Chapter 1

Introduction

Three-dimensional arrays of data are a convenient and widely-used representation for information. Medical imaging technologies such as magnetic resonance (MR) and computed tomography (CT) can produce 3D arrays of data containing detailed representations of internal organs, allowing doctors to make a diagnosis without invasive surgery. Confocal microscopy can produce data revealing the 3D internal structure of individual cells in a biological sample with details as small as 0.1 microns. In seismic exploration, acoustic waves triggered by an explosive can be used to produce 3D maps of geologic structures below the surface of the Earth. As a final example, simulations based on computational fluid dynamics produce data on a 3D grid that can be used to predict the aerodynamics of a proposed automobile design or the regional weather.

All of these measurement and simulation techniques can produce very large arrays of numbers that are difficult to understand due to the sheer quantity of data. Data visualization techniques specialized for large 3D arrays are necessary. One emerging technique is volume rendering, a method for producing an image from a 3D array of sampled scalar data. Figure 1.1 shows a volume rendering produced from a CT scan of a human head. Volume rendering is a robust and versatile method for visualizing 3D arrays, but it has one key disadvantage that has prevented its widespread use: existing volume rendering algorithms are computationally expensive.

The high computational cost of volume rendering stems from the large size of typical

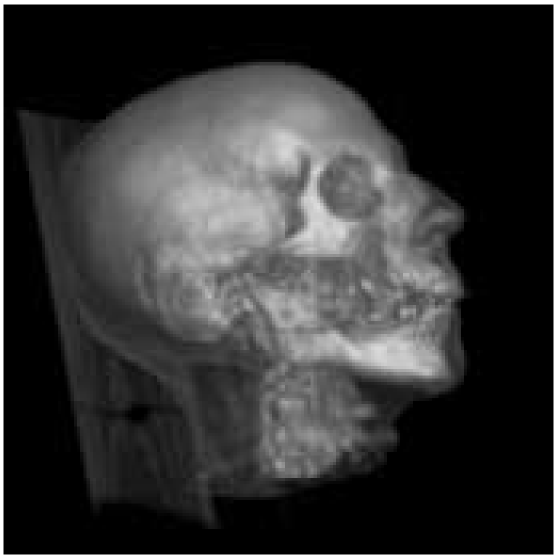


Figure 1.1: Example of a volume rendering produced from a CT scan of a human head. The input data contains 15 million samples.

data sets. Arrays containing many megabytes of data are common, and rendering such quantities of data quickly enough for interactive applications requires large amounts of processing power and high memory bandwidth. Past approaches to speeding up volume rendering have included trading off image quality for speed, or using large, expensive multiprocessors or special-purpose hardware.

This thesis presents a family of new volume rendering algorithms with lower computational costs and improved memory performance. The performance improvements stem from high-level algorithmic optimizations that exploit coherence in the data more efficiently than previous approaches. The new algorithms achieve rendering rates that are fast enough for interactive applications on desktop workstations, without using specialized hardware and without significantly compromising image quality. Parallel versions of the new algorithms running on small- to medium-scale multiprocessors achieve real-time rendering rates.

The rest of this chapter describes volume rendering in more detail, motivates the need for fast algorithms, and summarizes the contributions of this thesis.

1.1 Volume Rendering

The basic steps in any volume rendering algorithm consist of assigning a color and an opacity to each sample in a 3D input array, projecting the samples onto an image plane, and then blending the projected samples. The foundation of this visualization technique is a physically-based model for the propagation of light in a colored, semi-transparent material. In Section 1.1.1 we develop the model by writing an energy balance equation and reducing it to the volume rendering equation. In Section 1.1.2 we show how to simplify the volume rendering equation to a form known as the volumetric compositing approximation. This approximation is the starting point for the algorithms in this thesis.

1.1.1 The Volume Rendering Equation

The input to a volume rendering algorithm is a 3D array of scalars (as opposed to vectors which have a directional component). The array is called a *volume* and each element of the array is called a *voxel*. Although some volume rendering algorithms assume that each voxel represents a small cube in space with a constant scalar value, we will assume a voxel represents a point sample of a continuous scalar function.

Volume rendering is an approximate simulation of the propagation of light through a *participating medium* represented by the volume (Figure 1.2). The medium can be thought of as a block of colored, semi-transparent gel in which the color and opacity are functions of the scalar values in the input array. As light flows through the volume it interacts with the gel via several processes: light can be absorbed, scattered or emitted by the volume. Many other types of interactions are also possible, such as phosphorescence (absorption and re-emission of energy after a short time delay) and fluorescence (absorption and re-emission of energy at a different frequency). However since the goal of volume rendering is data visualization, not accurate simulation of the physics, we will omit many parts of the full optical model that are not necessary for the visualization. Given a simplified optical model, a volume rendering algorithm produces an image by computing how much light reaches each point on an image plane.

Light transport is governed by a special case of the Boltzmann equation from *transport theory*, which is the study of how a statistical distribution of particles (such as photons) flows

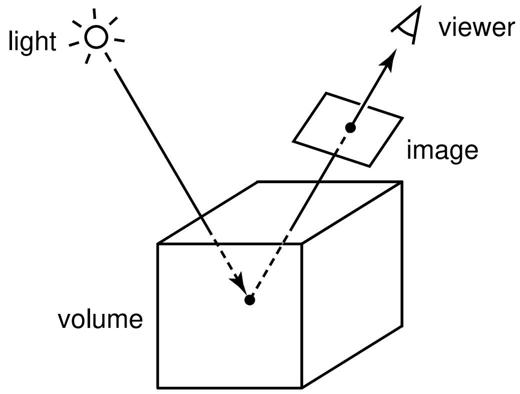


Figure 1.2: A physical model for volume rendering: A light ray propagates through a cube of semi-transparent gel and scatters onto the image plane.

through an environment. Arvo [1993], Glassner [1995, Ch. 12] and Cohen & Wallace [1993, Ch. 2] contain introductions to transport theory and its application to computer graphics, and Siegel & Howell [1992] contains a more detailed treatment of the underlying physics. Here we summarize the major results.

The flow of photons in a fixed environment rapidly reaches equilibrium, so the number of photons flowing through a given region of space in a particular range of directions must be constant over time. Thus if we consider the photons in a differential volume about a point $\mathbf{r}$ traveling in a differential range of angles about a direction $\vec{\omega}$ and we sum the change in the number of photons due to each possible type of interaction with the volume then the net change must be zero.

Instead of counting photons we will write an energy balance equation in terms of *radiance*, designated by $L(\mathbf{r}, \vec{\omega})$. Radiance describes the density of power transmitted by the photons at a particular point in a given direction. More precisely it is the power per unit area (projected onto the direction of flow) per unit solid angle, and it has units[1] of W/m^2-sr.

[1] Units will be expressed using the MKS system. The meter (m) is the unit of distance and the watt (W) is the unit of power (energy per unit time). The steradian (sr) is the unit of solid angle.

The energy balance equation states that the directional derivative of the radiance along a particular ray equals the sum of three terms representing losses and gains due to interactions with the volume:

$$\vec{\omega} \cdot \nabla L(\mathbf{r}, \vec{\omega}) = -\phi_t(\mathbf{r}) L(\mathbf{r}, \vec{\omega}) + \epsilon(\mathbf{r}, \vec{\omega}) + \int_{\mathcal{S}^2} k(\mathbf{r}, \vec{\omega}\,' \to \vec{\omega}) L(\mathbf{r}, \vec{\omega}\,')\, d\omega' \tag{1.1}$$

$\phi_t(\mathbf{r})$ is the *extinction coefficient* (with units of m^{-1}) which equals the probability per unit distance that a photon traveling along the ray will be either absorbed or scattered into a different direction by the volume.

$\epsilon(\mathbf{r}, \vec{\omega})$ is the *emission function* (with units of W/m^3-sr) that accounts for photons emitted within the volume.

$k(\mathbf{r}, \vec{\omega}\,' \to \vec{\omega})$ is the *scattering kernel* (with units of $\mathrm{sr}^{-1}\ \mathrm{m}^{-1}$) which is the probability per unit solid angle per unit distance that a photon moving in direction $\vec{\omega}\,'$ will scatter into direction $\vec{\omega}$. We integrate this function against the incoming radiance from all directions (denoted by $\mathcal{S}^2$) to account for the photons scattered into the ray.

Equation 1.1 is a first order differential equation known as the *differential form of the equation of transfer*. It can be written in an equivalent *integral form* as follows:

$$L(\mathbf{r}, \vec{\omega}) = e^{-\tau(\mathbf{r}, \mathbf{r_B})} L_B(\mathbf{r_B}, \vec{\omega}) + \int_{\Gamma(\mathbf{r}, \mathbf{r_B})} e^{-\tau(\mathbf{r}, \mathbf{r}')} Q(\mathbf{r}', \vec{\omega})\, dr' \tag{1.2}$$

$\tau(\mathbf{r}, \mathbf{s})$ is the integral of the extinction coefficient along the straight-line path between points $\mathbf{r}$ and $\mathbf{s}$:

$$\tau(\mathbf{r}, \mathbf{s}) \equiv \int_{\Gamma(\mathbf{r}, \mathbf{s})} \phi_t(\mathbf{r}')\, dr'$$

The path between the two points is denoted by $\Gamma(\mathbf{r}, \mathbf{s})$. We note that $e^{-\tau(\mathbf{r}, \mathbf{s})}$ is the integrating factor used to convert Equation 1.1 into Equation 1.2.

$L_B(\mathbf{r}, \vec{\omega})$ is a function specifying boundary conditions over a closed surface surrounding the volume. The point $\mathbf{r_B}$ is the intersection between the closed surface and the ray from $\mathbf{r}$ in direction $\vec{\omega}$.

$Q(\mathbf{r}, \vec{\omega})$ is short-hand for the sum of the emission and scattering terms:

$$Q(\mathbf{r}, \vec{\omega}) \equiv \epsilon(\mathbf{r}, \vec{\omega}) + \int_{\mathcal{S}^2} k(\mathbf{r}, \vec{\omega}\,' \to \vec{\omega}) L(\mathbf{r}, \vec{\omega}\,')\, d\omega'$$

This function can be thought of as a generalized source function that accounts for all of the gains in the energy balance equation.

The integral form of the equation of transfer states that the radiance at any point along a ray equals a contribution from the radiance entering at the boundary of the volume plus a sum of contributions from the source terms along the ray. All of these contributions are multiplied by an exponential attenuation function that depends on the distance between the source and the measurement point (located at $\mathbf{r}$).

Equation 1.2 is a generalization of many equations used in rendering. Kajiya's rendering equation [Kajiya 1986] is equivalent to the equation of transfer specialized for environments consisting of surfaces (represented by boundary conditions) separated by empty space, in which case volume absorption, emission and scattering drop out of the equation. Zonal radiosity is a method for solving the equation of transfer assuming that the scattering and emission functions are isotropic, i.e. independent of $\vec{\omega}$ [Rushmeier & Torrance 1987]. Finally, with a different set of assumptions we arrive at the volume rendering equation.

First, we only model single scattering, i.e. all of the photons reaching the image are assumed to have scattered only once after leaving a light source. Second, we ignore absorption between the light source and the scattering event (but not between the scattering event and the image plane). Third, we assume isotropic absorption. Fourth and finally, we choose a simple boundary condition: we assume that the only energy entering the volume comes from a finite set of point light sources.

The first two assumptions allow us to drop the scattering integral from the equation of transfer since we can use the emission term to model the single scattering event. In this model the light sources deposit energy at each voxel and the voxels then re-emit the energy. The emission function describes the amount of re-emitted energy as a function of the viewing angle. Typical volume rendering algorithms compute the value of the emission function at a voxel using a local illumination model that is a function of the voxel's scalar value and the positions of the light sources. Since we have ignored absorption between the light source

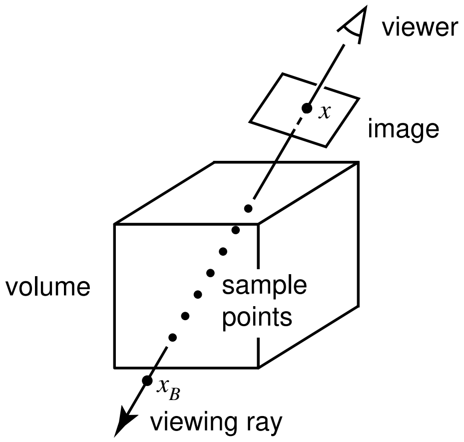


Figure 1.3: A simplified model for volume rendering: Each voxel in the volume emits light (computed using a local illumination model) and absorbs a fraction of the light passing through it, but scattering is ignored. The value of a pixel is computed by sampling the voxel values along a viewing ray from a point x on the image to a point x_B on the opposite boundary of the volume and then numerically evaluating the volume rendering integral.

and the voxel this model cannot be used to produce shadows. We discuss volume rendering algorithms that correctly render shadows in Chapter 7.

With all four assumptions Equation 1.2 reduces to the *volume rendering equation*:

$$L(x) = \int_{x}^{x_B} e^{-\int_{x}^{x'} \phi_t(x'')dx''} \epsilon(x')dx' \tag{1.3}$$

In this equation we have reparametrized the radiance in terms of a one-dimensional position variable x representing distance along a viewing ray. The upper limit of integration is x_B, the point at which the ray exits the volume.

A simple ray casting algorithm based on the volume rendering equation operates by tracing rays into the volume parallel to the viewing direction as shown in Figure 1.3. The rendering algorithm then numerically evaluates the integral in Equation 1.3 along each ray. The user of the volume renderer specifies a mapping from the scalar value associated with each voxel to the parameters in the physical model, such as the absorption coefficient and the parameters of the illumination model for computing the emission function. In the next section we will describe a common method for evaluating the integral, called volumetric compositing, and we will refine the ray casting algorithm.

The volume rendering equation derived in this section is equivalent to the volume rendering models proposed by Blinn [1982], Levoy [1989] and Sabella [1988]. Other volume rendering models based on the more general equation of transfer, which includes multiple scattering, have also been proposed: Kajiya & Von Herzen [1984] derive an approximation for the case of a volume containing high-albedo (highly-reflective) particles, Krüger [1990] evaluates the full equation using Monte Carlo integration techniques, and Sobierajski & Kaufman [1994] perform volumetric ray tracing including ideal specular reflections at isosurfaces in the volume.

A key advantage of volume rendering is that the volume data need not be thresholded, in contrast to surface rendering techniques. Surface rendering techniques for volume data operate by fitting polygons to an isosurface in the volume (using, for instance, the marching cubes algorithm [Lorensen & Cline 1987]), and then rendering the polygonal model with traditional polygon rendering techniques. The surface-fitting process requires making binary decisions, i.e. the volume must be thresholded to produce regions that are either inside or outside the isosurface. If the volume contains fuzzy or cloud-like objects then a polygonal surface will be a poor approximation. In contrast, volume rendering algorithms never explicitly detect surfaces so they naturally handle fuzzy data as well as sharply-defined surfaces.

1.1.2 The Volumetric Compositing Approximation

In the previous section we began with an energy balance equation, we simplified the equation by ignoring some of the physical interactions, and we arrived at the volume rendering equation. In this section we derive an approximate numerical solution to the volume rendering equation called the *volumetric compositing equation*. Many volume rendering algorithms use this approximation, including the new algorithms in this thesis.

The volume rendering equation derived in the previous section is:

$$L(x) = \int_{x}^{x_B} e^{-\int_{x}^{x'} \phi_t(x'')dx''} \epsilon(x')dx'$$

The integrals in this equation can be evaluated using the rectangle rule:

$$L(x) = \sum_{i=0}^{n-1} e^{-\sum_{j=0}^{i-1} \phi_j \Delta x} \cdot \epsilon_i \Delta x = \sum_{i=0}^{n-1} \epsilon_i \Delta x \cdot \prod_{j=0}^{i-1} e^{-\phi_j \Delta x}$$

where

$$\begin{aligned} \epsilon_i &\equiv \epsilon(x + i\Delta x) \\ \phi_i &\equiv \phi_t(x + i\Delta x) \end{aligned}$$

for some sample spacing Δx along the viewing rays. The expression on the right side of the equation can be written using the "over" operator from digital compositing [Porter & Duff 1984] as we can see by making the following substitutions. First define:

α_i	$\equiv$	$1 - e^{-\phi_i \Delta x}$	the *opacity* of sample i
C_i	$\equiv$	$(\epsilon_i / \alpha_i) \cdot \Delta x$	the *color* of sample i
c_i	$\equiv$	$C_i \alpha_i$	the premultiplied color and opacity

The premultiplied color and opacity is a convenient quantity for digital compositing [Porter & Duff 1984]. We have defined the "color" of a sample to be proportional to its emission normalized by its opacity. Intuitively a voxel with vanishing opacity must contain empty space so its emission should also vanish and this ratio is well-defined. This definition of color has no rigorous physical interpretation but it does have the same units as radiance (mapping radiances to displayable colors is a general problem in physically-based renderers, most of which perform some form of ad-hoc scaling just before displaying an image).

Given these definitions the volume rendering equation can be rewritten as follows:

$$\begin{aligned} L(x) &= \sum_{i=0}^{n-1} c_i \cdot \prod_{j=0}^{i-1} 1 - \alpha_j \\ &= c_0 + c_1(1-\alpha_0) + c_2(1-\alpha_0)(1-\alpha_1) + \cdots \\ &\qquad + c_{n-1}(1-\alpha_0)\cdots(1-\alpha_{n-2}) \\ &= c_0 \text{ over } c_1 \text{ over } c_2 \text{ over } \cdots \text{ over } c_{n-1} \end{aligned} \tag{1.4}$$

Equation 1.4 is the volumetric compositing equation. It describes a method for numerically evaluating the volume rendering equation.

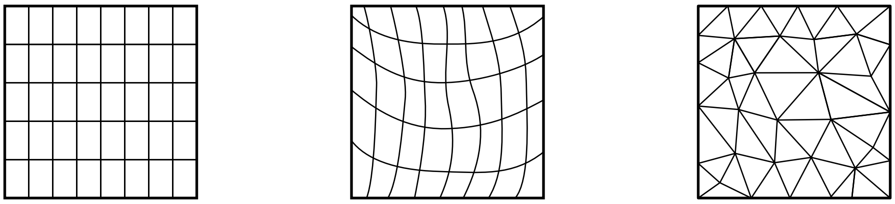
If we incorporate the volumetric compositing equation into the ray casting algorithm introduced in the previous section, we can render a volume as follows:

1. For each image pixel, cast a ray through the volume.
2. At regularly-spaced sample points along the ray compute a color C_i and an opacity α_i from the scalar value in the volume.
3. Combine the colors and opacities along the ray using Equation 1.4.

This is a simple brute-force algorithm. If the volume contains n voxels per side and the image is computed by casting n^2 rays with n sample points per ray then the algorithm requires $O(n^3)$ operations. In this thesis we will propose methods to compute the same image with lower computational costs.

Several other compositing equations have been proposed for volume rendering. Upson & Keeler [1988] use the trapezoid rule and Novins & Arvo [1992] use higher-order quadrature rules to evaluate Equation 1.3 more accurately. There are also two common alternatives to volumetric compositing that model neither absorption nor scattering: X-ray projections and maximum intensity projections (MIP) [Laub & Kaiser 1988, Keller et al. 1989, Laub 1990]. In an X-ray projection the value of a pixel equals the integral of the emission function, and in a MIP rendering the value of a pixel equals the maximum emission of all samples along the ray.

X-ray projections can be computed rapidly using a theorem from Fourier analysis called the Fourier projection-slice theorem [Dunne et al. 1990, Malzbender 1993]. The advantage of the method is its low computational complexity: $O(n^2 \log n)$ operations to project an arbitrary n^3 voxel volume onto an arbitrary image plane. The greatest disadvantage is that the algorithm cannot be used to render images with occlusions as in conventional volume rendering since absorption is ignored, although algorithms to introduce alternative depth cues have been proposed [Levoy 1992, Totsuka & Levoy 1993]. The algorithms in this thesis can be adapted to use X-ray projections and MIP, but we will focus on methods for accelerating volume rendering algorithms based on the volumetric compositing equation.



(a) regular grid (b) curvilinear grid (c) unstructured grid

Figure 1.4: Examples of sampling grids in 2D: (a) regular grid, (b) curvilinear grid, (c) unstructured grid.

1.1.3 Data Representation and Sampling

In the previous discussion we have implicitly assumed that the physical parameters associated with a volume, e.g. the emission and absorption coefficients, are defined everywhere within the volume. In practice volume data is usually represented as an array of discrete samples, and the samples in the volume do not in general correspond to the sample points required to evaluate the volumetric compositing equation. Thus we must consider the representation of the volume and the method for determining values at arbitrary sample points within the volume.

There are several common sampling grids used for volume data (Figure 1.4). A *regular grid* consists of uniformly-spaced sample points located on a rectangular lattice. A *curvilinear grid* is a regular grid that has been warped with a non-linear transformation so that the sides of each cell need not be straight. Such grids are often used in computational fluid dynamics to conform the shape of the grid to the surface of an arbitrary object. An *unstructured grid* is an arbitrary collection of sample points with no implicit connectivity (although connectivity may be specified explicitly). Hybrid grids, in which several different grids are stitched together, are also possible.

In this thesis we restrict our attention to volume rendering algorithms for regular grids, which are the simplest grids and hence lead to the fastest algorithms.

Volumes sampled on other grid types can be resampled to a regular grid, although errors introduced during resampling may be a concern. Another problem is that multi-resolution hybrid grids used in computational fluid dynamics often have cells which range in size over many orders of magnitude. In this case a fixed-resolution grid with a sampling rate sufficiently high to accurately represent the highest-resolution regions of the volume would result in an impractically large resampled data set.

A possible solution to both of these problems is to choose a regular grid with the same resolution as the display device and to resample a subset of the data set that contains the user's current region of interest. If the user zooms in or out (requiring a change of resolution) or pans to a new region of interest then the appropriate portion of the original data set must be resampled to the regular grid. The loss of accuracy due to resampling should be no worse than the inherent limits of the display device. Volume rendering algorithms for curvilinear and unstructured grids exist but are significantly slower than algorithms for regular grids [Wilhelms et al. 1990, Garrity 1990] so the computational cost of on-the-fly resampling may not be prohibitive.

Given a volume defined on a regular grid we must now find a way to determine values at arbitrary sample points within the volume. Suppose the volume was originally produced by sampling a continuous function. A basic theorem from sampling theory states that if the sampling frequency is at least twice the maximum spatial frequency in the continuous function (called the Nyquist rate) then the continuous function can be reconstructed perfectly from the samples [Bracewell 1986]. If the sampling rate is too low then *aliasing* artifacts such as jagged edges appear in the reconstructed volume.

In practice two problems prevent perfect reconstruction. First, the continuous function must be *band limited*, meaning that it cannot contain arbitrarily-high frequencies. To guarantee this property the original function must be low-pass filtered *before* sampling, and we can only reconstruct the low-pass filtered version of the original function. Many sources of volume data, such as medical imaging scanners, produce a sample by averaging their input signal over a small region of space (thereby applying a low-pass filter). We will therefore assume that the samples in a volume represent a band-limited continuous function.

The second problem is that the ideal reconstruction procedure (using a sinc filter [Bracewell 1986]) is expensive. The sinc filter uses a weighted contribution of every input sample to compute one output value. Cheaper, less-accurate filters use a small number of samples in the vicinity of the output location to compute each output value. The number of samples required by the filter is called the *support* of the filter, and the location of the samples in the input array is called the *footprint* of the filter. The most common reconstruction filters used for volume rendering are the nearest-neighbor and trilinear interpolation filters, which produce acceptable results provided the frequencies in the band-limited signal are well below the Nyquist rate.

Reconstruction is one part of the procedure used to calculate samples along a viewing ray from samples in the original volume. The complete process is called *resampling*. We call the coordinate system of the volume samples *object space* and the coordinate system of the viewing ray samples *image space*. Resampling consists of four conceptual steps: (1) reconstruction of the continuous signal from the object-space samples, (2) warping of the continuous signal to image space, (3) band limiting the warped signal (only if the image-space sampling rate is less than the object-space sampling rate), and (4) sampling the band-limited signal to produce image-space samples. In real implementations the image-space samples are usually computed directly from the object-space samples without explicitly computing the intermediate signals. The four steps are combined by convolving the two filters into a single filter and then evaluating the combined filter at the image-space sample points.

Sampling issues will become important in Chapter 5 when we discuss image quality.

1.1.4 The Visualization Process

Since the goal of this thesis is to enable interactive volume rendering applications it is important to understand the steps involved in visualizing a data set. Figure 1.5 is a flow chart that shows the actions a user performs in a typical visualization session.

First the user acquires an array of input data. The acquisition process may include preparation steps such as resampling the volume to a regular grid, interpolating missing voxel values, and applying image processing operators to improve contrast.

Next the user *classifies* the volume. Classification is the process of assigning an opacity to each voxel (α in Equation 1.4). We will focus primarily on classification algorithms in which the user chooses a transfer function to compute the opacities from the scalar values. An alternative method is to partition the volume into specific structures (using a *segmentation* algorithm) and then to assign opacities to each structure.

Proper choice of the classification function is difficult. A good classification function reveals structures in the volume or highlights some subset of the volume in an informative

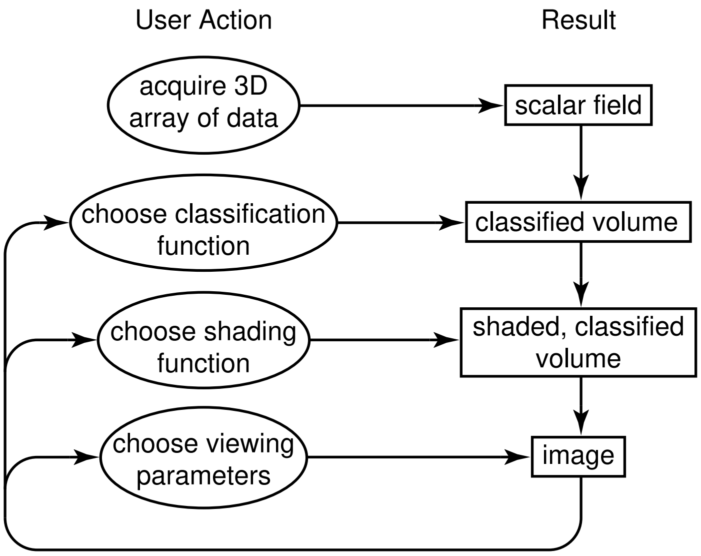


Figure 1.5: The visualization process is a feedback loop involving the user and three steps in the volume rendering algorithm: classification, shading, and rendering an image.

way. The appropriate choice depends both on the data itself and on the goals of the user. Unsuccessful classification can result in an image portraying a cloud-like fog which provides little information about the data. Experimentation with different classification functions is often necessary to produce an informative visualization, and interactive exploration can also lead to unexpected discoveries about the data.

After classifying the data the user chooses a shading function. The shading function specifies the illumination model and a rule for determining the color of each voxel. The rendering algorithm uses this information to compute C in Equation 1.4. Proper choice of the shading function is also a subjective process, depending partly on the information the user wishes to convey and partly on characteristics of the human visual system. Careful use of visual cues, such as specular highlights, depth cueing and shadows can greatly improve the effectiveness of a visualization.

Finally the user chooses viewing parameters: the viewpoint, the type of projection (parallel or perspective), clipping planes, and so on. With this information and the shaded, classified volume, the rendering algorithm produces an image.

At this point the user will almost certainly want to adjust the parameters to change or improve the image, as indicated by the feedback arcs in Figure 1.5. Because of the subjective choices involved in classification and shading it is difficult to decide on a set of parameters *a priori*, so the user must rely on trial and error to produce a good visualization. Furthermore, the user often does not know exactly what information exists in a new data set before exploring it with a visualization tool. The visualization application should support exploration of the parameter space in search of new insights about the data. User interaction with immediate visual feedback is essential for this process.

In the past, applications have used 2D slices or low-quality 3D renderings to give users visual feedback during selection of classification and shading parameters, coupled with a slow non-interactive volume renderer to produce final images. However, direct interaction with a 3D volume via a high-quality rendering provides better feedback and a simpler user interface. Our goal is to support interactive manipulation of all rendering parameters while immediately updating the rendered image. We wish to achieve this goal for high-resolution volumes using readily-available hardware. Given the current state of technology, we define our canonical problem as follows: we want to render a volume with 256^3 eight-bit voxels on a mid-range desktop workstation[2] in less than one second. Achieving this goal requires a volume renderer that is an order of magnitude faster than the best previous methods.

1.2 A New Family of Fast Volume Rendering Algorithms

Many acceleration techniques for volume rendering have been proposed. The most successful have used spatial data structures to reduce the number of voxels processed, thereby reducing the computational cost of volume rendering by an order of magnitude compared to brute-force methods [Levoy 1990a, Subramanian & Fussell 1990, Zuiderveld et al. 1992]. Such algorithms achieve rendering times of several seconds to several tens of seconds for our canonical problem on current workstations. While an improvement, these algorithms are still not fast enough for a fully interactive application. Multiprocessors can deliver real-time volume rendering, but only very large and expensive machines have the necessary processing power.

[2]Current mid-range machines have integer performance ratings of about 100 SPECint92 and floating point performance ratings of about 100 SPECfp92. The machine we use for our performance results is a 150 MHz SGI Indigo2.

The primary contribution of this thesis is a new family of fast volume rendering algorithms. The algorithms achieve sub-second rendering times on a desktop workstation without graphics accelerators, and can generate images at real-time frame rates on small- to medium-size general-purpose parallel processors. The algorithms achieve these speed improvements through high-level algorithmic optimizations that reduce computational costs. As in previous work we use spatial data structures to exploit coherence in the volume. However, the new algorithms use an efficient scanline-order traversal to stream through the data structures in storage order, thereby reducing overhead incurred by accessing the data structures. These scanline algorithms rely on a factorization of the viewing transformation into a 3D shear, a projection to form a 2D intermediate image, and a 2D warp. The factorization enables simultaneous scanline-order access to both the volume and the image.

We also introduce separate algorithms optimized for different stages in the rendering process: an algorithm for rendering while experimenting with classification functions, an algorithm for rendering while experimenting with shading functions and viewpoints, and variants of the algorithms for rendering parallel and perspective projections. The family of algorithms proposed in this thesis addresses each step of the visualization process and enables an interactive volume visualization system on general-purpose hardware.

1.3 Organization

Chapter 2 begins with a brief survey of previous volume rendering algorithms and acceleration techniques. We present a simple taxonomy of volume rendering algorithms and highlight the advantages and disadvantages of each class in the taxonomy.

Next, in Chapter 3 we describe the technique that forms the basis of our new algorithms: volume rendering using the shear-warp factorization. The factorization allows us to combine the advantages of the different classes of existing algorithms. We describe previous uses of the factorization for rendering parallel projections and present a new extension to perspective projections.

Chapter 4 introduces three new volume rendering algorithms based on the shear-warp factorization. The algorithms use the shear-warp factorization and spatial data structures to maximize efficiency with minimal loss of image quality.

Chapter 5 contains a performance analysis of the shear-warp algorithms, including a detailed comparison to brute-force algorithms and an octree-accelerated ray casting algorithm. This chapter uses an optimized ray caster as an example to show how the shear-warp algorithms eliminate overheads inherent in other volume rendering algorithms. Chapters 3-5 expand on material previously published in Lacroute & Levoy [1994].

Chapter 6 describes a parallel volume rendering algorithm for shared memory multiprocessors based on the new serial algorithms. We describe implementations for two multiprocessors, the SGI Challenge and the Stanford DASH Multiprocessor, and analyze the performance of the algorithms. This chapter appears in slightly-modified form in Lacroute [1995].

Chapter 7 describes a collection of extensions to the basic rendering algorithms. The extensions include a fast, flexible shading system based on lookup tables, a fast depth cueing algorithm, and a method for rendering shadows with the shear-warp algorithm. The chapter also sketches algorithms for rendering volumes with clipping planes and mixed data sets containing both volume data and polygons.

Chapter 8 gives an overview of the design of VolPack, a volume rendering software library based on the algorithms in this thesis. The chapter focuses on the tradeoff between flexibility and speed when designing an application programmer's interface for volume rendering.

Finally, Chapter 9 summarizes the conclusions of this work and discusses the implications for future research.