Hardware-Accelerated Volume Rendering

Hanspeter Pfister, Mitsubishi Electric Research Laboratories *

Abstract

In this chapter, we discuss methods for hardware-accelerated rendering of rectilinear volume datasets: ray casting, texture slicing, shear-warp and shear-image rendering, and splatting. We give sufficient background to enable a novice to understand the details and tradeoffs of each algorithm. As much as possible, we refer to the literature for more information.

1 Introduction

Over the last decade, volume rendering has become an invaluable visualization technique for a wide variety of applications in medicine, bio-technology, engineering, astrophysics, and other sciences. Examples include visualization of 3D sampled medical data (CT, MRI), seismic data from oil and gas exploration, or computed finite element models. While volume rendering is very popular, the lack of interactive frame rates has long limited its widespread use. Fortunately, advances in graphics hardware have lead to interactive and even real-time volume rendering performance, even on personal computers.

High frame rates are essential for the investigation and understanding of volume datasets. Real-time rotation of 3D objects in space under user control makes the renderings appear more realistic due to kinetic depth effects [Sollenberg and Milgram 1993]. Immediate visual feedback allows for interactive experimentation with different rendering parameters, such as transfer functions [Pfister et al. 2001]. Dynamically changing volume data can now be visualized, for example, data from interactive surgical simulation or real-time 3D ultrasound. And the image quality of hardware accelerated volume rendering rivals or equals that of the best software algorithms.

In this chapter, we review different hardware accelerated methods and architectures for volume rendering. Our discussion will focus on direct volume rendering techniques. A survey of iso-surface methods can be found in Chapter **X**. Direct volume rendering has the ability to give a qualitative feel for the density changes in the data. It precludes the need for segmenting the data [Levoy 1988] – indeed it is particularly adept at showing structurally weak and "fuzzy" information.

Section 2 reviews the basics of direct volume rendering. It provides background and terminology used throughout this chapter. The following sections then describe several approaches to volume rendering that have been successfully accelerated by hardware. We will discuss ray casting, 3D and 2D texture slicing, shear-warp rendering and its implementation on VolumePro, and splatting. We will focus on the underlying rendering algorithms and principles, and refer to the literature for implementation details. The chapter ends with conclusions and an outlook on future work in Section 9.

2 Volume Rendering Basics

2.1 Volume Data

A volumetric dataset consists of information sampled at discretized positions in space. The information may be a *scalar* (such as density in a computed tomography (CT) scan), or a *vector* (such as velocity in a flow field), or a higher order *tensor* (such as energy, density, and momentum in computational fluid dynamics). The space is usually three-dimensional, either consisting of three spatial dimensions or another combination of spatial and frequency dimensions.

In many applications the data is sampled on a *rectilinear grid*, represented as a 3D grid of volume elements, so called *voxels*. Voxels are assumed to be zero-dimensional scalar values defined at integer coordinates. This is in contrast to an alternative definition, where a voxel is interpreted as a small unit cube of volume with a constant value. There are many good arguments why such a definition may lead to errors and confusion [Smith 1995]. To describe voxels as volume points in 3D is consistent with signal processing theory [Möller et al. 1997b] and makes it easy to combine them with point-sampled surface models [Zwicker et al. 2002].

If all the voxels of a rectilinear dataset are spaced identically in each dimension, the dataset is said to be *regular*. Otherwise, the data is called *anisotropic*. Anisotropic volume data is commonly found in medical and geophysical applications. For example, the spacing of CT slices along the axis of the patient is determined by the (adjustable) speed of the table, while the spacing within a slice is determined by the geometry of the scanner. In addition, the gantry of a CT scanner may be tilted with respect to the axis of the patient. The resulting (rectilinear) data is called *sheared*, because the axes are not at right angles.

Other types of datasets can be classified into *curvilinear grids*, which can be thought of as resulting from a warping of a rectilinear grid, and *unstructured grids*, which consist of arbitrary shaped cells with no particular relation to rectilinear grids [Speary and Kennon 1990]. We restrict our discussion in this chapter to hardware-accelerated rendering of scalar voxels stored on a rectilinear volume grid, including anisotropic and sheared data.

2.2 Coordinate Systems and Transformations

Every volume rendering technique maps the data onto the image plane through a sequence of intermediate steps where the data is transformed to different coordinate systems. We introduce the basic terminology in Figure 1. Note that the terms *space* and *coordinate system* are synonymous. The volume data is stored in *source space*.

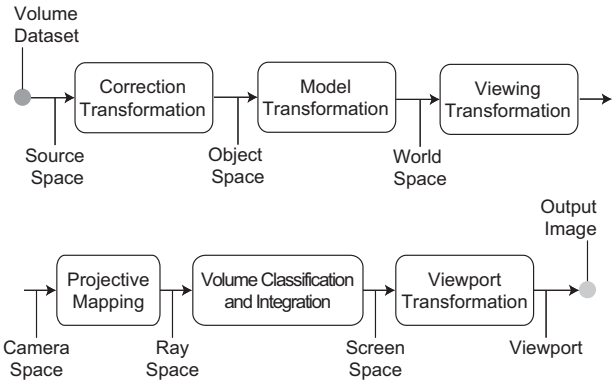

Figure 1: *The volume rendering pipeline.*

*201 Broadway, Cambridge MA. Email: pfister@merl.com

The correction transformation C transforms source space to *object space*, correcting for anisotropy and shear in the volume data. The model transformation M transforms object space to *world space*. This transformation allows to place multiple volume and polygon objects in the scene. To render the scene from an arbitrary viewpoint, the world space is mapped to *camera space* using the viewing transformation V. The camera coordinate system is defined such that its origin is at the center of projection.

The volume rendering algorithm projects the data and evaluates the volume rendering integral. The details of this integration will be discussed in Section 2.3. For now, we use the projection transformation P to transform the data to *ray space* (see Figure 2). Ray

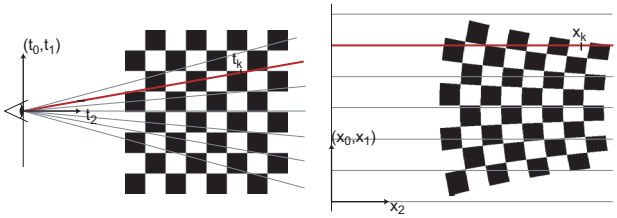

Figure 2: *Transforming the volume from camera to ray space. Left: camera space. Right: ray space.*

space is a non-cartesian coordinate system that enables an easy formulation of the volume rendering integral. In ray space, the viewing rays are parallel to a coordinate axis, facilitating analytical integration of the volume function. We denote a point in ray space by a vector $\mathbf{x} = (x_0, x_1, x_2)^T$, where the first two coordinates specify a point in screen space and can be abbreviated as $\hat{\mathbf{x}} = (x_0, x_1)^T$. The third coordinate x_2 specifies the Euclidean distance from the camera to a point on the viewing ray. Because the projection transformation P is similar to the projective transform used in rendering pipelines such as OpenGL, it is also called the projective mapping. For orthographic or parallel projection, P is the identity matrix.

Evaluating the volume rendering integral results in a 2D image in *screen space*. In the final step, this image is transformed to *viewport coordinates* using the viewport transformation VP. We will ignore the viewport transformation in the remainder of this chapter.

2.3 The Volume Rendering Integral

Volume rendering algorithms typically rely on the low-albedo approximation to how the volume data generates, scatters, or occludes light. If the albedo is low, most of the in-scattering of light comes from the direction of the viewing ray [Blinn 1982; Kajiya and Herzen 1984; Williams and Max 1992; Max 1995]. Effects of the light interaction are integrated along the viewing rays in ray space according to the *volume rendering integral*. The equation describes the light intensity $I_\lambda(\hat{\mathbf{x}})$ at wavelength λ that reaches the center of projection along the ray $\mathbf{x}$ with length L:

$$I_\lambda(\hat{\mathbf{x}}) = \int_0^L c_\lambda(\hat{\mathbf{x}},\xi) g(\hat{\mathbf{x}},\xi) e^{-\int_0^\xi g(\hat{\mathbf{x}},\mu)\,d\mu}\,d\xi, \tag{1}$$

where $g(\mathbf{x})$ is the *extinction function* that models the attenuation of light per unit length along the ray due to scattering or extinction. $c_\lambda(\mathbf{x})$ is an *emission coefficient*, modeling the light added per unit length along the ray, including self-emission, scattered, and reflected light.

The exponential term can be interpreted as an *attenuation factor* that models the absorption of light between a point along the ray and the eye. The product $c_\lambda(\mathbf{x})g(\mathbf{x})$ is also called the *source term* [Williams and Max 1992; Max 1995], describing the light intensity scattered in the direction of the ray $\mathbf{x}$ at the point x_2. In the remainder of this chapter we will omit the parameter λ, implying that (1) has to be evaluated for different wavelengths separately.

We now assume that the extinction function is given as a weighted sum of coefficients g_k and reconstruction kernels $r_k(\mathbf{x})$:

$$g(\mathbf{x}) = \sum_k g_k r_k(\mathbf{x}). \tag{2}$$

This corresponds to the *source-attenuation* physical model [Max 1995] where the volume consists of individual particles that absorb and emit light. The reconstruction kernels r_k reflect position and shape of individual particles. The particles can be irregularly spaced and may differ in shape, hence the model is not restricted to regular datasets.

Depending on how (1) is evaluated, volume rendering algorithms can be divided into *backward mapping* and *forward mapping* methods. Backward mapping algorithms shoot rays through pixels on the image plane into the volume data, and forward mapping algorithms map the data onto the image plane.

2.4 Backward Mapping

Backward mapping (or *image-order*) algorithms iterate over all pixels of the output image and determine the contributions of the integral to the current pixel [Levoy 1988; Sabella 1988; Danskin and Hanrahan 1992]. Ray casting is the most commonly used backward mapping technique. It simulates optical projections of light rays through the dataset, yielding a simple and visually accurate mechanism for volume rendering.

The integral (1) can be evaluated using a Riemann sum approximation. By approximating the exponential function with the first two terms of its Taylor expansion ($e^{-x} \approx 1 - x$), we arrive at this equation:

$$I(\hat{\mathbf{x}}) = \sum_{l=0}^{L} \left(c(\mathbf{x}_l) \sum_k g_k r_k(\mathbf{x}_l) \prod_{j=0}^{l-1} (1 - \sum_m g_m r_m(\mathbf{x}_j)) \right). \tag{3}$$

The inner summation $\sum_k g_k r_k(\mathbf{x}_l)$ computes the sum of volume reconstruction kernels using (2) at position $\mathbf{x}_l$ on the viewing ray. As described in Section 3.2, this is typically implemented by trilinear interpolation. The product over j is the attenuation due to of all sample points $\mathbf{x}_j$ that lie in front of the current position $\mathbf{x}_l$. The weighted sums of reconstruction kernels are typically replaced with the *opacity* α at the sample position. Thus we arrive at the familiar equation:

$$I(\hat{\mathbf{x}}) = \sum_{l=0}^{L} \left(c(\mathbf{x}_l) \alpha_l \prod_{j=0}^{l-1} (1 - \alpha_j) \right). \tag{4}$$

2.5 Forward Mapping

Forward mapping (or *object-order*) algorithms iterate over the volume data and determine the contribution of each reconstruction kernel to the screen pixels. The traditional forward mapping algorithm is splatting, introduced by Westover [Westover 1991]. It convolves every voxel in object space with the reconstruction kernel and accumulates their contributions on the image plane.

Because of the linearity of integration, substituting (2) into (1) yields:

$$I(\hat{\mathbf{x}}) = \sum_k g_k \left(\int_0^L c(\hat{\mathbf{x}},\xi) r_k(\hat{\mathbf{x}},\xi) \prod_j e^{-g_j \int_0^\xi r_j(\hat{\mathbf{x}},\mu)\,d\mu}\,d\xi \right), \tag{5}$$

which can be interpreted as a weighted sum of projected reconstruction kernels.

To compute this integral numerically, splatting algorithms make a couple of simplifying assumptions. Usually, the reconstruction

kernels $r_k(\mathbf{x})$ have local support. The splatting approach assumes that these local support areas do not overlap along a ray, and the reconstruction kernels are ordered front to back. We also assume that the emission coefficient is constant in the support of each reconstruction kernel along a ray, hence we have $c_k(\hat{\mathbf{x}}) = c(\mathbf{x})$. Again, we approximate the exponential function with the first two terms of its Taylor expansion, thus $e^{-x} \approx 1 - x$. Finally, we ignore self-attenuation. Under these assumptions, we can rewrite (5) to:

$$I(\mathbf{x}) = \sum_k \left(g_k c_k(\mathbf{x}) q_k(\hat{\mathbf{x}}) \prod_{j=0}^{k-1} \left(1 - g_j q_j(\hat{\mathbf{x}})\right) \right), \tag{6}$$

where $q_k(\hat{\mathbf{x}})$ denotes an integrated reconstruction kernel:

$$q_k(\hat{\mathbf{x}}) = \int_{\mathbb{R}} r_k(\hat{\mathbf{x}}, x_2)\, dx_2. \tag{7}$$

The difference between backward and forward mapping is apparent by comparing (3) and (6). In backward mapping, the evaluation of the volume rendering integral (1) is a Riemann sum along viewing rays. In forward mapping, we assume that the reconstruction kernels do not overlap and can be integrated separately using (7). The volume rendering integral (1) is then a sum of pre-integrated reconstruction kernels, also called *footprints*.

3 The Volume Rendering Pipeline

Volume rendering can be viewed as a set of pipelined processing steps. *Pipelining* is an important concept in hardware design and for the design of efficient parallel algorithms with local communication. A pipeline consists of a sequence of so called *stages* through which a computation and data flow. New data is input at the start of the pipeline while other data is being processed throughout the pipeline. In this section we look in more detail at the pipeline stages that are commonly found in volume rendering algorithms. The order in which these stages are arranged varies among implementations.

3.1 Data Traversal

A crucial step of the any volume rendering algorithm is to generate addresses of *resampling locations* throughout the volume. The resampling locations in object space are most likely not positioned on voxel locations, which requires *interpolation* from surrounding voxels to estimate sample values at non-integer positions.

3.2 Interpolation

Interpolation at a resampling location involves a convolution of neighboring voxel values with a reconstruction filter (see (2)). There is a wealth of literature that deals with the theory and application of appropriate reconstruction filters in computer graphics [Glassner 1995; Wolberg 1990] and volume visualization [Neumann 1993; Bentum 1995; Möller et al. 1997b]. In practice, due to the prohibitive computational cost of higher order filters, the most commonly used filters for ray-casting are *nearest neighbor interpolation* and linear interpolation in three dimensions, also called *tri-linear interpolation*. Note that tri-linear interpolation is a non-linear, cubic function in three-dimensions [Möller et al. 1997b]. This has consequences for the order of volume classification, as discussed below.

3.3 Gradient Estimation

To approximate the surface normals necessary for shading and classification requires the computation of a *gradient*. Given a continuous function $f(x,y,z)$, the gradient ∇f is defined as the partial derivative with respect to all three coordinate directions. Due to the sampled nature of volumetric data the computation of this continuous gradient has to be approximated using discrete *gradient filters*.

Most gradient filters are straight-forward 3D extensions of the corresponding two-dimensional edge detection filters, such as the Laplacian, Prewitt, or Zucker-Hummel [Zucker and Hummel 1981] operators. The Sobel operator [Sobel 1995] is one of the most widely used gradient filters for volume rendering. In practice, and due to computational considerations, most volume rendering algorithms use the *central-difference gradient*, which is computed by local differences between voxel or sample values in all three dimensions [Höhne and Bernstein 1986]. Detailed analysis of several gradient filters for volume rendering can be found in [Goss 1994; Bentum 1995; Möller et al. 1997a; Lichtenbelt et al. 1998].

3.4 Classification

Classification is the process of mapping physical properties of the volume, such as different material types, to the optical properties of the volume rendering integral, such as emission (color, RGB) and absorption (opacity, α). We distinguish between *pre- and post-classification*, depending if the voxel values of the volume are classified before or after interpolation.

Pre-Classification

In pre-classification, voxels may be mapped directly to RGBα values, which are then interpolated. Alternatively, voxels may be augmented by attributes that correspond to disjoint materials [Drebin et al. 1988], which is common for medical image data that contains separate anatomical parts. Typically, these attributes are computed in a separate *segmentation* process using accurate statistical methods [Wells et al. 1996; Dengler et al. 1995]. Such segmentation prevents *partial voluming* [Jacq and Roux 1997], one of the main sources of error in direct volume rendering. In partial voluming, single voxels ostensibly represent multiple materials, or tissue types in the case of medical data. Segmented volumes contain indices and associated probabilities for different material types, which can then be mapped to different colors and opacities during pre-classification [Tiede et al. 1998].

However, the individual interpolation of color and opacity after pre-classification can lead to image artifacts [Wittenbrink et al. 1998]. The solution is to pre-multiply the color of each voxel with its opacity before interpolation. The resulting vector (Rα, Gα, Bα, α) is called *associated color* or *opacity-weighted color* [Drebin et al. 1988; Blinn 1994]. If we denote original colors with C, we will use the notation $\tilde{C} = C\alpha$ for associated colors. Wittenbrink et al. [Wittenbrink et al. 1998] present an efficient method to interpolate associated colors. Interpolation with associated colors is also necessary for pre-integration techniques (see Section 4.3 and Chapter **X**).

Post-Classification

In post-classification, the mapping to RGBα values is applied to a continuous, interpolated scalar field. Post-classification is easier to implement than pre-classification and mostly used in the hardware-accelerated algorithms described below. Note that post-classification does not require the use of associated colors, although it is still possible to do so. In that case, the transfer functions (see below) are stored for associated colors.

As discussed by Engel et al. [Engel et al. 2001], pre- and post-classification produce different results because classification is in general a non-linear operation. The non-linearity of tri-linear interpolation may lead to artifacts if it is applied to pre-classified data. For post-classification, the evaluation of the non-linear classification function in a linearly interpolated scalar field produces the

correct result [Neumann 1993; Bentum 1995]. However, as noted above, pre-classification remains a very important tool in medical imaging, and it is important that the hardware-accelerated volume rendering method is able to support both options.

Transfer Functions

The mapping which assigns a value for optical properties like $g(\mathbf{x})$ or $c(\mathbf{x})$ is called a *transfer function*. The transfer function for $g(\mathbf{x})$ is called *opacity transfer function*, typically a continuously varying function of the scalar value s along the ray: $g(\mathbf{x}) = T_o(s(\mathbf{x}))$. Often it is useful to include the gradient magnitude $|\nabla s|$ as an additional parameter for classification. This approach has been widely used in the visualization of bone or other tissues in medical datasets or for the iso-surface visualization of electron density maps [Levoy 1988]. In the simplest case, the opacity is optionally multiplied with the gradient magnitude, which also called *gradient magnitude modulation*, to emphasize surface boundaries or to minimize the visual impact of noisy data [Gelder and Kim 1996; Pfister et al. 1999] (see Figure 3). Kindlmann et al. [Kindlmann and Durkin 1998] and

Figure 3: *CT scan of a human foot, rendered on VolumePro 1000 with gradient magnitude modulation of opacity. Image courtesy of Yin Wu and Jan Hardenbergh, TeraRecon Inc.*

Kniss et al. [Kniss et al. 2001] use higher order derivatives for semi-automatic transfer function design. Easy transfer function design still remains one of the main obstacles to make volume rendering more accessible to non-expert users [Pfister et al. 2001].

The emission term $c(\mathbf{x})$ can also be specified as a transfer function of the scalar s: $c(\mathbf{x}) = T_c(s(\mathbf{x}))$. The simplest emission term is direction independent, representing the glow of a hot gas [Max 1995]. It may have red, green, and blue components, with their associated *color transfer functions* $f_{red}(s)$, $f_{green}(s)$, and $f_{blue}(s)$. More sophisticated emission models include *multiple scattering* and *anisotropic scattering* terms [Max 1995; Harris and Lastra 2001] – mostly used for rendering of clouds – and *shading* effects.

3.5 Shading

Volume shading can substantially add to the realism and understanding of volume data (see Figure 4). Most volume shading

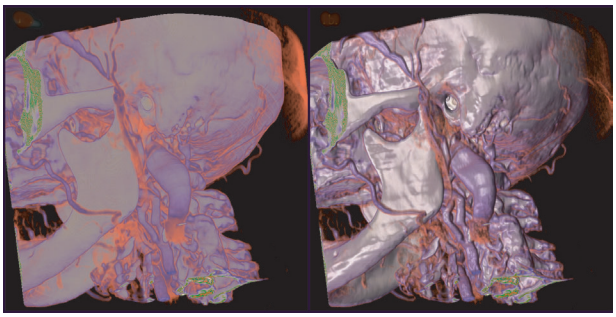

Figure 4: *CT scan of a human head. Left: volume rendering with a simple emission model without shading. Right: including Phong shading. Images rendered on VolumePro 1000; courtesy of Yin Wu and Jan Hardenbergh, TeraRecon Inc.*

is computed by the well-known Phong [Phong 1975] or Blinn-Phong [Blinn 1977] illumination models. The resulting color is a function of the gradient, light and view directions, ambient, diffuse, and specular shading parameters. It is typically added to the color that results from classification. Higher-order shading models, which include the physical effects of light-material interaction [Cook and Torrance 1982], are computationally too expensive to be considered for volume rendering. More illumination models for interactive systems can be found in [Akenine-Möller and Haines 2002].

Care has to be taken for sheared and anisotropic volume data. The different voxel spacing and alignment leads to incorrect gradients because the vector components are not orthonormal in object space. One remedy is to use full three-dimensional convolution kernels for gradient computation, which may be prohibitive in real-time systems. The alternative is to use separable kernels and to apply *gradient correction* before classification and lighting calculations. For example, gradients can be transformed from voxel space to world space, including the correction transformation C. Shading is then computed using world-space versions of the eye and light vectors [Pfister et al. 1999]. However, this transformation requires multiplication with a 3×3 matrix per gradient vector.

Alternatively, Wu et al. [Wu et al. 2003] describe an elegant and efficient solution for gradient correction using an intermediate lighting space. The product MC of model and correction transformation is decomposed into a shear-scale transformation L and a rotation R. Gradients are transformed to lighting space by $(L^{-1})^T$, similar to how surface normals are transformed in polygon graphics [Francis S. Hill 2000], while light and eye vectors are transformed from world space to lighting space by R^{-1}. Note that this rotation preserves dot products, which enables to pre-compute some shading calculations. The transformation $(L^{-1})^T$ is upper triangular and requires only six multiplications per gradient. For anisotropic but non-sheared volumes this reduces to three multiplications per gradient.

3.6 Compositing

Compositing is the recursive evaluation of the numerical integration in (4) and (6). It was first introduced in the context of digital image compositing, where it was formulated using the *"over" operator* [Porter and Duff 1984]. The composition of n associated

color samples $\tilde{C}_i = C_i \alpha_i$ is described by:

$$\begin{aligned} \tilde{C}(0,n-1) &= \sum_{x=0}^{n-1} \tilde{C}_x \prod_{t=0}^{x-1}(1-\alpha_t) \\ &= \tilde{C}_0 + \tilde{C}_1(1-\alpha_0) + \tilde{C}_2(1-\alpha_0)(1-\alpha_1) + \cdots \\ &\quad + \tilde{C}_{n-1}(1-\alpha_0)\cdots(1-\alpha_{n-2}) \\ &= \tilde{C}_0 \text{ over } \tilde{C}_1 \text{ over } \tilde{C}_2 \text{ over } \cdots \tilde{C}_{n-1}. \end{aligned} \tag{8}$$

Because of the associativity of the "over" operator, the composition of – for example – four samples $\tilde{C}_i$ can be computed in three different ways [Wittenbrink and Harrington 1994]:

Front-to-back:	$\tilde{C} = (((\tilde{C}_1 \text{ over } \tilde{C}_2) \text{ over } \tilde{C}_3) \text{ over } \tilde{C}_4)$
Back-to-front:	$\tilde{C} = (\tilde{C}_1 \text{ over } (\tilde{C}_2 \text{ ; over } (\tilde{C}_3 \text{ ; over } \tilde{C}_4)))$
Binary tree:	$\tilde{C} = ((\tilde{C}_1 \text{ ; over } \tilde{C}_2) \text{ ; over } (\tilde{C}_3 \text{ ; over } \tilde{C}_4))$

The *front-to-back* or *back-to-front* formulations are used in most volume rendering algorithms. The last formulation as a *binary tree* is especially useful for parallel implementations algorithms, where partial results of segments along the ray can be computed on different processors [Pfister et al. 1994; Hsu 1993; Wittenbrink and Somani 1993]. The final composition of the partial results yields the same image as sequential compositing along the ray.

Compositing is expressed algorithmically using recursion. The front-to-back formulation is:

$$\begin{aligned} \hat{t}_0 = (1-\alpha_0);\ \hat{C}_0 &= \tilde{C}_0 \\ \hat{C}_i &= \hat{C}_{i-1} + \hat{t}_{i-1}\,\tilde{C}_i \\ \hat{t}_i &= \hat{t}_{i-1}\,(1-\alpha_i), \end{aligned} \tag{9}$$

where $\hat{C}_i$, $\hat{t}_i$ indicate the results of the current iteration, $\hat{C}_{i-1}$, $\hat{t}_{i-1}$ the accumulated results of the previous iteration, and $\tilde{C}_i$ and α_i the associated sample color and opacity values at the current resampling location. Note that $\hat{t}$ is the accumulated *transparency*. Substituting $\hat{t}_i = (1-\hat{\alpha}_i)$ leads to the less efficient – but more familiar – formulation with accumulated opacities $\hat{\alpha}_i$.

When compositing back-to-front, the accumulated transparencies of Equation 9 do not need to be maintained. However, they are useful if the final image is composited over a new background or for mixing volumes and polygons (see Section 4.5). The recursive back-to-front formulation for n sample points is:

$$\begin{aligned} \hat{t}_n = (1-\alpha_n);\ \hat{C}_n &= \tilde{C}_n \\ \hat{C}_i &= \hat{C}_{i-1}(1-\alpha_i) + \tilde{C}_i \\ \hat{t}_i &= \hat{t}_{i-1}\,(1-\alpha_i). \end{aligned} \tag{10}$$

Because the extinction coefficient measures the volumetric light absorption per unit length, the opacity value must be adapted to the distance between interpolated samples. This scaling is called *opacity correction*. If opacities are defined for a distance d_{old}, and samples are spaced by a distance d_{new}, the scaling becomes [Lacroute 1995]:

$$\alpha_{corrected} = 1 - (1-\alpha_{stored})^{\frac{d_{old}}{d_{new}}}. \tag{11}$$

This can be efficiently implemented using a pre-computed lookup table that stores $\alpha_{corrected}$ as a function of α_{stored} and d_{new}.

As discussed by Schulze et al. [Schulze et al. 2003], associated colors have to be corrected correspondingly:

$$\tilde{C}_{corrected} = \tilde{C}_{stored}\frac{\alpha_{corrected}}{\alpha_{stored}}. \tag{12}$$

Orthographic projections typically lead to constant sample distances throughout the volume. Maintaining constant sample distance for perspective projections leads to spherical shells of samples around the from the center of projection (see Figure 2). While some approaches have used spherical shell sampling [LaMar et al. 1999] for uniform sample spacing, it is more common to correct opacities by evaluating (11) and (12).

There are several alternatives to volumetric compositing that have proven useful. In *X-ray* or *weighted sum* projections, the value of the pixel equals the sum of the intensities. *Maximum Intensity Projections* (MIP) project the maximum intensity along the ray into a pixel. Other options includes *first opaque projection*, *minimum intensity projection*, and *weighted sum projection* [Gasparakis 1999].

4 Advanced Techniques

Given the high performance requirements of volume rendering, it becomes clear that a brute-force implementation requires an excessive amount of processing. It is therefore not surprising that many optimizations have been developed.

4.1 Early Ray Termination

Early ray termination is a widely used method to speed up ray casting [Levoy 1990a; Arvo and Kirk 1990; Danskin and Hanrahan 1992]. The accumulation of new samples along a ray is terminated as soon as their contribution towards the currently computed pixel becomes minimal. Typically, the ray is terminated as soon as the accumulated ray opacity reaches a certain threshold since any further samples along the ray would be occluded. More general methods terminate rays according to a probability that increases with increasing accumulated ray opacity [Arvo and Kirk 1990; Danskin and Hanrahan 1992], or decrease the sampling rate as the optical distance to the viewer increases [Danskin and Hanrahan 1992]. The performance of early ray termination is dataset and classification dependent.

4.2 Space Leaping

Empty or transparent regions of the volume data may be skipped using pre-computed data structures. In *content-based space leaping*, samples are skipped that are invisible by virtue of opacity assignment or filtering. Typically, the opacity transfer function is used to encode non-transparent areas of the data into hierarchical [Meagher 1982; Levoy 1990a; Danskin and Hanrahan 1992; Subramanian and Fussell 1990] or run-length encoded [Reynolds et al. 1987; Lacroute and Levoy 1994] data structures. Inherent problems of content-based space leaping are that its performance is classification dependent, and that changes of the opacity transfer function lead to lengthy re-computation of the data structures.

In contrast, *geometry-based space leaping* skips empty space depending on the position of samples, not based on their values. Levoy [Levoy 1990a] uses an octree data structure to skip empty subvolumes of the data during ray-casting. Avila et al. [Avila et al. 1992] use convex polyhedral shapes as bounding volumes and graphics hardware to efficiently skip space by rendering the bounding volume into the depth buffer. Other methods use pre-computed distance functions to indicate the radial distance from each voxel in the data [Sramek 1994; Zuiderveld et al. 1992] or fast discrete line algorithms to progress quickly through empty regions [Yagel et al. 1992].

4.3 Pre-Integration

The discrete approximation of (1) will converge to the correct results only for high volume sampling rates, i.e., if the spacing between samples is sufficiently small. As discussed by Engel et al. [Engel et al. 2001], the Nyquist frequency for correct sampling is roughly the product of the Nyquist frequencies of the scalar field

and the maximum of the Nyquist frequencies of the two transfer functions for $g(\mathbf{x})$ and $c(\mathbf{x})$. In other words, non-linear transfer functions require very high sampling rates. Artifacts may still occur unless the transfer functions are smooth and the volume is band-limited.

To address the problem, Max et al. [Max et al. 1990] and Roettger et al. [Roettger et al. 2000] introduced a technique called *pre-integration* (see Chapter **X**). The volume rendering integral between two samples is a function of their interpolated values and the distance between the samples. For each combination of scalar values, the integration of the opacity or color transfer function is pre-computed and stored in a 2D lookup table (for constant sampling distance). The computation of the integral can be accelerated by graphics hardware [Roettger and Ertl 2002; Guthe et al. 2002a]. The value of the pre-integrated opacity or color values is looked up during post-classification. Meissner et al. [Meissner et al. 2002a] also apply pre-integration to the ambient, diffuse, and specular lighting parameters of the Phong shading model.

Despite pre-integration, artifacts may still occur from high frequencies that are present in the scalar field [Knittel 2002] and from the non-linear effects of tri-linear interpolation and shading. Consequently, the distance between samples needs to be decreased to *over-sample* the volume.

4.4 Volume Clipping

Volume clipping is an important operation that helps the understanding of 3D volumetric data. Clipping helps to uncover important details in the data by cutting away selected parts of the volume based on the position of *clip geometry*. The simplest clip geometry are one or more *clipping planes* that reveal slices and cross-sections of the volume data. *Cropping* is an easy way of specifying a rectilinear region of interest with multiple clipping planes parallel to the volume faces [Pfister et al. 1999]. During cropping, cross-sections of the volume may be combined by taking intersections, unions, and inverses to define elaborate regions of visibility of the volume data set. In its most general form, volume clipping uses arbitrary clip geometry that may be defined by a polygon mesh or by an additional volume (see Figure 9). Weiskopf et al. [Weiskopf et al. 2003] provide a good overview of volume clipping methods.

4.5 Mixing Polygons and Volumes

The incorporation of polygonally defined objects into a volumetric scene is often important, especially in medical applications such as virtual endoscopy [Geiger and Kikinis 1995]. Volume data – such as CT or MR images – can be directly combined with synthetic objects – such as surgical instruments, probes, catheters, prostheses, and landmarks displayed as glyphs. In some instances, preoperatively derived surface models for certain anatomical structures such as skin can be more efficiently stored and better visualized as a polygon mesh. A straightforward way of mixing volume and polygons is to convert the polygonal models into sampled volumes and then render them using a volume rendering method [Kaufman et al. 1990]. Another way is to simultaneously cast rays through both the polygonal and volume data, at the same sample intervals, and then composite the colors and opacities in depth sort order [Levoy 1990b]. Bhalerao at al. [Bhalerao et al. 2000] provide an overview of recent methods and propose a hardware-accelerated method for static views. All of the techniques described in this chapter are amenable to mixing volumes with polygons, although some with less efficiency than others.

We now will discuss several hardware-accelerated volume rendering algorithms for rectilinear grids: ray casting, texture slicing, shear-warp and shear-image rendering, and splatting. Our discussion will focus on how these methods implement each stage of the volume rendering pipeline (Section 3) followed by an overview of the available extensions (Section 4). Throughout the chapter, we will rarely quote performance numbers, unless we are confident they will not change over time. For an additional comparison between most of these algorithms, including image-quality evaluations, see [Meissner et al. 2000].

5 Ray Casting

Ray casting is the most commonly used image-order technique. It simulates optical projections of light rays through the dataset, yielding a simple and visually accurate mechanism for volume rendering [Levoy 1988].

Data Traversal: Rays are cast from the viewpoint (also called center of projection) through screen pixels into the volume. The ray directions can be computed from the model and viewing transformations using standard computer graphics techniques [Francis S. Hill 2000]. Empty space between the viewing plane and the volume can be skipped by rendering a polygonal bounding box of the volume into a depth buffer [Avila et al. 1992]. The ray starting points and their normalized directions are then used to generate evenly spaced resampling locations along each ray (see Figure 5).

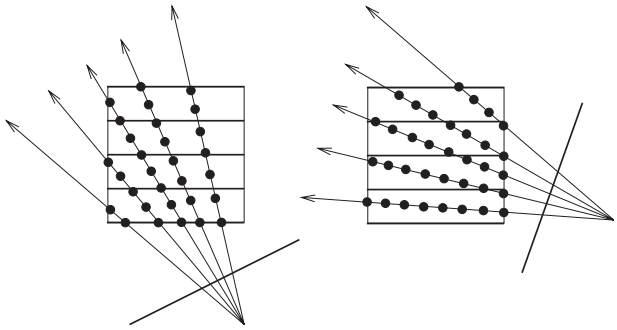

Figure 5: *Ray casting sample generation.*

Interpolation: At each resampling location, the data is interpolated, typically by tri-linear interpolation in *cells* of $2 \times 2 \times 2$ voxels. Note that tri-linear interpolation can be efficiently evaluated using caching of cell data among neighboring rays [Pfister et al. 1999].

Gradient Estimation: Gradient vectors are usually pre-computed at voxel locations and stored with the volume data. Alternatively, they are computed during rendering using the central difference gradient filter. The gradient vectors of a cell are interpolated to the nearest resampling locations. Optionally, the gradient magnitude is computed for gradient magnitude modulation during post-classification.

Classification: If scalar values were interpolated, color and opacity are assigned to each sample using a post-classification lookup. Optionally, the sample opacity is modulated by the gradient magnitude. If the data was pre-classified, no further classification of the interpolated associated colors is necessary.

Shading: Using the gradient as the surface normal approximation and the classified color as the emitted (or primary) color, a local shading model is applied to each sample. For example, the Phong illumination model for directional lights can be efficiently implemented using pre-computation and table lookup in so-called *reflectance maps* [Voorhies and Foran 1994]. The reflectance map implementation supports an unlimited number of directional light sources, but no positional lights.

Compositing: All samples along the ray are composited into pixel values – typically in front-to-back order – to produce the final image. For higher image quality, multiple rays per pixel are cast and combined using high-quality image filters [Francis S. Hill 2000].

Ray casting offers high image quality and is conceptually easy to implement. Unfortunately, these advantages come at the price of high computational requirements. The volume data is not accessed in storage order because of the arbitrary traversal direction of the viewing rays. This leads to poor spatial locality of data references, especially for sample and gradient interpolation.

There has been a lot of research on special-purpose hardware for volume ray casting. General surveys on early work in this field can be found in [Hesser et al. 1995; Ray et al. 1999]. The Cube project at SUNY Stony Brook resulted in VolumePro 500 (see Section 7.1), the first commercial real-time volume rendering engine, and various proposals for improvements [Kreeger and Kaufman 1998; Kreeger and Kaufman 1999a; Dachille and Kaufman 2000] The VIZARD project [Knittel and Strasser 1997; Meissner et al. 1998] at the University of Tübingen lead to the successful implementation of the VIZARD II hardware [Meissner et al. 2002b]. VIZARD II uses reconfigurable field-programmable gate arrays (FPGAs) for fast design changes and low cost development. The system can be configured for high quality perspective ray casting of volume data or for medical image reconstruction.

Recently, Roettger et al. [Roettger et al. 2003] presented the first implementation of volume ray casting on off-the-shelf graphics hardware (see Figure 6). All rays are processed in parallel in front-to-back order. The bounding box of the volume is rendered to provide starting locations for all rays. The parameters for ray traversal are stored in floating point textures, which are subsequently updated. The optimal step size for each ray is pre-computed and stored in a so-called importance volume. The step size depends on the pre-integrated emission and opacity value as well as second-order gradients. This technique, called *adaptive pre-integration*, is able to guarantee error bounds on the ray integration. Rays are terminated early or when they leave the volume by setting the z-buffer for the corresponding pixels such that further computation is avoided. Additional rendering passes are necessary to determine if all rays have terminated.

Figure 6: *Hardware-accelerated ray casting of a CT scan of a bonsai (*256^3*) with adaptive pre-integration. Image courtesy of Stefan Roettger, University of Stuttgart, Germany.*

The number of rendering passes for this approach is $2n-1$, where n is the maximum number of samples along a ray. The performance on modern graphics hardware is interactive, and it outperforms a software ray caster with pre-integration. The image quality is higher than comparable texture slicing methods (see Section 6), and additional performance increases can be expected with future hardware improvements.

6 Texture Slicing

Texture slicing on programmable graphics processing units (GPUs) [Lindholm et al. 2001] is the predominant hardware-accelerated volume rendering method. Texture-based volume rendering approaches can be implemented using 3D or 2D texture mapping functionality.

6.1 3D Texture Slicing

3D texture methods traverse the volume using *image-aligned texture slices* [Akeley 1993; Cullip and Neumann 1993; Wilson et al. 1994; Guan and Lipes 1994; Cabral et al. 1994] (see Figure 7).

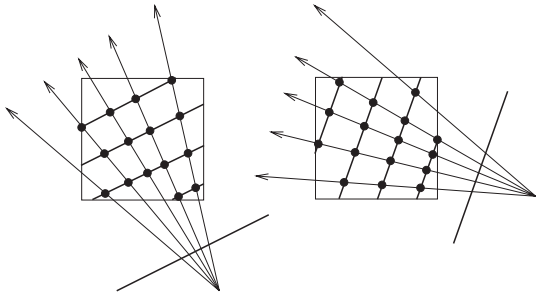

Figure 7: *Image-aligned 3D texture slicing.*

Data Traversal: The volume is stored in 3D texture memory and sampled during rendering by polygons parallel to the image plane. The view-aligned polygons are generated on the CPU and clipped against the volume bounding box. The clipping operation, which has to be performed for each frame, requires an efficient algorithm [Rezk-Salama 2001]. The texture coordinates of each polygon vertex with respect to the 3D texture parameters in the range $[0,1]$ are computed.

Interpolation: The polygons with associated 3D texture coordinates are projected by the graphics hardware using the standard transformations of the polygon graphics pipeline (see Section 2.2). The volume data is automatically resampled by the 3D texture hardware during polygon projection using tri-linear interpolation. Note that the sampling rate along viewing rays for orthographic projections is constant, whereas the sampling rate varies per ray for perspective projections [Rezk-Salama 2001]. This may lead to some artifacts, depending on the transfer functions and the data.

Gradient Estimation: Current GPUs do not support the computation of 3D gradients in hardware. Gradients are pre-computed, scaled and biased into the range $[0,1]$, and stored in the 3D texture. Typically, the RGB channel is used for the volume gradients, while the scalar values are stored in the alpha channel [Westermann and Ertl 1998]. Since gradients are projected and interpolated the same way as scalar values, subsequent shading operations are computed per pixel in screen space.

Gradients can also be stored using an integer encoding of the quantized gradient vector [Fletcher and Robertson 1993; Glassner 1990; Gelder and Kim 1996]. However, a non-linear shading function is very sensitive to quantization errors of the gradients. Another

alternative is to use a shading approximation without gradients by pairwise subtracting co-planar texture slices, one shifted in direction of the light source [Peercy et al. 1997].

Classification: Most texture mapping methods use post-classification (see Figure 8) by storing a one- or higher-dimensional transfer function as a texture [Meissner et al. 1999]. The interpolated scalar value is stored as a texture, which is then used as a lookup coordinate into the transfer function texture. This is also called *dependent texture lookup* because the texture coordinates for the second texture are obtained from the first texture If pre-

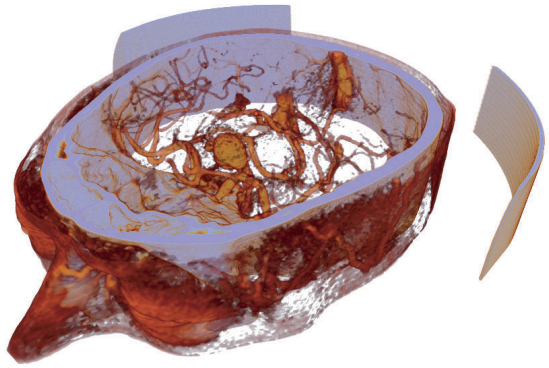

Figure 8: *CT Angiography of a human brain (*$512^2 \times 128$*). Rendered on an ATI Radeon 9700 with 3D texture slicing and post-classification using dependent textures. Image courtesy of Christof Rezk-Salama, University of Erlangen, Germany.*

classification is used, a 3D texture with associated colors is stored in addition to a 3D gradient texture. Alternatively, paletted textures can be used, where the texture format is defined by an index to a color palette that maps scalar values to colors [Berger et al. 2003]. Opacity correction is applied based on the distance between texture slices. It can be implemented using a dependent lookup into a one-dimensional floating point texture.

Shading: Before the introduction of programmable graphics hardware, shading of volumes stored as 3D textures was either ignored or performed in a pre-processing step [Gelder and Kim 1996]. However, pre-shaded volumes need to be recomputed whenever the light position and viewing direction change. A more fundamental problem is that classification and shading are in general non-linear operations. Interpolation of the pre-computed values degrades the image quality when compared to the evaluation of the non-linear functions in a linearly interpolated scalar field [Neumann 1993; Bentum 1995].

Dachille et al. [Dachille et al. 1998] use hardware for interpolation and compositing, and compute shading on the CPU during volume rendering. Westermann and Ertl [Westermann and Ertl 1998] introduced a hardware-accelerated ambient and diffuse shading technique for iso-surface rendering. Meissner at al. [Meissner et al. 1999] first expanded this technique for semi-transparent volume data, and then proposed an efficient technique to compute the full Phong illumination model using cube maps [Meissner et al. 2002a]. They also point out the need for normalized gradients in shading computations, and propose an efficient solution for pre-integrated classification. Engel et al. [Engel et al. 2001] compute diffuse and specular lighting for iso-surface rendering. They observe that memory requirements can be reduced for static lighting by storing the dot products of light and gradient vectors per voxel in luminance-alpha textures. Behrens et al. [Behrens and Ratering 1998] add shadows to the lighting model. Their multi-pass method works with 2D and 3D texture mapping hardware.

Compositing: The resampled RGBα textures are accumulated into the frame buffer using back-to-front compositing [Porter and Duff 1984]. If front-to-back compositing is used, accumulated opacities need to be stored for each pixel in the image.

Engel et al. [Engel et al. 2001] apply pre-integration of opacity and color transfer functions to texture slicing. Their method produces high quality images for semi-transparent and iso-surface volume rendering (see Chapter **X**). The pre-integration is applied to successive volume *slabs*, taking the scalar values at the entry and exit points and the distance between the two slices into account. Roettger et al. [Roettger et al. 2003] improve the image quality even further by internally blending the results of multiple pre-integrated slabs before writing the results to the framebuffer. They suggest that two-times over-sampling has almost the same performance as no over-sampling because of increased cache coherency. Using their multi-step slicing approach [Roettger et al. 2003], four-times over-sampling yields the best quality-performance tradeoff.

Weiskopf et al. [Weiskopf et al. 2002; Weiskopf et al. 2003] propose several techniques for volume clipping in 2D and 3D texture slicing methods. Arbitrary clip geometry can be defined by polygons, voxelized geometry, or the iso-surface of another volume. They also present a high-quality shading technique for clip boundaries (see Figure 9). Roettger et al. [Roettger et al. 2003] extend

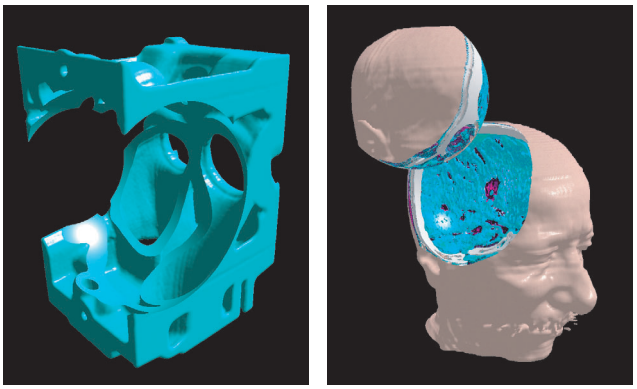

Figure 9: *Volume clipping applied to a CT scan of an engine (left,* $256^2 \times 110$*) and an MRI scan of a human head (right,* 256^3*). The cutting surfaces are enhanced by combining surface-based and volumetric shading. Image courtesy of Daniel Weiskopf, University of Stuttgart, Germany.*

some of their methods to work with pre-integrated classification.

Kreeger and Kaufman [Kreeger and Kaufman 1999b] developed a texture slicing method that renders opaque and translucent polygons embedded within volumes. Thin slabs of translucent polygons are rendered between volume slices and composited in the correct order.

A significant amount of texture memory is required to store the volume gradients. Typically, the storage increases by a factor of two to three. Visualization of very large volume data is also an issue for the limited memory of today's GPUs. Meissner et al. [Meissner et al. 2002a] use lossy texture compression to compress the volume with a corresponding loss in image quality. LaMar et al. [LaMar et al. 1999] propose a multi-resolution framework based on an octree, where each node is rendered using 3D texture slicing. Weiler et al. [Weiler et al. 2000] improve this algorithm to prevent discontinuity artifacts between different multi-resolution levels. Guthe et al. [Guthe et al. 2002b] use a hierarchical wavelet decomposition, on-the-fly decompression, and 3D texture slicing. Their implemen-

tation is able to render very large datasets at interactive rates on PCs, although with a loss in image quality.

6.2 2D Texture Methods

Historically, 3D texture mapping was not available on PC graphics cards, and 2D texture mapping methods had to be used instead [Cabral et al. 1994; Rezk-Salama et al. 2000]. For example,

Brady et al. [Brady et al. 1998] present a technique for interactive volume navigation that uses ray casting accelerated with 2D texture mapping. Despite the availability of 3D texture mapping on all modern GPUs, 2D texture slicing is still used for volume rendering today, and it outperforms 3D texture slicing for large volumes.

2D texture methods traverse the volume using *object-aligned* texture slices [Cullip and Neumann 1993; Guan and Lipes 1994; Cabral et al. 1994] (see Figure 10).

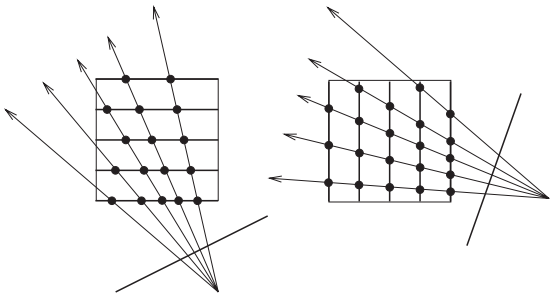

Figure 10: *Object-aligned 2D texture slicing.*

Data Traversal: Similar to 3D texture methods, each texture slice is defined by a polygon. In 2D texture slicing the polygons are always parallel to the face of the volume data that is most parallel to the viewing plane, which is also called the *base plane*. It can be easily determined by computing the minimum angle between viewing direction and face normals [Rezk-Salama 2001]. An arbitrary choice can be made in case of a tie at 45^o. Each polygon vertex is assigned the texture coordinates of the corresponding 2D volume slice in texture memory. In contrast to 3D texture methods, three copies of the volume have to be stored in texture memory, one for each slicing direction.

Interpolation: The texture mapped slices are interpolated by the graphics hardware during projection of the polygon to screen space. Object-aligned 2D texture slicing requires only bi-linear instead of tri-linear interpolation, which leads to higher performance due to the coherent memory accesses.

The lack of interpolation between slices may lead to aliasing artifacts if the scalar field or transfer functions contain high frequencies [Lacroute 1995]. Rezk-Salama et al. [Rezk-Salama et al. 2000] improve the image quality by interpolating additional slices during rendering using multiple texture units in one pass (see Figure 11). The tri-linear interpolation of inbetween slices is decomposed into two bi-linear interpolations (performed by 2D texture units in the graphics hardware) and one linear interpolation between slices (performed in the pixel shader of the GPU).

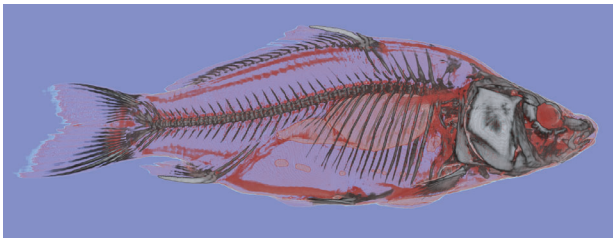

Figure 11: *CT scan of a carp (512^3) rendered on an ATI Radeon 9700 with 2D multi-textures and post-classification. Image courtesy of Christof Rezk-Salama, University of Erlangen, Germany.*

Gradient Estimation and Classification: Gradients and classification are computed similar as in 3D texture slicing. Precomputed gradients are stored in the RGB channel and bi-linearly interpolated to screen space during polygon rasterization. Classification can take place pre- or post-interpolation.

For opacity correction, Rezk-Salama et al. [Rezk-Salama et al. 2000] show that scaling the opacities linearly according to the distance between samples is a visually adequate approximation. They also describe an algorithm for fast shaded iso-surface display using multi-stage rasterization (see Figure 12). Engel et al. [Engel et al. 2001] improve the quality of 2D texture methods with pre-integrated classification.

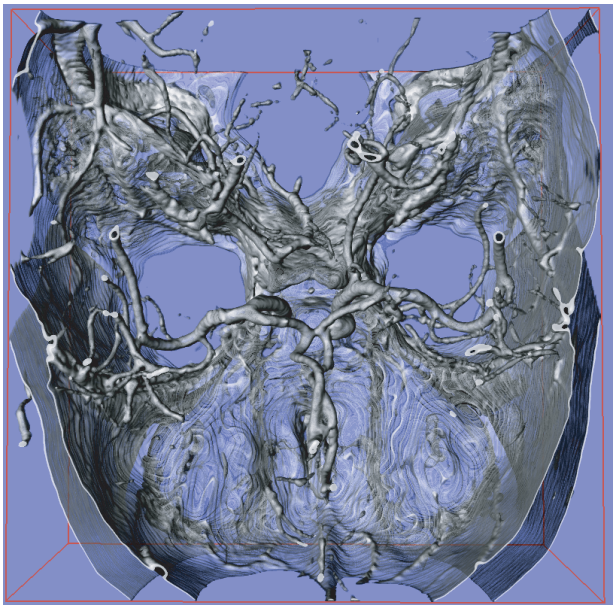

Figure 12: *CT Angiography of a human brain ($512^2 \times 128$). Transparent rendering of a non-polygonal shaded iso-surface with 2D multi-textures on an NVIDIA GeForce-4Ti. Image courtesy of Christof Rezk-Salama, University of Erlangen, Germany.*

Shading and Compositing: The same methods are used for shading and compositing as in 3D texture slicing. For high image quality, the gradients need to be normalized by the fragment shader after projection [Meissner et al. 2002a]. The texture-mapped slices are composited onto the image plane using the texture blending modes of the graphics hardware.

When the viewing direction suddenly changes from one slicing direction to another, the sampling through the volume changes as well. This may lead to *popping artifacts*, which are sudden changes in pixel intensity with changes in slicing direction (see Figure 13). The problem is worse for anisotropic volume data. Note that 3D texture slicing methods avoid popping artifacts by gradually adjusting the slice directions with the viewing angle. Rezk-Salama et al. [Rezk-Salama et al. 2000] virtually eliminate popping artifacts in 2D texture slicing by interpolating and shifting inbetween slices such that the sample spacing along viewing rays is practically constant independent of the viewing direction.

7 Shear-Warp Rendering

Shear-warp rendering algorithms resample the volume data from object space to the image coordinate space so that the resampled

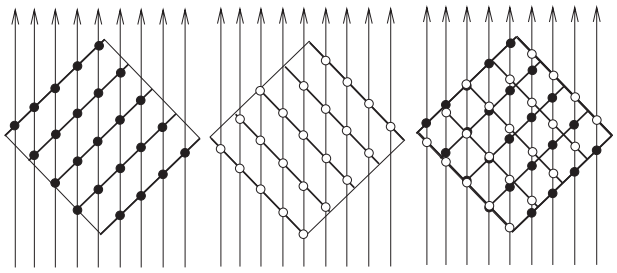

Figure 13: *Popping artifacts in 2D texture slicing. The samples along rays may not be aligned after a small change in viewing angle leads to a change of slicing direction. The superposition on the right shows that the location of resampling locations abruptly changes, which leads to sudden changes in pixel intensity. Figure suggested by Christof Rezk-Salama, University of Erlangen, Germany.*

voxels line up on the viewing axis in image space [Drebin et al. 1988; Upson and Keeler 1988] (see Figure 14). The interpolated

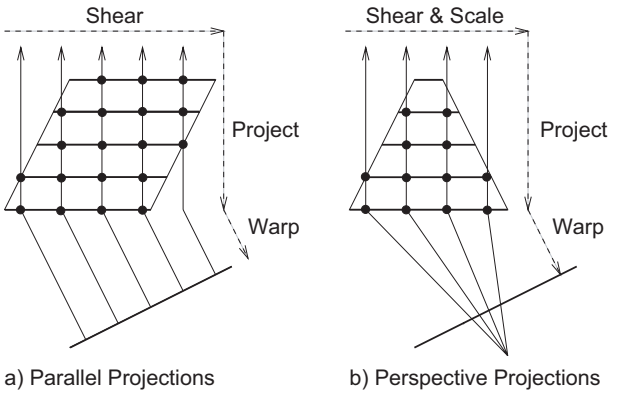

Figure 14: *Shear-warp factorization.*

samples are then composited onto the viewing plane along axis-aligned viewing rays. The 3D affine transformation between object space and image space can be decomposed into three sequential 1D shear operations [Hanrahan 1990]. Alternatively, the viewing transformation can be decomposed into a shear and a 2D image warping operation [Reynolds et al. 1987; Lacroute and Levoy 1994]. Perspective projections require an additional transformation, typically in the form of a scale operation of the sheared data slices [Vézina et al. 1992; Lacroute and Levoy 1994]. Shear-warp algorithms are very efficient due to the combination of object-order volume traversal and scanline-order resampling. More recently, they have been extended for improved image quality [Sweeney and Mueller 2002] and pre-integration [Schulze et al. 2003].

7.1 VolumePro 500

The VolumePro 500 system [Pfister et al. 1999] is based on the Cube-4 architecture developed at SUNY Stony Brook [Pfister and Kaufman 1996]. Mitsubishi Electric licensed the technology, improved it [Osborne et al. 1997], and started production shipments of the VolumePro 500 in 1999 [Pfister et al. 1999]. The technology was subsequently acquired by TeraRecon Inc., which released the VolumePro 1000 system in 2002 [Wu et al. 2003].

Data Traversal: VolumePro 500 uses the standard shear-warp factorization [Reynolds et al. 1987; Lacroute and Levoy 1994] for orthographic projections. Instead of casting rays from image space, rays are sent into the data set from pixels on the base plane. The ray traversal mechanism ensures that ray samples are aligned on slices parallel to the base plane [Yagel and Kaufman 1992]. A key feature of the VolumePro architecture is the special memory address arithmetic called 3D skewing [Kaufman and Bakalash 1988] and a highly optimized memory interface [Osborne et al. 1997]. This enables to efficiently read any blocks and axis-aligned voxel slices while storing only one copy of the volume data.

Interpolation: To prevent undersampling, VolumePro 500 uses tri-linear interpolation between volume slices. On-chip slice buffers and a the axis-aligned processing order allow maximum memory coherent accesses. The viewing rays can start at sub-pixel locations, which prevents popping artifacts during base plane switches and allows over-sampling of the volume in x, y, or z direction.

Gradient Estimation: VolumePro 500 has hardware for on-the-fly central-difference gradient estimation at each voxel. The gradients are then tri-linearly interpolated to resampling locations. The hardware includes gradient correction and gradient magnitude computation. The gradient magnitude is mapped by a lookup table to a user-specified piece-wise linear function. This function can be used to highlight particular gradient magnitude values or to attenuate the modulation effect. The lookup table is also used to automatically correct the gradient magnitudes in anisotropic volumes.

Classification: VolumePro 500 implements post-classification using a $4k \times 36$ bit classification lookup table that outputs 24-bit color and 12-bit α values. That precision is necessary for high accuracy during rendering of low opacity volumes. Because of the uniform sample spacing, opacity correction can be applied in software for each frame. The opacity and color lookup tables can be dynamically loaded using double buffering in hardware.

Shading: The hardware implements Phong shading at each sample point at the rate of one illuminated sample per clock cycle. The diffuse and specular illumination are looked up in reflectance maps, respectively [van Scheltinga et al. 1995; Voorhies and Foran 1994]. Each reflectance map is a pre-computed table that stores the amount of illumination due to the sum of all of the light sources of the scene. Reflectance maps need to be reloaded when the object and light positions change with respect to each other, or to correct the eye vector for anisotropic volumes (see Figure 15).

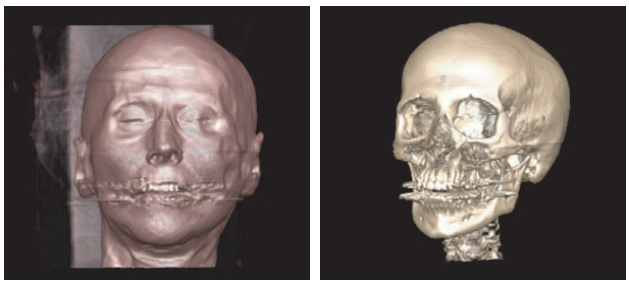

Figure 15: *CT scan of a human head (256^3) rendered on VolumePro 500 with Phong shading and different transfer functions.*

Compositing: The ray samples are accumulated into base plane pixels using front-to-back alpha blending or Minimum and Maximum Intensity Projections (MIP). The warping and display of the final image is performed by an off-the-shelf 3D graphics card using 2D texture mapping.

VolumePro 500 renders 256^3 or smaller volumes at 30 frames per second. Due to the brute-force processing, the performance is independent of the classification or data. In order to render a larger volume, the driver software first partitions the volume into smaller blocks. Each block is then rendered independently, and their resulting images are automatically combined to yield the final image. VolumePro 500 also provides various volume clipping and cropping features to visualize slices, cross-sections, or other regions-of-interest of the volume.

7.2 VolumePro 1000

The VolumePro 1000 system [Wu et al. 2003] uses a novel *shear-image order* ray casting approach (see Figure 16).

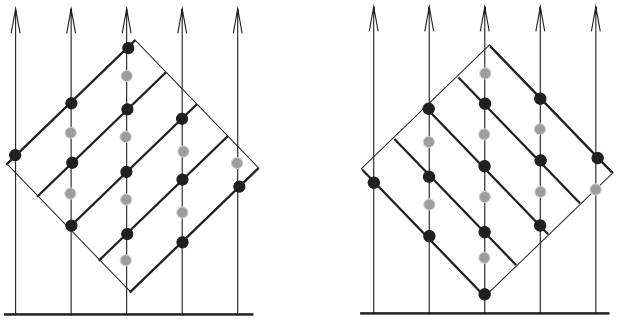

Figure 16: *Shear-image order ray casting. Grey samples are interpolated inbetween original volume slices.*

Data Traversal: Shear-image ray casting casts ray directly through the centers of pixels, but keeps the slices parallel to the base plane, similar to 2D texture mapping methods. However, the 3D viewing transformation is explicitly decomposed into two matrices: a transformation from voxel coordinates to an intermediate coordinate system called *sample space*, and a transformation to adjust the depth values of sample points to reflect their distance from the image plane. A detailed derivation of these transformations is given by Wu et al. [Wu et al. 2003].

Sample space is coherent with image and voxel space, and the final image does not have to be warped because samples are aligned along viewing rays from image plane pixels. This leads to higher image quality than the traditional shear-warp factorization (see Figure 17).

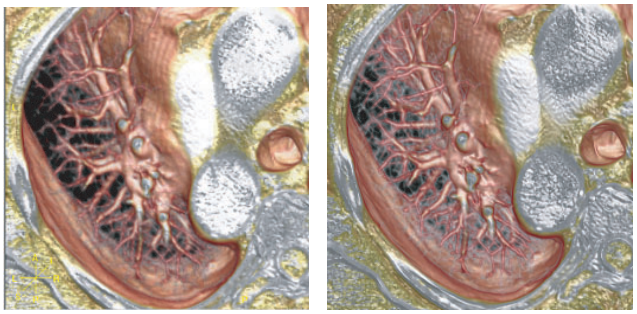

Figure 17: *Comparison of shear-warp (left, rendered by VolumePro 500) and shear-image order ray casting (right, by VolumePro 1000). Images courtesy of Yin Wu and Jan Hardenbergh, TeraRecon Inc.*

Interpolation and Gradient Estimation: Similar to VolumePro 500, the resampling of the volume proceeds in object-space for high memory coherence. VolumePro 1000 performs tri-linear interpolation of the volume data and computes gradient vectors in hardware. Similar to VolumePro 500 and the 2D texture slicing method of Rezk-Salama et al. [Rezk-Salama et al. 2000], additional interpolated slices can be generated inbetween original voxel slices. Since slices can be shifted with sub-pixel accuracy, this method avoids popping artifacts and keeps the ray spacing and sample spacing constant, also for anisotropic and sheared volumes.

Classification: VolumePro 1000 uses a set of cascaded lookup tables that can be combined by a hierarchy of arithmetic-logic units [Gasparakis 1999]. Voxels can have up to four fields, and each field is associated with its own lookup table. The classification and interpolation stage are cross-connected to allow the application to choose pre- or post-classification (see Figure 18). The hardware also supports opacity correction and gradient magnitude modulation of opacity.

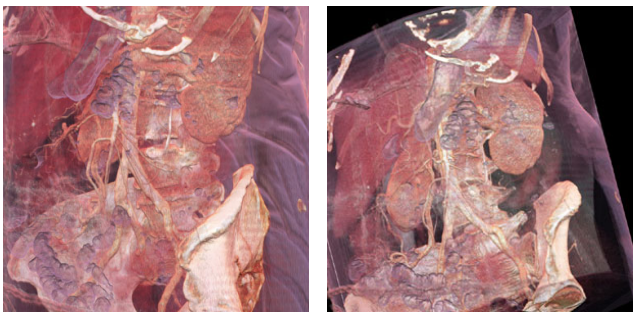

Figure 18: *CT scans of a human torso, pre-classified with different transfer functions per material type. Images courtesy of Yin Wu and Jan Hardenbergh, TeraRecon Inc.*

Shading: VolumePro 1000 uses similar Phong shading hardware as VolumePro 500. Great care is taken to ensure correct gradient and Phong shading calculations for sheared and anisotropic data using lighting space [Wu et al. 2003] (see Section 3.5).

Compositing: In addition to the blending modes of VolumePro 500, the hardware also supports early ray termination for increased performance. VolumePro 1000 also implements geometry-based space leaping, volume clipping and cropping, and perspective projections using a variation of the shear-warp transformation. VolumePro 1000 is capable of rendering 10^9 samples per second.

For embedding of polygons into the volume data, the depth of volume samples can be compared with a polygon depth-buffer (see Figure 19). The implementation uses multiple rendering passes:

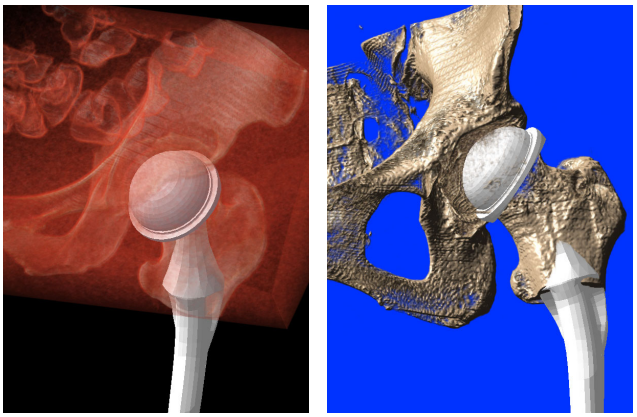

Figure 19: *Embedding a polygon prosthesis into a CT scan of a human hip. Images courtesy of Yin Wu and Jan Hardenbergh, TeraRecon Inc.*

first, the polygons are rendered into the depth buffer. Next, rays are cast into the volume starting at the image plane and ending at the captured depth buffer. The color buffers of the polygon and volume rendering are then blended. In the second pass, rays are initialized with the result of the blending pass. They start at the depth buffer and end at the background to render the portion of the volume behind the polygon. The result is an image of the volume with embedded polygons. VolumePro 1000 also supports embedding of

multiple translucent polygons using a dual depth buffers [Wu et al. 2003].

8 Splatting

Splatting, introduced by Westover [Westover 1991], convolves every voxel in object space with a 3D reconstruction filter and accumulates the voxels contribution on the image plane (see Chapter **X** and Figure 20).

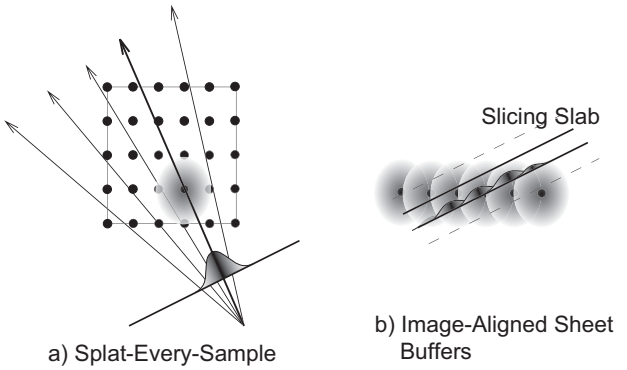

Figure 20: *Splatting algorithm. Left: 3D reconstruction kernels are integrated into 2D footprints, projected, and composited onto the image plane. Right: Image-aligned sheet buffers slice through the kernels. The contributions of 3D reconstruction kernels within a slab are added. The result of each slab is then composited onto the image plane.*

Data Traversal: Data traversal in splatting depends on the compositing method (see below). In its simplest form, voxels are traversed in object space and projected onto the screen (see Figure 20a). However, this leads to the wrong compositing order of the projected splats. Typically, traversal proceeds through the volume slice by slice, in approximate back-to-front order, similar to 2D texture slicing. For more advanced splatting methods, such as image-aligned sheet buffers, the traversal order is similar to 3D texture slicing (see Figure 20b).

Interpolation: Splatting is attractive because of its efficiency, which it derives from the use of pre-integrated reconstruction kernels. For simple splatting, the 3D kernel can be pre-integrated into a generic 2D footprint that is stored as a 2D texture.

Splatting also facilitates the use of higher quality kernels with a larger extent than trilinear kernels. Three-dimensional Gaussian reconstruction kernels are preferable because they are closed under convolution and integration [Zwicker et al. October 2001]. I.e., the convolution of two Gaussians is another Gaussian, and the integration of a 3D Gaussian is a 2D Gaussian.

Additional care has to be taken if the 3D reconstruction kernels are not radially symmetric, as is the case for sheared, anisotropic, curvilinear, or irregular grids. In addition, for an arbitrary position in 3D, the contributions from all kernels must sum up to one in the image. Zwicker et al. [Zwicker et al. 2002] discuss these issues in more detail and present a solution for Gaussian reconstruction kernels.

Gradient Estimation, Classification, and Shading: Typically, splatting uses pre-classification and pre-shading of the volume data. Each voxel stores the resulting RGBα values, which are then multiplied with the footprint before projection. Mueller et al. [Mueller et al. 1999a] propose a method for post-classification and shading in screen space. The gradients are either projected to screen space using so-called gradient splats, or they are computed in screen space using central differencing.

Compositing: Compositing is more complicated for splatting than for other volume rendering methods. While the principle is easy, it is more difficult to achieve high image quality.

The easiest compositing approach is called *splat-every-sample* (see Figure 20a). The 2D footprint of the kernel is multiplied by the scalar voxel value, projected to screen space, and blended onto the image plane using graphics hardware [Crawfis and Max 1992]. However, this leads to visible artifacts, such as color bleeding from background objects, because of incorrect visibility determination [Westover 1989].

To solve this problem, Westover [Westover 1990] introduces *sheet buffer splatting*. 2D footprints are added (not composited) onto sheet buffers that are parallel to the base plane. Traversal proceeds in back-to-front order, and subsequent sheet buffers are composited onto the image plane. The approach solves color bleeding, but similar to 2D texture slicing it introduces *popping artifacts* when the slice direction suddenly changes.

Mueller and Crawfis [Mueller and Crawfis October 1998] proposed to use *image-aligned sheet buffers* (see Figure 20b). A slab parallel to the image plane traverses the volume. The contributions of 3D reconstruction kernels between slab planes are added to the slab buffer, and the result is composited onto the image plane. This technique is similar to 3D texture slicing (see Section 6) and resolves the popping artifacts. But intersecting the slab with the 3D reconstruction kernels has a high computational cost.

Mueller and Yagel [Mueller and Yagel October 1996] combine splatting with ray casting techniques to accelerate rendering with perspective projection. Laur and Hanrahan [Laur and Hanrahan 1991] describe a hierarchical splatting algorithm enabling progressive refinement during rendering. Furthermore, Lippert [Lippert and Gross 1995] introduced a splatting algorithm that directly uses a wavelet representation of the volume data. For more extensions see Chapter **X**.

Westover's original framework does not deal with sampling rate changes due to perspective projections. Aliasing artifacts may occur in areas of the volume where the sampling rate of diverging rays falls below the volume grid sampling rate. The aliasing problem in volume splatting has first been addressed by Swan et al. [Swan et al. 1997] and Mueller et al. [Mueller et al. 1998]. They use a distance-dependent stretch of the footprints to make them act as low-pass filters.

Zwicker et al. [Zwicker et al. 2002] develop EWA splatting along similar lines to the work of Heckbert [Heckbert 1989], who introduced EWA filtering to avoid aliasing of surface textures. They extended his framework to represent and render texture functions on irregularly point-sampled surfaces [Zwicker et al. 2001], and to volume splatting [Zwicker et al. October 2001]. EWA splatting results in a single 2D Gaussian footprint in screen space that integrates an elliptical 3D Gaussian reconstruction kernel and a 2D Gaussian low-pass filter. This screen-space footprint is analytically defined and can efficiently be evaluated. By flattening the 3D Gaussian kernel along the volume gradient, EWA volume splats reduce to surface splats that are suitable for high quality iso-surface rendering.

Ren et al. [Ren et al. 2002] derive an object space formulation of the EWA surface splats and describe its efficient implementation on graphics hardware. For each point in object-space, quadrilaterals that are texture-mapped with a Gaussian texture are deformed to result in the correct screen-space EWA splat after projection. A similar idea can be applied to EWA volume splatting, as shown in Figure 21. The EWA volume splat is evaluated in screen space by deforming a texture-mapped screen-space quadrilateral. The projection of samples and the deformation of the screen-space quads can be performed efficiently on modern GPUs [Chen et al. 2004].

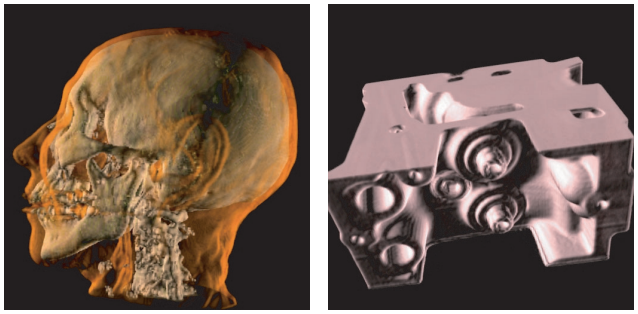

Figure 21: *CT scan of a human head (*$256^2 \times 225$*) and of an engine (*$256^2 \times 110$*) rendered with hardware-accelerated EWA volume splatting on a GeForce FX Ultra 5900. Image courtesy of Wei Chen, Zhejiang University, China, and Liu Ren, Carnegie Mellon University, USA.*

9 Conclusions

Without a doubt, the availability of programmable graphics hardware on PCs has changed the field of hardware-accelerated volume rendering. It is has lead to the great popularity of texture slicing methods. More recently, it has become feasible to implement ray casting on the GPU, including space leaping and early ray termination. The rapid progress of GPU hardware will address some remaining performance and image quality issues soon. The recent introduction of procedural shading languages [Mark et al. 2003] will increase productivity and portability of code across hardware from different vendors.

A more serious issue is the continuing growth of volume data compared to the limited memory on the GPU and its low download / upload bandwidth. The availability of increasingly powerful computers and high resolution scanners result in highly accurate and detailed data. For example, CT scanners now capture thousands of images with 512×512 resolution, supercomputers are producing terabytes of simulation data, and seismic scans for the oil and gas industry contain gigabytes or terabytes of data [Volz 2000]. All of the GPU accelerated algorithms presented in this chapter, such as texture slicing, multiply these memory requirements many fold by storing gradients and other auxiliary volumes. Interesting directions to solve these problems are multi-resolution techniques [LaMar et al. 1999; Guthe et al. 2002b], compression-domain volume rendering [Chiueh et al. 1994], and image-based volume rendering (IBVR) [Mueller et al. 1999b; Chen et al. 2001; Harris and Lastra 2001].

On the high end, VolumePro remains the only commercially available solution. In its current incarnation it can store up to 1 GB of volume data on board; that memory size will undoubtedly increase with new releases. The business challenge is to make this hardware widely available in dedicated, high-end visualization systems, such as PACS or geo-physical workstations, and 3D ultrasound systems. This challenge will increase with the continuing pressure from cheap, ever more powerful GPUs.

As hardware-accelerated techniques for rectilinear volumes mature, researchers focus their attention on the interactive or real-time rendering of unstructured volume data [Guthe et al. 2002a; Weiler et al. 2003a; Weiler et al. 2003b], time-varying data [Ma 2003], and non-traditional, illustrative volume rendering [Lu et al. 2003]. If the rapid pace of innovation continues, the chapter on hardware-accelerated volume rendering will have to be expanded in the very near future.

10 Acknowledgements

I would like to thank the many people who provided images for this chapter, namely (in alphabetical order) Wei Chen, Tom Ertl, Jan Hardenbergh, Liu Ren, Christof Rezk-Salama, Stefan Roettger, Manfred Weiler, Daniel Weiskopf, and Yin Wu. A big thank you goes to Matthias Zwicker and Christof Rezk-Salama for stimulating and enlightening discussions. I also would like to thank the editors – Chuck Hansen and Chris Johnson – and Piper West for the opportunity to contribute to this book and for their tremendous patience. Finally, I would like to thank Lilly Charlotte Pfister for coming into the world – I could not have wished for a more beautiful excuse to procrastinate this project.

References

AKELEY, K. 1993. RealityEngine graphics. In *Computer Graphics*, Proceedings of SIGGRAPH 93, 109–116.

AKENINE-MÖLLER, T., AND HAINES, E. 2002. *Real-Time Rendering*, 2 ed. A. K. Peters Ltd.

ARVO, J., AND KIRK, D. 1990. Particle transport and image synthesis. In *Computer Graphics*, Proceedings of SIGGRAPH 90, 63–66.

AVILA, R., SOBIERAJSKI, L., AND KAUFMAN, A. 1992. Towards a comprehensive volume visualization system. In *Proceedings of Visualization 92*, 13–20.

BEHRENS, U., AND RATERING, R. 1998. Adding shadows to a texture-based volume renderer. In *IEEE Symposium on Volume Visualization*, ACM Press, 39–46.

BENTUM, M. 1995. *Interactive Visualization of Volume Data*. PhD thesis, University of Twente, Enschede, The Netherlands.

BERGER, C., HADWIGER, M., AND HAUSER, H. 2003. A flexible framework for hardware-accelerated high-quality volume rendering. Tech. Rep. TR VRVIS 2003 001, VrVis, Austria.

BHALERAO, A., PFISTER, H., HALLE, M., AND KIKINIS, R. 2000. Fast re-rendering of volume and surface graphics by depth, color, and opacity buffering. *Journal of Medical Image Analysis 4*, 235–251.

BLINN, J. 1977. Models of light reflection for computer synthesized pictures. *Computer Graphics 11*, Annual Conference Series, 192–198.

BLINN, J. 1982. Light reflection functions for simulation of clouds and dusty surfaces. In *Computer Graphics*, vol. 16 of *SIGGRAPH 82 Proceedings*, 21–29.

BLINN, J. 1994. Compositing – theory. *Computer Graphics & Applications 14*, 5 (Sept.), 83–87.

BRADY, M., JUNG, K., NGUYEN, H. T., AND NGUYEN, T. P. 1998. Interactive volume navigation. *IEEE Transactions on Visualization and Computer Graphics 4*, 3 (July–September), 243–257.

CABRAL, B., CAM, N., AND FORAN, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *1994 Workshop on Volume Visualization*, 91–98.

CHEN, B., KAUFMAN, A., AND TANG, Q. 2001. Image-based rendering of surfaces from volume data. In *Volume Graphics 2001*, K. Mueller and A. Kaufman, Eds., 279–295.

CHEN, W., REN, L., ZWICKER, M., AND PFISTER, H. 2004. Hardware-accelerated adaptive ewa volume splatting. In *Proceedings of IEEE Visualization 2004*.

CHIUEH, T., HE, T., KAUFMAN, A., AND PFISTER, H. 1994. Compression domain volume rendering. Tech. Rep. TR.94.01.04R, State University of New York at Stony Brook, Computer Science Department, Stony Brook, NY 11794-4400, Apr.

COOK, R. L., AND TORRANCE, K. E. 1982. A reflectance model for computer graphics. *ACM Transactions on Graphics 1*, 1 (Jan.), 7–24.

CRAWFIS, R., AND MAX, N. 1992. Direct volume visualization of three-dimensional vector fields. *Workshop on Volume Visualization* (Oct.), 55–50.

CULLIP, T. J., AND NEUMANN, U. 1993. Accelerating volume reconstruction with 3D texture mapping hardware. Tech. Rep. TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill.

DACHILLE, F., AND KAUFMAN, A. 2000. Gi-cube: An architecture for volumetric global illumination and rendering. In *SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, 119–128.

DACHILLE, F., KREEGER, K., CHEN, B., BITTER, I., AND KAUFMAN, A. 1998. High-quality volume rendering using texture mapping hardware. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 69–76.

DANSKIN, J., AND HANRAHAN, P. 1992. Fast algorithms for volume ray tracing. In *Workshop on Volume Visualization*, A. Kaufman and W. L. Lorensen, Eds., 91–98.

DENGLER, J., WARFIELD, S., ZAERS, J., GUTTMANN, C., WELLS, W., ETTINGER, G., HILLER, J., AND KIKINIS, R. 1995. Automatic identification of grey matter structures from MRI to improve the segmentation of white matter lesions. In *Proceedings of Medical Robotics and Computer Assisted Surgery*, 140–147.

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. Volume rendering. *Computer Graphics 22*, 4 (Aug.), 65–74.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/Eurographics Workshop on Graphics hardware*, ACM Press, 9–16.

FLETCHER, P. A., AND ROBERTSON, P. K. 1993. Interactive shading for surface and volume visualization on graphics workstations. In *Proceedings of Visualization 93*, IEEE Computer Society Press, San Jose, CA, 291–298.

FRANCIS S. HILL, J. 2000. *Computer Graphics Using OpenGL*, 2 ed. Prentice Hall. ISBN: 0023548568.

GASPARAKIS, C. 1999. Multi-resolution multi-field ray tracing: A mathematical overview. In *Proceedings of IEEE Visualization 99*, IEEE Computer Society Press, San Francisco, CA, 199–206.

GEIGER, B., AND KIKINIS, R. 1995. Simulation of endoscopy. In *Comp. Vision Virtual Reality and Robotics in Medicine*, 227–281.

GELDER, A. V., AND KIM, K. 1996. Direct volume rendering with shading via three-dimensional textures. In *ACM/IEEE Symposium on Volume Visualization*, 23–30.

GLASSNER, A. S., Ed. 1990. *Graphics Gems V*. Academic Press, Inc., New York, ch. Normal Coding, 257–264.

GLASSNER, A. S. 1995. *Principles of Digital Image Synthesis*, first ed., vol. one and two of *The Morgan Kaufman Series in Computer Graphics and Geometric Modeling*. Morgan Kaufman Publishers, Inc.

GOSS, M. E. 1994. An adjustable gradient filter for volume visualization image enhancement. In *Graphics Interface '94*, 67–74.

GUAN, S., AND LIPES, R. 1994. Innovative volume rendering using 3d texture mapping. In *Image Capture, Formatting, and Display*, SPIE 2164.

GUTHE, S., ROETTGER, S., SCHIEBER, A., STRASSER, W., AND ERTL, T. 2002. High-quality unstructured volume rendering on the pc platform. In *Eurographics/SIGGRAPH Graphics Hardware Workshop*, 119–125.

GUTHE, S., WAND, M., GONSER, J., AND STRASSER, W. 2002. Interactive rendering of large volume data sets. In *Proceedings of IEEE Visualization '02*, IEEE Press, 53–60.

HANRAHAN, P. 1990. Three-pass affine transforms for volume rendering. *Computer Graphics 24*, 5 (Nov.), 71–78.

HARRIS, M. J., AND LASTRA, A. 2001. Real-time cloud rendering. In *Computer Graphics Forum (Eurographics 2001 Proceedings)*, vol. 20.

HECKBERT, P. 1989. *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, University of California at Berkeley, Department of Electrical Engineering and Computer Science.

HESSER, J., MÄNNER, R., KNITTEL, G., STRASSER, W., PFISTER, H., AND KAUFMAN, A. 1995. Three architectures for volume rendering. In *Proceedings of Eurographics '95*, European Computer Graphics Association, C–111–C–122.

HÖHNE, K. H., AND BERNSTEIN, R. 1986. Shading 3D-images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging MI-5*, 1 (Mar.), 45–47.

HSU, W. M. 1993. Segmented ray-casting for data parallel volume rendering. In *Proceedings of the 1993 Parallel Rendering Symposium*, 7–14.

JACQ, J., AND ROUX, C. 1997. A direct multi-volume rendering method aiming at comparisons of 3-D images and models. *IEEE Trans. on Information Technology in Biomedicine 1*, 1, 30–43.

KAJIYA, J. T., AND HERZEN, B. P. V. 1984. Ray tracing volume densities. In *Computer Graphics*, H. Christiansen, Ed., vol. 18 of *SIGGRAPH '84 Proceedings*, 165–174.

KAUFMAN, A., AND BAKALASH, R. 1988. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics & Applications 8*, 6 (Nov.), 10–23.

KAUFMAN, A., YAGEL, R., AND COHEN, D. 1990. Intermixing surface and volume rendering. In *3D Imaging in Medicine: Algorithms, Systems, Applications*, K. H. Höhne, H. Fuchs, and S. M. Pizer, Eds. June, 217–227.

KINDLMANN, G., AND DURKIN, J. 1998. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings 1998 IEEE Symposium on Volume Visualization*, 79–86.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization*, 255–262.

KNITTEL, G., AND STRASSER, W. 1997. Vizard – visualization accelerator for real-time display. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, 139–146.

KNITTEL, G. 2002. Using pre-integrated transfer functions in an interactive software system for volume rendering. In *Eurographics 2002 Short Presentations*, 119–123.

KREEGER, K., AND KAUFMAN, A. 1998. Pavlov: A programmable architecture for volume processing. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 77–85.

KREEGER, K., AND KAUFMAN, A. 1999. Hybrid volume and polygon rendering with cube hardware. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 15–24.

KREEGER, K., AND KAUFMAN, A. 1999. Mixing translucent polygons with volumes. In *Proceedings of IEEE Visualization 99*, IEEE Computer Society Press, 191–198.

LACROUTE, P., AND LEVOY, M. 1994. Fast volume rendering using a shear-warp factorization of the viewing transform. In *Computer Graphics*, Proceedings of SIGGRAPH 94, 451–457.

LACROUTE, P. 1995. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform*. PhD thesis, Stanford University, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford, CA 94305–4055. CSL-TR-95-678.

LAMAR, E., HAMANN, B., AND JOY, K. 1999. Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of the 1999 IEEE Visualization Conference*, 355–362.

LAUR, D., AND HANRAHAN, P. 1991. Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Computer Graphics*, SIGGRAPH '91 Proceedings, 285–288.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics & Applications 8*, 5 (May), 29–37.

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

LEVOY, M. 1990. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics & Applications 10*, 2 (Mar.), 33–40.

LICHTENBELT, B., CRANE, R., AND NAQVI, S. 1998. *Introduction to Volume Rendering*. Hewlett-Packard Professional Books. Prentice Hall, Los Angeles, USA.

LINDHOLM, E., KILGARD, M., AND MORETON, H. 2001. A user-programmable vertex engine. In *Computer Graphics*, SIGGRAPH 2001 Proceedings, 149–158.

LIPPERT, L., AND GROSS, M. 1995. Fast wavelet based volume rendering by accumulation of transparent texture maps. In *Computer Graphics Forum*, Proceedings of Eurographics 95, C–431 – C–443.

LU, A., MORRIS, C., TAYLOR, J., EBERT, D., HANSEN, C., RHEINGANS, P., AND HARTNER, M. 2003. Illustrative interactive stipple rendering. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (April–June), 127–138.

MA, K.-L. 2003. Visualizing time-varying volume data. *IEEE Transactions on Visualization and Computer Graphics 5*, 2 (March/April), 34–42.

MARK, B., GLANVILLE, S., AKELEY, K., AND KILGARD, M. 2003. Cg: A system for programming graphics hardware in a c-like language. vol. 22, 896–908.

MAX, N., HANRAHAN, P., AND CRAWFIS, R. 1990. Area and volume coherence for efficient visualization of 3D scalar functions. *Computer Graphics 24*, 5 (Nov.), 27–34.

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

MEAGHER, D. 1982. Efficient synthetic image generation of arbitrary 3D objects. In *Proceedings of IEEE Computer Society Conference on Pattern Recognition and Image Processing*.

MEISSNER, M., KANUS, U., AND STRASSER, W. 1998. Vizard ii, a pci card for real-time volume rendering. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware*, 61–68.

MEISSNER, M., HOFFMANN, U., AND STRASSER, W. 1999. Enabling classification and shading for 3d texture mapping based volume rendering using opengl and extensions. In *Proceedings of the 1999 IEEE Visualization Conference*, 207–214.

MEISSNER, M., HUANG, J., BARTZ, D., MUELLER, K., AND CRAWFIS, R. 2000. A practical evaluation of popular volume rendering algorithms. In *IEEE Symposium on Volume Visualization*, 81–90.

MEISSNER, M., GUTHE, S., AND STRASSER, W. 2002. Interactive lighting models and pre-integration for volume rendering on pc graphics accelerators. In *Proceedings of Graphics Interface 2002*, 209–218.

MEISSNER, M., KANUS, U., WETEKAM, G., HIRCHE, J., EHLERT, A., STRASSER, W., DOGGETT, M., AND PROKSA, R. 2002. A reconfigurable interactive volume rendering system. In *Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware*.

MÖLLER, T., MACHIRAJU, R., MUELLER, K., AND YAGEL, R. 1997. A comparison of normal estimation schemes. In *Proceedings of IEEE Visualization '97*, 19–26.

MÖLLER, T., MACHIRAJU, R., MUELLER, K., AND YAGEL, R. 1997. Evaluation and design of filters using a taylor series expansion. *IEEE Transactions on Visualization and Computer Graphics 3*, 2, 184–199.

MUELLER, K., AND CRAWFIS, R. October 1998. Eliminating popping artifacts in sheet buffer-based splatting. *IEEE Visualization '98*, 239–246.

MUELLER, K., AND YAGEL, R. October 1996. Fast perspective volume rendering with splatting by utilizing a ray-driven approach. *IEEE Visualization '96*, 65–72.

MUELLER, K., MOELLER, T., SWAN, J., CRAWFIS, R., SHAREEF, N., AND YAGEL, R. 1998. Splatting errors and antialiasing. *IEEE Transactions on Visualization and Computer Graphics 4*, 2 (April-June), 178–191.

MUELLER, K., MOELLER, T., AND CRAWFIS, R. 1999. Splatting without the blur. In *Proceedings of the 1999 IEEE Visualization Conference*, 363–370.

MUELLER, K., SHAREEF, N., HUANG, J., AND CRAWFIS, R. 1999. Ibr-assisted volume rendering. In *Proceedings of IEEE Visualization Late Breaking Hot Topics*, 5–8.

NEUMANN, U. 1993. *Volume Reconstruction and Parallel Rendering Algorithms: A Comparative Analysis*. PhD thesis, Computer Science Department, University of North Carolina at Chapel Hill.

OSBORNE, R., PFISTER, H., LAUER, H., MCKENZIE, N., GIBSON, S., HIATT, W., AND OHKAMI, T. 1997. EM-Cube: An architecture for low-cost real-time volume rendering. In *Proceedings of the Siggraph/Eurographics Workshop on Graphics Hardware*, 131–138.

PEERCY, M., AIREY, J., AND CABRAL, B. 1997. Efficient bump mapping hardware. In *Computer Graphics*, Proceedings of SIGGRAPH '97, 303–306.

PFISTER, H., AND KAUFMAN, A. 1996. Cube-4 – A scalable architecture for real-time volume rendering. In *1996 ACM/IEEE Symposium on Volume Visualization*, 47–54.

PFISTER, H., KAUFMAN, A., AND CHIUEH, T. 1994. Cube-3: A real-time architecture for high-resolution volume visualization. In *1994 ACM/IEEE Symposium on Volume Visualization*, 75–83.

PFISTER, H., HARDENBERGH, J., KNITTEL, J., LAUER, H., AND SEILER, L. 1999. The volumepro real-time ray-casting system. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 99)*, ACM Press/Addison-Wesley Publishing Co., 251–260.

PFISTER, H., LORENSEN, W., BAJAJ, C., KINDLMANN, G., SCHROEDER, W., AVILA, L. S., MARTIN, K., MACHIRAJU, R., AND LEE, J. 2001. The transfer function bake-off. *IEEE Computer Graphics and Applications* (May / June), 16–22.

PHONG, B. T. 1975. Illumination for computer generated pictures. *Communications of the ACM 1*, 18 (June), 311–317.

PORTER, T., AND DUFF, T. 1984. Compositing digital images. *Computer Graphics 18*, 3 (July).

RAY, H., PFISTER, H., SILVER, D., AND COOK, T. A. 1999. Ray-casting architectures for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 5*, 3 (July/September), 210–223.

REN, L., PFISTER, H., AND ZWICKER, M. 2002. Object-space ewa surface splatting: A hardware accelerated approach to high quality point rendering. In *Computer Graphics Forum*, vol. 21, 461–470. Proceedings of Eurographics 2002.

REYNOLDS, R. A., GORDON, D., AND CHEN, L.-S. 1987. A dynamic screen technique for shaded graphics display of slice-represented objects. *Computer Vision, Graphics, and Image Processing 38*, 3, 275–298.

REZK-SALAMA, C., ENGEL, K., BAUER, M., GREINER, G., AND ERTL, T. 2000. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 109–118.

REZK-SALAMA, C. 2001. *Volume Rendering Techniques for General Purpose Graphics Hardware*. PhD thesis, University of Erlangen, Germany.

ROETTGER, S., AND ERTL, T. 2002. A two-step approach for interactive pre-integrated volume rendering of unstructured grids. In *IEEE Symposium on Volume Visualization*, ACM Press, Boston, MA, 23–28.

ROETTGER, S., KRAUS, M., AND ERTL, T. 2000. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of IEEE Visualization*, 109–116.

ROETTGER, S., GUTHE, S., WEISKOPF, D., ERTL, T., AND STRASSER, W. 2003. Smart hardware-accelerated volume rendering. In *Eurographics/IEEE TCVG Symposium on Visualization 2003*, Eurographics.

SABELLA, P. 1988. A rendering algorithm for visualizing 3D scalar fields. *Computer Graphics 22*, 4 (Aug.), 59–64.

SCHULZE, J., KRAUS, M., LANG, U., AND ERTL, T. 2003. Integrating pre-integration into the shear-warp algorithm. In *Proceedings of the Third International Workshop on Volume Graphics*, 109–118.

SMITH, A. R. 1995. A pixel is not a little square, a pixel is not a little square, a pixel is not a little square! (and a voxel is not a little cube)! Tech. rep., Microsoft, Inc.

SOBEL, I. 1995. An isotropic 3x3x3 volume gradient operator. Unpublished manuscript, May.

SOLLENBERG, R. L., AND MILGRAM, P. 1993. Effects of stereoscopic and rotational displays in a three-dimensional path tracing task. In *Human Factors*, 483–499.

SPEARY, D., AND KENNON, S. 1990. Volume probes: Interactive data exploration on arbitrary grids. *Computer Graphics 24*, 5–12.

SRAMEK, M. 1994. Fast surface rendering from raster data by voxel traversal using chessboard distance. In *Proceedings of Visualization '94*, 188–195.

SUBRAMANIAN, K. R., AND FUSSELL, D. S. 1990. Applying space sudivision techniques to volume rendering. In *Proceedings of Visualization '90*, 150–159.

SWAN, J. E., MUELLER, K., MÖLLER, T., SHAREEF, N., CRAWFIS, R., AND YAGEL, R. 1997. An anti-aliasing technique for splatting. In *Proceedings of the 1997 IEEE Visualization Conference*, 197–204.

SWEENEY, J., AND MUELLER, K. 2002. Shear-warp deluxe: The shear-warp algorithm revisited. In *Eurographics / IEEE TCVG Symposium on Visualization*, 95–104.

TIEDE, U., SCHIEMANN, T., AND HÖHNE, K. 1998. High quality rendering of attributed volume data. In *Proceedings of IEEE Visualization*, 255–262.

UPSON, C., AND KEELER, M. 1988. V-BUFFER: Visible volume rendering. *Computer Graphics 22*, 4 (Aug.), 59–64.

VAN SCHELTINGA, J., SMIT, J., AND BOSMA, M. 1995. Design of an on-chip reflectance map. In *Proceedings of the 10th Eurographics Workshop on Graphics Hardware*, 51–55.

VÉZINA, G., FLETCHER, P., AND ROBERTSON, P. 1992. Volume rendering on the MasPar MP-1. In *1992 Workshop on Volume Visualization*, 3–8.

VOLZ, W. 2000. Gigabyte volume viewing using split software/hardware interpolation. In *Volume Visualization and Graphics Symposium 2000*, 15–22.

VOORHIES, D., AND FORAN, J. 1994. Reflection vector shading hardware. In *Computer Graphics*, Proceedings of SIGGRAPH 94, 163–166.

WEILER, M., WESTERMANN, R., HANSEN, C., ZIMMERMAN, K., AND ERTL, T. 2000. Level-of-detail volume rendering via 3d textures. In *Volume Visualization and Graphics Symposium 2000*, 7–13.

WEILER, M., KRAUS, M., MERZ, M., AND ERTL, T. 2003. Hardware-based ray casting for tetrahedral meshes. In *To appear in the Proceedings of IEEE Visualization*.

WEILER, M., KRAUS, M., MERZ, M., AND ERTL, T. 2003. Hardware-based view-independent cell projection. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (April–June), 163–175.

WEISKOPF, D., ENGEL, K., AND ERTL, T. 2002. Volume clipping via per-fragment operations in texture-based volume rendering. In *Visualization 2002*, 93–100.

WEISKOPF, D., ENGEL, K., AND ERTL, T. 2003. Interactive clipping techniques for texture-based volume visualization and volume shading. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (July/September), 298–313.

WELLS, W., VIOLA, P., ATSUMI, H., NAKAJIMA, S., AND KIKINIS, R. 1996. Multi-modal volume registration by maximization of mutual information. *Medical Image Analysis 1*, 1, 35–51.

WESTERMANN, R., AND ERTL, T. 1998. Efficiently using graphics hardware in volume rendering applications. In *Computer Graphics*, SIGGRAPH '98 Proceedings, 169–178.

WESTOVER, L. 1989. Interactive volume rendering. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, University of North Carolina at Chapel Hill, Chapel Hill, NC, C. Upson, Ed., 9–16.

WESTOVER, L. 1990. Footprint evaluation for volume rendering. In *Computer Graphics*, Proceedings of SIGGRAPH 90, 367–376.

WESTOVER, L. A. 1991. *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, The University of North Carolina at Chapel Hill, Department of Computer Science. Technical Report, TR91-029.

WILLIAMS, P. L., AND MAX, N. 1992. A volume density optical model. *Workshop on Volume Visualization* (Oct.), 61–68.

WILSON, O., GELDER, A. V., AND WILHELMS, J. 1994. Direct volume rendering via 3D textures. Ucsc-crl-94-19, Jack Baskin School of Eng., University of California at Santa Cruz.

WITTENBRINK, C., AND HARRINGTON, M. 1994. A scalable MIMD volume rendering algorithm. In *Eighth International Parallel Processing Symposium*, 916–920.

WITTENBRINK, C., AND SOMANI, A. 1993. Permutation warping for data parallel volume rendering. In *Parallel Rendering Symposium, Visualization '93*, 57–60.

WITTENBRINK, C., MALZBENDER, T., AND GOSS, M. 1998. Opacity-weighted color interpolation for volume sampling. In *Symposium on Volume Visualization*, 135–142.

WOLBERG, G. 1990. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California.

WU, Y., BHATIA, V., LAUER, H. C., AND SEILER, L. 2003. Shear-image order ray casting volume rendering. In *Symposium on Interactive 3D Graphics*, 152–162.

YAGEL, R., AND KAUFMAN, A. 1992. Template-based volume viewing. *Computer Graphics Forum, Proceedings Eurographics 11*, 3 (Sept.), 153–167.

YAGEL, R., COHEN, D., AND KAUFMAN, A. 1992. Discrete ray tracing. *IEEE Computer Graphics & Applications* (Sept.), 19–28.

ZUCKER, S. W., AND HUMMEL, R. A. 1981. A three-dimensional edge operator. *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 3 (May), 324–331.

ZUIDERVELD, K. Z., KONING, A. H. J., AND VIERGEVER, M. A. 1992. Acceleration of ray casting using 3D distance transform. In *Proceedings of Visualization in Biomedical Computing*, SPIE, Vol. 1808, Chapel Hill, NC, R. A. Robb, Ed., 324–335.

ZWICKER, M., PFISTER., H., BAAR, J. V., AND GROSS, M. 2001. Surface splatting. In *Computer Graphics*, SIGGRAPH 2001 Proceedings, 371–378.

ZWICKER, M., PFISTER, H., BAAR, J. V., AND GROSS, M. 2002. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics 8*, 3, 223–238.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. October 2001. Ewa volume splatting. *IEEE Visualization 2001*, 29–36.