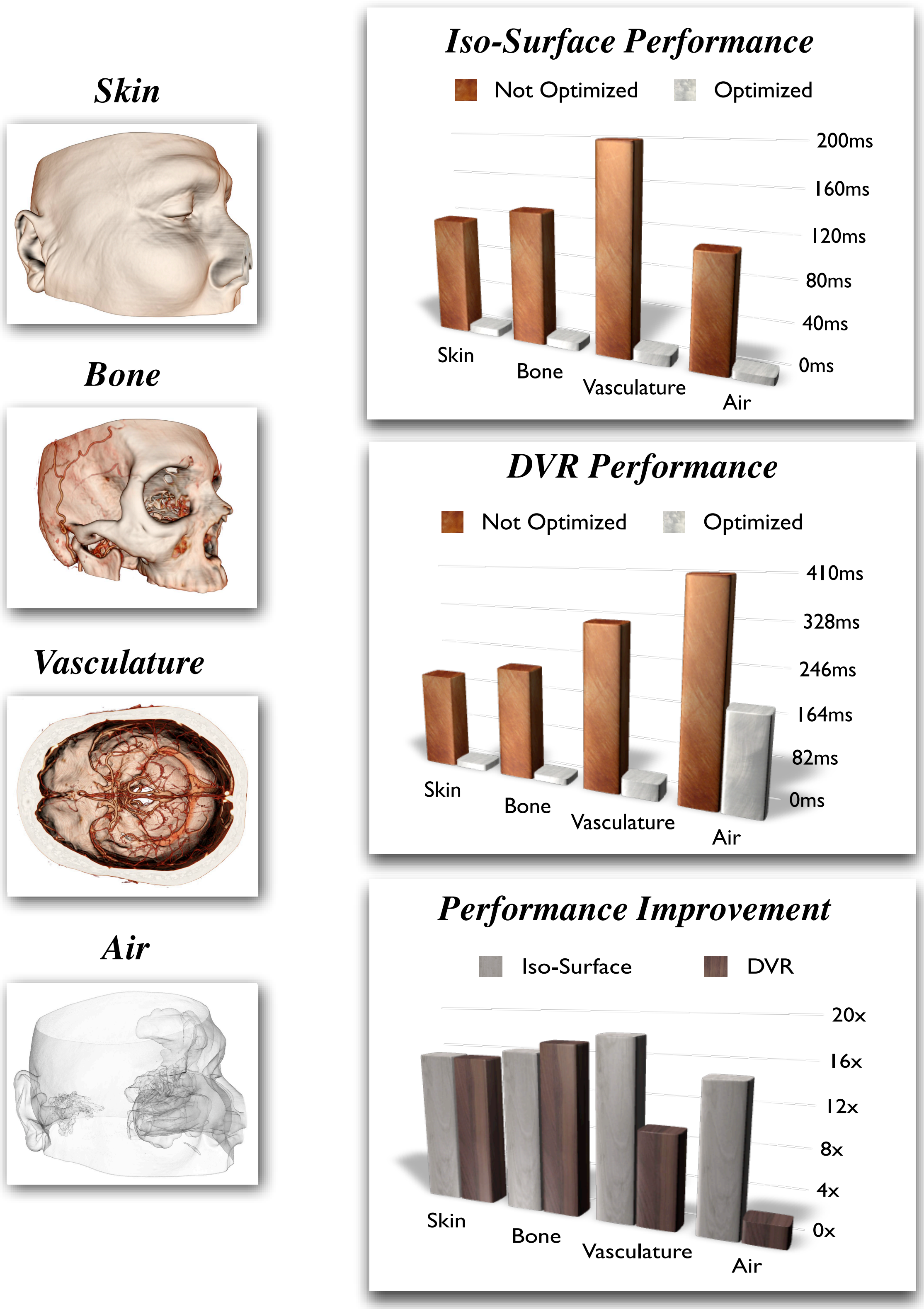


Figure 5.12: Illustrative graphs from the results in figure 5.11. Iso-surface rendering performance is consistently improved since there is no internal empty space. DVR improvement drops off in data where large amounts of internal empty space is present. These tests were all performed on a GeForce 8800 card.

5.6 Discussion

We have presented a novel approach to optimizing GPU ray-casting which addresses many of the concerns with prior GPU empty space skipping approaches. Rather than simply porting existing ray-tracing algorithms to the GPU - which have shown to very poorly utilize the GPU's total arithmetic throughput - we approached the problem by adapting algorithms that are known to perform well on the GPU to the new problem of empty space skipping.

Compared to optimization approaches which have become very complicated and require large amounts of CPU/GPU communication, our approach runs entirely on the GPU and can fit into a conventional ray-casting system by simply adding a few extra shaders and one extra volume texture (the OMMO). By utilizing all possible hardware accelerated instructions, the empty space skipping shaders used in our system boil down to just a few carefully chosen instructions. We have also presented several optimizations to transfer function lookups and min-max visibility tests which will benefit many other approaches as well.

The one area we wished to address further was the issue of skipping over large amounts of internal empty space. We found that we could easily gain another 2-4X speedup for shell-like datasets by inserting empty space skipping instructions into our main compositing shaders. However, since this cost is directly associated with each compositing ray, rather than being amortized over large groups of rays, we found this had a considerable impact on worst-case performance. However, as discussed in section 5.2 this cost can potentially be mitigated by adding a detection step and choosing the most appropriate method.

We also looked briefly into extending coherent frustum casting to find regions of empty space within the volume. We found this to be the most promising approach, but found this also had a large impact on worst-case performance, as the entire volume needs to be searched for empty internal regions, instead of just stopping at the first visible cell. This time is largely wasted if volume is largely opaque and compositing rays actually terminate near the surface. Again, a detection step could be employed to turn this feature on or off, but we would like to investigate adding a conservative ray-termination step into the frustum casting passes that would prevent this wasted effort while still providing the same benefits.

Chapter 6

Dynamic Ambient Occlusion and Soft Shadows

This chapter discusses one of the main contributions of this thesis, which is a novel method for approximating dynamic ambient occlusion and soft shadows in iso-surface volume rendering. This chapter also reviews additional relevant literature. We first discuss the importance of shadows on our perception, followed by a review of current shadowing methods for volume rendering. We then turn to our technique and discuss the reasoning behind and the implementation details of our approach. We conclude this chapter with a discussion of our technique and future work. This chapter was the basis of a publication for Volume Graphics 2008, which can be found in Appendix A.

6.1 Motivation

Cast shadows are known to play a key role in human perception of the 3D world. The first thorough analysis of shadows was likely performed by Leonaro Da Vinci [101] (see figure 6.1). The early work of Lambert is also worth mentioning for its description of the geometry underlying shadows [58]. To qualitatively understand the importance of shadows in our perception of the world, several studies and experiments have been conducted to understand how shadows shape our perception and understanding of a scene. Through these experiments, the importance of shadows has been demonstrated in understanding the position, size and geometry of both the shadow caster and shadow receiver (see figures 6.2 and 6.3). Hubona *et al.* [43] discuss the role of shadows in general 3D visualization. Wanger *et al.* [105] study the effect of shadow quality on the perception of object relationships. Kersten *et al.* [47] demonstrate that adjusting the motion of a

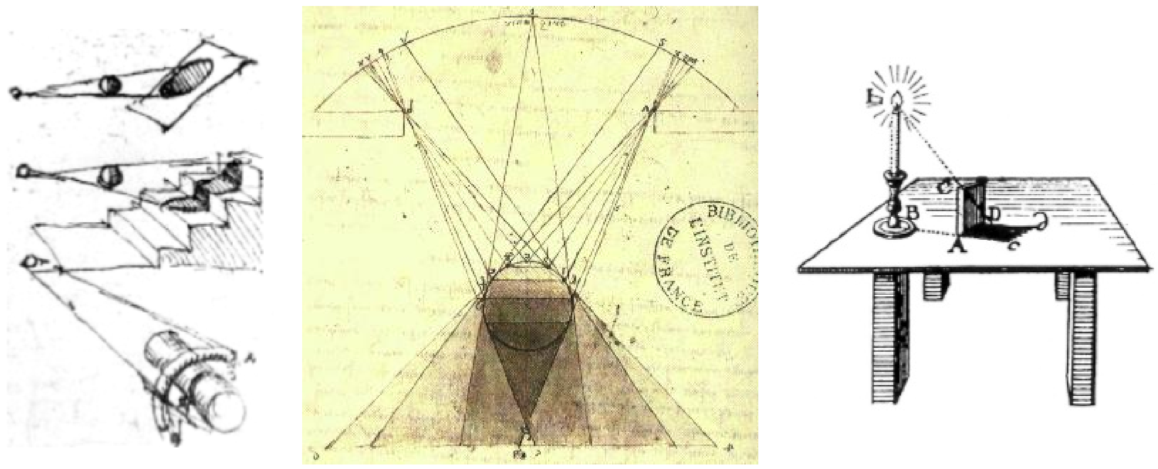


Figure 6.1: Left/Center: Early study of shadows by Leonardo da Vinci [101]. Right: Study of the geometry of shadows by Lambert [58].

shadow can dramatically effect the apparent trajectory of a shadow casting object. For the interested reader, a comprehensive discussion of real-time geometry based shadowing algorithms with references to their perceptual importance can be found in [37].

These experiments have convincingly established the importance of shadows in computer graphics applications. Since then, advances in computer graphics technology and the development of consumer graphics processing units have made real-time 3D graphics a reality. However, incorporating shadows and especially realistic soft shadows into these applications still remains difficult and has generated a large amount of research effort.

The most traditional method of computing shadows is with ray tracing using a point light source. Using this method, a ray is traced from each surface point back to the point light that illuminates the surface. If an object obscures the path of the ray, the surface is rendered without the light's contribution. Unfortunately, shadows from simple point sources produce stark discontinuities which aren't present under normal lighting conditions. The depth cues provided also vary depending on the viewing direction. For example, if the camera location aligns with the light location all the shadows become hidden behind the objects and no extra cues are provided. Methods such as shadow

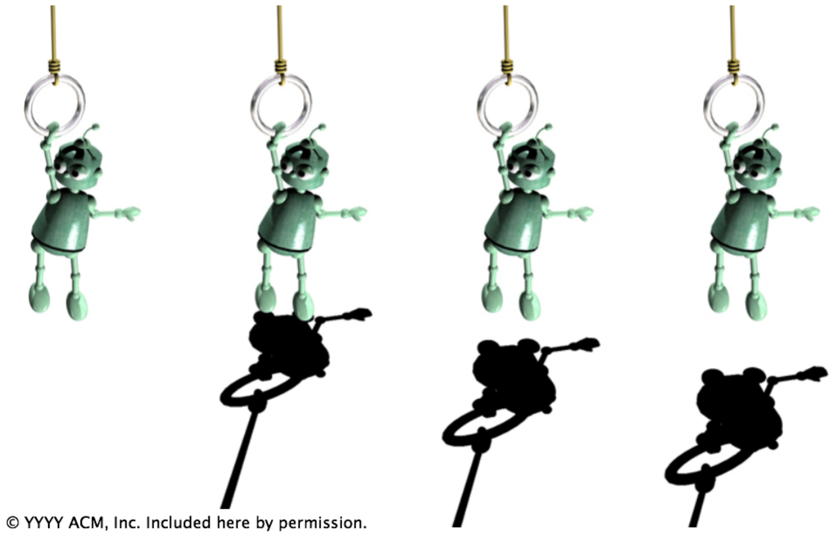


Figure 6.2: Illustration of the importance of shadow in perceiving relative position. Without shadows an object appears to float above the ground at an unknown distance; shadows clarify the relative locations [37].

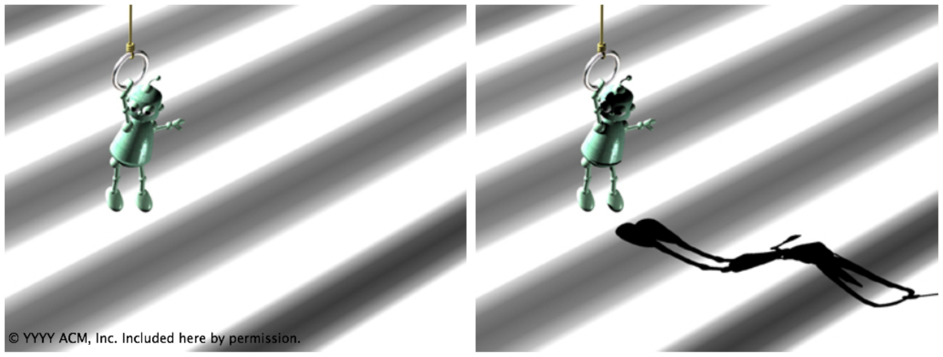


Figure 6.3: Illustration of the importance of shadows in perceiving structure. With shadows the shape of the ground becomes clear [37].

maps [107] and shadow volumes [18] can be used to accelerate point-lighting in real-time applications and produce similar results.

More realistic shadows are provided with more complex models such as *ambient occlusion* or *global illumination*. It has been shown that these realistic approaches provide better perception of many shapes than with simple point lighting. Ambient occlusion simulates light arriving equally from all directions or "light on a cloudy day" and is also referred to as *uniform diffuse lighting*. Global illumination simulates lighting from complex area light sources as well as diffuse inter-reflections and caustics. While there have been methods to compute good approximations of these lighting methods for a long time, traditional methods have always had very high computational costs. Usually the calculation involves a monte-carlo simulation of hundreds or even thousands of rays as opposed to the single ray used for a point or directional light source (see figure 6.4). In order

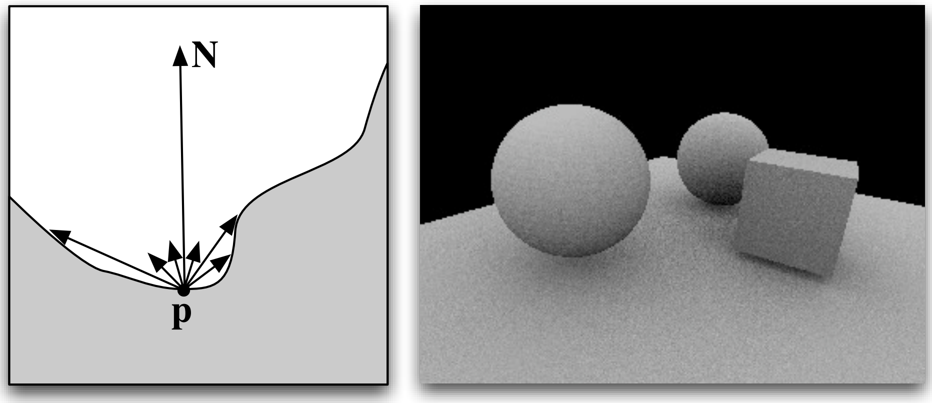


Figure 6.4: Left: Illustration of calculating ambient occlusion using monte-carlo ray-tracing. Rays are chosen randomly in the hemisphere above a surface point. The percentage of occluded rays represents ambient occlusion. Right: Example using 36 randomly distributed rays. Notice the noise artifacts due to under-sampling.

to capture these effects in interactive applications, many methods based on the theory of light transfer have been developed to enable the pre-computation of ambient occlusion and diffuse inter-reflection. Some even allow for arbitrary modification of light and camera parameters. However, the application of these approaches to changing geometry requires a new pre-computation for any change.

To address these limitations, approximations have been proposed which are not strictly physically motivated, but lead to visually convincing and plausible results while remaining feasible to compute in real-time. For example, Bunnell *et al.* [13] represent geometry as a hierarchical tree of discs for which simple form factors can be calculated to approximate occlusion. Shanmugam *et al.* [92] and Mittring *et al.* [73] even go so far as to compute occlusion entirely from the depth buffer as a post-process (see figure 6.5). While these approximate methods are far from physically correct, they add a surprising amount of realism and accurate depth cues to the scene.

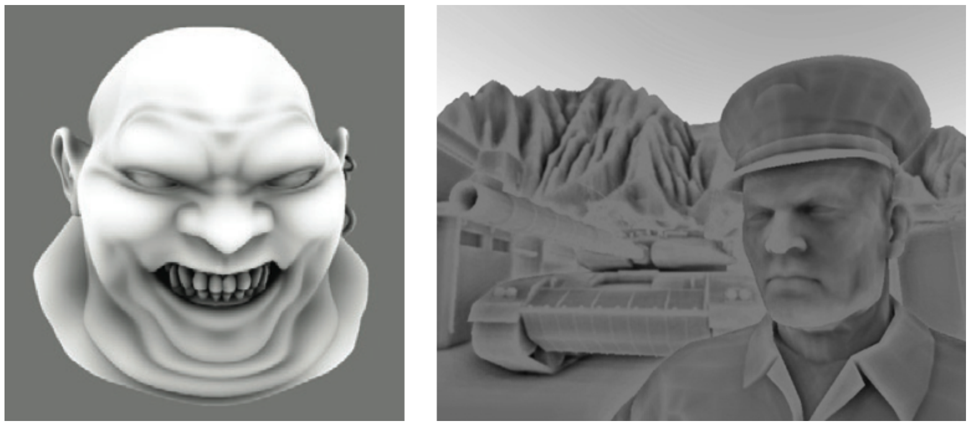


Figure 6.5: Left: Dynamic ambient occlusion from [13] using a hierarchy of discs to represent geometry. Right: Screen-space ambient occlusion, which uses only the depth buffer [73]. Although both methods are not physically correct they produce plausible results which still improve the depth perception of the scene.

6.2 Shadows in Volume Rendering

Due to the added computational complexity involved with computing shadows most medical volume rendering applications will only utilize a local illumination model due to its low computational cost. This involves illuminating the volume using one or more point light sources. In a local lighting model the light at each point in the volume is calculated as the sum of diffuse and specular components calculated from a bidirectional reflectance distribution function (BRDF) model. A BRDF model such as the popular Blinn-Phong model [7] provides a means to calculate the amount of locally reflected light based on the directions of the light source, L, the viewer, V, and the surface normal (or gradient), N.

Local illumination methods provide good perceptual cues to the *orientation* of a surface within the volume, due to the diffuse $N \cdot L$ term. Surfaces are bright if lit from directly above, and dark if illuminated from a steep angle. However, as discussed above local illumination methods provide poor cues to the *relationships* between neighboring surfaces. It can be difficult to tell whether a neighboring surface is above or below an adjacent surface. Due to the extra perceptual information they can provide, adding shadows to real-time volume rendering applications has resulted in significant research effort.

Unfortunately, these efforts are complicated by the fact that different volume rendering methods exist and have an impact on how and when light can be propagated. As already discussed, the two primary volume rendering styles are iso-surface and direct volume rendering (DVR), while the two primary volume rendering methods are texture slicing and ray-casting. Simple point light shadows are easily added to iso-surface ray-casting by casting an extra ray from the surface point back to the light. Soft shadows, and shadows within DVR ray-casting have proven much more difficult. A number of more advanced lighting techniques have been developed for slice-based DVR using a technique called half-angle slicing [51]. Half-angle slicing keeps track of rays from both the light's and eye's point of view and advancing them on the same slicing plane. The algorithm cleverly switches between front-to-back and back-to-front compositing depending relative angle of the view and light vectors. Unfortunately, slice-based renderers are known to suffer from several rendering artifacts and are very difficult to optimize. Conversely, ray-casting is known to produce artifact free images and, as we have shown, can be optimized.

To maintain the quality and performance benefits of ray-casting, considerable research effort has been spent on incorporating efficient lighting algorithms into ray-casting engines. Stewart *et al.* pre-compute diffuse ambient lighting which they call *vicinity shading* in a separate volume which is used as a lookup during iso-surface rendering [96]. They accelerated the computation using a 3D version of Bresenham's line drawing algorithm [11]. Wyman *et al.* and Banks *et al.* furthered this technique by pre-computing or lazily computing global illumination lighting in a separate volume [5,108]. Unfortunately, pre-computed lighting effects can result in significant aliasing artifacts (see Figures 6.7, 6.8 and 6.6) unless at least twice the original resolution is used, resulting in an 8X-100X memory footprint depending on the type of pre-computed lighting [108]. Hadwiger *et al.* [33] pre-compute *deep shadow maps* that represent a compressed attenuation curve from the light's point of view and can represent area light sources. Unfortunately, these maps must be recalculated and compressed each time the light is moved or transfer function is altered.

Very recently, approximate lighting techniques are also starting to show up in volume rendering research. Desgranges *et al.* use a summed area table of the volume's opacity to quickly perform variable width blurring operations to approximate ambient occlusion [22]. Unfortunately the summed area table must be recomputed whenever the transfer function or isosurface is changed. Ropinski *et al.* compute approximate ambient occlusion by quantizing all the possible combinations of neighboring voxels such that they can apply the transfer function dynamically [86]. This method can approximate ambient occlusion as well as color bleeding but suffers from a lengthy compression process and quantization artifacts during rendering.

6.3 Pre-Computed Volume Lighting

One of the key insights in our research was the identification of an inherent flaw in trying to naively pre-compute volume lighting in a separate volume. While one would expect the Shannon-Nyquist sampling rate of pre-computed lighting should be related to the resolution of the original volume, we demonstrate that it is actually related to the local image gradient at each voxel, making it significantly more difficult to pre-compute. Figure 6.7 demonstrates how ambient occlusion can change rapidly at neighboring image voxels, while figure 6.8 formalizes why this is happening and presents a worst case. Essentially, the number of iso-surfaces passing through a given voxel is related to the dynamic range of the image and the gradient magnitude at each voxel, rather than the resolution of the original voxel as one might expect, and the number of iso-surfaces determines the Shannon-Nyquist sampling rate.

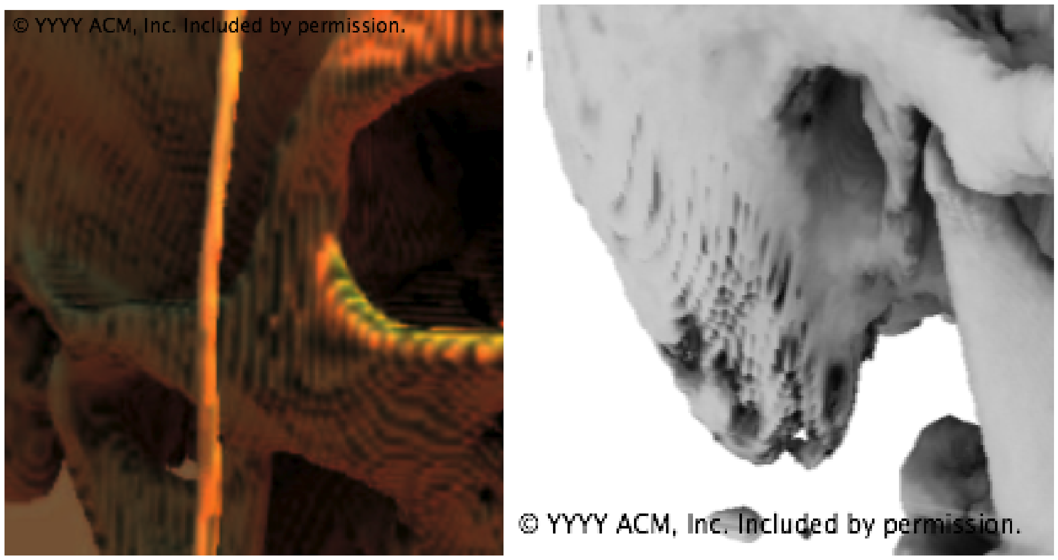

© YYYY ACM, Inc. Included by permission.
© YYYY ACM, Inc. Included by permission.

Figure 6.6: These images depict the result of aliasing when using a volume to pre-compute lighting (from [96, 108] respectively). This tends to occur in areas of high gradient magnitude, since those areas contain the most iso-surfaces.

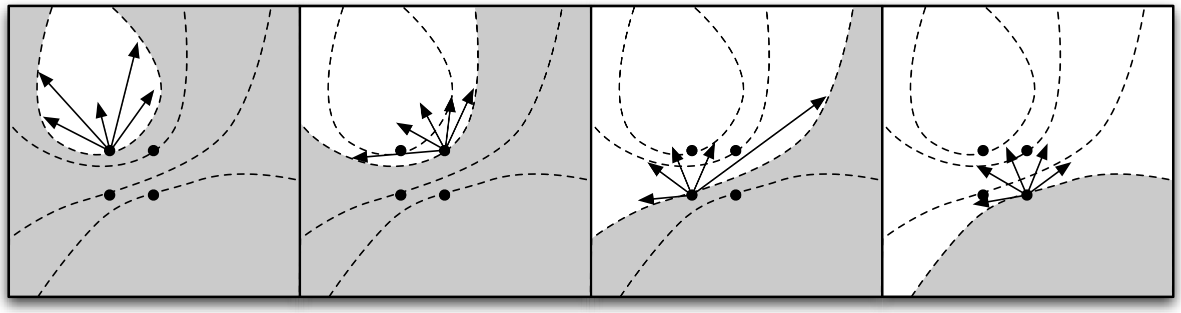


Figure 6.7: Illustration of how aliasing occurs in pre-computed volume lighting. Neighboring sampling locations can have highly different occlusion. The dotted lines indicate the iso-surfaces that are used to calculate occlusion for neighboring voxels. Aliasing occurs when trying to interpolate between occlusion for such different surfaces.

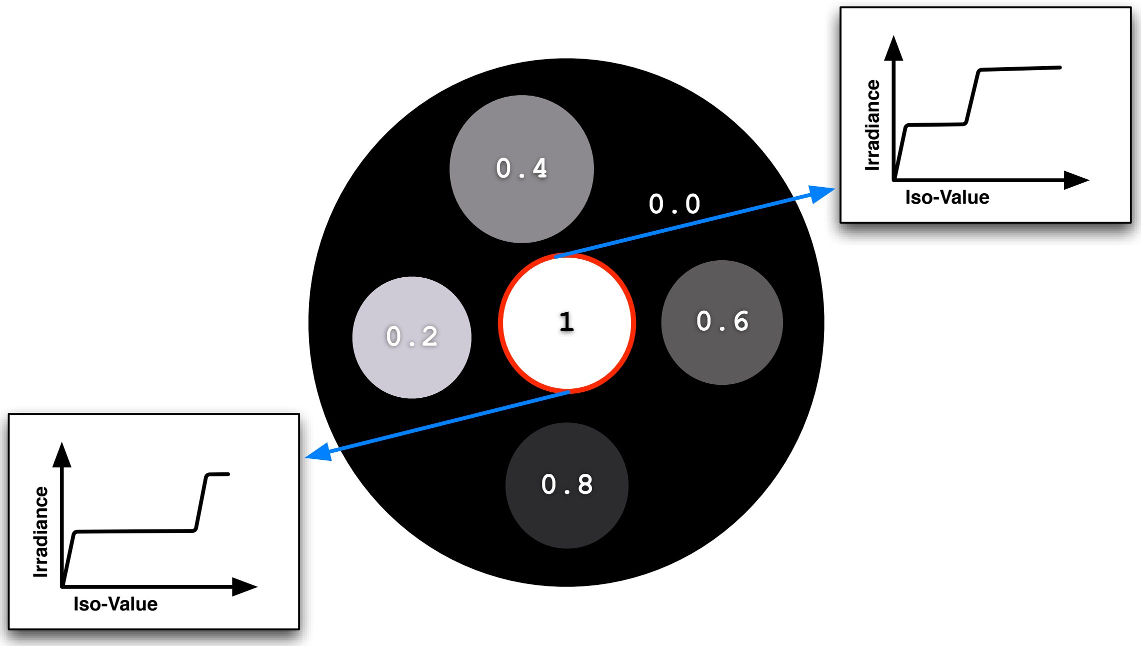


Figure 6.8: Illustration of a worse case volume where almost no resolution would be sufficient to eliminate aliasing; the one-voxel-thick red circle contains *all* iso-surfaces. The small graphs indicate the occlusion for the range of iso-surfaces at those locations. Our key insight is that the adequate sampling rate for pre-computed lighting is based on the local image gradient rather than the resolution of the image. The idea behind our technique is to try to approximate these occlusion graphs at each voxel.

6.4 Filterable Occlusion Maps

While we could utilize these insights into pre-computed lighting to design a better method for pre-computation, our goal is to take it a step further and also drastically accelerate the calculation of volume ambient occlusion. While meshes are stored in a sparse format as a collection of vertices and triangles, volumes are stored in a 3D image grid which is suitable for a host of image processing operations and volume measurements that aren't possible on meshes. Our goal is to very quickly preprocess the volume in such a way that we can quickly extract information about the number of occluding voxels without having to traverse hundreds or thousands of rays. Generally speaking, we wish to replace the geometric definition of ambient occlusion with a statistical approximation that can be computed using image-processing operations. We have focused on computing ambient occlusion in iso-surface rendering, but we explain how this method can be used in direct-volume rendering as well.

6.4.1 Ambient Occlusion

Recall from equation 4.5 that volume rendering integral used in most applications uses only local reflectance $q(p)$ to represent the light leaving a given point p in the direction of the eye. In order to account for complex occlusions from neighboring structures we would like to replace this with a new term $q'(p)$ that takes into account the irradiance arriving at a surface from all angles. This can be represented as:

$$q'(p) = q(p) \cdot \int_{\Omega} L(\omega) d\omega \tag{6.1}$$

where $L(\omega)$ is the radiance arriving from direction ω, and Ω is the set of directions above the surface point (where $N \cdot \omega > 0$ and N is the surface normal). Since we are interested in the diffuse reflection of the incoming irradiance, a *cosine-weighted* contribution of incoming radiance is often used:

$$q'(p) = q(p) \cdot \int_{\Omega} N \cdot L(\omega) d\omega \tag{6.2}$$

In this case $L(\omega)$ is represented as a vector with magnitude equal to the radiance. This equation can be approximately evaluated using ray casting by discretizing the domain Ω into k sectors of equal solid angle $\Delta\omega$.

$$q'(p) \approx q(p) \cdot \sum_{i=0}^{k} N \cdot L(\omega_i) \Delta\omega \tag{6.3}$$

Since we have chosen to focus on an iso-surface model, computing $L(\omega_i)$ is heavily simplified. If a ray ever enters the iso-surface then it immediately becomes fully occluded. After an occluding iso-surface is found, we can proceed to evaluate the amount of occlusion using an *all-or-nothing* method or a *partial occlusion* method. In the all-or-nothing method $L(\omega_i) = 0$ when ω_i hits an occluding voxel. Otherwise, $L(\omega_i) = 1$. In the partial occlusion method $L(\omega_i)$ uses the unobstructed distance to the first occlusion to determine the amount of occlusion using a basic linear or quadratic fall off function. The partial occlusion method works better in completely occluded spaces or tight spaces where the all-or-nothing method would result in complete occlusion.

6.4.2 Neighborhood Approximations

Computing the integral mentioned above would require casting many rays. Instead, we would like to make use of the discretized nature of the volume to compute occlusion. The basic assumption we rely on in doing this is that the percentage of occluding voxels surrounding the surface provides a good approximation to the percentage of rays that would be occluded while traversing through the same space. While this is not strictly correct in all cases, it is a commonly used assumption in approximate techniques [92] [73] where real-time performance is required.

Under this assumption, we can now define occlusion using the neighborhood around a surface point and define the irradiance at a point p as:

$$\frac{1}{|V|}\int_V L'(v)dv \tag{6.4}$$

Here, V is a volume above the surface we are evaluating, $L'(x)$ is now simply a binary function of the iso-surface c defined as:

$$L'(x) = \begin{cases} 1 & \text{if } x \leq c \text{ (not occluding)}, \\ 0 & \text{otherwise (occluding)}. \end{cases} \tag{6.5}$$

By discretizing the volume into k voxels we can express this discretely as:

$$\sum_{i=0}^{k} L'(v_i)\Delta v \tag{6.6}$$

Whereas the ray-based approximation from equation 6.3 is dependent on a hemisphere of angles Ω, our approximation in 6.6 is dependent only on a hemispherical or spherical volume V which lies above a surface point (see figure 6.9).

6.4.3 Filterable Occlusion Maps

While our new definition of occlusion allows us to determine the amount of occlusion using a simple neighborhood around a surface point, computing this directly by testing each voxel in the region would still be prohibitively expensive. We note here that one voxel of a conventional volume data-set can only represent the image value at that one location. If we instead had knowledge of a *distribution* of values in a spherical region surrounding each point, we could do one test against the distribution instead of one

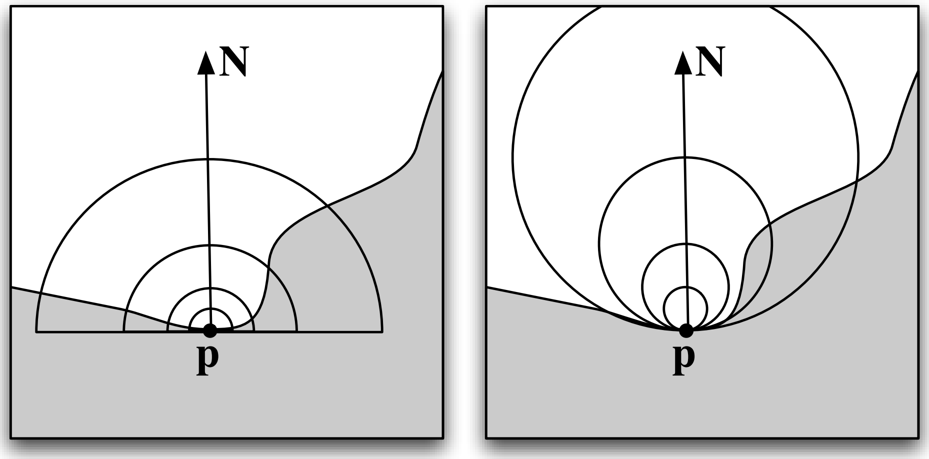


Figure 6.9: Occlusion is calculated using the percentage of the a region that is within the iso-surface (grey area). Left: Hemispherical regions. Right: Spherical regions. Different sizes can be combined to better localize occlusions.

test for every voxel. The optimal function for performing this test is the *cumulative distribution function* (CDF). If we think of a region as a distribution of values X, the CDF is defined as:

$$CDF(x) = P(X \leq x) = \sum_{x_i \leq x} P(X = x_i) = \sum_{x_i \leq x} p(x) \tag{6.7}$$

Given this CDF function we can determine the exact percentage of voxels that are inside/outside the iso-surface in the region. If we look at the properties of the CDF, we can see it is actually equivalent to our new definition of ambient irradiance from equation 6.6. Unfortunately, while pre-computing CDF functions for each spherical region would allow us to lookup the occlusion with one lookup, the memory required to store all these CDFs uncompressed would be monumental.

Thankfully, by reducing the ambient occlusion problem to storing and evaluating a CDF, we can make use of a lot of research from a slightly different domain. Shadow mapping algorithms [107] are faced with the exact same comparison problem when testing an object's depth against a shadow map. In fact a soft shadow technique known as percentage closer filtering [82] performs a brute force depth test against a region in the shadow map which is identical to the test described in equation 6.6. The only difference is that the shadow map region is in 2D and the volume region is in 3D (see figure 6.10).

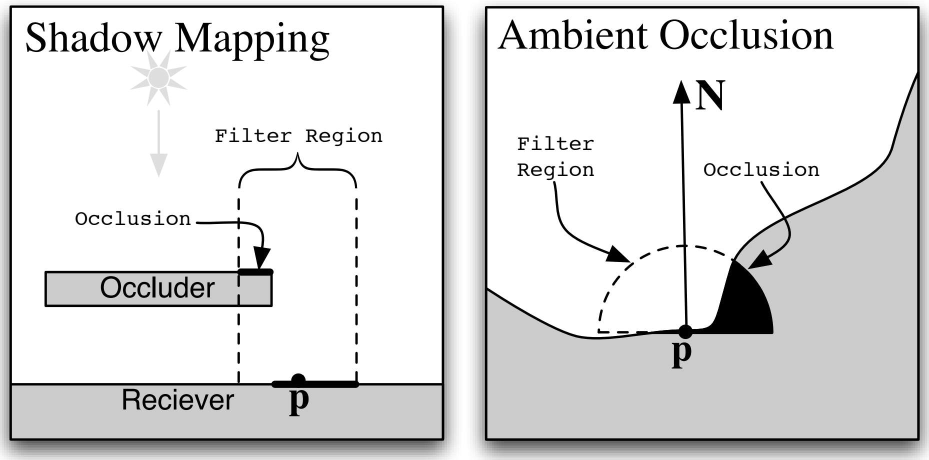


Figure 6.10: Comparison of the soft shadow mapping problem to our ambient occlusion problem. Left: In shadow mapping, we want to determine the percentage of non-occluding shadow map texels (greater than or equal to our reciever depth). Right: In our ambient occlusion approximation we wish to determine the percentage of non-occluding voxels (less than the iso-value). Both can be determined using the CDF of the filter region.

6.4.4 Variance Occlusion Maps

We have chosen to use a technique described by Donnelly *et al.* [23] to represent a distribution of image values and approximately query the CDF. To very compactly approximate a distribution, they store only the first two moments of the distribution: the image value and the squared image value. The advantage of this representation is that it can approximate the average of several distributions by averaging the moments. This means that the image may simply be blurred to generate a distribution centered at each pixel, where the blurring/filtering kernel represents the image region that will be represented by the distribution.

As in [23] we can describe these two moments M_1 and M_2 as:

$$M_1 = E(x) = \int_{-\infty}^{\infty} xp(x)dx \tag{6.8}$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2p(x)dx \tag{6.9}$$

We can then calculate the mean μ and variance σ^2 of the distribution:

$$\mu = E(x) = M_1 \tag{6.10}$$

$$\sigma^2 = E(x^2) - E(x)^2 = M_2 - M_1 \tag{6.11}$$

Since the variance gives us a measure of the width of the distribution, we can place a bound on how much of the distribution can be found a certain distance away from the mean. Chebychev's inequality states this bound precisely as:

$$P(x \leq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} \tag{6.12}$$

Donnelly *et al.* further demonstrate that while this only provides an upper bound on the CDF, it is very accurate in the case of a bi-modal distribution containing only two image values d_1 and d_2. This occurs in shadow maps when there is one occluding object at d_1 and one partially occluded object at d_2. We have found that this is also very often the case in volume data sets where different materials are represented by different iso-values.

In this case we have:

$$\mu = E(x) = pd_2 + (1-p)d_1 \tag{6.13}$$

$$E(x^2) = pd_2^2 + (1-p)d_1^2 \tag{6.14}$$

$$\sigma = E(x^2) - \mu^2 = pd_2^2 + (1-p)d_1^2 - (pd_2 + (1-p)d_1)^2 \tag{6.15}$$

$$= (p - p^2)(d_2 - d_1)^2 \tag{6.16}$$

When we test d_2 against μ and σ using equation 6.12 we get:

$$p_{max}(d_2) = \frac{\sigma^2}{\sigma^2 + (d_2 - \mu)^2} \tag{6.17}$$

$$= \frac{(p-p^2)(d_2-d_1)^2}{(p-p^2)(d_2-d_1)^2 + (pd_2 + (1-p)d_1 - d_2)^2} \tag{6.18}$$

$$= \frac{(p-p^2)(d_2-d_1)^2}{(p-p^2)(d_2-d_1)^2 + (1-p)^2(d_2-d_1)^2} \tag{6.19}$$

$$= \frac{p-p^2}{1-p} \tag{6.20}$$

$$= p \tag{6.21}$$

In this simple bi-modal case, the inequality becomes an equality and gives the exact percentage of d_2 in the region, which is the same result as the CDF. This represents the percentage of occluding voxels in the shadow map region and can be used to calculate the amount of irradiance. Of course this becomes less accurate as the distribution takes different forms, but it provides a surprisingly good approximation in many data-sets, especially when only a small region of the volume is considered.

To verify that this approximation is valid, we analyzed a number of data-sets (see figure 6.11). We found that in medical data-sets, CT values were heavily clustered around three clusters correlating to air, soft tissue, and bone. We found that non-medical data-sets such as the stanford bunny data-set were even more clustered, having almost entirely bi-modal distributions. However, MRI data-sets and distance-field data-sets were much less clustered and thus the variance test will be less valid for these types of data. Other filterable approximations may still apply however (see section 6.7).

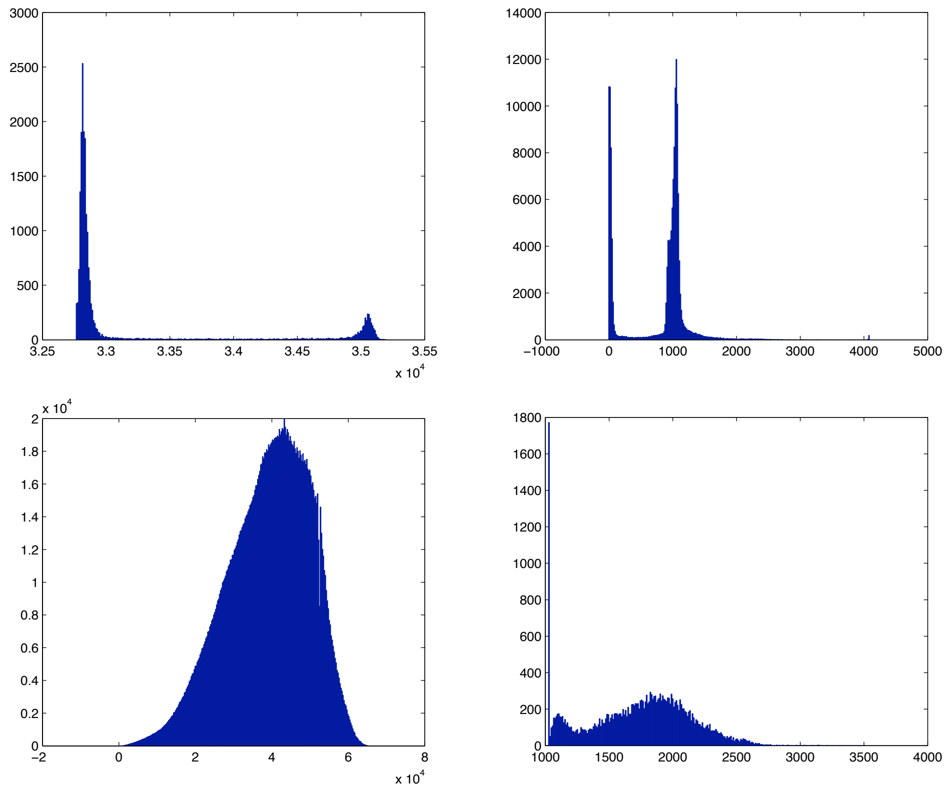


Figure 6.11: Histograms of volume data-sets. We found CT data tends to be heavily clustered and works well with our technique. Top-Left: Stanford Bunny CT. Top-Right: Head CT. Bottom-Left: Signed Distance Field. Bottom-Right: Head T1 Weighted MRI

6.5 Implementation

6.5.1 Preprocessing

As a preprocessing step, we create a new volume containing the two moments of the distribution (value and value squared). We then perform separable convolutions with gaussian kernels to generate local distributions centered at each voxel. Storing many of these volumes would take a large amount of memory, so we chose to double the gaussian kernel size at each iteration and down-sample by a factor of two at each iteration. This allows us to store the entire variance ambient map in a mip-mapped two channel floating-point texture. Since we are interested in fairly large filter regions, we also found that we could easily get away with starting at half the resolution of the original volume or even smaller resolutions if only low frequency occlusions are required. At half resolution variance ambient map is roughly 0.6X the size of the original volume including mip-maps.

6.5.2 Blurring Kernel

As discussed above, by representing our occlusion regions using a filterable representation, we can generate occlusion distributions using a simple image blur. An important optimization used in our approach is the choice of blurring kernel we used to generate occlusion distributions.

While we would normally want to use a hemispherical filter region with cosine and distance weighted occlusion region, this would require a complicated non-separable blurring process based on the surface normal at each voxel. Since the surface normal can change rapidly, this representation would also suffer from aliasing artifacts under linear interpolation of neighboring samples (see figure 6.12). This becomes especially apparent when lower resolution volumes or mipmaps are used.

Our approach uses a novel method which doesn't store the occlusion values at the surface, but rather offset from the surface. We accomplish this by using a radially symmetric kernel and offsetting the occlusion sampling location from the surface along the surface normal. This reduces aliasing while still allowing high-frequency changes in the surface normal. It also allows us to preform a separable blur which reduces the complexity of the blur from $n^3 \cdot v$ to $n \cdot v$ where n is the radius of the occlusion region and v is the number of voxels. Alternatively we can also perform the blur in the fourier

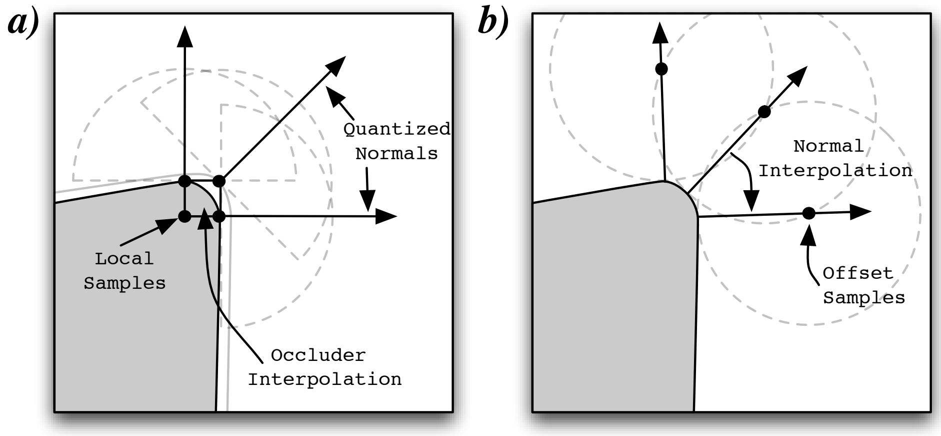


Figure 6.12: Illustration of different blurring kernels. (a) While a hemisphere is normally used to search for occluders, it can result in artifacts due to the interpolation of very different occluding regions. (b) By using a spherical kernel and offsetting the sampling location from the surface, the correct normal is used to determine the angle of the occlusion region.

domain with a complexity of *vlnv*. The hemispherical blur can't be performed in the fourier domain because it is spatially varying.

The only drawback of our approach is that the occlusion weighting becomes slightly more concentrated in the direction of the surface normal. However since a cosine weighted distribution does this anyway, we found this approximation to work very well. Being able to adjust the surface normal during interactive rendering also allows us to perform some other interesting effects like averaging the normal and view vector to get view dependent ambient occlusion.

6.5.3 Rendering

During interactive rendering, we first search to the iso-surface and then perform a binary search to refine the surface location. We then calculate the gradient using central-differencing to perform local lighting. We also use the gradient to calculate offsets for querying the variance ambient map. Since many isosurfaces tend to fall on partial-

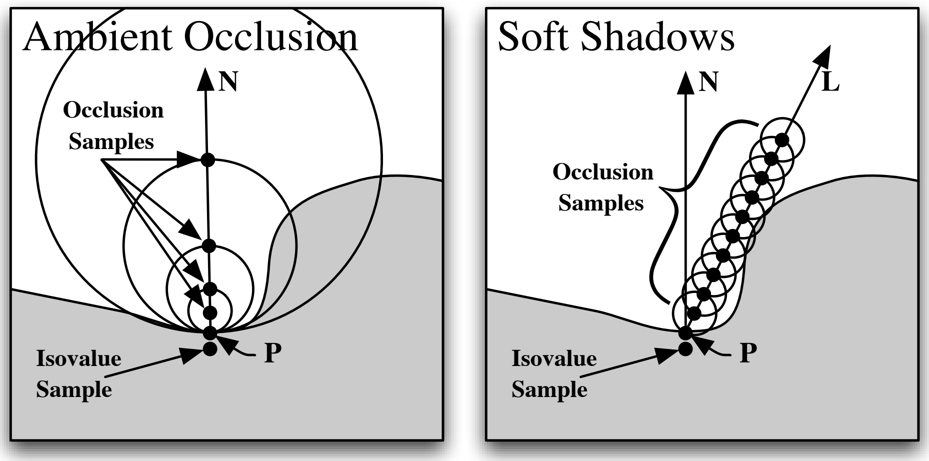


Figure 6.13: Illustration of our dynamic ambient occlusion and soft shadow algorithms. The isovalue is chosen one half of a voxel behind the isosurface. Ambient occlusion requires only 3-4 samples of different sizes in the direction of the surface normal. Soft shadows requires one ray cast in the direction of the light. In both cases the occlusion sample with the most occlusion is chosen.

volumes where one material meets another, we take an extra sample one half of a voxel under the surface in the direction of the gradient (see figure 6.13) and use this sample as the isovalue to evaluate the amount of occlusion in the variance occlusion map. In the case of ambient occlusion we can evaluate four instances of equation 6.12 in parallel (one sample from the first 4 mip levels) and then take the maximum occlusion value. In the case of soft shadows, we cast a ray four steps at a time evaluating equation 6.12 until the ray leaves the volume. To simulate soft shadows from area light sources, the mip level can be varied as it travels towards the light.

6.6 Results

All of the images in this section were generated on a Mac Book Pro with a single NVIDIA GeForce 8800 GTS. Figure 6.14 shows 4 offset occlusion samples from seperate mip-levels in the occlusion map. Figure 6.15 shows the combined result with diffuse local lighting.

Figure 6.17 shows soft shadows calculated with one ray combined with ambient occlusion.

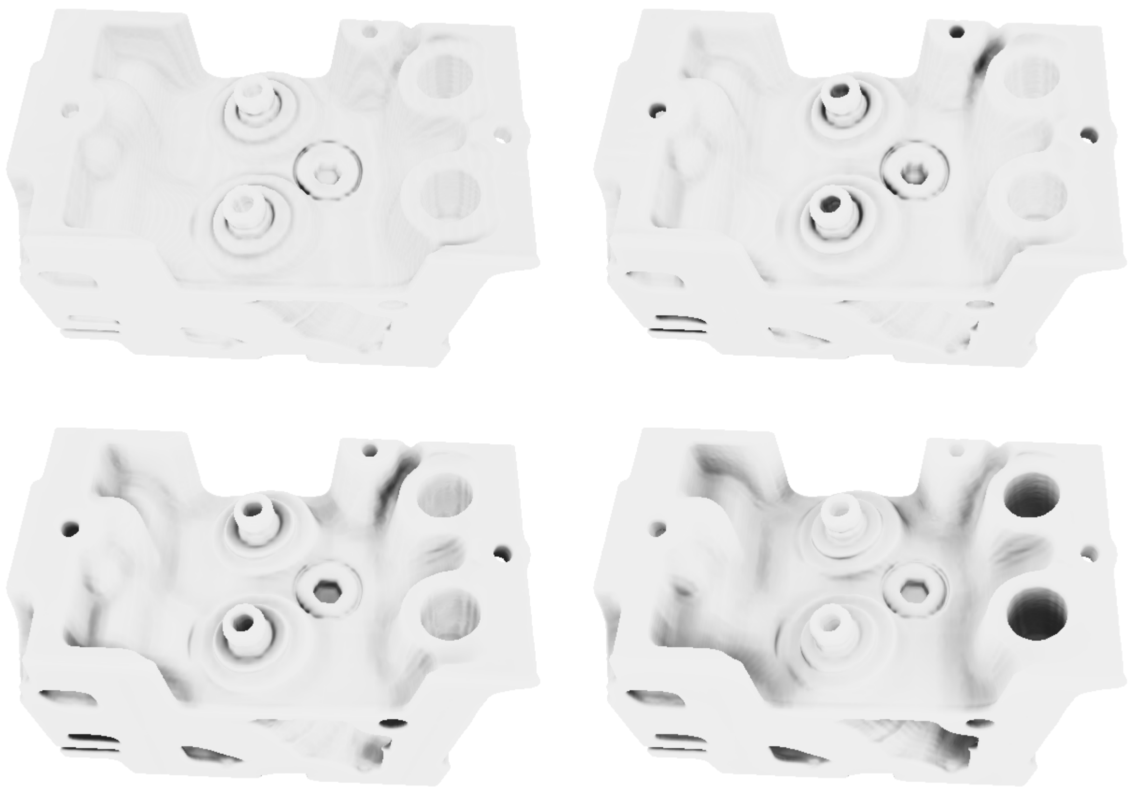


Figure 6.14: Occlusion sampled from the first 4 levels of the occlusion map. These are combined by taking the maximum occlusion (minimum irradiance).

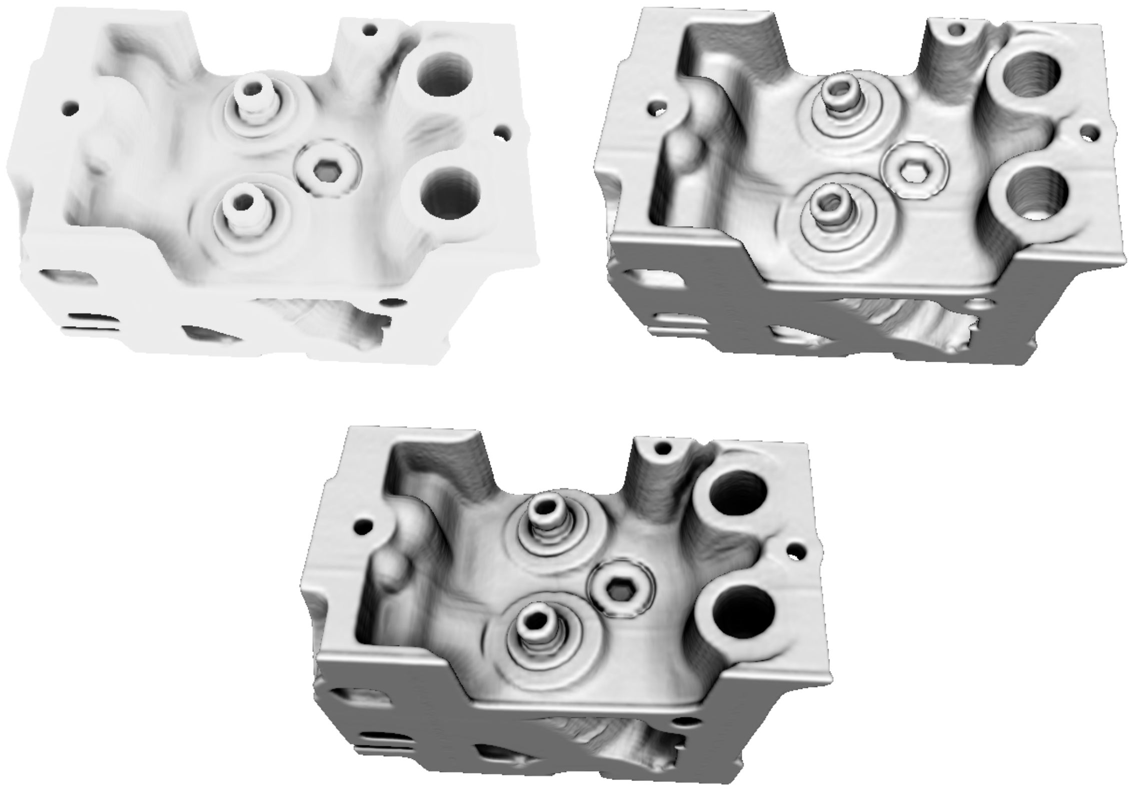


Figure 6.15: Top Left: Combined occlusion from figure 6.14. Top Right: Simple diffuse lighting. Bottom: Diffuse combined with ambient occlusion.

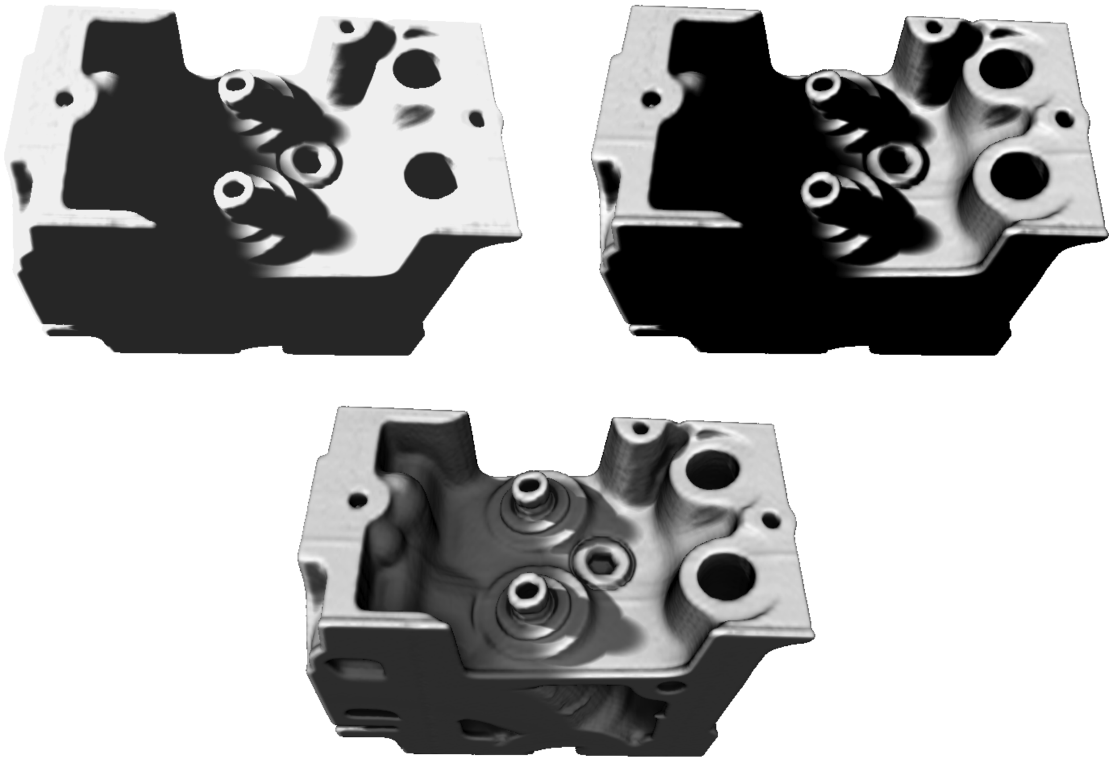


Figure 6.16: Top Left: Approximate soft shadows using only one ray. Top Right: Diffuse lighting with shadows. Bottom: Diffuse lighting, ambient occlusion and soft shadows.

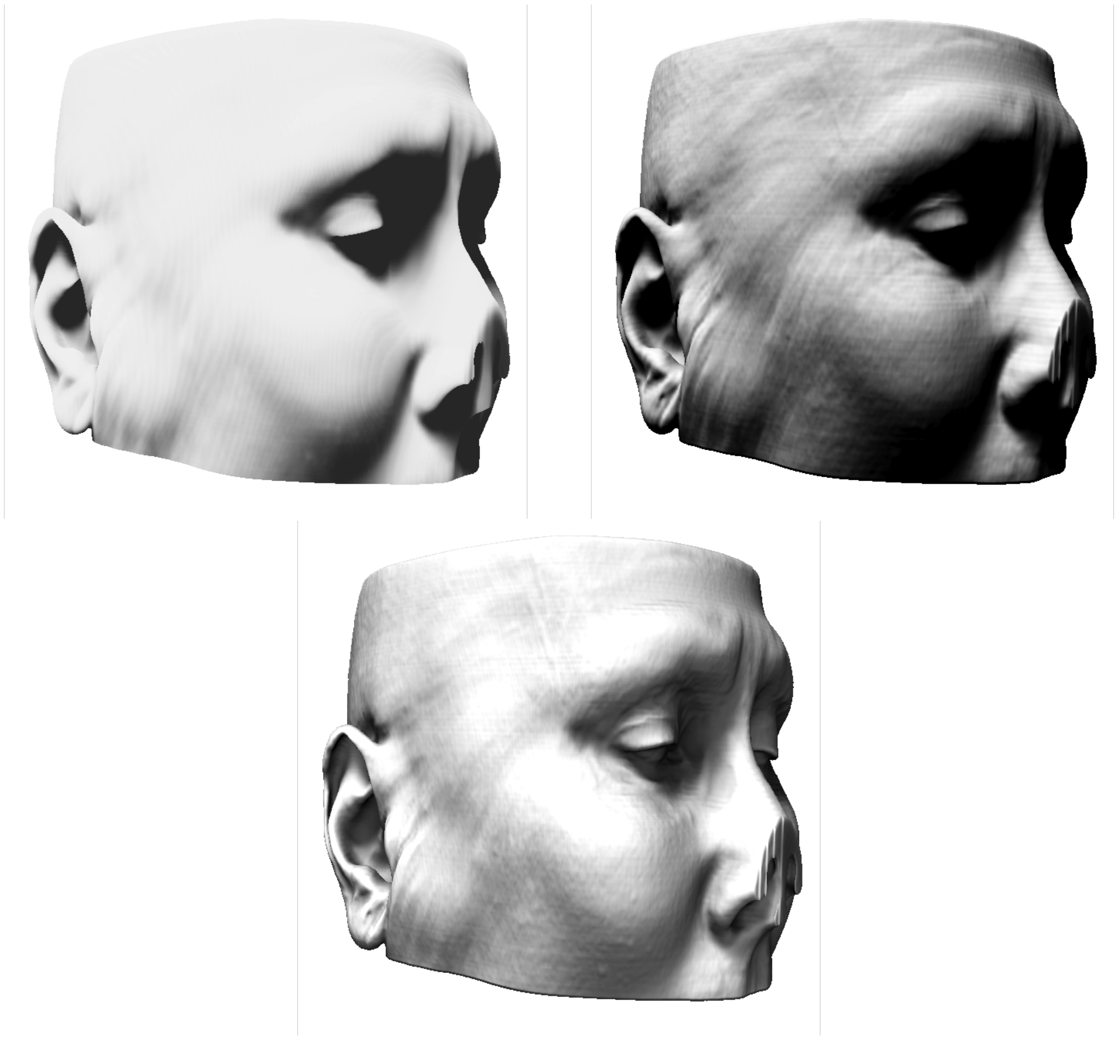


Figure 6.17: Top Left: Approximate soft shadows using only one ray. Top Right: Diffuse lighting with shadows. Bottom: Diffuse lighting, ambient occlusion and soft shadows.

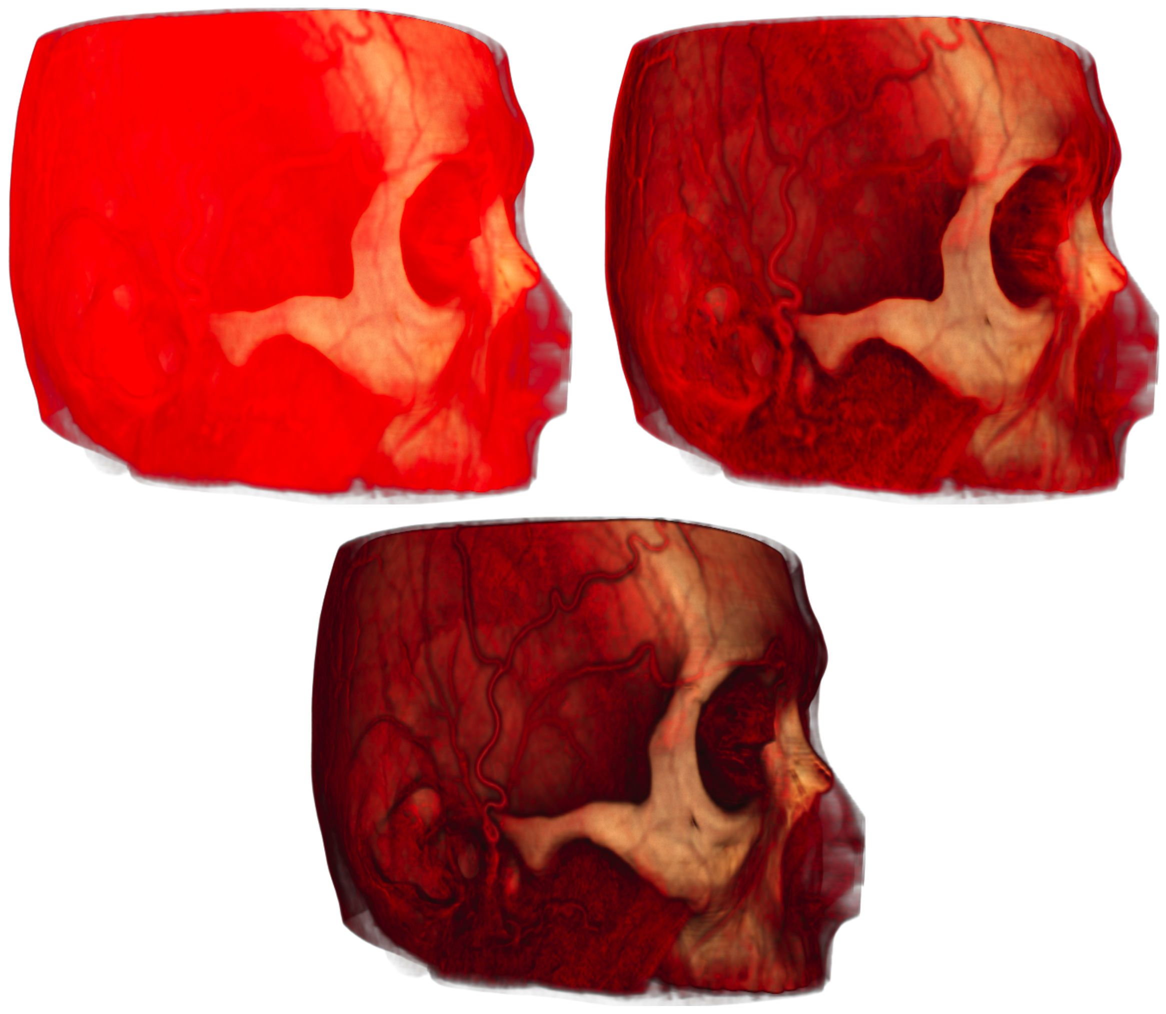
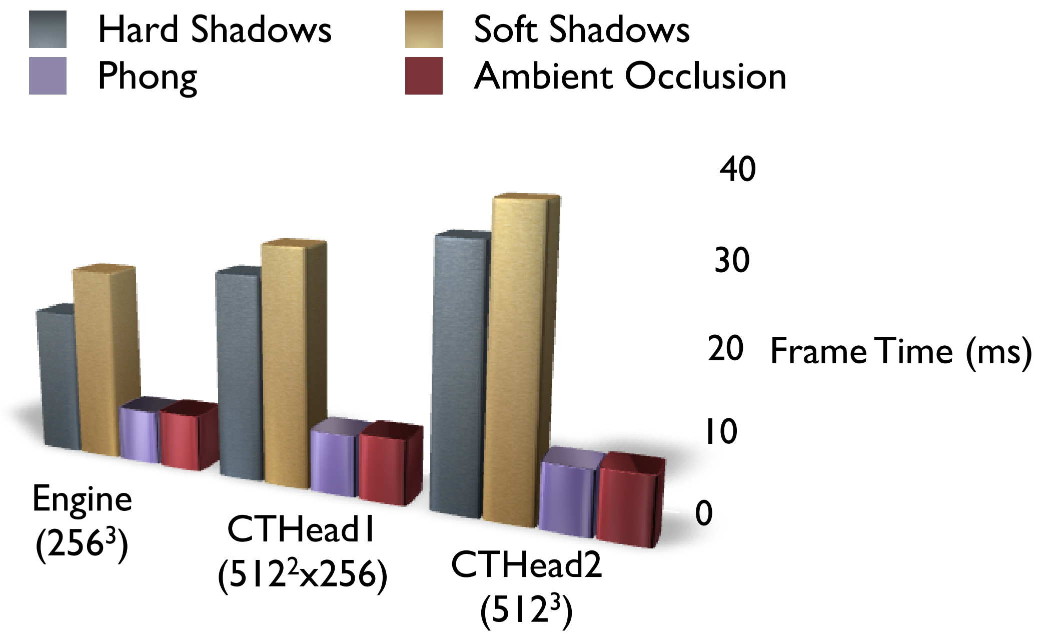
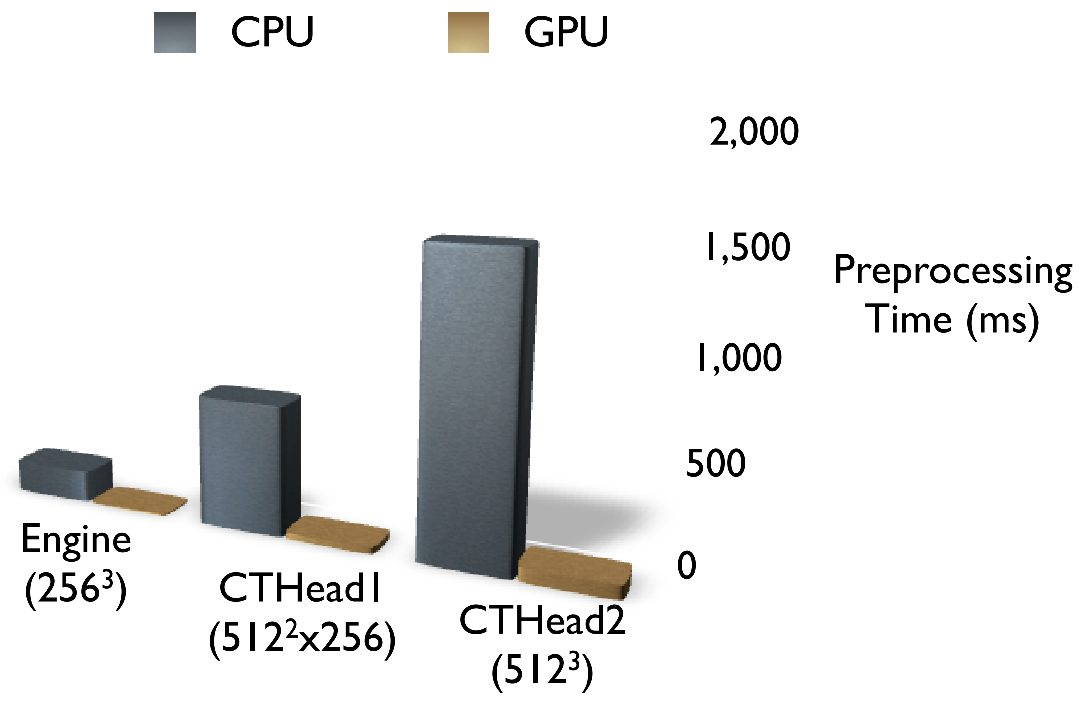


Figure 6.18: Top Left: DVR. Top Right: DVR with ambient occlusion. Bottom: DVR with ambient occlusion and local phong lighting. If transparency is intended to provide detail-in-context rather than specify transparency to incoming light, then our occlusion method is still useful within DVR. We also found it useful to adjust the shadow contribution based on transparency.



Rendering Time (in milliseconds)				
Dataset	Phong	AO	HS	SS
Engine (256^3)	7.1	7.4	19.1	25.2
CTHead1 (512^2x256)	7.8	8.2	26.4	30.3
CTHead2 (512^3)	8.0	8.3	32.5	37.0

Figure 6.19: Rendering time using phong, ambient occlusion(AO), hard shadows(HS) and soft shadows(SS) using a GeForce 8800 card.



Preprocessing Time (in milliseconds)		
Dataset	CPU	GPU
Engine (256^3)	170	8.6
Small Head (512^2x256)	678	34.5
Big Head (512^3)	1524	74.1

Figure 6.20: Preprocessing time on the CPU and GPU using a GeForce 8800 card.

6.7 Discussion

We have presented a novel method for generating approximate iso-surface ambient occlusion and soft shadows. Compared with other approaches which may require hours or days of pre-computation time and suffer from aliasing and/or quantization artifacts, our approach can load a dataset within seconds and has very good runtime performance.

Since we have reduced the ambient occlusion problem to the same sampling problem faced by shadow map filtering, a good direction for future research will be to apply newer shadow filtering methods to reduce the memory footprint and improve on the statistical approximation. Convolution Shadow Maps (CSMs) [2] , Exponential Shadow Maps (ESMs) [3], Layered Variance Shadow Maps (LVSMs) [59] and Exponential Variance Shadow maps (EVSMs) [59] are potential candidates. These approaches should also work for datasets with more complex datasets such as MRI, although there is a memory and performance tradeoff with each approach.

Another possible avenue of future research would be to utilize the insights we have presented to more accurately store physically accurate pre-computed lighting. To most efficiently remove artifacts, an occlusion function should be stored for each voxel, rather than trying to increase the spatial resolution of the pre-computed volume (as presented in [108]). Interestingly, the same shadow mapping approaches can be used to compress physically accurate lighting as well, making our work apply in both cases.

To support more lighting effects, we found that adjusting the offset of the spherical distribution produced very interesting effects similar to subsurface scattering, but we were unable to do this because the variance test would begin to behave incorrectly as the mean of the distribution approached the isovalue. Other approximations such as Convolution Shadow Maps show promise in eliminating this problem and allowing for arbitrary queries of the distribution.

In conclusion, we believe that filterable occlusion maps can provide a very quick way to approximate complex volumetric lighting effects, and can be applied to solve a host of problems with pre-computed or dynamic volume lighting.

Chapter 7

Conclusion

While chapters 5 and 6 have discussed future work in their respective areas within medical imaging and volume rendering, this chapter serves to reach general conclusions and summarize the contributions made in this thesis.

Interactive 3D medical visualization is an effective way to gain new insight and interact with a medical image, and is crucial to enabling many new important medical techniques such as surgical simulation, image guided surgery and virtual endoscopy. Medical visualization can be improved by addressing three key factors: diagnostic quality, interactivity and realism. This work contributes to the field in several ways.

7.1 Image Quality

To achieve very high quality images without common volume rendering artifacts, we designed our software from the ground up with our own version of pre-integrated transfer functions based on mip-maps, and a fast cubic b-spline filtering method which fully utilizes built in GPU hardware for tri-linear texture sampling. While prior work has achieved these enhancements in only final non-interactive rendering modes, our optimized implementations, as well as empty space skipping optimizations, allow us to enable these features during an interactive session.

7.2 Interactivity

To improve interactive frame rates drastically we have presented a novel method to cast dozens or even hundreds of rays through large regions of empty space. We have combined the positive aspects of coherent ray-tracing but achieved them by designing our algorithm from the ground up in a GPU friendly way. The resulting hierarchical frustum casting algorithm and a new data structure called the overlapped min-max octree (OMMO) can be implemented entirely on the GPU with no complicated CPU/GPU communication. This allows for seamless editing of the transfer function while maintaining constant real-time frame rates, and can easily be supported in any ray-casting engine.

7.3 Realism

To improve image quality and add realism and depth cues, we have designed a new algorithm for approximating ambient occlusion and soft shadows. Previous techniques based on physically correct ambient occlusion have required hours or days of pre-computation time, and suffer from several artifacts which can only be fixed by increasing the pre-computed lighting resolution to unreasonably high levels. We have identified the inherent flaw in trying to pre-compute volume lighting, which is that the image gradient, rather than image resolution, determines the Shannon-Nyquist sampling rate for pre-computed lighting. Our work demonstrates how this can be overcome, through storage of an approximate occlusion function at every image voxel. We also go further however, and demonstrate how an approximate occlusion function for iso-surfaces can be designed using simple image sampling methods found in shadow mapping algorithms. Our technique's pre-computation step achieves real-time rates, potentially allowing our technique to be used in surgical simulation applications where the image volume is constantly changing.

7.4 Putting it All Together

To summarize our contributions, Figure 7.1 puts together a set of images captured from one session in our software using only one CT scan. Generating high quality, illustrative images such as these has been one of the inspirations in choosing this field of research and we hope that our contributions will one day make a difference for real physicians and patients.

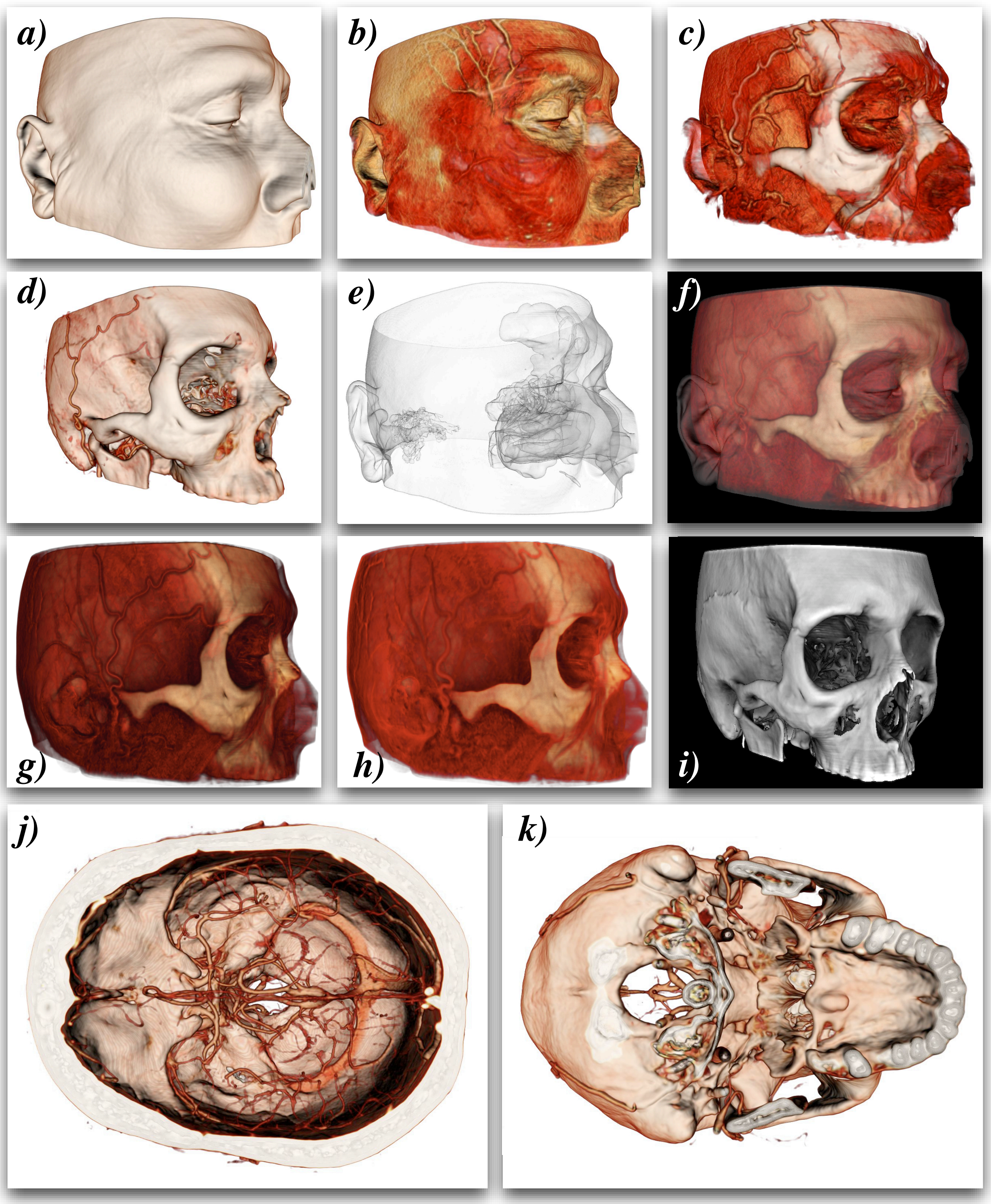


Figure 7.1: A set of of images generated from one session using our software to illustrate the contributions of this thesis. These are all taken from a single CT scan by enabling the features we have implemented and by modifying the transfer function. Images a,b,c,d and e illustrate choosing surfaces of interest while f combines two transfer functions (d and e). Images g,h, and i illustrate our ambient occlusion algorithm for DVR and iso-surface rendering. Images j and k illustrate our high quality b-spline interpolation when diagnosing ischemic stroke.

Bibliography

[1] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Proceedings of the Eurographics Symposium on Rendering*, pages 3–10, 1987.

[2] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proceedings of the Eurographics Symposium on Rendering*, pages 51–60, 2007.

[3] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *Proceedings of Graphics Interface*, pages 155–161, 2008.

[4] ATI. Amd stream computing whitepaper. http://ati.amd.com/technology/streamcomputing/firestream-sdk-whitepaper.pdf, 2007. Accessed Dec 22, 2009.

[5] David C. Banks and Kevin Beason. Fast global illumination for visualizing isosurfaces with a 3D illumination grid. *Computing in Science and Engineering*, 9(1):48–54, 2008.

[6] Praveen Bhaniramka and Yves Demange. Opengl volumizer: a toolkit for high quality volume rendering of large data sets. In *Proceedings of the IEEE Symposium on Volume Visualization and Graphics*, pages 45–54, 2002.

[7] James F. Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 192–198, 1977.

[8] David Blythe. The Direct3D 10 system. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 724–734, 2006.

[9] Solomon Boulos, Dave Edwards, J Dylan Lacewell, Joe Kniss, Jan Kautz, Ingo Wald, and Peter Shirley. Packet-based whitted and distribution tay tracing. In *Proceedings of Graphics Interface*, pages 177 – 184, 2007.

[10] Richard P. Brent. Algorithms for minimization without derivatives. *IEEE Transactions on Automatic Control*, 19(5):632–633, October 1974.

[11] J. E. Bresenham. Algorithm for computer control of a digital plotter. In *Seminal Graphics: Poineering Efforts that Shaped the Field*, pages 1–6, 1965.

[12] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Mike, and H. Pat. Brook for gpus: stream computing on graphics hardware. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 777 – 786, 2004.

[13] Michael Bunnell. *GPU Gems 2*, chapter Dynamic Ambient Occlusion and Indirect Lighting. Addison-Wesley, 2005.

[14] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 Symposium on Volume visualization*, pages 91–98, 1994.

[15] Sonny Chan. Three-dimensional medical image registration on modern graphics processors. Master's thesis, University of Calgary, Calgary, April 2007.

[16] Min Chen, Carlos Correa, and Deborah Silver. Feature aligned volume manipulation for illustration and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1069–1076, 2006.

[17] Daniel Cohen and Zvi Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer*, 11(1):27–38, 1994.

[18] Franklin C. Crow. Shadow algorithms for computer graphics. *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, 11(2):242–248, 1977.

[19] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 207–212, 1984.

[20] Nicolas Cuntz and Andreas Kolb. Fast hierarchical 3D distance transforms on the gpu. In *Proceedings of the Eurographics Symposium on Rendering*, pages 93–96, 2007.

[21] John Danskin and Pat Hanrahan. Fast algorithms for volume ray tracing. In *Proceedings of the Workshop on Volume Visualization*, pages 91–98, 1992.

[22] Engel K. Desgranges P. Fast ambient occlusion for direct volume rendering. US patent application 2007/0013696 A1, 2007.

[23] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 161–165, 2006.

[24] Christoph Dräger. A chain mail algorithm for direct volume deformation in virtual endoscopy applications. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2005.

[25] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron E. Lefohn, Christof Rezk Salama, and Daniel Weiskopf. Real-time volume graphics. In *ACM Siggraph Course Notes*, 2004.

[26] Klaus Engel, Martin Kraus, and Thomas Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM Siggraph/Eurographics Workshop on Graphics Hardware*, pages 9–16, 2001.

[27] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science, September 2004.

[28] Heiko Friedrich, Ingo Wald, and Philipp Slusallek. Interactive iso-surface ray tracing of massive volumetric data sets. In *Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, pages 186–192, 2007.

[29] Allen Van Gelder and Kwansik Kim. Direct volume rendering with shading via three-dimensional textures. In *Proceedings of the 1996 Symposium on Volume Visualization*, pages 23–30, 1996.

[30] Francisco González, Mateu Sbert, and Miquel Feixas. An information-theoretic ambient occlusion. In *Computational Aesthetics*, pages 29–36, 2007.

[31] Sören Grimm. *Real-time mono- and multi-volume rendering of large medical datasets on standard PC hardware.* PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, April 2005.

[32] Markus Hadwiger, Joe M. Kniss, Christof Rezk-salama, Daniel Weiskopf, and Klaus Engel. *Real-time Volume Graphics.* A. K. Peters, Ltd., 2006.

[33] Markus Hadwiger, Andrea Kratz, Christian Sigg, and Katja Bühler. Gpu-accelerated deep shadow maps for direct volume rendering. In *Proceedings of the Eurographics Symposium on Graphics hardware*, pages 49–52, 2006.

[34] Markus Hadwiger, Christian Sigg, Henning Scharsach, Katja Bhler, and Markus Gross. Real-rime ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[35] John C. Hart. Ray tracing implicit surfaces. In *Siggraph Course notes*, pages 1 – 13, 1993.

[36] John C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[37] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadow algorithms. *Computer Graphics Forum*, 22(4):753–774, December 2003.

[38] P. Hastreiter and T. Ertl. Integrated registration and visualization of medical image data. In *Proceedings of Computer Graphics International*, pages 78–85, 1998.

[39] P. Hastreiter, C. Rezk-Salama, B. Tomandl, K. Eberhard, and T. Ertl. Interactive direct volume rendering of the inner ear for the planning of neurosurgery. In *Proceedings of the Workshop on Image Processing in Medicine*, pages 192–196, 1999.

[40] Wei Hong, Xianfeng Gu, Feng Qiu, Miao Jin, and Arie Kaufman. Conformal virtual colon flattening. In *Proceedings of the ACM Symposium on Solid and Physical Modeling*, pages 85–93, 2006.

[41] Daniel Reiter Horn, Jeremy Sugerman, Mike Houston, and Pat Hanrahan. Interactive k-d tree gpu raytracing. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pages 167–174, 2007.

[42] Joseph Hornak. *The Basics of MRI*. http://www.cis.rit.edu/, 2009. Accessed Dec 22, 2009.

[43] G. Hubona, G. Shirah, M. Brandt, and P. Wheeler. The role of object shadows in promoting 3D visualization. *ACM Transactions on Computer-Human Interaction*, 6(1):214–242, 1999.

[44] Thiago Ize, Andrew Kensler, and Ingo Wald. A coherent grid traversal approach to visualizing particle-based simulation data. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):758–768, 2007.

[45] Radu Jianu, Cagatay Demiralp, and David Laidlaw. Exploring 3D DTI fiber tracts with linked 2D representations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1449–1456, 2009.

[46] Alexander Keller and Wolfgang Heidrich. Interleaved sampling. In *Proceedings of the Eurographics Workshop on Rendering*, pages 269–276, June 2001.

[47] D. Kersten, P. Mamassian, and D.C. Knill. Moving cast shadows and the perception of relative depth. Technical report, Max Planck Institute for Biological Cybernetics, June 1994.

[48] Adam G. Kirk and Okan Arikan. Real-time ambient occlusion for dynamic character skins. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 178–184, April 2007.

[49] Thomas Klein, Magnus Strengert, Simon Stegmaier, and Thomas Ertl. Exploiting frame-to-frame coherence for accelerating high-quality volume raycasting on graphics hardware. In *Proceedings of IEEE Visualization*, pages 29–35, 2005.

[50] Joe Kniss. *Interactive volume rendering techniques.* PhD thesis, University of Utah, May 2002.

[51] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *Proceedings of IEEE Visualization*, pages 109–116, 2002.

[52] Aaron Knoll, Charles D Hansen, and Ingo Wald. Coherent multi-resolution isosurface ray tracing. Technical report, University of Utah, 2007.

[53] Aaron Knoll, Ingo Wald, Steven G Parker, and Charles D Hansen. Interactive isosurface ray tracing of large octree volumes. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 115–124, 2006.

[54] Kevin Kreeger, Ingmar Bitter, Frank Dachille, Baoquan Chen, and Arie Kaufman. Adaptive perspective ray casting. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 55–62, 1998.

[55] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *Proceedings of IEEE Visualization*, pages 38–05, 2003.

[56] Heewon Kye, Helen Hong, and Yeong-Gil Shin. Efficient interactive pre-integrated volume rendering. In *International Conference on Computational Science*, pages 834–837, 2005.

[57] Sarang Lakare and Arie Kaufman. Light weight space leaping using ray coherence. In *Proceedings of IEEE Visualization*, pages 19–26, 2004.

[58] Johann Heinrich Lambert. Die freye perspektive, 1759.

[59] Andrew Lauritzen and Michael McCool. Layered variance shadow maps. In *Proceedings of Graphics Interface*, pages 139–146, 2008.

[60] Sylvain Lefebvre and Hugues Hoppe. Perfect spatial hashing. In *Proceedings of*

the Annual Conference on Computer Graphics and Interactive Techniques, pages 579–588, 2006.

[61] A.E. Lefohn, J.M. Kniss, C.D. Hansen, and R.T. Whitaker. A streaming narrow-band algorithm: Interactive deformation and visualization of level sets. *IEEE Transactions on Visualization and Computer Graphics*, 10(1):422–433, 2004.

[62] M. S. Levoy. *Display of surfaces from volume data*. PhD thesis, University of North Carolina, Chapel Hill, NC, USA, 1989.

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

[64] Patric Ljung. *Efficient methods for direct volume rendering of large data sets*. PhD thesis, Linkping University, Department of Science and Technology, September 2006.

[65] Eric Lum, Brett Wilson, and Kwan-Liu Ma. High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings of the Joint Eurographics-IEEE TVCG Symposium on Visualization*, pages 25–34, 2004.

[66] Muhammad Muddassir Malik, Torsten Möller, and M. Eduard Gröller. Feature peeling. In *Proceedings of Graphics Interface*, pages 273–280, 2007.

[67] Tom Malzbender. Fourier volume rendering. *ACM Transactions on Graphics*, 12(1):233–250, 1993.

[68] W. Mark, S. Glanville, and K. Akeley. Cg: A system for programming graphics hardware in a c-like language. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 896–907, August 2003.

[69] Stephen R. Marschner and Richard J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of the Conference on Visualization*, pages 100–107, 1994.

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

[71] Michael McCool and Stefanus Du Toit. *Metaprogramming GPUs with Sh*. AK Peters Ltd, 2004.

[72] Microsoft. Hlsl - high level shading language. http://msdn.microsoft.com/en-us/library/bb509561(VS.85).aspx, 2008. Accessed Dec 22, 2009.

[73] Martin Mittring. Finding next gen: Cryengine 2. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 97–121, 2007.

[74] Y. Miyawaki, H. Uchida, O. Yamashita, M. Sato, Y. Morito, H. Tanabe, N. Sadato, and Y. Kamitani. Visual image reconstruction from human brain activity using a combination of multiscale local image decoders. *Neuron*, 60(5):915–929, December 2008.

[75] Ken Museth, David E. Breen, Ross T. Whitaker, and Alan H. Barr. Level set surface editing operators. *ACM Transactions on Graphics*, 21(3):330–338, 2002.

[76] Ulrich Neumann. Parallel volume-rendering algorithm performance on mesh-connected multi-computers. In *Proceedings of the Symposium on Parallel Rendering*, pages 97–104, 1993.

[77] NVIDIA. Cuda programming guide. http://developer.download.nvidia.com/, 2007. Accessed Dec 22, 2009.

[78] Owens, D. John, Luebke, David, Govindaraju, Naga, Harris, Mark, Kruger, Jens, Lefohn, E. Aaron, Purcell, and J. Timothy. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, March 2007.

[79] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 251–260, 1999.

[80] John Plate, Thorsten Holtkaemper, and Bernd Froehlich. A flexible multi-volume shader framework for arbitrarily intersecting multi-resolution datasets. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1584–1591, 2007.

[81] RapidMind. Rapidmind. `http://www.rapidmind.net`, 2008. Accessed Dec 22, 2009.

[82] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 283–291, 1987.

[83] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. A multi-level ray tracing algorithm. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 1176–1185, 2005.

[84] Stefan Roettger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Strasser. Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation*, pages 231–238, 2003.

[85] Guodong Rong and Tiow-Seng Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 109–116, 2006.

[86] Timo Ropinski, Jennis Meyer-Spradow, Stefan Diepenbrock, Jörg Mensmann, and Klaus H. Hinrichs. Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Eurographics Computer Graphics Forum*, 27(2):567–576, 2008.

[87] Randi J. Rost. *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional, January 2006.

[88] D. Ruijterss and A. Vilanova. Optimizing gpu volume rendering. In *Winter School of Computer Graphics*, pages 9–16, 2006.

[89] Henning Scharsach. Advanced gpu ray-casting. In *Proceedings of Central European Seminar on Computer Graphics*, pages 69–76, 2005.

[90] Henning Scharsach, Markus Hadwiger, André Neubauer, Stefan Wolfsberger, and Katja Bühler. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In *Proceedings of EuroVis Visualization Symposium*, pages 315–322, 2006.

[91] Thorsten Scheuermann and Justin Hensley. Efficient histogram generation using scattering on gpus. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 33–37, 2007.

[92] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pages 73–80, 2007.

[93] Peter Shirley. Interactive display of isosurfaces with global illumination. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):186–196, 2006.

[94] Christian Sigg and Markus Hadwiger. *GPU Gems 2: Fast Third-Order Texture Filtering*, chapter 20, pages 313–329. Addison-Wesley Professional, 2005.

[95] Barton T. Stander and John C. Hart. A lipschitz method for accelerated volume rendering. In *Proceedings of the 1994 Symposium on Volume Visualization*, pages 107–114, 1994.

[96] A. James Stewart. Vicinity shading for enhanced perception of volumetric data. In *Proceedings of IEEE Visualization*, pages 47–53, October 2003.

[97] Paul Suetens. *Fundamentals of Medical Imaging.* Medical Image Computing, 2002.

[98] Natalya Tatarchuk. Advanced real-time rendering in 3D graphics and games. In *ACM Siggraph Courses*, 2006.

[99] Thomas Theußl, Torsten Möller, and Meister Eduard Gröller. Optimal regular volume sampling. In *Proceedings of IEEE Visualization*, pages 91–98, 2001.

[100] Thomas Theußl, Torsten Möller, and Meister Eduard Gröller. Reconstruction schemes for high quality raycasting of the body-centered cubic grid. Technical Report TR-186-2-02-11, Institute of Computer Graphics and Algorithms, Vienna University of Technology, December 2002.

[101] Leonardo Da Vinci. Codex urbinas, 1490.

[102] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In *Proceedings of Eurographics Computer Graphics Forum*, pages 153–164, 2001.

[103] Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G Parker. Ray tracing animated scenes using coherent grid traversal. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 485–493, 2006.

[104] Ingo Wald, Timothy J. Purcell, Joerg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime ray tracing and its use for interactive global illumination. In *Eurographics State of the Art Reports*, 2003.

[105] Leonard Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *Symposium on Interactive 3D Graphics*, pages 39–42, 1992.

[106] Daniel Weiskopf, Manfred Weiler, and Thomas Ertl. Maintaining constant frame rates in 3D texture-based volume rendering. In *Proceedings of the Computer Graphics International*, pages 604–607, 2004.

[107] Lance Williams. Casting curved shadows on curved surfaces. *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, 12(3):270–274, 1978.

[108] Chris Wyman, Steven Parker, and Charles Hansen. Interactive display of isosurfaces with global illumination. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):186–196, 2006.

[109] Qi Zhang, Roy Eagleson, and Terry M. Peters. Rapid voxel classification methodology for interactive 3D medical image visualization. In *Proceedings of Medical Image Computing and Computer Assisted Intervention*, pages 86–93, 2007.

.1 Appendix A: Isosurface Ambient Occlusion and Soft Shadows with Filterable Occlusion Maps. Volume Graphics 2008

IEEE/ EG Symposium on Volume and Point-Based Graphics (2008)
H.- C. Hege, D. Laidlaw, R. Pajarola, O. Staadt (Editors)

Isosurface Ambient Occlusion and Soft Shadows with Filterable Occlusion Maps

Eric Penner[1] Ross Mitchell [2]

[1]Department of Computer Science, University of Calgary
[2]Department of Clinical Neurological Sciences and Radiology, University of Calgary

Abstract

Volumetric data sets are often examined by displaying isosurfaces*, surfaces where the data or function takes on a given value. We propose a new method for rendering isosurfaces at interactive rates while supporting dynamic ambient occlusion and/or soft shadows and requiring minimal pre-computation time. By approximating the occlusion in a region as the percentage of occluding voxels in that region, we reduce the ambient occlusion problem to the same problem faced in soft shadow mapping algorithms. In order to quickly extract the number of occluding voxels in an image region, we propose representing distributions using filterable representations such as variance shadow maps or convolution shadow maps. By choosing different sampling patterns from these maps we can dynamically approximate ambient occlusion and/or soft shadows.*

1. Introduction

Cast shadows are known to play a key role in human perception of the 3D world. To qualitatively and quantitatively understand the importance of shadows in our perception of the world, several studies and experiments have been conducted to understand how shadows shape our perception and understanding of a scene. Through these experiments, the importance of shadows has been demonstrated in understanding the position, size and geometry of both the shadow caster and shadow receiver. Hubona *et al.* [HSBW99] discuss the role of shadows in general 3D visualization. Wanger *et al.* [Wan92] study the effect of shadow quality on the perception of object relationships. Kersten *et al.* [KMK94] demonstrate that adjusting the motion of a shadow can dramatically effect the apparent trajectory of a shadow casting object. For the interested reader, a comprehensive discussion of real-time geometry based shadowing algorithms with references to their perceptual importance can be found in [HLHS03].

The most traditional method of computing shadows is with ray tracing using a point light source. Using this method, a ray is traced from each surface point back to the point light that illuminates the surface. If an object obscures the path of the ray, the surface is rendered without the light's contribution. Unfortunately, shadows from simple point sources produce stark discontinuities which aren't present under normal lighting conditions. The depth cues provided also vary depending on the viewing direction. For example, if the camera location aligns with the light location all the shadows become hidden behind the objects and no extra cues are provided. Methods such as shadow maps [Wil78] and shadow volumes [Cro77] can be used to accelerate point-lighting in real-time applications and produce similar results.

More realistic shadows are provided with more complex models such as *ambient occlusion* or *global illumination*. It has been shown that these realistic approaches provide better perception of many shapes than with simple point lighting. Ambient occlusion simulates light arriving equally from all directions or "light on a cloudy day" and is also referred to as *uniform diffuse lighting*. Global illumination simulates lighting from complex area light sources as well as diffuse inter-reflections and caustics. While there have been methods to compute good approximations of these lighting methods for a long time, traditional methods have always had very high computational costs. Usually the calculation involves a monte-carlo simulation of hundreds or even thousands of rays as opposed to the single ray used for a point or directional light source (see figure 1).

In order to capture these effects in interactive applications, many methods based on the theory of light transfer have

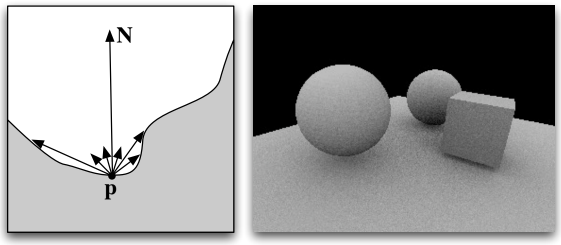


Figure 1: *Left: Illustration of calculating ambient occlusion using monte-carlo ray-tracing. Rays are chosen randomly in the hemisphere above a surface point. The percentage of occluded rays represents ambient occlusion. Right: Example using 36 randomly distributed rays. Notice the noise artifacts due to under-sampling.*

been developed to enable the pre-computation of ambient occlusion and diffuse inter-reflection. Some even allow for arbitrary modification of light and camera parameters. However, the application of these approaches to deformable geometry is much more constrained and thus usually requires a new pre-computation for any deformation.

The goal of our work is to drastically accelerate the calculation of volume ambient occlusion by taking advantage of the image based representation of medical volume data sets. While meshes are stored in a sparse format as a collection of vertices and triangles, volumes are stored in a 3D image grid which is suitable for a host of image processing operations and volume measurements that aren't possible on meshes. Our goal is to very quickly preprocess the volume in such a way that we can quickly extract information about the number of occluding voxels without having to traverse hundreds or thousands of rays. Generally speaking, we wish to replace the geometric definition of ambient occlusion with a statistical approximation that can be computed using image-processing operations.

This paper is structured as follows. In the next section we will discuss the related work with a focus on volume illumination methods. In order to approximate the occlusion for a point on an iso-surface, we query from a very compact representation of the distribution of values in the region above the surface. This is discussed in section 3. The implementation and some special considerations are discussed in section 4 and our results are discussed in section 5 before concluding in section 6 .

2. Related Work

Levoy and Cabral presented some of the first work on light transport for volumetric data sets and accelerated rendering using texture mapping [Lev90] [CCF94]. Max extended the optical models for direct volume rendering to include shadowing, single scattering an multiple scattering effects [Max95]. Max states that lighting from neighboring structures are important in volume rendering. Due to the added computational complexity involved with computing complex lighting effects most medical volume rendering applications only utilize a local illumination model due to its low computational cost. This involves illuminating the volume using one or more point light sources. In a local lighting model the light at each point in the volume is calculated as the sum of diffuse and specular components calculated from a bidirectional reflectance distribution function (BRDF) model. A BRDF model such as the popular Blinn-Phong model [Bli77] provides a means to calculate the amount of locally reflected light based on the directions of the light source, L, the viewer, V, and the surface normal (or gradient), N.

Local illumination methods provide good perceptual cues to the *orientation* of a surface within the volume, due to the diffuse $N \cdot L$ term. Surfaces are bright if lit from directly above, and dark if illuminated from a steep angle. However, as discussed above local illumination methods provide poor cues to the *relationships* between neighboring surfaces. It can be difficult to tell whether a neighboring surface is above or below an adjacent surface. Due to the extra perceptual information and realism they can provide, adding shadows and complex lighting to real-time volume rendering applications has resulted in significant research effort.

Unfortunately, these efforts are complicated by the fact that different volume rendering methods exist and have an impact on how and when light can be propagated. The two primary volume rendering styles are iso-surface and direct volume rendering (DVR), while the two primary volume rendering methods are texture slicing and ray-casting. Simple point light shadows are easily added to iso-surface ray-casting by casting an extra ray from the surface point back to the light. Soft shadows, and shadows within DVR ray-casting have proven much more difficult. A number of more advanced lighting techniques have been developed for slice-based DVR using a technique called half-angle slicing [KPHE02]. Half-angle slicing keeps track of rays from both the light's and eye's point of view and advancing them on the same slicing plane. Unfortunately, slice-based renderers are known to suffer from several rendering artifacts and are very difficult to optimize. Conversely, ray-casting is known to produce very high quality images and is much more easily optimized.

To maintain the quality and performance benefits of ray-casting, considerable research effort has been spent on incorporating efficient lighting algorithms into ray-casting engines. Stewart *et al.* pre-compute diffuse ambient lighting which they call *vicinity shading* in a separate volume which is used as a lookup during iso-surface rendering [Ste03]. They accelerated the computation using a 3D version of Bresenham's line drawing algorithm. Wyman *et al.* and Banks *et al.* furthered this technique by pre-computing or lazily computing global illumination lighting in a separate volume [WPH06] [BB08]. Unfortunately, pre-computed lighting effects can result in significant aliasing artifacts (see figure 2) unless twice the original resolution is used, resulting in an 8X-100X memory footprint depending on the type of pre-computed lighting [WPH06]. Hadwiger *et al.* [HKSB06] pre-compute *deep shadow maps* that represent a compressed attenuation curve from the light's point of view and can represent area light sources. Unfortunately, these maps must be recalculated and compressed each time the light is moved or transfer function is altered which is undesirable for an interactive application.

To address some of the above limitations, approximations have been proposed which are not strictly physically motivated, but lead to visually convincing and plausible results while still being feasible in real-time. For polygonal models, Bunnell *et al.* [Bun05] represent geometry as a hierarchical tree of discs for which simple form factors can be calculated to approximate occlusion. Shanmugam *et al.* [SA07] and Mittring *et al.* [Mit07] even go so far as to compute occlusion from the depth buffer as a post-process. While these approximate methods can be far from physically correct, they add a surprising amount of realism and accurate depth cues to the scene. In volume rendering, Desgranges *et al.* use a summed area table of the volume's opacity to quickly perform variable width blurring operations to approximate ambient occlusion [DP07]. Unfortunately the summed area table must be recomputed whenever the transfer function or isosurface is changed. Ropinski *et al.* compute approximate ambient occlusion by quantizing all the possible combinations of neighboring voxels such that they can apply the transfer function dynamically [RMSD*08]. This method can approximate ambient occlusion as well as color bleeding but suffers from a lengthy compression process and quantization artifacts during rendering.

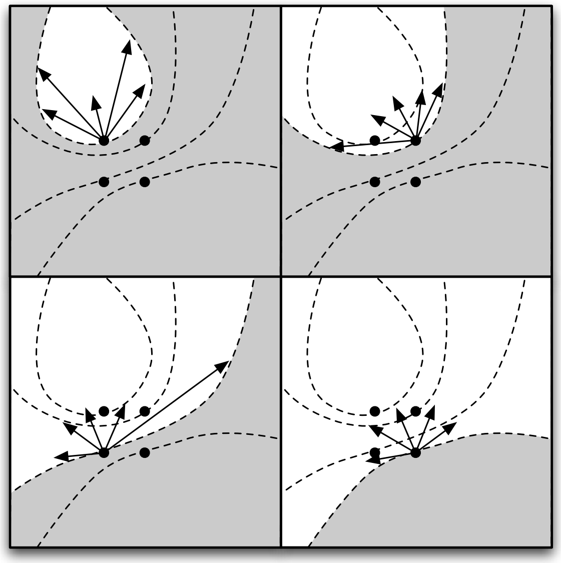


Figure 2: *Illustration of pre-computing volume lighting: Neighboring sampling locations can have highly different occlusion. In order to capture these high frequencies a higher resolution must be used for pre-computed lighting effects.*

3. Algorithm Overview

3.1. Ambient Occlusion

The lighting model used in most applications uses only local reflectance to represent the light leaving a given point p in the direction of the eye. In order to account for complex occlusions from neighboring structures we would like to replace this with a new term that takes into account the irradiance arriving at a surface from all angles. This can be represented as:

$$\int_{\Omega} L(\omega)d\omega \qquad (1)$$

where $L(\omega)$ is the radiance arriving from direction ω, and Ω is the set of directions above the surface point (where N is the surface normal and $N \cdot \omega > 0$). Since we are interested in the diffuse reflection of the incoming irradiance, a *cosine-weighted* contribution of incoming radiance is often used:

$$\int_{\Omega} N \cdot L(\omega)d\omega \qquad (2)$$

In this case $L(\omega)$ is represented as a vector with magnitude equal to the radiance. This equation can be approximately evaluated using ray casting by discretizing the domain Ω into k sectors of equal solid angle $\Delta\omega$.

$$\sum_{i=0}^{k} N \cdot L(\omega_i)\Delta\omega \qquad (3)$$

Since we have chosen to focus on an iso-surface model, computing $L(\omega_i)$ is heavily simplified. If a ray ever enters the iso-surface then it immediately becomes fully occluded. After an occluding iso-surface is found, we can proceed to evaluate the amount of occlusion using an *all-or-nothing* method or a *partial occlusion* method. In the all-or-nothing method $L(\omega_i) = 0$ when ω_i hits an occluding voxel. Otherwise, $L(\omega_i) = 1$. In the partial occlusion method $L(\omega_i)$ uses the unobstructed distance to the first occlusion to determine the amount of occlusion using a basic linear or quadratic fall off function. The partial occlusion method works better in completely occluded spaces or tight spaces where the all-or-nothing method would result in complete occlusion.

3.2. Neighborhood Approximation

Computing the integral mentioned above would require casting many rays. Instead, we would like to make use of the discretized nature of the volume to compute occlusion. The basic assumption we rely on in doing this is that the percentage of occluding voxels surrounding the surface provides a good approximation to the percentage of rays that would be occluded while traversing through the same space. While this is not strictly correct in all cases, it is a commonly used assumption in approximate techniques [SA07] [Mit07] where real-time performance is required.

Under this assumption, we can now define occlusion using the neighborhood around a surface point and define the irradiance at a point p as:

$$\frac{1}{|V|}\int_V L'(v)dv \tag{4}$$

Here, V is a volume above the surface we are evaluating, $L'(x)$ is now simply a binary function of the iso-surface c defined as:

$$L'(x) = \begin{cases} 1 & \text{if } x \le c \text{ (not occluding)}, \\ 0 & \text{otherwise (occluding)}. \end{cases} \tag{5}$$

By discretizing the volume into k voxels we can express this discretely as:

$$\sum_{i=0}^{k} L'(v_i)\Delta v \tag{6}$$

Whereas the ray-based approximation from equation 3 is dependent on a hemisphere of angles Ω, our approximation in 6 is dependent only on a hemispherical or spherical volume V which lies above a surface point (see figure 3).

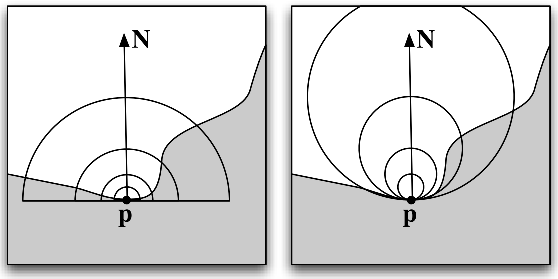


Figure 3: *Occlusion is calculated using the percentage of the a region that is within the iso-surface (grey area). Left: Hemispherical regions. Right: Spherical regions. Different sizes can be combined to better localize occlusions.*

3.3. Filterable Occlusion Maps

While our new definition of occlusion allows us to determine the amount of occlusion using a simple neighborhood around a surface point, computing this directly by testing each voxel in the region would still be prohibitively expensive. We note here that one voxel of a conventional volume data-set can only represent the image value at that one location. If we instead had knowledge of a *distribution* of values in a spherical region surrounding each point, we could do one test against the distribution instead of one test for every voxel. The optimal function for performing this test is the *cumulative distribution function* (CDF). If we think of a region as a distribution of values X, the CDF is defined as:

$$CDF(x) = P(X \le x) = \sum_{x_i \le x} P(X = x_i) = \sum_{x_i \le x} p(x) \tag{7}$$

Given this CDF function we can determine the exact percentage of voxels that are inside/outside any iso-surface in the region. If we look at the properties of the CDF, we can see it is actually equivalent to our new definition of ambient irradiance from equation 6. Unfortunately, while pre-computing CDF functions for each spherical region would allow us to lookup the occlusion with one lookup, the memory required to store all these CDFs uncompressed would be monumental.

Thankfully, by reducing the ambient occlusion problem to storing and evaluating a CDF, we can make use of a lot of research from a slightly different domain. Shadow mapping algorithms [Wil78] are faced with a similar comparison problem when testing an object's depth against a shadow map. In fact a soft shadow technique known as percentage closer filtering [RSC87] performs a brute force depth test against a region in the shadow map which is identical to the test described in equation 6. The main difference is that the shadow map region is in 2D and the volume region is in 3D (see figure 4). A significant amount of recent research has focussed on representing distributions in compact filterable forms.

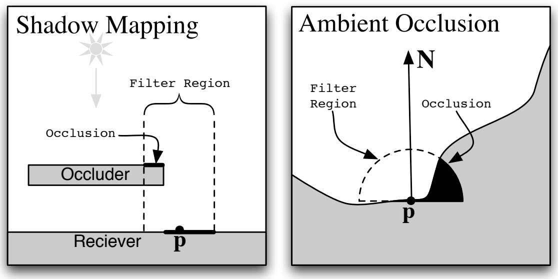


Figure 4: *Comparison of the soft shadow mapping problem to the approximate ambient occlusion problem. Left: In shadow mapping, we want to determine the percentage of non-occluding shadow map texels (greater than or equal to our reciever depth). Right: In our ambient occlusion approximation we wish to determine the percentage of non-occluding voxels (less than the iso-value). Both can be determined using the CDF of the filter region.*

3.4. Variance Occlusion Maps

We chose to use a technique described by Donnelly *et al.* [DL06] to represent a distribution of image values and approximately query the CDF. To very compactly approximate a distribution, they store only the first two moments of the distribution: the image value and the squared image value. The advantage of this representation is that it can approximate the average of several distributions by averaging the moments. This means that the image may simply be blurred to generate a distribution centered at each pixel, where the blurring/filtering kernel represents the image region that will be represented by the distribution. As in [DL06] we can describe these two moments M_1 and M_2 as:

$$M_1 = E(x) \quad = \int_{-\infty}^{\infty} x p(x) dx \tag{8}$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2 p(x) dx \tag{9}$$

We can then calculate the mean μ and variance σ^2 of the distribution:

$$\mu = E(x) = M_1 \tag{10}$$

$$\sigma^2 = E(x^2) - E(x)^2 = M_2 - M_1 \tag{11}$$

Since the variance gives us a measure of the width of the distribution, we can place a bound on how much of the distribution can be found a certain distance away from the mean. Chebychev's inequality states this bound precisely as:

$$P(x \leq t) \leq p_{max}(t) \equiv \begin{cases} 1 & \text{if } t \leq \mu\,, \\ \frac{\sigma^2}{\sigma^2 + (t-\mu)^2} & \text{otherwise} \end{cases} \tag{12}$$

Donnelly *et al.* further demonstrate that while this only provides an upper bound on the CDF, it is very accurate in the case of a bi-modal distribution containing only two image values d_1 and d_2. This occurs in shadow maps when there is one occluding object at d_1 and one partially occluded object at d_2. We have found that this is also very often the case in volume data sets where different materials are represented by different iso-values.

In this simple bi-modal case, the inequality becomes an equality and querying equation 12 with d_2 gives the exact percentage of d_2 in the region, which is the same result as the CDF. This represents the percentage of occluding voxels in the region. Of course this becomes less accurate as the distribution takes different forms or the query differs from d_2, but it provides a surprisingly good approximation in many data-sets, especially when only a small carefully chosen region of the volume is considered.

To verify that this approximation is valid, we analyzed a number of data-sets (see figure 5). We found that in medical data-sets, CT values were heavily clustered around different materials such as air, soft tissue, and bone. We found that non-medical data-sets such as the stanford bunny data-set were even more clustered, having almost entirely bi-modal distributions. However, MRI data-sets and distance-field data-sets were much less clustered and thus this technique will be less valid for these types of data.

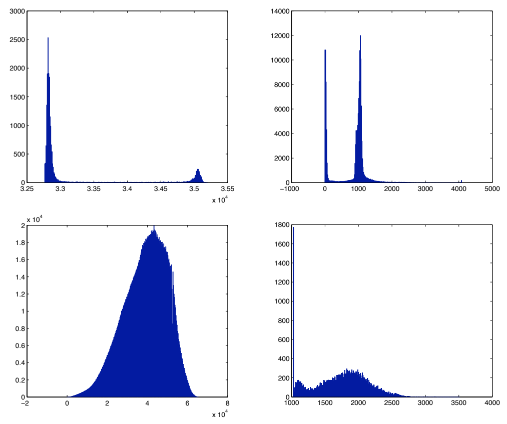


Figure 5: *Histograms of volume data-sets. We found CT data tends to be heavily clustered and works well with our technique. Top-Left: Stanford Bunny CT. Top-Right: Head CT. Bottom-Left: Signed Distance Field. Bottom-Right: Head T1 Weighted MRI*

3.5. Handling Arbitrary Isosurfaces

Unfortunately, even though our data is clustered such that distributions tend to be bimodal, there is no limitation on the isovalue that is chosen by the user. Thus isosurfaces can fall anywhere in between d_1 and d_2. As the isovalue moves towards the mean of the distribution it becomes less accurate. We found this to be a major drawback of this representation but found that it could still be quite useful if the region is chosen carefully (see section 4) .

4. Implementation

4.1. Preprocessing

As a preprocessing step, we create a new volume containing the two moments of the distribution (value and value squared). We then perform separable convolutions with gaussian kernels to generate local distributions centered at each voxel. Storing many of these volumes would take a large amount of memory, so we chose to double the gaussian kernel size at each iteration and down-sample by a factor of two at each iteration. This allows us to store the entire variance ambient map in a mip-mapped two channel floating-point texture. Since we are interested in fairly large filter regions, we also found that we could easily get away with starting at half the resolution of the original volume or even smaller resolutions if only low frequency occlusions are required. At one half resolution the variance ambient map is roughly 0.6X the size of the original volume including mipmaps if a two channel 32-bit floating point texture is used.

4.2. Blurring Kernel

It is worthwhile to discuss the choice of blurring kernel we used to generate distributions. While we would normally want to use a hemispherical filter region with cosine and distance weighted occluders, this would require a complicated non-seperable blurring process based on the surface normal of each voxel. Since the surface normal can change rapidly, this representation would suffer from aliasing artifacts under linear interpolation of neighboring samples (see figure 6). This becomes especially apparent when lower resolution volumes or mipmaps are used. By using a radially symmetric kernel and offsetting the sampling location from the surface location we can eliminate this aliasing while still allowing high-frequency changes in the surface normal. The one drawback is that the occlusion becomes slightly more concentrated in the direction of the surface normal. Being able to adjust the surface normal during interactive rendering allows us to perform some other interesting effects like averaging the normal and view vector to get view dependent ambient occlusion. Anisotropic spacing can be supported by varying the width of the kernel.

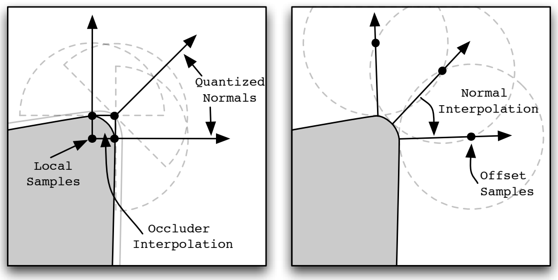


Figure 6: *Illustration of different blurring kernels. While a hemisphere is normally used to search for occluders, it can result in artifacts due to the interpolation of very different occluding regions. By using a spherical kernel and offsetting the sampling location from the surface, the correct normal is used to determine the angle of the occlusion region.*

4.3. Rendering

During interactive rendering, we first search to the isosurface and then perform a binary search to refine the surface location. We then calculate the gradient using central-differencing to perform local lighting. We also use the gradient to calculate offsets for querying the variance ambient map. Since many isosurfaces tend to fall on partial-volumes where one material meets another, we take an extra sample one half of a voxel under the surface in the direction of the gradient (see figure 7) and use this sample as the isovalue to evaluate the amount of occlusion in the variance occlusion map. In the case of ambient occlusion we can evaluate four instances of equation 12 in parallel (one sample from the first 4 mip levels) and then take the maximum occlusion value. In the case of soft shadows, we cast a ray four steps at a time evaluating equation 12 until the ray leaves the volume. To simulate soft shadows from area light sources, the mip level can be increased logarithmically as it travels towards the light.

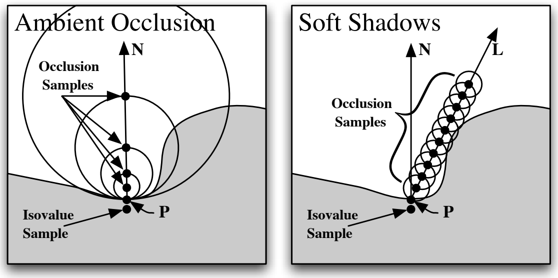


Figure 7: *Illustration of our dynamic ambient occlusion and soft shadow algorithms. The isovalue is chosen one half of a voxel behind the isosurface. Ambient occlusion requires only 3-4 samples of different sizes in the direction of the surface normal. Soft shadows requires one ray cast in the direction of the light. In both cases the occlusion sample with the most occlusion is chosen.*

5. Results

All of the images and timings in this section were generated on a Mac Pro with a single NVIDIA GeForce 8800 GTS. Figure 8 shows 4 offset occlusion samples from separate mip-levels in the occlusion map. Figure 9 shows the combined result with diffuse local lighting. Figure 10 shows soft shadows calculated with one ray combined with ambient occlusion.

5.1. Performance

Preprocessing the volume simply involves blurring and down sampling the volume. We used a 5x5x5 separable gaussian blur to generate variance volumes. This processing can be greatly accelerated on the GPU. Since we are dealing with iso-surfaces, lighting can be calculated as a post process, independent of volume size. The one exception is our soft shadow algorithm which requires an extra ray to be cast until it exits the volume or becomes fully occluded.

6. Conclusion and Discussion

We have presented a novel method for generating approximate isosurface ambient occlusion and soft shadows. Compared with other approaches which may require hours or

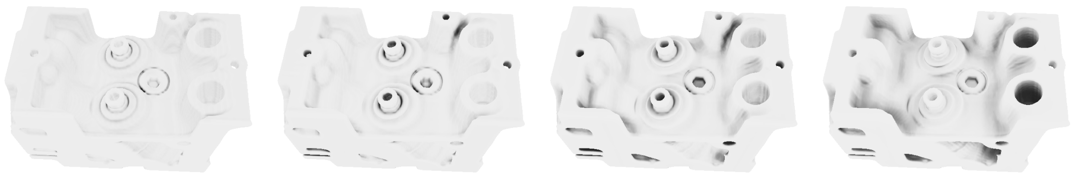


Figure 8: *Occlusion sampled from the first 4 levels of the occlusion map. These are combined by taking the maximum occlusion (minimum irradiance) or average occlusion.*

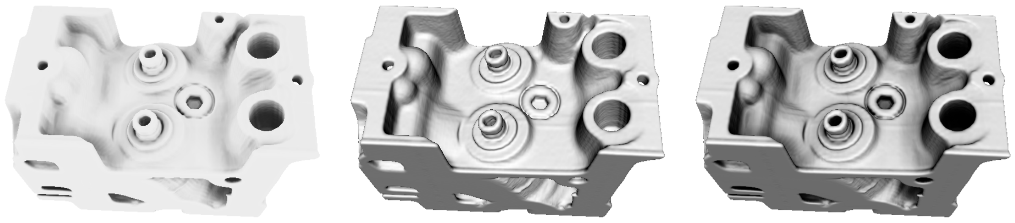


Figure 9: *Left: Combined occlusion from figure 8 using maximum occlusion. Center: Simple diffuse lighting. Right: Diffuse combined with ambient occlusion.*

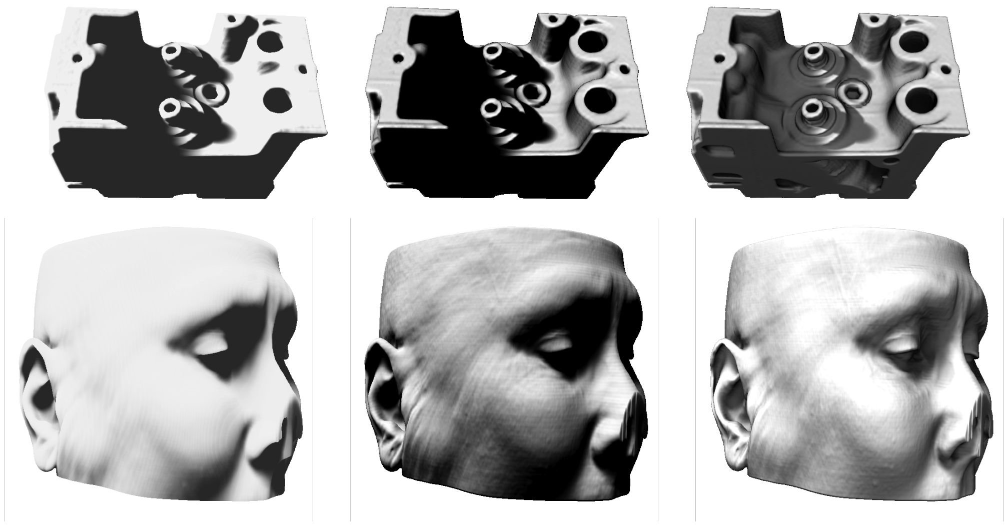


Figure 10: *Left: Approximate soft shadows using only one ray. Center: Diffuse lighting with shadows. Right: Diffuse lighting, ambient occlusion and soft shadows.*

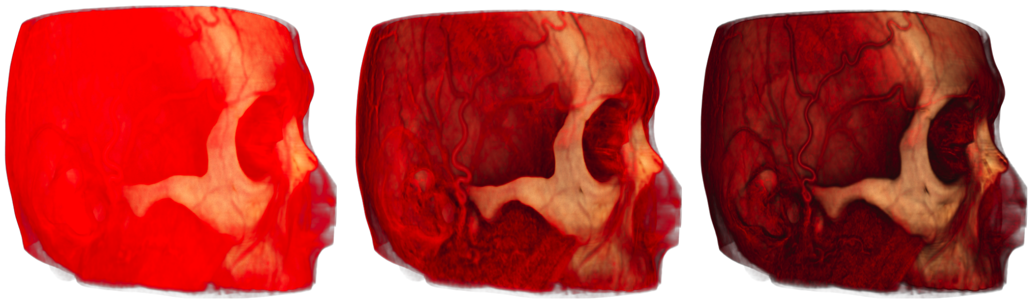


Figure 11: *Left: DVR. Center: DVR with AO. Right: DVR with AO and local phong lighting. If transparency is intended to provide detail-in-context rather than specify transparency to incoming light, then our occlusion method is still useful within DVR. We also found it useful to adjust the shadow contribution based on transparency.*

Preprocessing Time (in milliseconds)		
Dataset	CPU	GPU
Engine (256^3)	170	8.6
Small Head (512^2x256)	678	34.5
Big Head (512^3)	1524	74.1

Table 1: *Preprocessing time on the CPU and GPU*

Rendering Time (in milliseconds)				
Dataset	Phong	AO	HS	SS
Engine (256^3)	7.1	7.4	19.1	25.2
CTHead1 (512^2x256)	7.8	8.2	26.4	30.3
CTHead2 (512^3)	8.0	8.3	32.5	37.0

Table 2: *Rendering time using phong, ambient occlusion(AO), hard shadows(HS) and soft shadows(SS)*

days of pre-computation time and suffer from aliasing and/or quantization artifacts, our approach can load a data set within seconds, has very good performance and doesn't suffer from the aliasing of pre-computed lighting techniques.

We found the most serious limitation of our work to be the statistical approximation provided by the first two moments. We found that adjusting the offset of the spherical distribution produced very interesting effects similar to subsurface scattering, but we were often unable to do this because the variance test would begin to behave incorrectly as the mean of the local distribution approached the isovalue.

In conclusion, we believe that filterable occlusion maps can provide a very quick way to approximate complex lighting effects, but if it is to be used in practice a better CDF approximation will be needed. We are currently investigating Convolution Shadow Maps since they represent the entire CDF and aren't restricted to bi-modal distributions.

References

[BB08] BANKS D. C., BEASON K.: Fast global illumination for visualizing isosurfaces with a 3d illumination grid. *Computing in Science and Engg. 9*, 1 (2008).

[Bli77] BLINN J. F.: Models of light reflection for computer synthesized pictures. In *SIGGRAPH '77* (New York, NY, USA, 1977), ACM, pp. 192–198.

[Bun05] BUNNELL M.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Dynamic Ambient Occlusion and Indirect Lighting.

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94.* (New York, NY, USA, 1994), ACM, pp. 91–98.

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph. 11*, 2 (1977).

[DL06] DONNELLY W., LAURITZEN A.: Variance shadow maps. In *I3D '06* (New York, NY, USA, 2006), ACM, pp. 161–165.

[DP07] DESGRANGES P. E. K.: Us patent application 2007/0013696 a1: Fast ambient occlusion for direct volume rendering, 2007.

[HKSB06] HADWIGER M., KRATZ A., SIGG C., BÜHLER K.: Gpu-accelerated deep shadow maps for direct volume rendering. In *GH '06* (New York, NY, USA, 2006), ACM, pp. 49–52.

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (dec 2003), 753–774.

[HSBW99] HUBONA G., SHIRAH G., BRANDT M., WHEELER P.: The role of object shadows in promoting 3-d visualization, 1999.

[KMK94] KERSTEN D., MAMASSIAN P., KNILL D.: *Moving Cast Shadows and the Perception of Relative Depth.* Tech. Rep. 6, Max Planck Institute for Biological Cybernetics, T§bingen, Germany, jun 1994.

[KPHE02] KNISS J., PREMOZE S., HANSEN C., EBERT D.: Interactive translucent volume rendering and procedural modeling. In *IEEE Vis* (2002), pp. 109–116.

[Lev90] LEVOY M.: Efficient ray tracing of volume data. *ACM Trans. Graph. 9*, 3 (1990), 245–261.

[Max95] MAX N.: Optical models for direct volume rendering. *IEEE Vis. 1*, 2 (1995), 99–108.

[Mit07] MITTRING M.: Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, pp. 97–121.

[RMSD*08] ROPINSKI T., MEYER-SPRADOW J., DIEPENBROCK S., MENSMANN J., HINRICHS K. H.: Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Eurographics 27*, 2 (2008).

[RSC87] REEVES W. T., SALESIN D. H., COOK R. L.: Rendering antialiased shadows with depth maps. In *SIGGRAPH '87* (New York, NY, USA, 1987), ACM.

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *I3D* (New York, NY, USA, 2007), ACM, pp. 73–80.

[Ste03] STEWART A. J.: Vicinity shading for enhanced perception of volumetric data. In *IEEE Vis* (10 2003).

[Wan92] WANGER L.: The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *SI3D '92* (New York, NY, USA, 1992), ACM.

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH 12*, 3 (1978), 270–274.

[WPH06] WYMAN M.-C., PARKER M.-S., HANSEN S. M.-C.: Interactive display of isosurfaces with global illumination. *IEEE Vis 12*, 2 (2006), 186–196.