(version before final edits by the publisher)

Overview of Volume Rendering

(Chapter for *The Visualization Handbook,* eds. C. Johnson and C. Hansen, Academic Press, 2005)

Arie Kaufman and Klaus Mueller

Center for Visual Computing   Computer Science Department   Stony Brook University

INTRODUCTION

Volume visualization is a method of extracting meaningful information from volumetric data using interactive graphics and imaging. It is concerned with volume data representation, modeling, manipulation, and rendering [31][100][101][176]. Volume data are 3D (possibly time-varying) entities that may have information inside them, might not consist of tangible surfaces and edges, or might be too voluminous to be represented geometrically. They are obtained by sampling, simulation, or modeling techniques. For example, a sequence of 2D slices obtained from Magnetic Resonance Imaging (MRI), Computed Tomography (CT), functional MRI (fMRI), or Positron Emission Tomography (PET), is 3D reconstructed into a volume model and visualized for diagnostic purposes or for planning of treatment or surgery. The same technology is often used with industrial CT for non-destructive inspection of composite materials or mechanical parts. Similarly, confocal microscopes produce data which is visualized to study the morphology of biological structures. In many computational fields, such as in computational fluid dynamics, the results of simulations typically running on a supercomputer are often visualized as volume data for analysis and verification. Recently, the area of *volume graphics* [104] has been expanding, and many traditional geometric computer graphics applications, such as CAD and flight simulation, have been exploiting the advantages of volume techniques.

Over the years many techniques have been developed to render volumetric data. Since methods for displaying geometric primitives were already well-established, most of the early methods involve approximating a surface contained within the data using geometric primitives. When volumetric data are visualized using a surface rendering technique, a dimension of information is essentially lost. In response to this, volume rendering techniques were developed that attempt to capture the entire 3D data in a single 2D image. Volume rendering convey more information than surface rendering images, but at the cost of increased algorithm complexity, and consequently increased rendering times. To improve interactivity in volume rendering, many optimization methods both for software and for graphics accelerator implementations, as well as several special-purpose volume rendering machines, have been developed.

VOLUMETRIC DATA

A volumetric data set is typically a set V of samples (x,y,z,v), also called *voxels*, representing the value v of some property of the data, at a 3D location (x,y,z). If the value is simply a 0 or an integer i within a set I, with a value of 0 indicating background and the value of i indicating the presence of an object O_i, then the data is referred to as binary data. The data may instead be multi-valued, with the value representing some measurable property of the data, including, for example, color, density, heat or pressure. The value v may even be a vector, representing, for example, velocity at each location, results from multiple scanning modalities, such as anatomical (CT, MRI) and functional imaging (PERT, fMRI), or color (RGB) triples, such as the Visible Human cryosection dataset [91]. Finally, the volume data may be time-varying, in which case V becomes a 4D set of samples (x,y,z,t,v).

In general, the samples may be taken at purely random locations in space, but in most cases the set V is isotropic containing samples taken at regularly spaced intervals along three orthogonal axes. When the spacing between samples along each axis is a constant, but there may be three different spacing constants for the three axes the set V is anisotropic. Since the set of samples is defined on a regular grid, a 3D array (also called the *volume buffer*, *3D raster*, or simply the *volume*) is typically used to store the values, with the element location indicating position of the sample on the grid. For this reason, the set V will be referred to as the array of values $V(x, y, z)$, which is defined only at grid locations. Alternatively, either rectilinear, curvilinear (structured), or unstructured grids, are employed (e.g., [240]). In a *rectilinear* grid the cells are axis-aligned, but grid spacings along the axes are arbitrary. When such a grid has been non-linearly transformed while preserving the grid topology, the grid becomes *curvilinear*. Usually, the rectilinear grid defining the logical organization is called *computational space*, and the curvilinear grid is called *physical space*. Otherwise the grid is called *unstructured* or *irregular*. An unstructured or irregular volume data is a collection of cells whose connectivity has to be specified explicitly. These cells can be of an arbitrary shape such as tetrahedra, hexahedra, or prisms.

RENDERING VIA GEOMETRIC PRIMITIVES

To reduce the complexity of the volume rendering task, several techniques have been developed which approximate a surface contained within the volumetric data by ways of geometric primitives, most commonly triangles, which can then be rendered using conventional graphics accelerator hardware. A surface can be defined by applying a binary segmentation function $B(v)$ to the volumetric data, where $B(v)$ evaluates to 1 if the value v is considered part of the object, and evaluates to 0 if the value v is part of the background. The surface is then contained in the region where $B(v)$ changes from 0 to 1.

Most commonly, $B(v)$ is either a step function $B(v) = 1, \forall v \geq v_{iso}$ (where v_{iso} is called the *iso-value*), or an interval $[v_1,v_2]$ in which $B(v) = 1, \forall v \in [v_1, v_2]$ (where $[v_1,v_2]$ is called the *iso-interval*). For the former, the resulting surface is called the *iso-surface,* while for the latter the resulting structure is called the *iso-contour*. Several methods for extracting and rendering iso-surfaces have been developed, a few are briefly described here. The *Marching Cubes* algorithm [136] was developed to approximate an iso-valued surface with a triangle mesh. The algorithm breaks down the ways in which a surface can pass through a grid cell into 256 cases, based on the $B(v)$ membership of the 8 voxels that form the cell's vertices. By ways of symmetry, the 256 cases reduce to 15 base topologies, although some of these have duals, and a technique called *Asymptotic Decider* [185] can be applied to select the correct dual case and thus prevent the incidence of holes in the triangle mesh. For each of the 15 cases (and their duals), a generic set of triangles representing the surface is stored in a look-up table. Each cell through which a surface passes maps to one of the base cases, with the actual triangle vertex locations being determined using linear interpolation of the cell vertices on the cell edges (see Fig. 1). A normal value is estimated for each triangle vertex, and standard graphics hardware can be utilized to project the triangles, resulting in a relatively smooth shaded image of the iso-valued surface.

When rendering a sufficiently large data set with the March-

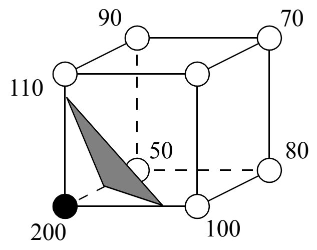


Figure 1: A grid cell with voxel values as indicated, intersected by an iso-surface (iso-value=125). This is base case #1 of the Marching Cubes algorithm: a single triangle separating surface interior (black vertex) from exterior (white vertices). The positions of the triangle vertices are estimated by linear interpolation along the cell edges.

ing Cubes algorithm, with an average of 3 triangles per cell, millions of triangles may be generated, and this can impede interactive rendering of the generated polygon mesh. To reduce the number of triangles, one may either post-process the mesh by applying one of the many mesh decimation methods (see e.g., [63][88][220]), or produce a reduced set of primitives in the mesh generation process, via a feature-sensitive octree method [223] or discretized Marching Cubes [170]. The fact that during viewing many of the primitives may map to a single pixel on the image plane led to the development of screen-adaptive surface rendering algorithms that use 3D points as the geometric primitive. One such algorithm is Dividing Cubes [36], which subdivides each cell through which a surface passes into subcells. The number of divisions is selected such that the subcells project onto a single pixel on the image plane. Another algorithm which uses 3D points as the geometric primitive is the Trimmed Voxel Lists method [235]. Instead of subdividing, this method uses one 3D point (with normal) per visible surface cell, projecting that cell on up to three pixels of the image plane to insure coverage in the image.

The traditional Marching Cubes algorithms simply marches across the grid and inspects every cell for a possible iso-surface. This can be wasteful when users want to interactively change the iso-value v_{iso} and -surface to explore the different surfaces embedded in the data. By realizing that an iso-surface can only pass through a cell if at least one voxel has a value above or equal v_{iso} and at least one voxel has a value below or equal v_{iso}, one can devise data structures that only inspect cells where this criterion is fulfilled. Examples are the NOISE algorithm [135] that uses a K-D tree embedded into span-space for quickly identifying the candidate cells (this method was later improved by [35] who used an interval tree), as well as the ISSUE algorithm [224]. Finally, since often triangles are generated that are later occluded during the rendering process, it is advisable to visit the cells in front-to-back order and only extract and render triangles that fall outside previously occluded areas [62].

DIRECT VOLUME RENDERING: PRELUDE

Representing a surface contained within a volumetric data set using geometric primitives can be useful in many applications, however, there are several main drawbacks to this approach. First, geometric primitives can only approximate surfaces contained within the original data. Adequate approximations may require an excessive amount of geometric primitives. Therefore, a trade-off must be made between accuracy and space requirements. Second, since only a surface representation is used, much of the information contained within the data is lost during the rendering process. For example, in CT scanned data useful information is contained not only on the surfaces, but within the data as well. Also, amorphous phenomena, such as clouds, fog, and fire cannot be adequately represented using surfaces, and therefore must have a volumetric representation, and must be displayed using volume rendering techniques.

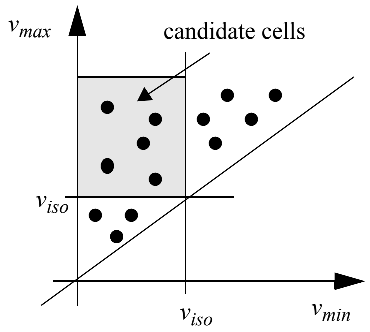


Figure 2: Each grid cell is characterized by its lowest (v_{min}) and its highest (v_{max}) voxel value, and represented by a point in span space. Given an iso-value v_{iso}, only cells that satisfy both $v_{min} \le v_{iso}$ and $v_{max} \ge v_{iso}$ contain the iso-surface and are quickly extracted from a K-D tree [135] or interval-tree [35] embedding of the span-space points.

However, before moving to techniques that visualize the data directly, without going through an intermediate surface extraction step, we first discuss in the next section some of the general principles that govern the theory of discretized functions and signals, such as the discrete volume data. We also present some specialized theoretical concepts, more relevant in the context of volume visualization.

VOLUMETRIC FUNCTION INTERPOLATION

The volume grid *V* only defines the value of some measured property *f(x,y,z)* at discrete locations in space. If one requires the value of *f(x,y,z)* at an off-grid location *(x,y,z)*, a process called *interpolation* must be employed to estimate the unknown value from the known grid samples *V(x,y,z)*. There are many possible interpolation functions (also called *filters* or *filter kernels*). The simplest interpolation function is known as zero-order interpolation, which is actually just a nearest-neighbor function. i.e., the value at any location *(x,y,z)* is simply that of the grid sample closest to that location:

$$f(x, y, z)) = V(round(x), round(y), round(z)) \tag{1}$$

which gives rise to a box filter (black curve in Fig. 4). With this interpolation method there is a region of constant value around each sample in *V*. The human eye is very sensitive to the jagged edges and unpleasant staircasing that result from a zero-order interpolation, and therefore this kind of interpolation gives generally the poorest visual results (see Fig. 3a).

Linear or first-order interpolation (magenta curve in Fig. 4) is the next-best choice, and its 2D and 3D versions are called bi-linear and tri-linear interpolation, respectively. It can be written as 3 stages of 7 linear interpolations, since the filter function is separable in higher dimensions. The first 4 linear interpolations are along *x*:

$$\begin{aligned} &f(u, v_{0,1}, w_{0,1}) \\ &\quad = (1-u)(V(0, v_{0,1}, w_{0,1}) + uV(1, v_{0,1}, w_{0,1})) \end{aligned} \tag{2}$$

Using these results, 2 linear interpolations along *y* follow:

$$f(u, v, w_{0,1}) = (1-v)f(u, 0, w_{0,1}) + vf(u, 1, w_{0,1}) \tag{3}$$

One final interpolation along z yields the interpolation result:

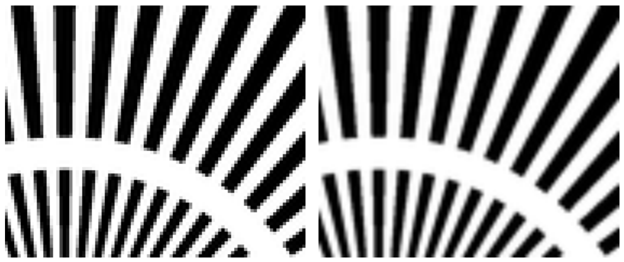


Figure 3: Magnification via interpolation with (a) a box filter; and (b) a bi-linear filter. The latter gives a much more pleasing result.

$$f(x, y, z) = f(u, v, w) = (1-w)f(u, v, 0) + wf(u, v, 1) \quad (4)$$

Here the u,v,w are the distances (assuming a cell of size 1^3, without loss of generality) of the sample at (x,y,z) from the lower, left, rear voxel in the cell containing the sample point (e.g., the voxel with value 50 in Fig. 1). A function interpolated with a linear filter no longer suffers from staircase artifacts (see Fig. 3b). However, it has discontinuous derivatives at cell boundaries, which can lead to noticeable banding when the visual quantities change rapidly from one cell to the next.

A second-order interpolation filter that yields a *f(x,y,z)* with a continuous first derivative is the cardinal spline function, whose 1D function is given by (see blue curve in Fig. 4):

$$h(u) = \begin{cases} (a+2)|u|^3 - (a+3)|u|^2 + 1 & 0 \le |u| < 1 \\ a|u|^3 - 5a|u|^2 + 8a|u| - 4a & 1 \le |u| \le 2 \\ 0 & |u| > 2 \end{cases} \quad (5)$$

Here, u measures the distance of the sample location to the grid points that fall within the extent of the kernel, and a=-0.5 yields the Catmull-Rom spline which interpolates a discrete function with the lowest third-order error [107]. The 3D version of this filter *h(u,v,w)* is separable, i.e., *h(u,v,w)=h(u)h(v)h(w)*, and therefore interpolation in 3D can be written as a 3-stage nested loop.

A more general form of the cubic function has two parameters and the interpolation results obtained with different settings of these parameters has been investigated by Mitchell and Netravali [165]. In fact, the choice of filters and their parameters always presents trade-offs between the sensitivity to noise, sampling frequency ripple, aliasing (see below), ringing, and blurring, and there is no optimal setting that works for all applications. Marschner and Lobb [151] extended the filter discussion to volume rendering and created a challenging volumetric test function with a uniform frequency spectrum that can be employed to visually observe the characteristics of different filters (see Fig. 5). Finally, Möller et al. [167] applied a Taylor series expansion to devise a set of optimal n-th order filters that minimize the *(n+1)*-th order error.

Generally, higher filter quality comes at the price of wider spatial extent (compare Fig. 4) and therefore larger computational effort. The best filter possible in the numerical sense is the *sinc* filter, but it has infinite spatial extent and also tends to noticeable ringing [165]. Sinc filters make excellent, albeit expensive, interpolation filters when used in truncated form and multiplied by a window function [151][252], possibly adaptive to local detail [148]. In practice, first-order or linear filters give satisfactory results for most applications, providing good cost-quality trade-offs, but cubic filters are also used. Zero-order filters give acceptable results when the discrete function has already been sampled at a very high rate, for example in high-definition function lookup tables [270].

All filters presented thus far are grid-interpolating filters, i.e.,

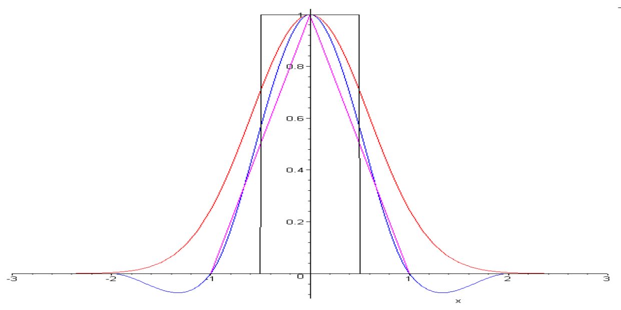


Figure 4: Popular filters in the spatial domain: box (black), linear (magenta), cubic (blue), Gaussian (red)

their interpolation yields *f(x,y,z) = V(x,y,z)* at grid points [254]. When presented with a uniform grid signal they also interpolate a uniform *f(x,y,z)* everywhere. This is not the case with a Gaussian filter function (red curve in Fig. 4) which can be written as:

$$h(u, v, w) = b \cdot e^{-a(u^2 + v^2 + w^2)} \quad (6)$$

Here, a determines the width of the filter and b is a scale factor. The Gaussian has infinite continuity in the interpolated function's derivative, but it introduces a slight ripple (about 0.1%) into an interpolated uniform function. The Gaussian is most popular when a radially symmetric interpolation kernel is needed [268][183] and for grids that assume that the frequency spectrum of *f(x,y,z)* is radially bandlimited [253][182].

It should be noted that interpolation cannot restore sharp edges that may have existed in the original function $f_{org}(x,y,z)$ prior to sampling into the grid. Filtering will always smooth or *lowpass* the original function somewhat. Non-linear filter kernels [93] or transformations of the interpolated results [180] are needed to recreate sharp edges, as we shall see later.

A frequent artifact that can occur is *aliasing*. It results from inadequate sampling and gives rise to strange patterns that did not exist in the sampled signal. Proper pre-filtering (bandlimiting) has to be performed whenever a signal is sampled below its *Nyquist limit*, i.e., twice the maximum frequency that occurs in the signal. Filtering after aliasing will not undo these adverse effects. Fig. 6 illustrates this by ways of an example, and the interested reader may consult standard texts, such as [281] and [57], for more detail.

The gradient of *f(x,y,z)* is also of great interest in volume visualization, mostly for the purpose of estimating the amount of light reflected from volumetric surfaces towards the eye (for example, strong gradients indicate stronger surfaces and therefore stronger reflections). There are three popular methods to estimate a gradient from the volume data [166]. The first computes the gradient vector at each grid point via a process called *central differencing*:

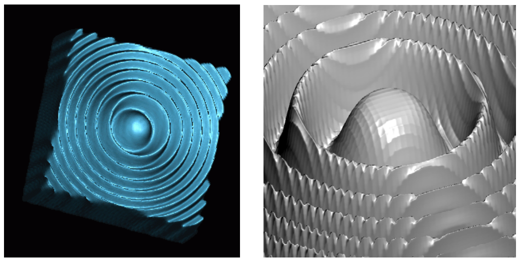


Figure 5: Marschner-Lobb test function, sampled into a 20^3 grid: (a) the whole function, (b) close-up, reconstructed and rendered with a cubic filter.

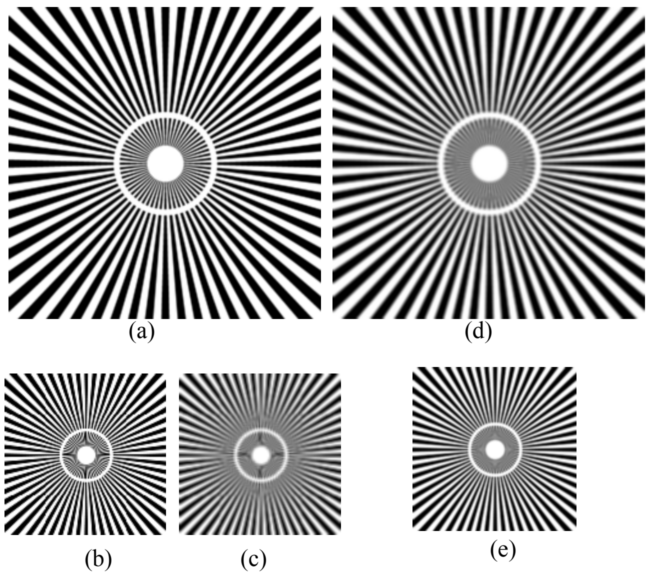


Figure 6: Anti-aliasing: (a) original image; (b) reduction by simple subsampling - disturbing patterns emerge, caused by aliasing the higher frequency content; (c) blurring of (b) does not eliminate patterns; (d) pre-filtering (blurring) of the original image reduces its high frequency content; (e) subsampling of (d) does not cause aliasing due to the prior bandlimiting operation.

$$\begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = \begin{bmatrix} V(x-1,y,z) \\ V(x,y-1,z) \\ V(x,y,z-1) \end{bmatrix} - \begin{bmatrix} V(x+1,y,z) \\ V(x,y+1,z) \\ V(x,y,z+1) \end{bmatrix} \tag{7}$$

and then interpolates the gradient vectors at a *(x,y,z)* using any of the filters described above. The second method also uses central differencing, but it does it at *(x,y,z)* by interpolating the required support samples on the fly. The third method is the most direct and employs a gradient filter [11] in each of the three axis directions to estimate the gradients. These three gradient filters could be simply the *(u,v,w)* partial derivatives of the filters described above or they could be a set of optimized filters [166]. The third method gives the best results since it only performs one interpolation step, while the other two methods have lower complexity and often have practical application-specific advantages. An important observation is that gradients are much more sensitive to the quality of the interpolation filter since they are used in illumination calculations, which consist of higher-order functions that involve the normal vectors, which in turn are calculated from the gradients via normalization [167].

VOLUME RENDERING TECHNIQUES

In the next subsections various fundamental volume rendering techniques are explored. *Volume rendering* or *direct volume rendering* is the process of creating a 2D image directly from 3D volumetric data, hence it is often called *direct volume rendering.* Although several of the methods described in these subsections render surfaces contained within volumetric data, these methods operate on the actual data samples, without generating the intermediate geometric primitive representations used by the algorithms in the previous section.

Volume rendering can be achieved using an *object-order*, an *image-order*, or a *domain-based* technique. Hybrid techniques have also been proposed. Object-order volume rendering techniques use a *forward mapping* scheme where the volume data is mapped onto the image plane. In image-order algorithms, a *backward mapping* scheme is used where rays are cast from each pixel in the image plane through the volume data to determine the final pixel value. In a domain-based technique the spatial volume data is first transformed into an alternative domain, such as compression, frequency, and wavelet, and then a projection is generated directly from that domain.

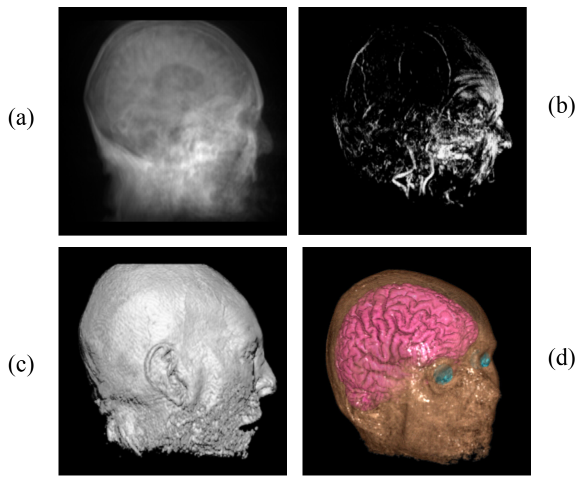


Figure 7: CT head rendered in the four main volume rendering modes: (a) X-ray; (b) MIP; (c) Iso-surface; (d) Translucent.

Image-Order Techniques

There are four basic volume rendering modes: *X-ray rendering, Maximum Intensity Projection (MIP)*, *iso-surface rendering* and *full volume rendering*, where the third mode is just a special case of the fourth. These four modes share two common operations: (i) They all cast rays from the image pixels, sampling the grid at discrete locations along their paths, and (ii) they all obtain the samples via interpolation, using the methods described in the previous section. The modes differ, however, in how the samples taken along a ray are combined. In X-ray, the interpolated samples are simply summed, giving rise to a typical image obtained in projective diagnostic imaging (Fig. 7a), while in MIP only the interpolated sample with the largest value is written to the pixel (Fig. 7b). In full volume rendering (Fig. 7c and d), on the other hand, the interpolated samples are further processed to simulate the light transport within a volumetric medium according to one of many possible models. In the remainder of this section, we shall concentrate on the full volume rendering mode since it provides the greatest degree of freedom, although rendering algorithms have been proposed that merge the different modes into a hybrid image generation model [80].

The fundamental element in full volume rendering is the volume rendering integral. In this section we shall assume the *low-albedo* scenario, in which a certain light ray only scatters once before leaving the volume. The low-albedo *optical model* was first described by [14] and [98], and then formally derived by [152]. It computes, for each cast ray, the quantity $I_\lambda(\boldsymbol{x},\boldsymbol{r})$, which is the amount of light of wavelength λ coming from ray direction ***r*** that is received at point ***x*** on the image plane:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \int_0^L C_\lambda(s)\mu(s)\exp\left(-\int_0^s \mu(t)dt\right)ds \tag{8}$$

Here L is the length of ray ***r***. We can think of the volume as being composed of particles with certain mass density values μ (Max

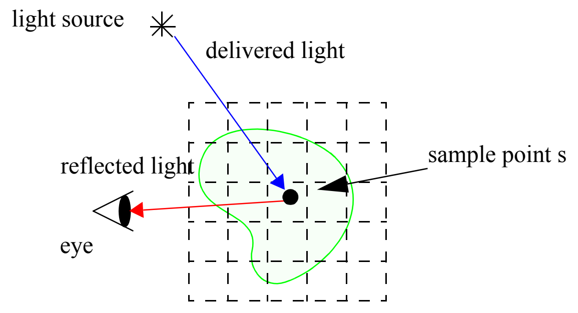


Figure 8: Transport of light to the eye.

calls them light extinction values [152]). These values, as well as the other quantities in this integral, are derived from the interpolated volume densities *f(x,y,z)* via some mapping function. The particles can contribute light to the ray in three different ways: via emission [215], transmission, and reflection [258], thus $C_\lambda(s)=E_\lambda(s)+T_\lambda(s)+R_\lambda(s)$. The latter two terms, T_λ and R_λ, transform light received from surrounding light sources, while the former, E_λ, is due to the light-generating capacity of the particle. The reflection term takes into account the specular and diffuse material properties of the particles. To account for the higher reflectivity of particles with larger mass densities, one must weight C_λ by μ. In low-albedo, we only track the light that is received on the image plane. Thus, in (8), C_λ is the portion of the light of wavelength λ available at location *s* that is transported in the direction of ***r***. This light then gets attenuated by the mass densities of the particles along ***r***, according to the exponential attenuation function.

$R_\lambda(s)$ is computed via the standard illumination equation [57]:

$$R(s) = k_a C_a + k_d C_l C_o(s)(N(s)\cdot L(s)) + k_s C_l (N(s)\cdot H(s))^{ns} \quad (9)$$

where we have dropped the subscript λ for reasons of brevity. Here, C_a is the ambient color, k_a is the ambient material coefficient, C_l is the color of the light source, C_o is the color of the object (determined by the density-color mapping function), k_d is the diffuse material coefficient, N is the normal vector (determined by the gradient), L is the light direction vector, k_s is the specular material coefficient, H is the halfvector, and *ns* is the Phong exponent.

Equation (8) only models the attenuation of light from s to the eye (blue ray in Fig. 8). But the light received at s is also attenuated by the volume densities on its path from the light source to s (red ray in Fig. 8). This gives rise to the following term for C_l in (9), which is now dependent on the location s:

$$C_l(s) = C_L \exp\left(-\int_s^T \mu(t)dt\right) \quad (10)$$

Here, C_L is the color of the lightsource and T is the distance from s to the light source (see Fig. 8). The inclusion of this term into (9) produces volumetric shadows, which give greater realism to the image [191][290] (see Fig. 9). In practice, applications that compute volumetric shadows are less common, due to the added computational complexity, but an interactive hardware-based approach has been recently proposed [113][114].

The analytic volume rendering integral cannot, in the general case, be computed efficiently, if at all, and therefore a variety of approximations are in use. An approximation of (8) can be formulated using a discrete Riemann sum, where the rays interpolate a set of samples, most commonly spaced apart by a distance Δs:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s - 1} C_\lambda(i\Delta s)\mu(i\Delta s)\Delta s \prod_{j=0}^{i-1} \exp(-\mu(j\Delta s)\Delta s) \quad (11)$$

A few more approximations make the computation of this equation more efficient. First, the *transparency* $t(i\Delta s)$ is defined as $\exp(-\mu(i\Delta s)\Delta s)$. Transparency assumes values in the range [0.0, 1.0]. The *opacity* $\alpha(i\Delta s) = (1-t(i\Delta s))$ is the inverse of the transparency. Further, the exponential term in (11) can be approximated by the first two terms of its Taylor series expansion: $t(i\Delta s) = \exp(-\mu(i\Delta s)\Delta s) \approx 1-\mu(i\Delta s)\Delta s$. Then, one can write: $\mu(i\Delta s)\Delta s \approx 1-t(i\Delta s) = \alpha(i\Delta s)$. This transforms (11) into the well-known compositing equation:

$$I_\lambda(\boldsymbol{x},\boldsymbol{r}) = \sum_{i=0}^{L/\Delta s - 1} C_\lambda(i\Delta s)\alpha(i\Delta s)\cdot \prod_{j=0}^{i-1} (1-\alpha(j\Delta s)) \quad (12)$$

This is a recursive equation in (1-α) and gives rise to the recursive *front-to-back compositing* formula [127][207]:

$$\begin{aligned} c &= C(i\Delta s)\alpha(i\Delta s)(1-\alpha)+c \\ \alpha &= \alpha(i\Delta s)(1-\alpha)+\alpha \end{aligned} \quad (13)$$

Thus, a practical implementation of volumetric ray would traverse the volume from front to back, calculating colors and opacities at each sampling site, weighting these colors and opacities by the current accumulated transparency (1-α), and adding these terms to the accumulated color and transparency to form the terms for the next sample along the ray. An attractive property of the front-to-back traversal is that a ray can be stopped once α approaches 1.0, which means that light originating from structures further back is completely blocked by the cumulative opaque material in front. This provides for accelerated rendering and is called *early ray termination*. An alternative form of (13) is the *back-to-front compositing* equation:

$$\begin{aligned} c &= c(1-\alpha(i\Delta s))+C(i\Delta s) \\ \alpha &= \alpha(1-\alpha(i\Delta s))+\alpha(i\Delta s) \end{aligned} \quad (14)$$

Back-to-front compositing is a generalization of the Painter's algorithm and does not enjoy speed-up opportunities of early ray termination and is therefore less frequently used.

Equation (12) assumes that a ray interpolates a volume that stores at each grid point a color vector (usually a (red, green, blue) = RGB triple) as well as an α value [127][128]. There, the colors are obtained by shading each grid point using (9). Before we describe the alternative representation, let us first discuss how the voxel densities are mapped to the colors C_o in (9).

The mapping is implemented as a set of mapping functions, often implemented as 2D tables, called *transfer functions*. By ways of the transfer functions, users can interactively change the properties of the volume dataset. Most applications give access to four mapping functions: *R(d)*, *G(d)*, *B(d)*, *A(d)*, where *d* is the value of a grid voxel, typically in the range of [0,255] for 8-bit volume data. Thus, users can specify semi-transparent materials by mapping their densities to opacities < 1.0, which allows rays to acquire a mix of colors that is due to all traversed materials. More advanced

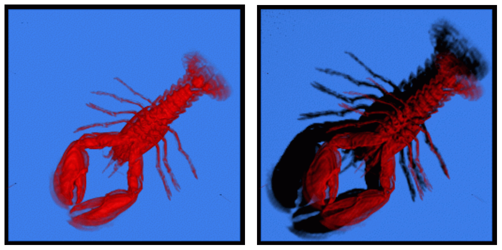
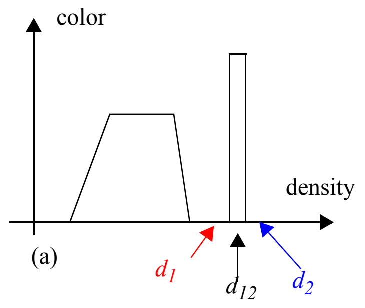


Figure 9: CT lobster rendered without shadows (left) and with shadows (right). The shadows on the wall behind the lobster as well as the self-shadowing of the legs creates greater realism.



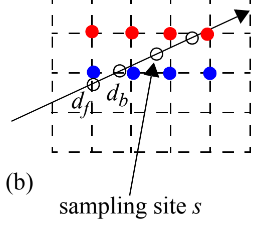


Figure 10: Transfer function aliasing. When the volume is rendered pre-classified, then both the red (density d_1) and the blue (density d_2) voxels receive a color of zero, according to the transfer function shown on the left. At ray sampling this voxel neighborhood at s would then interpolate a color of zero as well. On the other hand, in post-classified rendering, the ray at s would interpolate a density close to d_{12} (between d_1 and d_2) and retrieve the strong color associated with d_{12} in the transfer function.

applications give users also access to transfer functions that map $k_s(d)$, $k_d(d)$, $ns(d)$, and others. Wittenbrink pointed out that the colors and opacities at each voxel should be multiplied *prior* to interpolation to avoid artifacts on object boundaries [280].

The model in (12) is called the *pre-classified model*, since voxel densities are mapped to colors and opacities *prior* to interpolation. This model cannot resolve high frequency detail in the transfer functions (see Fig. 10 for an example), and also typically gives blurry images under magnification [180]. An alternative model that is more often used is the *post-classified model*. Here, the raw volume values are interpolated by the rays, and the interpolation result is mapped to color and opacity:

$$I_\lambda(\boldsymbol{x}, \boldsymbol{r}) = \sum_{i=0}^{L/\Delta s - 1} C_\lambda(f(i\Delta s), g(i\Delta s))\alpha(f(i\Delta s)) \prod_{j=0}^{i-1} (1 - \alpha(f(j\Delta s))) \qquad (15)$$

The function value $f(i\Delta s)$ and the gradient vector $g(i\Delta s)$ are interpolated from $f_d(x,y,z)$ using a 3D interpolation kernel, and C_λ and α are now the transfer and shading functions that translate the interpolated volume function values into color and opacity. This generates considerably sharper images (see Fig. 11).

A quick transition from 0 to 1 at some density value d_i in the opacity transfer function selects the iso-surface $d_{iso}=d_i$. Thus, iso-surface rendering is merely a subset of full volume rendering, where the ray hits a material with $d=d_{iso}$ and then immediately becomes opaque and terminates.

Post-classified rendering only eliminates some of the problems that come with busy transfer functions. Consider again Fig. 10a, and now assume a very narrow peak in the transfer function at d_{12}. With this kind of transfer function, a ray point-sampling the volume at s may easily miss to interpolate d_{12}, but may have interpolated it, had it just sampled the volume at $s+\delta s$. Pre-integrated transfer functions [55] solve this problem by pre-computing a 2D table that stores the analytical volume rendering integration for all possible density pairs (d_f, d_b). This table is then indexed during rendering by each ray sample pair (d_b, d_f), interpolated at sample locations Δs apart (see Fig. 10b). The pre-integration assumes a piecewise linear function within the density pairs, and thus guarantees that no transfer function detail falling between two interpolated (d_f, d_b) fails to be considered in the discrete ray integration.

Object-Order Techniques

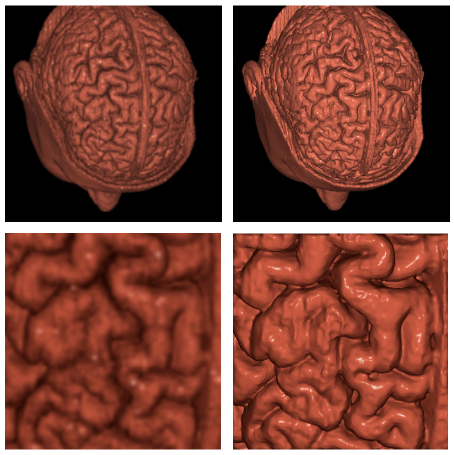


Figure 11: Pre-classified (left column) vs. post-classified rendering (right column). The latter yields sharper images since the opacity and color classification is performed after interpolation. This eliminates the blurry edges introduced by the interpolation filter.

Object-order techniques decompose the volume into a set of basis elements or *basis functions* which are individually projected to the screen and assemble into an image. If the volume rendering mode is X-ray or MIP, then the basis functions can be projected in any order, since in X-ray and MIP the volume rendering integral degenerates to a commutative sum or MAX operation. In contrast, depth ordering is required when solving for the generalized volume rendering integral (8). Early work represented the voxels as disjoint cubes, which gave rise to the *cuberille* representation [68][83]. Since a cube is equivalent to a nearest neighbor kernel, the rendering results were inferior. Therefore, more recent approaches have turned to kernels of higher quality.

To better understand the issues associated with object-order projection it helps to view the volume as a field of basis functions h, with one such basis kernel located at each grid point where it is modulated by the grid point's value (see Fig. 12 where two such kernels are shown). This ensemble of modulated basis functions then makes up the continuous object representation, i.e., one could interpolate a sample anywhere in the volume by simply adding up the contributions of the modulated kernels that overlap at the location of the sample value. Hence, one could still traverse this ensemble with rays and render it in image-order. However, a more efficient method emerges when realizing that the contribution of a voxel j with value d_j is given by $d_j \cdot \int h(s)ds$, where s follows the line of kernel integration along the ray. Further, if the basis kernel is radially symmetric, then the integration $\int h(s)ds$ is independent of the viewing direction. Therefore, one can perform a pre-integration of $\int h(s)ds$ and store the result into a lookup-table. This table is called the kernel *footprint*, and the kernel projection process is referred to as *kernel splatting* or simply, *splatting*. If the kernel is a Gaussian, then the footprint is a Gaussian as well. Since the kernel is identical for all voxels, we can use it for all voxels. We can generate an image by going through the list of object voxels in depth-order and performing the following steps for each (see again Fig. 12): (i) Calculate the screen-space coordinate of the projected grid point; (ii) center the footprint around that point and stretch it according to the image magnification factor; (iii) rasterize the footprint to the screen, using the pre-integrated footprint table and multiplying the indexed values by the voxel's value [268][269][270]. This rasterization can either be performed via fast DDA procedures [147][174], or in graphics hardware, by texture-mapping the footprint (basis image) onto a polygon [42].

There are three types of splatting: composite-only, axis-

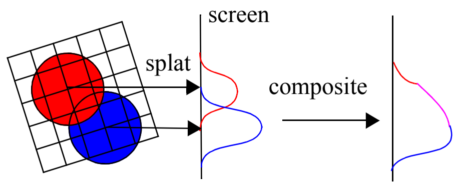


Figure 12: Object-order volume rendering with kernel splatting implemented as footprint mapping.

aligned sheet-buffered, and image-aligned sheet-buffered splatting. The composite-only method was proposed first [269] and is the most basic one (see Fig. 12). Here, the object points are traversed in either front-to-back or back-to-front order. Each is first assigned a color and opacity using the shading equation (9) and the transfer functions. Then, each point is splatted into the screen's color and opacity buffers and the result is composited with the present image (Equation (13)). In this approach, color bleeding and slight sparkling artifacts in animated viewing may be noticeable since the interpolation and compositing operations cannot be separated due to the pre-integration of the basis (interpolation) kernel [270].

An attempt to solve this problem gave way to the axis-aligned sheet-buffered splatting approach [268] (see Fig. 13a). Here, the grid points are organized into sheets (basically the volume slices most parallel to the image plane), assigned a color and opacity, and splatted into the sheet's color and opacity buffers. The important difference is that now all splats within a sheet are added and not composited, while only subsequent sheets are composited. This prevents potential color bleeding of voxels located in consecutive sheets, due to the more accurate reconstruction of the opacity layer. The fact that the voxel sheets must be formed by the volume slices most parallel to the viewing axis leads to a sudden switch of the compositing order when the major viewing direction changes and an orthogonal stack of volume slices must be used to organize the voxels. This causes noticeable popping artifacts where some surfaces suddenly reflect less light and others more [173]. The solution to this problem is to align the compositing sheet with the image plane at all times, which gives rise to the image-aligned sheet-buffered splatting approach [173] (see Fig. 13b). Here, a slab is advanced across the volume and all kernels that intersect the slab are sliced and projected. Kernel slices can be pre-integrated into footprints as well, and thus this sheet-buffered approach differs from the original one in that each voxel has to be considered more than once. The image-aligned splatting method provides the most accurate reconstruction of the voxel field prior to compositing and eliminates both color bleeding and popping artifacts. It is also best suited for post-classified rendering since the density (and gradient) field is reconstructed accurately in each sheet. However, it is more expensive due to the multiple splatting of a voxel.

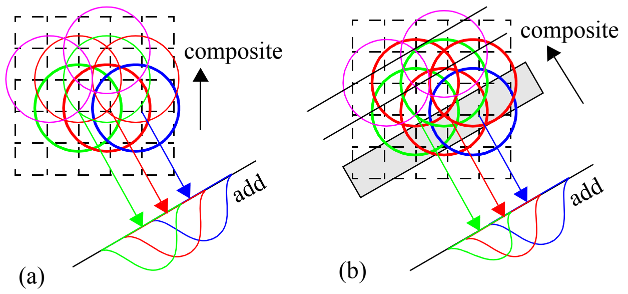


Figure 13: Sheet-buffered splatting: (a) axis-aligned - the entire kernel within the current sheet is added, (b) image-aligned - only slices of the kernels intersected by the current sheet-slab are added.

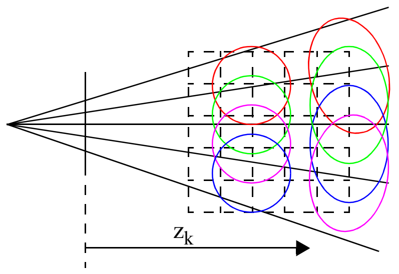


Figure 14: Stretching the basis functions in volume layers $z>z_k$, where the sampling rate of the ray grid is progressively less than the volume resolution.

The divergence of rays under perspective viewing causes undersampling of the volume portions further away from the viewpoint (see Fig. 14). This leads to aliasing in these areas. As was demonstrated in Fig. 6, lowpassing can eliminate the artifacts caused by aliasing and replace them by blur (see Fig. 15). For perspective rendering the amount of required lowpassing increases with distance from the viewpoint. The kernel-based approaches can achieve this progressive lowpassing by simply stretching the footprints of the voxels as a function of depth, since stretched kernels act as lowpass filters (see Fig. 14). EWA (Elliptical Weighted Average) Splatting [293] provides a general framework to define the screen-space shape of the footprints, and their mapping into a generic footprint, for generalized grids under perspective viewing. An equivalent approach for raycasting is to split the rays in more distant volume slices to always maintain the proper sampling rate [190]. Kreeger et al. [118] proposed an improvement of this scheme that splits and merges rays in an optimal way.

A major advantage of object-order methods is that only the points (or other basis primitives, such as tetrahedral or hexagonal cells [273]) which make up the object must be stored. This can be advantageous when the object has an intricate shape, with many pockets of empty space [159]. While raycasting would spend much effort traversing (and storing) the empty space, kernel-based or *point-based* objects will not consider the empty space, neither during rendering nor for storage. However, there are trade-offs, since the rasterization of a footprint takes more time than the commonly used trilinear interpolation of ray samples, since the radially symmetric kernels employed for splatting must be larger than the trilinear kernels to ensure proper blending. Hence, objects with compact structure are more favorably rendered with image-order methods or hybrid methods (see next section). Another disadvantage of object-order methods is that early ray termination is not available to cull occluded material early from the rendering pipeline. The

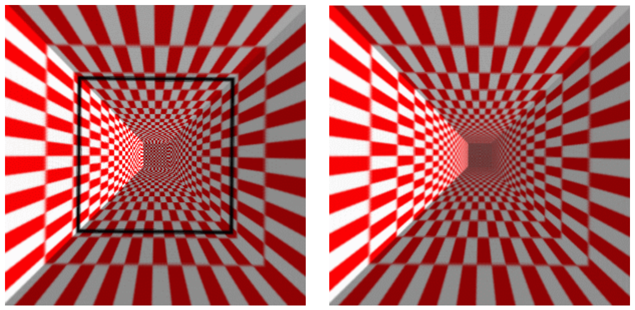


Figure 15: Anti-aliased splatting: (Left) A checkerboard tunnel rendered in perspective with equal sized splats. Aliasing occurs at distances beyond the black square. (Right) The same checkerboard tunnel rendered with scaled splats. The aliasing has been replaced by blur.

object-order equivalent is early point elimination, which is more difficult to achieve than early ray termination. Finally, image-order methods allow the extension of raycasting to raytracing, where secondary and higher-order rays are spawned at reflection sites. This facilitates mirroring on shiny surfaces, inter-reflections between objects, and soft shadows.

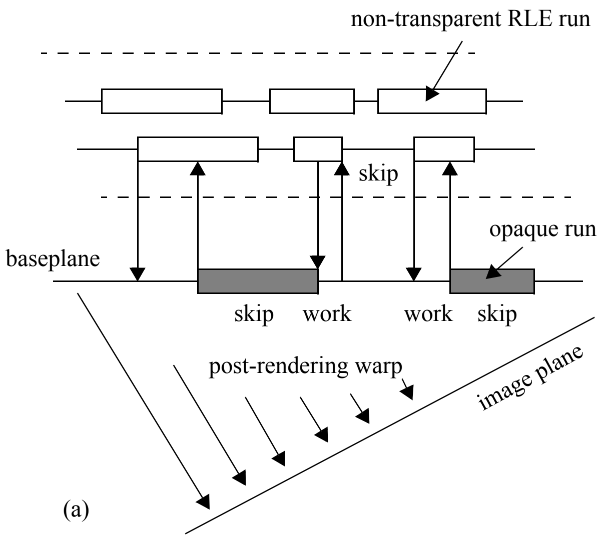
There are a number of ways to store and manage point-based objects. These schemes are mainly distinguished by their ability to exploit spatial coherence during rendering. The lack of spatial coherence requires more depth sorting during rendering and also means more storage for spatial parameters. The least spatial coherence results from storing the points sorted by density [41]. This has the advantage that irrelevant points, being assigned transparent values in the transfer functions, can be quickly culled from the rendering pipeline. However, it requires that *(x,y,z)* coordinates and, possibly gradient vectors, are stored along with the points since neighborhood relations are completely lost. It also requires that all points be view-transformed first before they can be culled due to occlusion or exclusion from the viewing pyramid. The method also requires that the points be depth-sorted during rendering, or at least, tossed into depth bins [177]. A compromise is struck by Ihm and Lee [94] who sort points by density within volume slices only, which gives implicit depth-ordering when used in conjunction with an axis-aligned sheet-buffer method. A number of approaches exist that organize the points into RLE (Run Length Encoded) lists, which allow the spatial coordinates to be incrementally computed when traversing the runs [108][182]. However, these approaches do not allow points to be easily culled based on their density value. Finally, one may also decompose the volume into a spatial octree and maintain a list of voxels in each node. This provides depth sorting on the node-level.

A number of surface-based splatting methods have also been described. These do not provide the flexibility of volume exploration via transfer functions, since the original volume is discarded after the surface has been extracted. They only allow a fixed geometric representation of the object that can be viewed at different orientations and with different shadings. A popular method is *shell-rendering* [259] which extracts from the volume (possibly with a sophisticated segmentation algorithm) a certain thin or thick surface or contour and represents it as a closed shell of points. Shell-rendering is fast since the number of points is minimized and its data structure used has high cache coherence. More advanced point-based surface rendering methods are QSplat [214], Surfels [201], and Surface Splats [292], which have been predominantly developed for point-clouds obtained with range scanners, but can also be used for surfaces extracted from volumes [293].

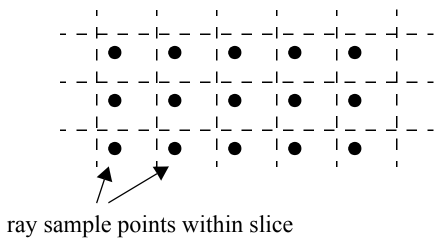
Hybrid Techniques

Hybrid techniques seek to combine the advantages of the image-order and object-order methods, i.e., they use object-centered storage for fast selection of relevant material (which is a hallmark of object-order methods) and they use early ray termination for fast occlusion culling (which is a hallmark of image-order methods).

The shear-warp algorithm [120] is such a hybrid method. In shear-warp, the volume is rendered by a simultaneous traversal of RLE-encoded voxel and pixel runs, where opaque pixels and transparent voxels are efficiently skipped during these traversals (see Fig. 16a). Further speed comes from the fact that sampling only occurs in the volume slices via bilinear interpolation, and that the ray grid resolution matches that of the volume slices, and therefore the same bilinear weights can be used for all rays within a slice (see Fig. 16b). The caveat is that the image must first be rendered from a sheared volume onto a so-called base-plane, that is aligned with the volume slice most parallel to the true image plane (Fig. 16a). After completing the base-plane rendering, the base



(a)



(b)

Figure 16: The shear-warp algorithm. (a) mechanism, (b) interpolation scheme.

plane image must be warped onto the true image plane and the resulting image is displayed. All of this combined enables framerates in excess of 10 frames/s on current PC processors, for a 128^3 volume. There are a number of compromises that had to be made in the process:

- Since the interpolation only occurs within one slice at a time, more accurate tri-linear interpolation reduces to less accurate bilinear interpolation and the ray sampling distance varies between 1 and $\sqrt{3}$, depending on the view orientation. This leads to aliasing and staircasing effects at viewing angles near 45°.
- Since the volume is run-length one needs to use three sets of voxel encodings (but it could be reduced to two [249]), one for the each major viewing direction. This triples the memory required for the runs, but in return, the RLE encoding saves considerable space.
- Since there is only one interpolated value per voxel-slice 4-neighborhood, zooming can only occur during the warping phase and not during the projection phase. This leads to considerable blurring artifacts at zoom factors greater than 2. The post-rendering magnification in fact is a major source of the speedup for the shear-warp algorithm.

An implementation of the shear-warp algorithm is publicly available as the *volpack* package [90] from Stanford University.

Domain Volume Rendering

In domain rendering, the spatial 3D data is first transformed into another domain, such as compression, frequency, and wavelet domain, and then a projection is generated directly from that domain or with the help of information from that domain. The frequency domain rendering applies the Fourier slice projection theorem, which states that a projection of the 3D data volume from a certain view direction can be obtained by extracting a 2D slice perpendicular to that view direction out of the 3D Fourier spectrum and then inverse Fourier transforming it. This approach obtains the 3D volume projection directly from the 3D spectrum of the data, and therefore reduces the computational complexity for volume rendering from $O(N^3)$ to $O(N^2 log(N))$ [50][149][256]. A major problem of frequency domain volume rendering is the fact that the resulting projection is a line integral along the view direction which does not exhibit any occlusion and attenuation effects. Totsuka and Levoy [256] proposed a linear approximation to the exponential attenuation [215] and an alternative shading model to fit the computation within the frequency-domain rendering framework.

The compression domain rendering performs volume rendering from compressed scalar data without decompressing the entire data set, and therefore reduces the storage, computation and transmission overhead of otherwise large volume data. For example, Ning and Hesselink [187][188] first applied vector quantization in the spatial domain to compress the volume and, then directly rendered the quantized blocks using regular spatial domain volume rendering algorithms. Fowler and Yagel [58] combined differential pulse-code modulation and Huffman coding, and developed a lossless volume compression algorithm, but their algorithm is not coupled with rendering. Yeo and Liu [288] applied discrete cosine transform based compression technique on overlapping blocks of the data. Chiueh et al. [33] applied the 3D Hartley transform to extend the JPEG still image compression algorithm [261] for the compression of subcubes of the volume, and performed frequency domain rendering on the subcubes before compositing the resulting sub-images in the spatial domain. Each of the 3D Fourier coefficients in each subcube is then quantized, linearly sequenced through a 3D zig-zag order, and then entropy encoded. In this way, they alleviated the problem of lack of attenuation and occlusion in frequency domain rendering while achieving high compression ratios, fast rendering speed compared to spatial volume rendering, and improved image quality over conventional frequency domain rendering techniques. More recently, Guthe et al. [73] and also Sohn and Bajaj [239] have used principles from MPEG encoding to render time-varying datasets in the compression domain.

Rooted in time-frequency analysis, wavelet theory [34][46] has gained popularity in the recent years. A wavelet is a fast decaying function with zero averaging. The nice features of wavelets are that they have local property in both spatial and frequency domain, and can be used to fully represent the volumes with small number of wavelet coefficients. Muraki [181] first applied wavelet transform to volumetric data sets, Gross et al. [71] found an approximate solution for the volume rendering equation using orthonormal wavelet functions, and Westermann [266] combined volume rendering with wavelet-based compression. However, all of these algorithms have not focused on the acceleration of volume rendering using wavelets. The greater potential of wavelet domain, based on the elegant multiresolution hierarchy provided by the wavelet transform, is to exploit the local frequency variance provided by wavelet transform to accelerate the volume rendering in homogeneous areas. Guthe and Strasser [74] have recently used the wavelet transform to render very large volumes at interactive frame rates, on texture mapping hardware. They employ a wavelet pyramid encoding of the volume to reconstruct, on the fly, a decomposition of the volume into blocks of different resolutions. Here, the resolution of each block is chosen based on the local error committed and the resolution of the screen area the block is projected onto. Each block is rendered individually with 3D texture mapping hardware, and the block decomposition can be used for a number of frames, which amortizes the work spent on the inverse wavelet transform to construct the blocks.

ACCELERATION TECHNIQUES

The high computational complexity of volume rendering has led to a great variety of approaches for its acceleration. In the current section, we will discuss general acceleration techniques that can benefit software as well as hardware implementations. We have already mentioned a few acceleration techniques in the previous section, such as early ray termination [127], post-rendering warps for magnified viewing [120], and the splatting of pre-integrated voxel basis functions [270]. The latter two gave rise to independent algorithms, that is, shear-warp [120] and splatting [270]. Acceleration techniques generally seek to take advantage of properties of the data, such as empty space, occluded space, and entropy, as well as properties of the human perceptional system, such as its insensitivity to noise over structural artifacts.

A number of techniques have been proposed to accelerate the grid traversal of rays in image-order rendering. Examples are the 3D DDA (Digital Differential Analyzer) method [1][59], in which new grid positions are calculated by fast integer-based incremental arithmetic, and the template-based method [284], in which templates of the ray paths are precomputed and used during rendering to quickly identify the voxels to visit. Early-ray termination can be sophisticated into a Russian Roulette scheme [45] in which some rays terminate with lower and others with higher accumulated opacities. This capitalizes on the human eye's tolerance to error masked as noise [146]. In the object-order techniques, fast differential techniques to determine the screen-space projection of the points as well as to rasterize the footprints [147][174] are also available.

Most of the object-order approaches deal well with empty space - they simply don't store and process it. In contrast, ray casting relies on the presence of the entire volume grid since it requires it for sample interpolation and address computation during grid traversal. Although opaque space is quickly culled, via early ray termination, the fast leaping across empty space is more difficult. A number of techniques are available to achieve this (see Fig. 17 for an illustration of the methods described in the following text). The simplest form of space leaping is facilitated by enclosing the object into a set of boxes, possibly hierarchical, and first quickly determine and test the rays' intersection with each of the boxes before

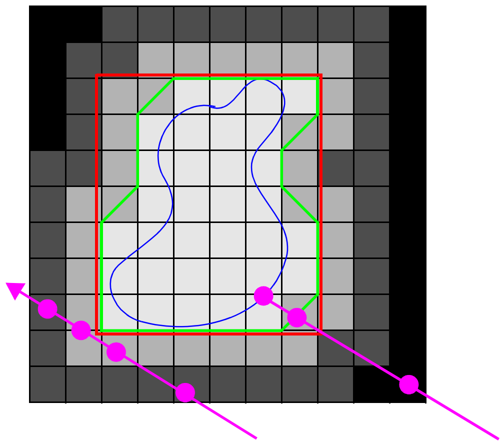


Figure 17: Various object approximation techniques: (blue) isosurface of the object, (lightly shaded) discretized object (proximity cloud =0), (red) bounding box, (green) polygonal hull used in PARC, (darker shaded areas) proximity clouds with grey level indicating distance to the object. Note also that while the right magenta ray is correctly sped up by the proximity clouds, the left magenta ray missing the object is unnecessarily slowed down.

engaging into more time-consuming volumetric traversal of the material within [105]. A better geometrical approximation is obtained by a polyhedral representation, chosen crudely enough to still maintain ease of intersection. In fact, one case utilize conventional graphics hardware to perform the intersection calculation, where one projects the polygons twice to create two Z- (depth) buffers. The first Z-buffer is the standard closest-distance Z-buffer, while the second is a farthest-distance Z-buffer. Since the object is completely contained within the representation, the two Z-buffer values for a given image plane pixel can be used as the starting and ending points of a ray segment on which samples are taken. This algorithm has been known as *PARC* (Polygon Assisted Ray Casting) [237] and it is part of the *VolVis* volume visualization system [4][5], which also provides a multi-algorithm progressive refinement approach for interactivity. By using available graphics hardware, the user is given the ability to interactively manipulate a polyhedral representation of the data. When the user is satisfied with the placement of the data, light sources, and viewpoint, the Z-buffer information is passed to the PARC algorithm, which produces a ray-cast image.

A different technique for empty-space leaping was devised by Zuiderfeld et al. [291] as well as Cohen and Shefer [37] who introduced the concept of *proximity clouds.* Proximity clouds employ a distance transform of the object to accelerate the rays in regions far from the object boundaries. In fact, since the volume densities are irrelevant in empty volume regions, one can simply store the distance transform values in their place and therefore storage is not increased. Since the proximity clouds are the iso-distance layers around the object's boundaries, they are insensitive to the viewing direction. Thus, rays that ultimately miss the object are often still slowed down. To address this shortcoming, Sramek and Kaufman [241] proposed a view-sensitive extension of the proximity clouds approach. Wan [262] places a sphere at every empty voxel position, where the sphere radius indicates the closest non-empty voxel. They apply this technique for the navigation inside hollow volumetric objects, as occurring in virtual colonoscopy [87], and reduce a ray's space traversal to just a few hops until a boundary wall is reached. Finally, Meissner [160] suggested an algorithm that quickly re-computes the proximity cloud when the transfer function changes.

Proximity clouds only handle the quick leaping across empty space, but methods are also available that traverse occupied space faster when the entropy is low. These methods generally utilize a hierarchical decomposition of the volume where each non-leaf node is obtained by low-pass filtering its children. Commonly this hierarchical representation is formed by an octree [155] since these are easy to traverse and store. An octree is the 3D extension of a quadtree [218], which is the 2D extension of a binary tree. Most often a non-leaf node stores the average of its children, which is synonymous with a box filtering of the volume, but more sophisticated filters are possible. Octree don't have to be balanced [274] nor fully expanded into a single root node or into single-voxel leaf nodes. The latter two give rise to a brick-of-bricks decomposition, where the volume is stored as a flat hierarchy of bricks of size n^3 to improve cache-coherence in the volume traversal. Parker et al. [194][195] utilize this decomposition for the raycasting of very large volumes, and they also gives an efficient indexing scheme to quickly find the memory address of the voxels located in the 8-neighborhood required for trilinear interpolation.

When octrees are used for entropy-based rendering, non-leaf node store either an entropy metric of its children, such as standard deviation [45], minimum-maximum range [274], or Lipschitz range [242], or a measure of the error committed when the children are not rendered, such as the root mean square or the absolute error [74]. The idea is to either have the user specify a tolerable error before the frame is rendered or to make the error dependent on the

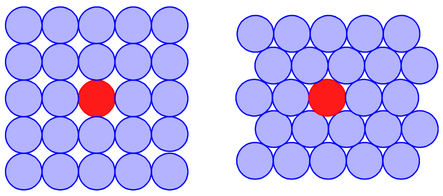


Figure 18: The cartesian grid (left) vs. the hexagonal grid (right) as two possible frequency domain lattices. The latter provides the tightest packing of a discrete 2D signal's circularly-bounded frequency spectrum. (Here, the dark, red circle contains the main spectrum, while the others contain the replicas or aliases.)

time maximally allowed to render the frame, which is known as *time-critical rendering*. In either case, the rays traversing the volume will advance across the volume, but also transcend up and down the octree, based on the metric used, which will either accelerate or decelerate them on their path. A method called β-acceleration will make this traversal also sensitive to the ray's accumulated opacity so far. The philosophy here is that the observable error from using a coarser node will be relatively small when it is weighted by a small transparency in (13).

Octrees are also easily used with object-order techniques, such as splatting. Laur and Hanrahan [124] have proposed an implementation that approximates non-leaf octree nodes by kernels of a radius that is twice the radius of the childrens' kernels, which gives rise to a magnified footprint. They store the childrens' average as well as an error metric based on their standard deviation in each parent node and use a pre-set error to select the nodes during rendering. While this approach uses non-leaves nodes during rendering, other splatting approaches only exploit them for fast occlusion culling. Lee and Ihm [125] as well as Mora et al. [171] store the volume as a set of bricks which they render in conjunction with a dynamically computed hierarchical occlusion map to quickly cull voxels within occluded bricks from the rendering pipeline. Hierarchical occlusion maps [289] are continuously updated during the rendering and thus store a hierarchical opacity map of the image rendered so far. Regions in which the opacity is high are tagged, and when octree nodes fall within such a region all voxels contained in them can be immediately culled. If the octree node does not fall into a fully opaque region then it has to be subdivided and its children are subjected to the same test. An alternative scheme that performs occlusion culling on a finer scale than the box-basis of an octree decomposition is to calculate an occlusion map in which each pixel represents the average of all pixels within the box-neighborhood covered by a footprint [177]. Occlusion of a particular voxel is then determined by indexing the occlusion map with the voxel's screen-space coordinate to determine if its footprint must be rasterized. One could attempt to merge these two methods to benefit both from the large-scale culling afforded by the octree-nodes and from the fine-scale culling of the average-occlusion map.

Hierarchical decomposition is not the only way to reduce the number of point primitives needed to represent an object for rendering. An attractive solution that does not reduce the volume's frequency content, by ways of averaging, is to exploit more space-efficient grids for storage. The most optimal regular lattices are the *face-centered cartesian (FCC)* lattices (see Fig. 19) [39]. The FCC lattices give the densest packings of a set of equal-sized spheres. If the frequency spectrum of the signal represented in the volume is spherical (and many of them are due to the sampling kernel used for volume generation), then they can be packed in the FCC lattice (see Fig. 18 for the 2D equivalent, the hexagonal lattice). The FCC lattice's dual in the spatial domain is the *body-centered cartesian (BCC)* lattice, and the spacing of samples there is the reciprocal of that in the frequency domain, according to the Fourier scaling theo-

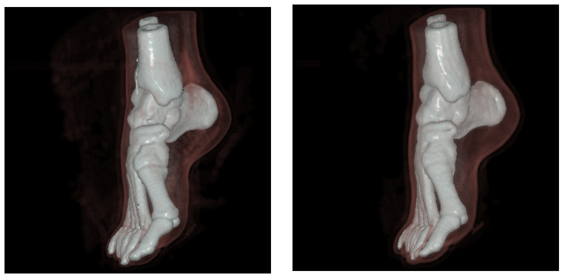


Figure 20: Foot dataset rendered on: (left) Cubic Cartesian (CC) grid, (right) Body Centered (BCC) grid. The renderings are almost identical, but the BCC rendering took 70% of the time of the CC rendering.

rem [17]. This BCC grid gives rise to two interleaved CC grids, each with a sampling interval of $\sqrt{2}$ .and $1/(\sqrt{2})$ apart, which implies that a volume, when sampled into a BCC grid, only requires $\sqrt{2}/2$ =71% of the samples of the usual cubic cartesian (CC) grid [182][253] (see Fig. 19 for an illustration of the grid and Fig. 20 for images). The theorem extends to higher dimensions as well, for example, a time-varying (4D) volume can be stored in a 4D BCC at only 50% of the 4D CC samples. The BCC grids are best used in conjunction with point-based object-order methods, since these use the spherical (radially symmetric) filter required to preserve the spherical shape of the BCC grid-sampled volume's frequency spectrum. The reconstruction of a BCC grid with trilinear filters can lead to aliasing since the trilinear filter's frequency response is not radially symmetric and therefore will include higher spectra when used for interpolation

A comprehensive system for accelerated software-based volume rendering is the UltraVis system devised by Knittel [115]. It can render 256^3 volume at 10 frames/s. It achieves this by optimizing cache performance during both volume traversal and shading, which is rooted in the fact that good cache management is key to achieve fast volume rendering, since the data are so massive. As we have mentioned before, this was also realized by Parker et al. [194][195], and it plays a key role in both custom and commodity hardware approaches as well, as we shall see later. The UltraVis system manages the cache by dividing it into four blocks: one block each for volume bricks, transfer function tables, image blocks, and temporary buffers. Since the volume can only map into a private cache block, it can never be swapped out by a competing data structure, such as a transfer function table or an image tile array. This requires that the main memory footprint of the volume is four times as high since no volume data may be stored in an address space that would map outside the volume's private cache slots. By using a bricked volume decomposition in conjunction with a flock of rays that are traced simultaneously across the brick, the brick's data will only have to be brought in once before it can be discarded when all rays have finished its traversal. A number of additional acceleration techniques give further performance.

Another type of acceleration is achieved by breaking the volume integral of (12) or (15) into segments and storing the composited color and opacity for each partial ray into a data structure. The idea is then to re-combine these partial rays into complete rays for images rendered at viewpoints near the one for which the partial rays were originally obtained (see Fig. 21). This saves the cost for fully integrating all rays for each new viewpoint and reduces it to the expense of compositing a few partial segments per ray, which is much lower. This method falls into the domain of *image-based rendering (IBR)* [29][30][154][221] and is, in some sense, a volumetric extension of the *lumigraph* [69] or *lightfield* [129], albeit dynamically computed. However, one could just as well store a set of partial rays into a static data structure to be used for volumetric-style lumigraph rendering. This idea of using a cache of partial rays for accelerated rendering was exploited by Brady et al. [19][20] for the volume rendering at great perspective distortions, such as found in virtual endoscopy applications [87]. Mueller et al. [178] stored the rays in form of a stack of depth-layered images and rendered these images warped and composited from novel viewpoints within a 30° view cone, using standard graphics hardware (see Fig. 22a). Since gaps may quickly emerge when the layers are kept planar, it helps to also compute, on the fly, a coarse polygonal mesh for each layer that approximates the underlying object, and then map the images onto this mesh when rendering them from a new viewpoint (see Fig. 22b and c). An alternative method that uses a precomputed triangle mesh to achieve similar goals for iso-surface volume rendering was proposed by Chen et al. [28], while Yagel and Shi [286] warped complete images to near-by viewpoints, aided by a depth buffer.

CLASSIFICATION AND TRANSFER FUNCTIONS

In volume rendering we seek to explore the volumetric data using visuals. This exploration process aims to discover and emphasize interesting structures and phenomena embedded in the data, while de-emphasizing or completely culling away occluding structures that are currently not of interest. Clipping planes and

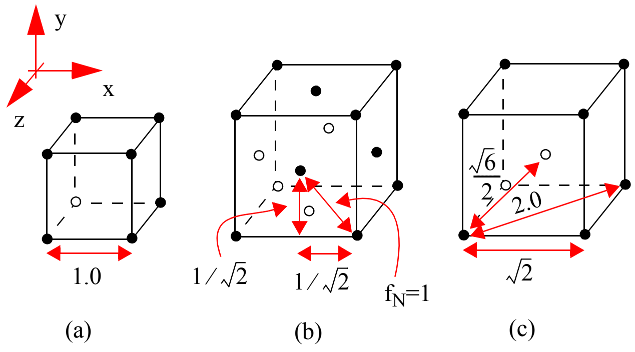


Figure 19: Various grid cells, drawn in relative proportions. We assume that the sampling interval in the CC grid is T=1. (a) Cubic cartesian (CC) for cartesian grids (all other grid cells shown are for grids that can hold the same spherically bandlimited, signal content); (b) Face-centered cubic (FCC); (c) Body-centered (BCC) cell.

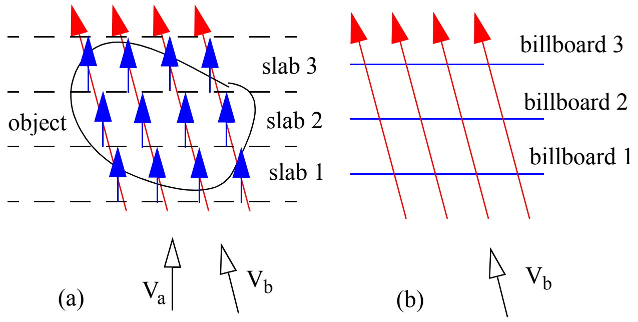


Figure 21: (a) The volume is decomposed into slabs, and each slab is rendered into an image from view direction V_a. The ray integrals for view direction V_b can now be approximated with higher accuracy by combining the appropriate partial ray integrals from view V_a (stored in the slab image). Interpolation is used to obtain partial integrals at non-grid positions. (b) The three billboard images can be composited for any view, such as V_b shown here.

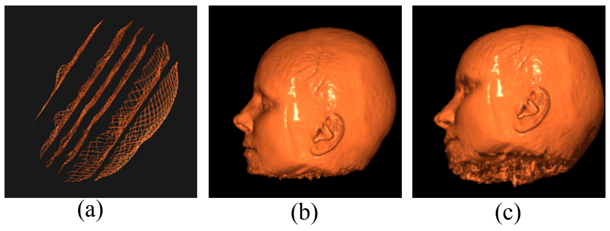


Figure 22: IBR-assisted volume rendering: (a) on-the-fly computed mesh derived from the slab's closest-voxel buffer, (b) head rendered from original view point, (c) head rendered from a view 30° away.

more general clipping primitives [264] provide geometric tools to remove or displace occluding structures in their entirety. On the other hand, transfer functions which map the raw volume density data to color and transparencies, can alter the overall look-and-feel of the dataset in a continuous fashion.

The exploration of a volume via transfer functions constitutes a navigation task, which is performed in a 4D transfer function space, assuming three axes for RGB color and one for transparency (or opacity). It is often easier to specify colors in HSV (Hue, Saturation, Value) color space, since it provides separate mappings for color and brightness. Simple algorithms exist to convert the HSV values into the RGB triples used in the volume rendering [57]. Fig. 23 shows a transfer function editor that also allows the mapping of the other rendering attributes in equation (9).

A generalization of the usual RGB color model has been pursued in *spectral volume rendering* [197], where the light transport occurs within any number of spectral bands. Noordmans [189] employed this concept to enable achromatic, elastic, and inelastic light scattering, which facilitates the rendering of inner structures through semi-transparent, yet solid (i.e., non-fuzzy) exterior structures. Bergner et al. [12] described a spectral renderer that achieves interactive speeds by factoring the illumination term out of the spectral volume rendering integral and using post-illumination for the final lighting (a related technique, in RGB space, using a Fourier series approach was presented by Kaneda et al. [99]). They describe a system which allows designers of a guided visualization to specify a set of lights and materials, whose spectral properties allow users to emphasize, de-emphasize, or merge specific structures by simply varying the intensity of the light sources.

Given the large space of possible settings, choosing an effective transfer function can be a daunting task. It is generally more convenient to gather more information about the data before the exploration via transfer functions begins. The easiest presentation

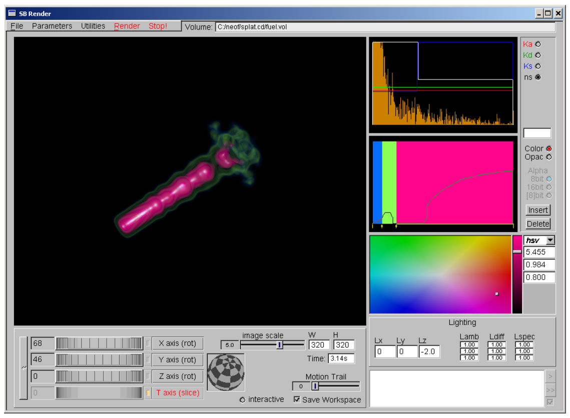


Figure 23: A transfer function editor with a HSV color palette and mapping of densities to various material properties.

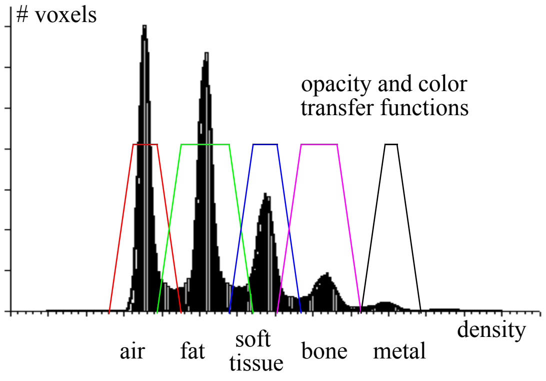


Figure 24: Histogram and a fuzzy classification into different materials.

of support data is in the form of 1D histograms, which are data statistics collected as a function of raw density, or some other quantity. A histogram of density values can be a useful indicator to point out dominant structures with narrow density ranges. A fuzzy classification function [48] can then be employed to assign different colors and opacities to these structures (see Fig. 24). This works well if the data are relatively noise-free, the density ranges of the features are well isolated, and not many distinct materials, such as bone, fat, and skin, are present. In most cases, however, this is not the case. In these settings, it helps to also include the first and second derivative into the histogram-based analysis [109]. The magnitude of the first derivative (the gradient strength) is useful since it peaks at densities where interfaces between different features exist (see Fig. 25). Plotting a histogram of first derivatives over density yields an arc that peaks at the interface density (see Fig. 26). Knowing the densities at which feature boundaries exist narrows down the transfer function exploration task considerably. One may now visualize these structures by assigning different colors and opacities within a narrow interval around these peaks. Levoy [127] showed that a constant width of (thick) surface can be obtained by making the width of the chosen density interval a linear function of the gradient strength (see Fig. 27). Kindlemann and Durkin [109] proposed a technique that uses the first and second derivative to generate feature-sensitive transfer functions automatically. This method provides a segmentation of the data, where the segmentation metric is a histogram of the first and second derivative. Tenginakai and Machiraju [251] extended the arsenal of metrics to higher order moments, and compute from them additional measures, such as kurtosis and skew, in small neighborhoods. These can provide better delineations of features in histogram space. Another proposed analysis method is based on maxima in cumulative Laplacian-weighted density histograms [198].

There are numerous articles (we can only reference a few

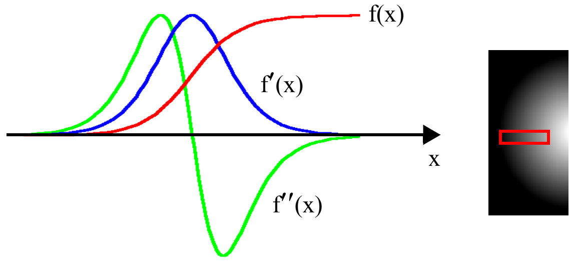


Figure 25: The relationship of densities and their first and second derivatives at a material interface.

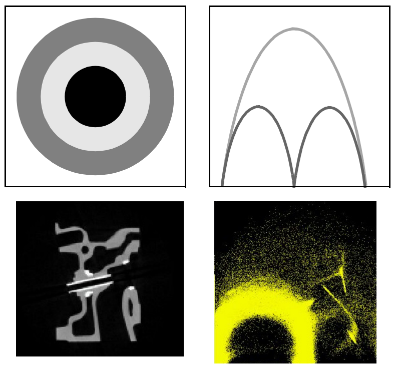


Figure 26: Histograms of (a) first and (b) second derivative strengths over density. In the concentric ring image (top row), the first arc is due to the background-outer ring interface, the second arc is due to the outer-inner ring interface, and the large arc is due to the background-inner ring interface that spans the widest density range. The second row shows the results of the same analysis for the CT engine volume.

here) on the topic of automatic segmentation of images and higher-dimensional datasets, using neural network-type approaches [142], statistical classifiers [222], region growing [117], the watershed algorithm [229], and many others. To that end, Tiede [255] describes an algorithm for rendering the tagged and segmented volumes at high quality. However, despite the great advances that have been made, automated segmentation of images and volumes remains a difficult task and is also in many cases observer and task dependent. In this regard, semi-supervised segmentation algorithms where users guide the segmentation process in an interactive fashion have a competitive edge. There are two examples for such systems: the PAVLOV architecture that implements an interactive region-grow to delineate volumetric features of interest [117], and the dual-domain approach of Kniss et al. [111][112], who embed Kindlemann's algorithm into an interactive segmentation application. Here, users work simultaneously within two domains, i.e., the histogram-coupled transfer function domain and the volume rendering domain, to bring out certain features of interest. To be effective, an interactive (hardware-based) volume renderer is required, and the technique could embed more advanced metrics as well [251].

Another way to analyze the data is to look for topological changes in the iso-contours of the volume, such as a merge of split of two contours (see Fig. 28). These events are called *critical points*. By topologically sorting the critical points as a function of density one can construct a *contour graph*, *contour tree*, or *Hyper Reeb Graph* which yields a roadmap for an exploration of the volume [7][26][60][119][227][250]. One can either use the contour graph to come up with an automatic transfer function (simply position an iso-surface between two nodes), or one can use it to guide users in the volume exploration process. A large number of critical points is potentially generated, especially when the data are noisy..

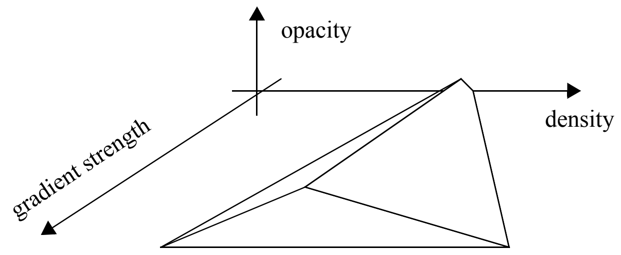


Figure 27: Gradient strength-dependent density range for iso-surface opacities [127].

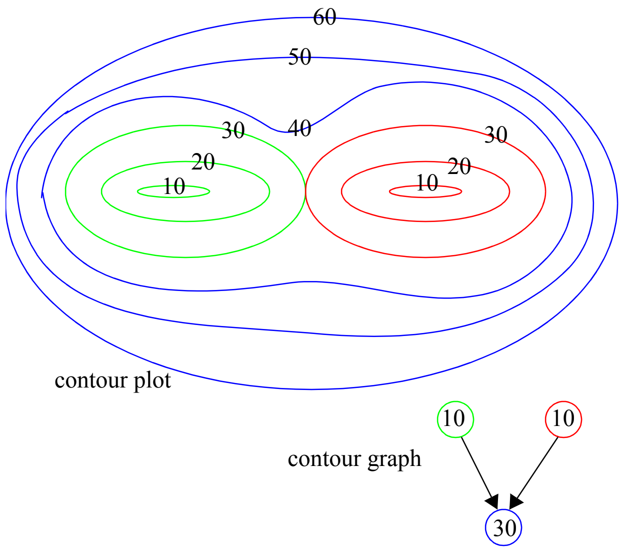


Figure 28: Simple contour graph. The first topological event occurs when the two inner contours are born at an iso-value of 10. The second topological event occurs at the iso-value at which the two inner contours just touch and give way to a single contour at iso-value=30.

There has also been a significant body of work on more specific segmentation and volume analysis processes, which aim to identify, track, and tag particular features of interest, such as vortices, streamlines, and turbulences [9][10][233][234][279]. Once extracted, the features can then be visualized in form of icons, glyphs, geometry, or volumetric objects. These data mining methods are particular attractive for the exploration of very large data sets, where volume exploration with conventional means can become intractable.

All of the methods presented so far base the transfer function selection on a prior analysis of the volume data. Another suggested strategy has been to render a large number of images with arbitrary transfer function settings and present these to the user, who then selects a subset of these for further refinement by application of genetic algorithms. This approach has been taken by the Design Galleries project [150], which is based, in part, on the method published by He et al. [81]. A good sample of all of the existing approaches (interactive trial-and-error, metric-based, contour graph, and design galleries) were squared off in a symposium panel [199]

VOLUMETRIC GLOBAL ILLUMINATION

In the local illumination equation (9), the global distribution of light energy is ignored and shading calculations are performed assuming full visibility of and a direct path to all light sources. While this is useful as a first approximation, the incorporation of global light visibility information (shadows, one instance of global illumination) adds a great deal of intuitive information to the image. This low albedo [98][236] lighting simulation has the ability to cast soft shadows by volume density objects. Generous improvements in realism are achieved by incorporating a high albedo lighting simulation [98][236], which is important in a number of applications (e.g., clouds [152], skin [75], and stone [47]). While some of these used hierarchical and deterministic methods, most of these simulations used stochastic techniques to transport lighting energy among the elements of the scene. We wish to solve the illumination transport equation for the general case of global illumination. The reflected illumination $I(\gamma,\omega)$ in direction ω at any voxel γ can be described as the integral of all incident radiation from directions ω', modulated by the phase function $q(\omega,\omega')$:

$$I(\gamma, \omega) = \iint_{V\Gamma} q(\omega, \omega')I(\gamma, \omega')d\omega' dv \tag{16}$$

where Γ is the set of all directions and V is the set of all voxels v. This means that the illumination at any voxel is dependent upon the illumination at every other voxel. In practice, this integral-equation is solved by finite repeated projection of energy among voxels. This leads to a finite energy transport path, which is generally sufficient for visual fidelity.

In physics, equations of this sort are solved via Monte-Carlo simulations. A large set of rays is cast from the energy sources into the volume and at each voxel a "dice is rolled" to determine how much energy is absorbed and how much energy is scattered and into what direction. After many iterations the simulation is stopped, and a final scattering of the radiosity volume is performed towards an arbitrarily positioned eye point. A practical implementation of this process is volumetric backprojection. Backprojection is usually performed on a voxel-by-voxel basis, since this is the most obvious and direct method of computation. For example, in volumetric ray tracing [236], as illumination is computed for a volume sample, rays are cast toward the light sources, sampling the partial visibility of each. In computing high-albedo scattering illumination, Max [152] used the method of discrete ordinates to transport energy from voxel to voxel. For calculations of volumetric radiosity, voxels are usually regarded as discrete elements in the usual radiosity calculation on pairs of elements, thereby computing on a voxel-by-voxel basis [213][236]. Particle tracing methods for global illumination track paths of scattered light energy through space starting at the light sources [97].

In many cases, the backprojection can be reorganized into a single sweep through the volume, processing slice-by-slice. Because sunlight travels in parallel rays in one direction only, Kajiya and Von Herzen [98] calculated the light intensity of a cloud-like volume one horizontal slice at a time. A similar technique was demonstrated as part of the Heidelberg ray-tracing model [157] in which shadow rays were propagated simultaneously slice-by-slice and in the same general direction as rendering. Dachille et al. [44] described a backprojection approach that scatters the energy in the volume by a multi-pass slice-by-slice sweep at random angles. He also devised a custom hardware architecture for a cache-efficient implementation of this algorithm.

Kniss et al. [113][114] proposed a single-pass algorithm that approximates the scattering of light within a volume by a recursive slice-blurring operation, starting at the light source. The profile of the blurring filter is determined by the user-specified phase function. The method exploits 3D texture mapping hardware in conjunction with a dual image buffer, and runs at interactive frame rates. One buffer, the repeatedly blurred (light) buffer, contains the transported and scattered light energy on its path away from the source, and the other (frame) buffer holds the energy headed for the eye and is attenuated by the densities along the path to the eye. At each path increment energy is transferred from the light buffer to the frame buffer.

RENDERING ON PARALLEL ARCHITECTURES

Much research towards parallel ray-casting has been reported in the literature, primarily due to the simplicity of the algorithm. To avoid volume data redistribution costs, researchers have proposed the distribution of data to processing nodes, where each node processes its own data for all frames or views. Each node generates a partial image with its data, which are then accumulated and composited into the final image [89][144][145][170][194][195].

Researchers have also investigated partitioning screen space into square tiles or contiguous scanlines, to be used as the basic task to be sent or assigned to processing nodes. For better load balancing, the task queue can be ordered in decreasing task size, such that the concurrency gets finer until the queue is exhausted [27]. Load balancing can also be achieved by having nodes steal smaller tasks from other nodes, once they have completed their own tasks [184][271]. Finally, time-out stamps for each node can be added, such that if the node cannot finish its task before the time-out, it takes the remnant of the task, re-partitions it and re-distributes it [40].

A parallel shear-warp implementation on shared-memory architectures has been reported in [121], with decent timing benchmarks. Amin et. al [2] ported the shear-warp algorithm onto a distributed memory architecture, by partitioning in sheared volume space and using an adaptive load balancing. The parallel shear-warp implementation has been improved on distributed memory architectures by dividing the volume data after the shear operation into subvolumes parallel to an intermediate image plane of the shear-warp factorization [219].

Splatting and cell projection methods have also been parallelized using a sort-last paradigm [168]. The community has researched parallel splatting algorithms [133] that do not utilize occlusion-based acceleration. The volume data is distributed in either slices (axis-aligned planes) [54] or blocks [145] to processing nodes. Those are then rendered, in parallel, to partial images which are composited for the final image by the master node. Speed-ups can further be achieved by only passing the non-empty parts of the partial images [54] or by parallelizing the final compositing stage using a screen space partitioning [133]. Hierarchical data structures such as a k-d tree can be applied to facilitate prompt compositing and occlusion culling [145]. Machiraju and Yagel [147] report a parallel implementation of splatting, where the tasks are defined by a subvolumes. Each processor is assigned a subvolume. The images are composited together in depth-sort order, also performed in parallel. This implementation splats all voxels, no matter if they are empty or occluded, while Huang [92] presents a parallel splatting algorithm that takes into account visibility and occlusion, which is considerably more challenging for load-balancing. PVR [230] is a parallel ray casting kernel that exploits image-space, object-space, and time-space parallelism. See also [143] for a tutorial article on two highly scalable, parallel software volume rendering algorithms for unstructured grids.

SPECIAL-PURPOSE RENDERING HARDWARE

The high computational cost of direct volume rendering makes it difficult for sequential implementations and general-purpose computers to deliver the targeted level of performance, although the recent advances in commodity graphics hardware have started to blur these boundaries (as we shall see in the next section). This situation is aggravated by the continuing trend towards higher and higher resolution datasets. For example, to render a dataset of 1024^3 16-bit voxels at 30 frames per second requires 2 GBytes of storage, a memory transfer rate of 60 GBytes per second and approximately 300 billion instructions per second, assuming 10 instructions per voxel per projection.

The same way as the special requirements of traditional computer graphics led to high-performance graphics engines, volume visualization naturally lends itself to special-purpose volume renderers that separate real-time image generation from general-purpose processing. This allows for stand-alone visualization environments that help scientists to interactively view their data on a single user workstation, augmented by a volume rendering accelerator. Furthermore, a volume rendering engine integrated in a graphics workstation is a natural extension of raster based systems into 3D volume visualization.

Several researchers have proposed special-purpose volume rendering architectures [100] (chapter 6) [287] [102] [67] [96] [156] [192] [245] [246] [116] [162] [163].

Most recent research focuses on accelerators for ray-casting of regular datasets. Ray-casting offers room for algorithmic improvements while still allowing for high image quality. More recent architectures [84] include VOGUE, VIRIM, Cube, and VIZARD. The VolumePro board [200] is a commercial implementation of the Cube architecture.

VOGUE [116], a modular add-on accelerator, is estimated to achieve 2.5 frames per second for 256^3 datasets. For each pixel a ray is defined by the host computer and sent to the accelerator. The VOGUE module autonomously processes the complete ray, consisting of evenly spaced resampling locations, and returns the final pixel color of that ray to the host. Several VOGUE modules can be combined to yield higher performance implementations. For example, to achieve 20 projections per second of 512^3 datasets requires 64 boards and a 5.2 GB per second ring-connected cubic network.

VIRIM [72] is a flexible and programmable ray-casting engine. The hardware consists of two separate units, the first being responsible for 3D resampling of the volume using lookup tables to implement different interpolation schemes. The second unit performs the ray-casting through the resampled dataset according to user programmable lighting and viewing parameters. The underlying ray-casting model allows for arbitrary parallel and perspective projections and shadows. An existing hardware implementation for the visualization of 256x256x128 datasets at 10 frames per second requires 16 processing boards.

The Cube project aims at the realization of high-performance volume rendering systems for large datasets and pioneered several hardware architectures. Cube-1, a first generation hardware prototype, was based on a specially interleaved memory organization [103], which has also been used in all subsequent generations of the Cube architecture. This interleaving of the n^3 voxels enables conflict-free access to any ray parallel to a main axis of n voxels. A fully operational printed circuit board (PCB) implementation of Cube-1 is capable of generating orthographic projections of 16^3 datasets from a finite number of predetermined directions in realtime. Cube-2 was a single-chip VLSI implementation of this prototype [8].

To achieve higher performance and to further reduce the critical memory access bottleneck, Cube-3 introduced several new concepts [203][205][206]. A high-speed global communication network aligns and distributes voxels from the memory to several parallel processing units and a circular cross-linked binary tree of voxel combination units composites all samples into the final pixel color. Estimated performance for arbitrary parallel and perspective projections is 30 frames per second for 512^3 datasets. Cube-4 [202][204] has only simple and local interconnections, thereby allowing for easy scalability of performance. Instead of processing individual rays, Cube-4 manipulates a group of rays at a time. As a result, the rendering pipeline is directly connected to the memory. Accumulating compositors replace the binary compositing tree. A pixel-bus collects and aligns the pixel output from the compositors. Cube-4 is easily scalable to very high resolutions of 1024^3 16-bit voxels and true real-time performance implementations of 30 frames per second.

EM-Cube [193] marked the first attempt to design a commercial version of the Cube-4 architecture. Its VLSI architecture features four rendering pipeline and four 64 Mbit SDRAMs to hold the volume data. VolumePro500 was the final design, in form of an ASIC, and was released to market by Mitsubishi Electric in 1999 [200]. VolumePro has hardware for gradient estimation, classification, and per-sample Phong illumination. It is a hardware implementation of the shear-warp algorithm, but with true trilinear interpolation which affords very high quality. The final warp is performed on the PC's graphics card. The VolumePro streams the data through four rendering pipelines, maximizing memory throughput by using a two-level memory block- and bank-skewing mechanism to take advantage of the burst mode of its SDRAMs. No occlusion culling or voxel skipping is performed. Advanced features such as gradient magnitude modulation of opacity and illumination, supersampling, cropping and cut planes are also available. The system renders 500 million interpolated, Phong illuminated, composited samples per second, which is sufficient to render volumes with up to 16 million voxels (e.g., 256^3 volumes) at 30 frames per second.

While the VolumePro uses a brute-force rendering mode in which all rays are cast across the volume, the VIZARD II architecture [162] implements an early ray-termination mechanism. It has been designed to run on a PCI board populated with four FPGAs, a DSP, and SDRAM and SRAM memory. In contrast to the VolumePro, it supports perspective rendering, but uses a table-based gradient vector lookup scheme to compute the gradients at sample positions. The VIZARD II board is anticipated to render a 256^3 dataset at interactive framerates. The VolumePro1000 [282] is the successor of the VolumePro500 and employs a different factorization of the viewing matrix, termed *shear-image* order ray casting, which is a method of ray casting that eliminates shear-warp's intermediate image and final warp step while preserving its memory access efficiency. VolumePro1000 uses empty space skipping and early ray termination, and it can render up to 10^9 samples/s.

The choice of whether one adopts a general-purpose or a special-purpose solution to volume rendering depends upon the circumstances. If maximum flexibility is required, general-purpose appears to be the best way to proceed. However, an important feature of graphics accelerators is that they are integrated into a much larger environment where software can shape the form of input and output data, thereby providing the additional flexibility that is needed. A good example is the relationship between the needs of conventional computer graphics and special-purpose graphics hardware. Nobody would dispute the necessity for polygon graphics acceleration despite its obvious limitations. The exact same argument can be made for special-purpose volume rendering architectures. The line between general-purpose and special-purpose, however, has become somewhat blurred in the past couple of years with to the arrival of advanced, programmable commodity GPUs (Graphics Processing Units). Although these boards do not, and perhaps never will, provide the full flexibility of a CPU, they gain more generality as a general computing machine with every new product cycle. In the following section, we shall discuss the recent revolution in GPUs in light of their impact on interactive volume rendering and processing.

GENERAL-PURPOSE RENDERING HARDWARE

Another opportunity to accelerate volume rendering is to utilize the texture mapping capability of graphics hardware. The first such implementation was devised by Cabral et al. [24] and ran on SGI Reality Engine workstations. There are two ways to go about this. Either one represents the volume as a stack of 2D textures, one texture per volume slice, or as one single 3D texture, which requires more sophisticated hardware. In the former case, three texture stacks are needed, one for each major viewing direction. An image is then rendered by choosing the stack that is most parallel to the image plane, and rendering the textured polygons to the screen in front-to-back or back-to-front order. If the machine has 3D texture capabilities, then one specifies a set of slicing planes parallel to the screen and composites the interpolated textures in depth order. The 3D texturing approach generally provides better images since the slice distance can be chosen arbitrarily small and no popping caused by texture stack switching can occur. While the early approaches did not provide any shading, VanGelder and Kim [65] introduced a fast technique to pre-shade the volume on the fly and then slice and composite a RGB volume to obtain an image with shading effects. Meißner et al. [161] provided a method to enable direct diffuse illumination for semi-transparent volume rendering. However, in this case multiple passes through the rasterization hardware led to a significant loss in rendering performance. Instead, Dachille et al. [43] proposed a one-pass approach that employs 3D texture hardware interpolation together with software shading and classification. Westermann and Ertl [267] introduced a fast multi-pass approach to display shaded isosurfaces. Both Boada et al. [15] and LaMar et al. [122] subdivide the texture space into an octree, which allows them to skip nodes of empty regions and use lower-resolution textures for regions far from the view point or of lower interest.

The emergence of advanced PC graphics hardware has made texture-mapped volume rendering accessible to a much broader community, at less than 2% of the cost of the workstations that were previously required. However, the decisive factor stemming the revolution that currently dominates the field was the manufacturer's (e.g., NVidia, ATI, and 3DLabs) decision to make two of the main graphics pipeline components programmable. These two components are the vertex shaders, the units responsible for the vertex transformations (GLs Modelview matrix), and the fragment shaders, which are the units that take over after the rasterizer (GLs Projection matrix). The first implementation that used these new commodity GPUs for volume rendering was published by Rezk-Salama et al. [209], who used the stack-or-textures approach since 3D texturing was not supported at that time. They overcame the undersampling problems associated with the large inter-slice distance at off-angles by interpolating, on-the-fly, intermediate slices, using the register combiners in the fragment shader compartment. Engel et al. [55] replaced this technique by the use of pre-integrated transfer function tables (see our previous section on transfer functions). The implementation can perform fully-shaded semi-transparent and iso-surface volume rendering at 1-4 frames per second for 256^3 volumes, using an NVidia GeForce3.

To compute the gradients required for shading, one must also load a gradient volume into the texture memory. The interpolation of a gradient volume without subsequent normalization is generally incorrect, but the artifacts are not always visible. Meißner and Guthe [158] use a shading cube texture instead, which eliminates this problem. Even the most recent texture mapping hardware cannot reach the performance of the specialized volume rendering hardware, such as the VolumePro500 and the new VolumePro 1000, at least not when volume are rendered brute-force. Therefore, current research efforts have concentrated on reducing the load for the fragment shaders. Level-of-detail methods have been devised that rasterize lower-resolution texture blocks whenever the local volume detail or projected resolution allow them to do so [74][126]. Li and Kaufman [130][131] proposed an alternative approach that approximates the object by a set of texture boxes, which efficiently clips empty space from the rasterization.

Commodity graphics hardware also found much use for the rendering of irregular grids and in non-photorealistic rendering, as will be discussed shortly. In addition, GPUs have also been extensively used for other non-graphics tasks, such as matrix computations [123], numerical simulations [16][79][132], and computed tomography [283]. These applications view the GPUs as general purpose SIMD machines, with high compute and memory bandwidth, and the latest feature: floating point precision. It should be noted, however, that the limited capacity of the texture memory (currently 128MB to 256MB) and the slow CPU-GPU AGP bus bandwidth currently present the bottlenecks.

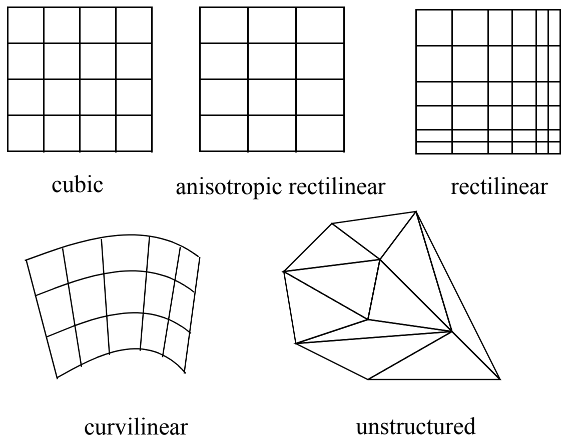


Figure 29: Various grid types in 2D.

IRREGULAR GRIDS

All the algorithms discussed above handle only regular gridded data. Irregular gridded data come in a large variety [240], including curvilinear data or unstructured (scattered) data, where no explicit connectivity is defined between cells (one can even be given a scattered collection of points that can be turned into an irregular grid by interpolation [186][153]). Fig. 29 illlustrates the most prominent grid types.

For rendering purposes, manifold (locally homeomorphic to R^3 grids composed of convex cells are usually necessary. In general, the most convenient grids for rendering purposes are tetrahedral grids and hexahedral grids. One disadvantage of hexahedral grids is that the four points on the side of a cell may not necessarily lie on a plane forcing the rendering algorithm to approximate the cells by convex ones during rendering. Tetrahedral grids have several advantages, including easier interpolation, simple representation (specially for connectivity information because the degree of the connectivity graph is bounded, allowing for compact data structure representation), and the fact that any other grid can be interpolated to a tetrahedral one (with the possible introduction of Steiner points). Among their disadvantages is the fact that the size of the datasets tend to grow as cells are decomposed into tetrahedra and sliver tetrahedra may be generated. In the case of curvilinear grids, an accurate (and naive) decomposition will make the cell complex contain five times as many cells.

As compared to regular grids, operations for irregular grids are more complicated and the effective visualization methods are more sophisticated in all fronts. Shading, interpolation, point location, etc., are all harder (and some even not well defined) for irregular grids. One notable exception is isosurface generation [136] that even in the case of irregular grids is fairly simple to compute given suitable interpolation functions. Slicing operations are also simple [240].

Volume rendering of irregular grids is a hard operation and there are several different approaches to this problem. The simplest and most inefficient is to resample the irregular grid to a regular grid. In order to achieve the necessary accuracy, a high enough sampling rate has to be used what in most cases will make the resulting regular grid volume too large for storage and rendering purposes, not mentioning the time to perform the re-sampling. To

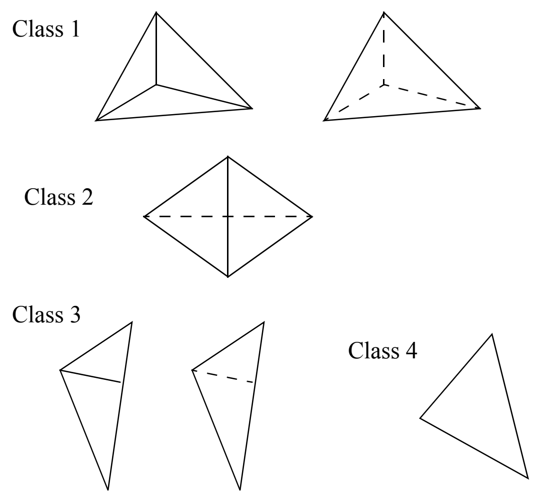


Figure 30: The four classes in tetrahedral projection.

overcome the need to fix the resolution of the regular grid to the smallest cell in the irregular grid, one can sample the data into a detail-adaptive octree whose local height is determined by the local granularity of the grid [126]. The octree decomposition also allows the grid to be rendered within a time-critical rendering framework.

One approach for rendering irregular grids is the use of feed-forward (or projection) methods, where the cells are projected onto the screen one by one, accumulating their contributions incrementally to the final image [275][153][273][228]. The projection algorithm that has gained popularity is the Projected Tetrahedra (PT) algorithm by Shirley and Tuchman [228]. It uses the projected profile of each tetrahedron with respect to the image plane to decompose it into a set of triangles. This gives rise to four classes of projections, which are shown in Fig. 30. The color and opacity values for each triangle vertex are approximated using ray integration through the thickest point of the tetrahedron. The resulting semi-transparent triangles are sorted in depth order and then rendered and composited using polygonal graphics hardware. Stein et al. [243] sort the cells before they are split into tetrahedra, and they utilize 2D texture mapping hardware to accelerate opacity interpolation and provide the correct per-pixel opacity values to avoid artifacts. While their method can only handle linear transfer functions without artifacts, Röttger et al. [211] introduced the concept of pre-integrated volume rendering to allow for arbitrary transfer functions. They create a 3D texture map to provide hardware support in interpolating along the ray between the front and back faces of a tetrahedral cell. In this texture map, two of the three coordinates correspond to values at the cell entry and exit points, with the third coordinate mapping to the distance through the cell. This texture map is then approximated using two-dimensional texture mapping.

Cell projection methods require a sorted list of cells to be passed to the hardware. Starting with Max et al.'s [153] and Williams's [276] works, there has been substantial progress in the development of accurate visibility ordering algorithms [232][38]. A graphics hardware architecture was also proposed, but not yet realized, by King et al. [110], which can both rasterize and sort tetrahedral meshes in hardware.

An alternative technique to visualize irregular grids is by ray-casting [64][260]. Ray-casting methods tend to be more exact than projective techniques since they are able to "stab" or integrate the cells in depth order, even in the presence of cycles. This is generally not possible in cell-by-cell projection methods. Many ray-casting approaches employ the plane sweep paradigm, which is based on processing geometric entities in an order determined by passing a line or a plane over the data. It was pioneered by Giertsen [66] for the use in volume rendering. It is based on a sweep plane that is orthogonal to the viewing plane (e.g., orthogonal to the *y*-axis). Events in the sweep are determined by vertices in the dataset and by values of *y* that correspond to the pixel rows. When the sweep plane passes over a vertex, an "Active Cell List" (ACL) is updated accordingly, so that it stores the set of cells currently intersected by the sweep plane. When the sweep plane reaches a *y*-value that defines the next row of pixels, the current ACL is used to process that row, casting rays, corresponding to the values of *x* that determine the pixels in the row, through a regular grid (hash table) that stores the elements of the ACL. This method has three major advantages: It is unnecessary to store explicitly the connectivity between the cells; it replaces the relatively expensive operation of 3D ray-casting with a simpler 2D regular grid ray-casting; and it exploits coherence of the data between scanlines. Since then, there has been a number of works that employ the sweep paradigm, most using a sweep plane that is parallel to the image plane. Some of these methods are assisted by hardware [285][267], while others are pure-software implementations [22][56][231]. The ZSweep [56] algorithm is very fast and has excellent memory efficiency. It sweeps the plane from front to back, and rasterizes the cell faces as they are encountered by the sweep plane. This keeps the memory footprint low since only the active cell set has be held in memory. Finally, Hong and Kaufman [85][86] proposed a very fast ray-casting technique, that exploits the special topology of cuvilinear grids.

TIME-VARYING AND HIGH-DIMENSIONAL DATA

A significant factor contributing to the growth in the size of computational science datasets is the fact that the time steps in the simulations have become increasingly finer in recent years. There have been significant developments in the rendering of time-varying volumetric datasets. These typically exploit time-coherency for compression and acceleration [3][74][141][225][247][266], but other methods have also been designed that allow general viewing [6][13][76][77][78][106][263] of high-dimensional (n-D) datasets and require a more universal data decomposition.

In n-D viewing, the direct projection from n-D to 2D (for n>3) is challenging. One major issue is that there are an infinite number of orderings to determine occlusion (for n=3 there are just two, the view from the front and the view from the back). In order to simplify the user interface and to eliminate the amount of occlusion explorations a user has to do, Bajaj et. al. [6] performed the n-D volume renderings as an X-ray projection, where ordering is irrelevant. The authors demonstrated that, despite the lack of depth cues, much useful topological information of the n-D space can be revealed in this way. They also presented a scalable interactive user interface that allows the user to change the viewpoint into n-D space by stretching and rotating a system of n axis vectors.

On the other end of the spectrum are algorithms [13] (and the earlier [263]) that first calculate an n-D hyper-surface (a tetrahedral grid in 4D) for a specific iso-value, which can then be interactively sliced along any arbitrary hyperplane to generate an opaque 3D polygonal surface for hardware-accelerated view-dependent display. This approach is quite attractive as long as the iso-value is kept constant. However, if the iso-value is modified, a new iso-tetrahedralization must be generated which can take on the order of tens of minutes [13].

Since 4D datasets can become quite large, a variety of methods to compress 4D volumes were proposed in recent years. Researchers used wavelets [73], DCT-encoding [141], RLE-encoding [3], and images [224][225]. All are lossy to a certain degree, depending on a set tolerance. An alternative compression strategy is the use of more efficient sampling grids, such as the BCC grids. Neophytou and Mueller [182] extended these grids for 4D volume rendering and use a 3D hyperslicer to extract 3D volumes for shaded and semi-transparent volume visualization with occlusion ordering.

Early work on 4D rendering includes a paper by Ke and Panduranga [106] who used the hyperslice approach to provide views onto the on-the-fly computed 4D Mandelbrot set. Another early work is a paper by Rossignac [212], who gave a more theoretical treatment of the options available for the rendering of 4D hypersolids generated, for example, by time-animated or colliding 3D solids. Hanson et al. [76][77][78] wrote a series of papers that use 4D lighting in conjunction with a technique that augments 4D objects with renderable primitives to enable direct 4D renderings. The images they provided in [77] are somewhat reminiscent to objects rendered with motion blur. The 4D isosurface algorithms proposed by Weigle and Banks [263] and Bhaniramka, Wenger, and Crawfis [13] both use a Marching Cubes-type approach and generalize it into n-D.

Methods that focus more on the rendering of the time-variant aspects of 3D datasets have stressed the issue of compression and time-coherence to facilitate interactive rendering speeds. Shen and Johnson [226] used difference encoding of time-variant volumes to reduce storage and rendering time. Westermann [266] used a wavelet decomposition to generate a multi-scale representation of the volume. Shen, Chiang, and Ma [225] proposed the Time-Space Partitioning (TSP) tree, which allows the renderer to cache and reuse partial (node) images of volume portions static over a time interval. It also enables the renderer to use data from sub-volumes at different spatial and temporal resolutions. Anagnostou [3] extended the RLE data encoding of the shear-warp algorithm [120] into 4D, inserting a new run block into the data-structure whenever a change is detected over time. They then composited the rendered run block with partial rays of temporally-unchanged volume portions. Sutton and Hansen [247] expanded the Branch-On-Need Octree (BONO) approach of Wilhelms and Van Gelder [274] to time-variant data to enable fast out-of-core rendering of time-variant isosurfaces. Lum, Ma, and Clyne [141] advocated an algorithm that DCT-compresses time-runs of voxels into single scalars that are stored in a texture map. These texture maps, one per volume slice, are loaded into a texture-map accelerated graphics board. Then, during time-animated rendering, the texture maps are indexed by a time-varying color palette that relates the scalars in the texture map to the current color of the voxel they represent. Although the DCT affords only a lossy compression, their rendering results are quite good and can be produced interactively. Another compression-based algorithm was proposed by Guthe and Straßer [74], who used a lossy MPEG-like approach to encode the time-variant data. The data were then decompressed on-the-fly for display with texture mapping hardware.

MULTI-CHANNEL AND MULTI-MODAL DATA

So far, we have assumed that a voxel had a scalar density value from which other multi-variate properties could be derived, for example, via transfer function lookup. We shall now extend this notion to datasets where the voxel data come originally in form of multi-variate vectors. In the context of this discussion, we shall distinguish between vectors of physical quantities, such as flow and strain, and vectors that store a list of voxel attributes. There is a large body of literature to visualize the former, including line integral convolution [23], spot noise [272], streamlines and streamballs [21], glyphs, texture splats [42], and many more. In this section, we shall focus on the latter scenario, that is, volumes composed of attribute vectors. These can be (i) multi-channel, such as the RGB color volumes obtained by cryosectioning the Visible Human [91] or multi-spectra remote sensing satellite data, or (ii) multi-modal, that is, volumes acquired by scanning an object with multiple modalities, such as MRI, PET, and CT.

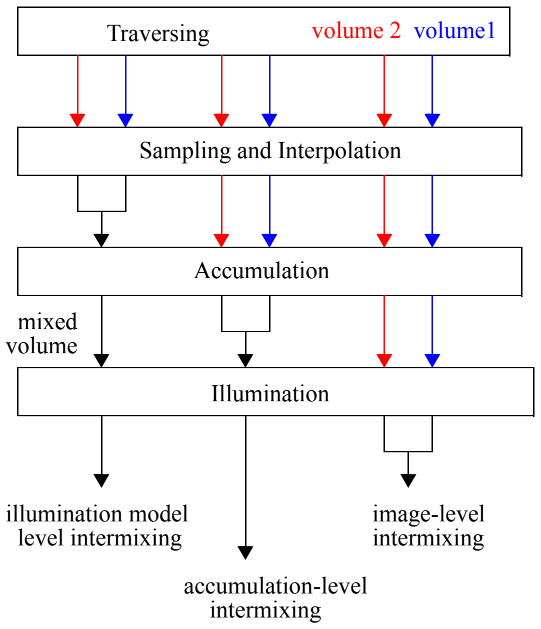

Traversing
volume 2
volume1
Sampling and Interpolation
Accumulation
mixed volume
Illumination
illumination model level intermixing
image-level intermixing
accumulation-level intermixing

Figure 31: Levels in the volume rendering pipeline at which data mixing can occur.

The rendering of multi-modal volumes requires the mixing of the data at some point in the rendering pipeline. There are at least three locations at which this can happen [25]. For the following discussion, let us assume a set of two co-located volumes, but this is not a limitation. The simplest mixing technique is *image-level intermixing*, i.e., to render each volume separately as a scalar dataset and then blend the two images according to some weighting function that possibly includes the z-buffer or opacity channel (see Fig a). This method is attractive since it does not require a modification of the volume renderer, but as Fig. 32a (top) shows, it gives results of limited practical value since depth ordering is not preserved. This can be fixed by intermixing the rendering results at every step along the ray, which gives rise to *accumulation level intermixing*. Here, we assign separate colors and opacities for each volume's ray sample, and then combine these according to some mixing function (see Fig. 32a (bottom)). A third method is *illumination model level intermixing*, where one combines the ray samples before colors and opacities are computed. One could just use a weighted sum of the two densities to look up opacities and colors, or one could have one of the volumes act as an emission volume and the other as an attenuation volume. This would work quite naturally, for example, for the visualization of the emissive metabolic activities in a SPECT volume within the spatial context of a CT attenuation volume. Cai and Sakas [25] demonstrate this method in the scenario of dose planning in radiation therapy, where they visualize an (emissive) radiation beam embedded in an (attenuating) CT volume.

Multi-channel data, such as RGB data obtained by ways of photographing slices of real volumetric objects, have the advantage that there is no longer a need to search for suitable color transfer functions to reproduce the original look of the data. On the other hand, the photographic data do not provide an easy mapping to densities and opacities, which are required to compute normals and other parameters needed to bring out structural object detail in surface-sensitive rendering. One can overcome the perceptional nonlinearities of the RGB space by computing gradients and higher derivatives in the perceptionally uniform color space L*u*v* [51]. In this method, the RGB data are first converted into the L*u*v* space, and the color distance between two voxels is calculated by

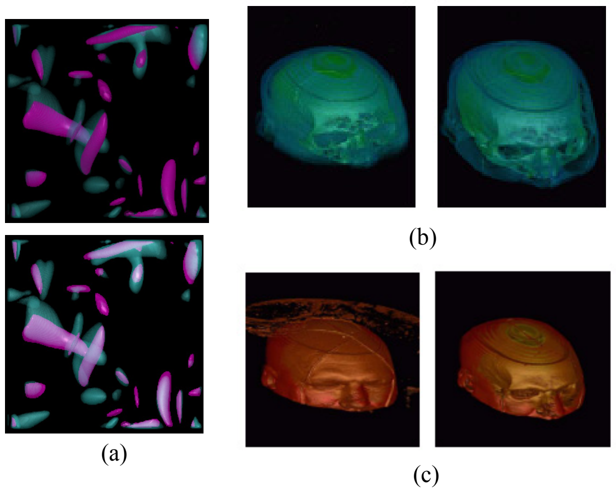


Figure 32: Multi-modal rendering with data intermixing:

(a) One time step of a time-varying volume (magenta) and volume motion-blurred across 10 time steps (blue). (top): image-level intermixing, (bottom): accumulation-level intermixing [182].
(b) Accumulation-level intermixing of the Visible Man's CT and a MRI dataset. Here we assign blue of CT>MRI and green if MRI>CT. (left): gradients specified on CT while MRI is rendered as a point cloud; (right): surfaces rendered with gradient modulation [70].
(c) Accumulation-level intermixing of the Visible Man's CT and a MRI dataset, rendered in inclusive opacity mode, i.e., $\alpha = 1-(1-\alpha_{CT})(1-\alpha_{MRI})$. (left) unweighted product of CT and MRI, (right) more CT than MRI [70].

their Euclidian distance in that color space. A gradient can then be calculated as usual via central differences, but replacing the voxel densities by the color distances. Although one cannot determine the direction of the normal vector with this method, this is not a limiting factor in practice. One can also derive more descriptive quantities, such as tensor gradients, since we are now dealing with vectors and not with densities in the gradient calculation. These can be used for segmentation, texture analysis, and others. Finally, opacities can be computed by using different functions of higher-level gradients to bring out different textural and structural aspects of the data [172].

NON-PHOTOREALISTIC VOLUME RENDERING

Non-photorealistic volume rendering (NPVR) is a relatively recent branch of volume rendering. It employs local image processing during the rendering to produce artistic and illustrative effects, such as feature halos, tone shading, distance color blending, stylized motion blur, boundary enhancements, fading, silhouettes, sketch lines, stipple patterns, and pen+ink drawings [52][53][137][138][95][140][139][244][257]. The overall goal of NPVR is to go beyond the means of photo-realistic volume rendering and produce images that emphasize critical features in the data, such as edges, boundaries, depth, and detail, to provide the user a better appreciation of the structures in the data. This is similar to the goals of medical and other illustrators, as well as related efforts in general computer graphics [277][278][216][217]. Since the set of parameters that can be tuned in NPVR is even larger than for traditional volume rendering, interactive rendering of the NPVR effects is crucial, and indeed a number or researchers have proposed interactive implementations that exploit the latest generations of commodity programmable graphics hardware [139][244].

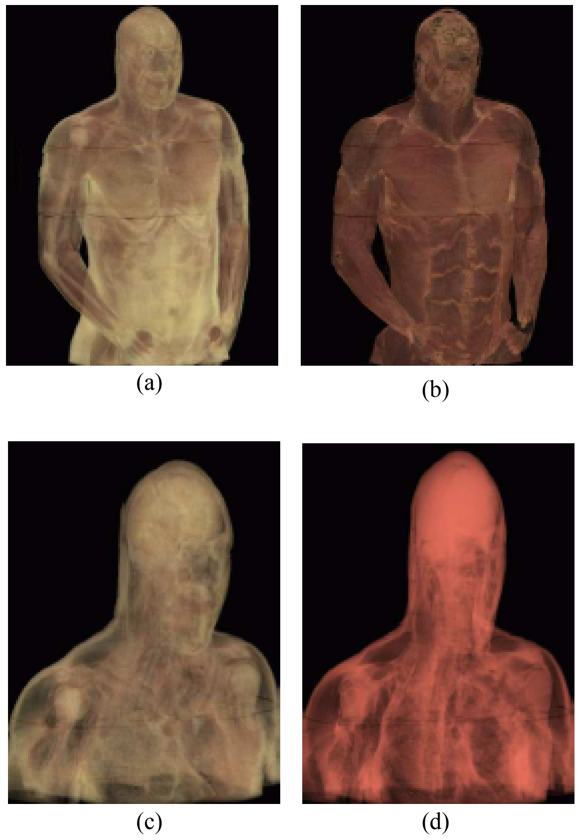


Figure 33: Rendering of multi-channel (photographic) data. (a) The L* component (related to brightness); (b) The u* component (related to the chromatic change in red-green colors); (c) Color difference gradient computed in RGB color space; (b) Gradients computed in L*u*v* space, using the second derivative along the gradient direction to compute opacity. (Images from [70]).

REFERENCES

[1] J. Amanatides, and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," *Eurographics'87*, 1987.

[2] M. Amin, A. Grama, V. Singh, "Fast Volume Rendering Using an Efficient, Scalable Parallel Formulation of the Shear-Warp Algorithm", *Proc. Parallel Rendering Symposium'95*, pp. 7-14, 1995.

[3] K. Anagnostou, T. Atherton, and A. Waterfall, "4D volume rendering with the Shear Warp factorization," *Symp. Volume Visualization and Graphics'00*, pp. 129-137, 2000.

[4] R. Avila, L. Sobierajski, and A. Kaufman, "Towards a Comprehensive Volume Visualization System", *Proc. of IEEE Visualization '92*, pp. 13-20, 1992.

[5] R. Avila, T. He, L. Hong, A. Kaufman, H. Pfister, C. Silva, L. Sobierajski, and S. Wang, "VolVis: A Diversified System for Volume Visualization," *Proc. of IEEE Visualization'94*, 1994.

[6] C. Bajaj, V. Pascucci, and D. Schikore, "The contour spectrum," *Proc. IEEE Visualization'97*, pp. 167-175, 1997.

[7] C. Bajaj, V. Pascucci, G. Rabbiolo, and D. Schikore, "Hypervolume visualization: A challenge in simplicity," *Proc. 1998 Symposium on Volume Visualization'98*, pp. 95-102, 1998.

[8] R. Bakalash, A. Kaufman, R. Pacheco, and H. Pfister, "An Extended Volume Visualization System for Arbitrary parallel

Projection," *Proc. of Eurographics Workshop on Graphics Hardware'92*, 1992.

[9] D. Banks, and B. Singer, "Vortex tubes in turbulent flows; identification, representation reconstruction," *Proc. of IEEE Visualization'94,* pp. 132-139, 1994.

[10] D. Bauer, and R. Peikert, "Vortex Tracking in Scale-Space," *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2002), VisSym'02*, 2002.

[11] M. Bentum, B.B.A. Lichtenbelt, and T. Malzbender, "Frequency analysis of gradient estimators in volume rendering," *IEEE Trans. on Visualization and Computer Graphics'96*, vol. 2, no. 3, pp. 242-254, 1996.

[12] S. Bergner, T. Möller, M. Drew, and G. Finlayson, "Interactive Spectral Volume Rendering", *Proc. of IEEE Visualization'02*, pp. 101-108, 2002.

[13] P. Bhaniramka, R. Wenger, and R. Crawfis, "Isosurfacing in higher dimensions," *Proc. of IEEE Visualization'00*, pp. 267-273, 2000.

[14] F. Blinn, "Light reflection functions for simulation of clouds and dusty surfaces," *Proc. of SIGGRAPH '82,* pp. 21-29, 1982.

[15] I. Boada, I. Navazo, and R. Scopigno, "Multiresolution volume visualization with a texture-based octree," *The Visual Computer,* vol. 17, no. 3, pp. 185-197, 2001.

[16] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "The GPU as numerical simulation engine," *SIGGRAH'03,* 2003.

[17] R. Bracewell, *The Fourier Transform and its Applications-3rd edition*, McGraw-Hill, 1999.

[18] M. Brady, K. Jung, H. Nguyen, and T. Nguyen, "Interactive Volume Navigation," *IEEE Transactions on Visualization and Computer Graphics* vol. 4, no. 3, pp. 243-256, 1998.

[19] M. Brady, K. Jung, H. Nguyen, and T. Nguyen, "Two-Phase Perspective Ray Casting for InteractiveVolume Navigation," *Visualization '97*, pp. 183–189, 1997.

[20] M. Brady, W. Higgins, K. Ramaswamy, and R. Srinivasan, "Interactive Navigation inside 3D radiological Images," *Proc. of BioMedical Visualization'99, Proc. of IEEE Visualization'99*, pp. 33-40 and p. 85 (color plate), 1995.

[21] M. Brill, V. Djatschin, H. Hagen, S. V. Klimenko, and H.-C. Rodrian, "Streamball techniques for flow visualization," *Proc. of IEEE Visualization '94*, pp. 225-231, 1994.

[22] P. Bunyk, A. Kaufman, and C. Silva, "Simple, fast, and robust ray casting of irregular grids," *Scientific Visualization*, pp. 30-36, 1997.

[23] B. Cabral, and L. Leedon, "Imaging vector fields using line integral convolution," *Proc. of SIGGRAPH '93,* pp. 263-272, 1993.

[24] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware", *Symp. on Volume Visualization'94*, pp. 91-98, 1994.

[25] W. Cai, and G. Sakas, "Data intermixing and multi-volume rendering," *Computer Graphics Forum*, *(Eurographics'99)*, vol. 18, no. 3, 1999.

[26] H. Carr, J. Snoeyink, and U. Axen, "Computing contour trees in all dimensions," *Computational Geometry Theory and Applications'02*, vol. 24, no. 2, pp. 75–94, 2003.

[27] J. Challinger, "Scalable Parallel Volume Raycasting for Nonrectilinear Computational Grids," *Proc. of Parallel Rendering Symposium'93*, pp. 81-88, 1993.

[28] B. Chen, A. Kaufman, and Q. Tang, "Image-based rendering of surfaces from volume data," *Workshop on Volume Graphics'01*, pp. 279-295, 2001.

[29] E. Chen, "QuickTime VR-An Image-Based Approach To Virtual Environment Navigation," *Proc. of SIGGRAPH '95*, pp. 29-38, 1995.

[30] E. Chen, and L. Williams, "View Interpolation For Image Synthesis," *Proc. of SIGGRAPH '93*, pp. 279-288, 1993.

[31] M. Chen, A. Kaufman, and R. Yagel (Eds.) *Volume Graphics*, Springer, London, February, 2000.

[32] K. Chidlow, and T. Möller, "Rapid emission volume reconstruction," (to appear) *Volume Graphics Workshop'03*, 2003.

[33] T. Chiueh, T. He, A. Kaufman, and H. Pfister, "Compression Domain Volume Rendering," *Tech. Rep.* 94.01.04, Computer science, SUNY Stony Brook, 1994.

[34] C. Chui, *An Introduction To Wavelets*, Academic Press, 1992.

[35] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno, "Optimal isosurface extraction from irregular volume data," *Symposium on Volume visualization'96*, pp.31-38, 1996.

[36] H. Cline, W. Lorensen, S. Ludke, C. Crawford, and B. Teeter, "Two Algorithms for the Three-Dimensional Reconstruction of Tomograms," *Med. Phys.*, vol. 15, pp. 320-327, 1988.

[37] D. Cohen, and Z. Shefer, "Proximity clouds - an acceleration technique for 3D grid traversal," *The Visual Computer*, vol. 10, no. 11, pp. 27-38, 1994.

[38] J. Comba, J. Klosowski, N. Max, J. Mitchell, C. Silva, and P Williams, "Fast polyhedral cell sorting for interactive rendering of unstructured grids," *Computer Graphics Forum*, vol. 18, no. 3, pp. 369-376, 1999.

[39] J. Conway, and N. Sloane, *Sphere Packings, Lattices and Groups*, Springer Verlag, 2nd edition, 1993.

[40] B. Corrie, and P. Mackerras, "Parallel Volume Rendering and Data Coherence", *Proc. of Parallel Rendering Symposium'93*, pp. 23-26, 1993.

[41] R. Crawfis, "Real-time Slicing of Data Space," *Proc. of IEEE Visualization'96*, pp. 271-277, 1996.

[42] R. Crawfis, and N. Max, "Texture splats for 3D scalar and vector field visualization," *Proc. of IEEE Visualization'93,* pp. 261-266, 1993.

[43] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman, "High-Quality Volume Rendering Using Texture Mapping Hardware," *SIGGRAPH/Eurographics Workshop on Graphics Hardware'98,* 1998.

[44] F. Dachille, K. Mueller, and A. Kaufman, "Volumetric Backprojection," *Volume Visualization Symposium'00*, pp. 109-117, 2000.

[45] J. Danskin, and P. Hanrahan, "Fast algorithms for volume ray tracing," *Workshop on Volume Visualization*, pp. 91-98, 1992.

[46] I. Daubechies, "Ten Lectures on Wavelets," *CBMS-NSF Reg. Conf. Ser. Appl. Math. SIAM.* 1992.

[47] J. Dorsey, A. Edelman, H. Jensen, J. Legakis, and H. Pederson, "Modeling and rendering of weathered stone," *Proc. of SIGGRAPH '99*, pp. 225-234, 1999.

[48] R. Drebin, L. Carpenter and P. Hanrahan, "Volume Rendering," *Proc. of SIGGRAPH'88*, vol. 22, no. 4, pp. 65-74, 1988.

[49] D. Dudgeon, and R. Mersereau, *Multi-dimensional Digital Signal Processing*, Prentice-Hall: Englewood Cliffs, 1984.

[50] S. Dunne, S. Napel, and B. Rutt, "Fast reprojection of volume data," *Proc. of IEEE Visualization in Biomed. Comput,* pp. 11-18, 1990.

[51] D. Ebert, C. Morris, P. Rheingans, and T. Yoo, "Designing effective transfer functions for volume rendering from photographics volumes," *IEEE Trans. on Visualization and Computer 3Graphics*, vol. 8, no. 2, pp. 183-197, 2002.

[52] D. Ebert, and P. Rheingans, "Volume Illustration: Non-Photorealistic Rendering of Volume Models," *IEEE Transactions on Visualization and Computer Graphics,* vol. 7, no. 3, pp. 253-265, 2001.

[53] D. Ebert, and P. Rheingans, "Volume Illustration: Non-Photorealistic Rendering of Volume Models," *Proc. of IEEE Visualization'00,* pp. 195-202, 2000.

[54] T. Elvins, “Volume Rendering on a Distributed Memory Parallel Computer”, *Proc. of Parallel Rendering Symposium’93*, pp. 93-98, 1993.

[55] K. Engel, M. Kraus, and T. Ertl, “High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading,” *Proc. of SIGGRAPH Graphics Hardware Workshop’01*, pp. 9-16, 2001.

[56] R. Farias, J. Mitchell, and C. Silva, “ZSWEEP: An Efficient and Exact Projection Algorithm for Unstructured Volume Rendering,” *ACM/IEEE Volume Visualization and Graphics Symposium*, pp. 91-99, 2000.

[57] J. Foley, A. Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice.* Addison-Wesley, 2nd edition, 1996.

[58] J. Fowler, and R. Yagel, “Lossless compression of volume data,” *Symp. of Volume Visualization’94*, pp. 43-50, 1994.

[59] A. Fujimoto, T. Tanaka, and K. Iwata, “ARTS: accelerated ray-tracing system,” *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16-26, 1986.

[60] I. Fujishiro, Y. Takeshima, T. Azuma, and S. Takahashi, “Volume Data Mining Using 3D Field Topology Analysis,” *IEEE Computer Graphics & Applications*, vol. 20, no. 5, pp. 46-51, 2000.

[61] A. Gaddipati, R. Machiraju, and R. Yagel, "Steering Image generation using Wavelet Based Perceptual Metric," *Computer Graphics Forum (Proc. of Eurographics `97)*, vol. 16, no. 3, pp. 241-251, 1997.

[62] J. Gao, and H. Shen, “Parallel View-Dependent Isosurface Extraction Using Multi-Pass Occlusion Culling,” *ACM/IEEE Symposium on Parallel and Large Data Visualization and Graphics*, 2001.

[63] M. Garland, and P. Heckbert, “Surface simplification using quadric error metrics,” *Proc. of the 24th annual conference on Computer graphics and interactive techniques*, pp.209-216, 1997.

[64] M. Garrity, “Raytracing irregular volume data,” *Computer Graphics*, pp. 35-40, November 1990.

[65] V. Gelder, and K. Kim, “Direct volume rendering via 3D texture mapping hardware,” *Proc. of Vol. Rend. Symp.’96,* pp. 23-30, 1996.

[66] C. Giertsen, “Volume visualization of sparse irregular meshes,” *IEEE Computer Graphics and Applications*, vol. 12, no.2, pp. 40-48, March 1992.

[67] S. Goldwasser, R. Reynolds, T. Bapty, D. Baraff, J. Summers, D. Talton, and E. Walsh, “Physician’s Workstation with Real-Time Performance,” *IEEE Computer Graphics & Applications*, vol. 5, no. 12, pp. 44-57, 1985.

[68] D. Gordon and R. Reynolds, “Image-space shading of 3-dimensional objects,“ *Computer Vision, Graphics, and Image Processing*, 29, pp. 361-376, 1985.

[69] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, “The Lumigraph,” *Proc. of SIGGRAPH ‘96*, pp. 43-54, 1996.

[70] A. Gosh, P. Prabhu, A. Kaufman, and K. Mueller, “Hardware Assisted Multichannel Volume Rendering,” (to be presented) *Computer Graphics International’03*, 2003.

[71] M. Gross, R. Koch, L. Lippert, and A. Dreger, “A new method to approximate the volume rendering equation using wavelet bases and piecewise polynomials,” *Computers & Graphics,* vol. 19 no. 1, pp.47-62, 1995.

[72] T. Guenther, C. Poliwoda, C. Reinhard, J. Hesser, R. Maenner, H. Meinzer, and H. Baur, “VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine,” *Proc. of the 9^{th} Eurographics Hardware Workshop*, pp. 103-108, 1994.

[73] S. Guthe, M. Wand, J. Gonser, and W. Strasser, “Interactive rendering of large volume datasets,” *Proc. of IEEE Visualization’02*, pp. 53-60, 2002.

[74] S. Guthe, and W. Straßer, “Real-time decompression and visualization of animated volume data,” *Proc. of IEEE Visualization’01*, 2001.

[75] P. Hanrahan, and W. Krueger, “Reflection from layered surfaces due to subsurface scattering,” *Computer Graphics (Proc. of SIGGRAPH’93)*, pp. 165-174, 1993.

[76] A. Hanson, and P. Heng, “Four-dimensional views of 3D scalar fields,” *Proc. of IEEE Visualization’92*, pp. 84-91, 1992.

[77] A. Hanson, and P. Heng, “Illuminating the fourth dimension,” *IEEE Computer Graphics & Applications*, vol. 12, no. 4, pp. 54-62, 1992.

[78] A. Hanson, and R. Cross, “Interactive visualization methods for four dimensions,” *Proc. of IEEE Visualization’93*, pp. 196-203, 1993.

[79] M. Harris, G. Coombe, T. Scheuermann, and A. Lastra, “Physically-Based Visual Simulation on Graphics Hardware,” *Proc. of 2002 SIGGRAPH/Eurographics Workshop on Graphics Hardware,* 2002.

[80] H. Hauser, L. Mroz, G. Bischi, and M. Gröller, “Two-level volume rendering-flushing MIP and DVR,” Proc. of IEEE Visualization’00, *pp. 211-218, 2000.*

[81] T. He, L. Hong, A. Kaufman, and H. Pfister, “Generation of Transfer Functions with Stochastic Search Techniques,” *Proc. of IEEE Visualization’96*, pp. 227-234, 1996.

[82] G. Herman, and H. Liu, “Three-dimensional display of human organs from computed tomograms,” *Comput. Graphics Image Process*, vol. 9, pp. 1-21, 1979.

[83] G. Herman and J. Udupa, “Display of three-dimensional discrete surfaces,“ Proceedings SPIE, 283, pp. 90-97, 1981.

[84] J. Hesser, R. Maenner, G. Knittel, W. Strasser, H. Pfister, and A. Kaufman, “Three Architectures for Volume Rendering,” *Computer Graphics Forum*, vol. 14, no. 3, pp. 111-122, 1995.

[85] L. Hong, and A. Kaufman, “Accelerated ray-casting for curvilinear volumes,” *Proc. of IEEE Visualization’98*, pp. 247-253, 1998.

[86] L. Hong, and A. Kaufman, “Fast Projection-Based Ray-Casting Algorithm for Rendering Curvilinear Volumes,” *IEEE Transactions on Visualization and Computer Graphics* vol. 5, no. 4, pp. 322-332, 1999.

[87] L. Hong, S. Muraki, A. Kaufman, D. Bartz, and T. He, “Virtual Voyage: Interactive Navigation in the Human Colon,” *Proc. of ACM SIGGRAPH’97*, pp. 27-34, August 1997.

[88] H. Hoppe, “Progressive Meshes,” *Proc. of SIGGRAPH’96*, pp. 99-108, 1996.

[89] W. Hsu, “Segmented Ray Casting for Data Parallel Volume Rendering”, *Proc. of Parallel Rendering Symposium’93*, pp. 93-98, 1993.

[90] http://graphics.stanford.edu/software/volpack/

[91] http://www.nlm.nih.gov/research/visible/visible_human.html

[92] J. Huang, N. Shareef, R. Crawfis, P. Sadayappan, and K. Mueller, “A Parallel Splatting Algorithm with Occlusion Culling,” *3rd Eurographics Workshop on Parallel Graphics and Visualization*, 2000.

[93] J. Huang, R. Crawfis, and D. Stredney, “Edge preservation in volume rendering using splatting,” *IEEE Volume Vis. ’98,* pp. 63-69, 1998.

[94] I. Ihm, and R. Lee, “On enhancing the speed of splatting with indexing,” *Proc. of IEEE Visualization '95*, pp. 69-76, October 1995.

[95] V. Interrante, “Illustrating surface shape in volume data via principal direction-driven 3D line integral convolution,” *Proc. of SIGGRAPH’97*, pp. 109-116, August, 1997.

[96] D. Jackel, “The Graphics PARCUM System: A 3D Memory Based Computer Architecture for Processing and Display of

Solid Models," *Computer Graphics Forum*, vol. 4, pp. 21-32, 1985.

[97] H. Jensen, and P. Christensen, "Efficient Simulation of Light Transport in Sciences with Participating Media Using Photon Maps," *Proc. of SIGGRAPH'98*, pp. 311-320, 1998.

[98] J. Kajiya, and B. Herzen, "Ray tracing volume densities," *Proc. of SIGGRAPH'84*, pp. 165-174, 1994.

[99] K. Kaneda, Y. Dobashi, K. Yamamoto, and H. Yamashita, "Fast volume rendering with adjustable color maps," *1996 Symposium on Volume Visualization*, pp. 7-14, 1996.

[100] A. Kaufman, *Volume Visualization*, IEEE Computer Society Press Tutorial, Los Alamitos, CA.

[101] A. Kaufman, "Volume visualization," *ACM Computing Surveys,* 28, 1 pp. 165-167, 1996.

[102] A. Kaufman, and R. Bakalash, "CUBE - An Architecture Based on a 3-D Voxel Map," *Theoretical Foundations of Computer Graphics and CAD*, R. A. Earnshaw (ed.), Springer-Verlag, pp. 689-701, 1985.

[103] A. Kaufman, and R. Bakalash, "Memory and Processing Architecture for 3-D Voxel-Based Imagery," *IEEE Computer Graphics & Applications*, vol. 8, no. 6, pp. 10-23, 1988. Also in Japanese, *Nikkei Computer Graphics*, vol. 3, no. 30, pp. 148-160, 1989.

[104] A. Kaufman, D. Cohen, and R. Yagel, "Volume graphics," *IEEE Computer,* vol. 26, no. 7, pp. 51-64, 1993.

[105] T. Kay, and J. Kajiya, "Ray Tracing Complex Scenes," *Proc. of SIGGRAPH'86*, p269-278, 1986.

[106] Y. Ke, and E. Panduranga, "A journey into the fourth dimension," *Proc. of IEEE Visualization'89*, pp. 219-229, 1989.

[107] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions. on Acoustics, Speech, Signal Processing,* vol. 29, no. 6, pp. 1153-1160, 1981.

[108] S. Kilthau, and T. Möller, "Splatting Optimizations", *Technical Report, School of Computing Science, Simon Fraser University*, (SFU-CMPT-04/01-TR2001-02), April 2001.

[109] G. Kindlmann, and J. Durkin, "Semi-automatic generation of transfer functions for direct volume rendering," *Symp. Volume Visualization'98*, pp. 79-86, 1998.

[110] D. King, C. Wittenbrink, and H. Wolters, "An Architecture for Interactive Tetrahedral Volume Rendering," *International Workshop on Volume Rendering'01*, 2001.

[111] J. Kniss, G. Kindlmann, and C. Hansen, "Interactive volume rendering using multidimensional transfer functions and direct manipulation widgets," *Proc. of IEEE Visualization'01*, pp. 255-262, 2001.

[112] J. Kniss, G. Kindlmann, and C. Hansen, "Multidimensional Transfer Functions for Interactive Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 270-285, 2002.

[113] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson, "Interactive Volume Light Transport and Procedural Modeling," *IEEE Transactions on Visualization and Computer Graphics*, 2003.

[114] J. Kniss, S. Premoze, C. Hansen, and D. Ebert, "Interactive Translucent Volume Rendering and Procedural Modeling," *Proc. of IEEE Visualization'02*, pp. 109-116, 2002.

[115] G. Knittel, "The ULTRAVIS system," *Proc. of Volume Visualization and Graphics Symposium'00,* pp. 71-80, 2000.

[116] G. Knittel, and W. Strasser, "A Compact Volume Rendering Accelerator," *Volume Visualization Symposium Proceedings*, pp. 67-74, 1994.

[117] K. Kreeger, and A. Kaufman, "Interactive Volume Segmentation with the PAVLOV Architecture," *Proc. of Parallel Visualization and Graphics Symposium'99*, 1999.

[118] K. Kreeger, I. Bitter, F. Dachille, B. Chen, and A. Kaufman, "Adaptive Perspective Ray Casting," *Volume Visualization Symposium'98*, pp. 55-62. 1998

[119] M. Kreveld, R. Oostrum, C. Bajaj, V. Pascucci, and D. Schikore, "Contour Trees and Small Seed Sets for Isosurface Traversal," *Proc. of the 13th ACM Symposium on Computational Geometry*, pp. 212-220, 1997.

[120] P. Lacroute, and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," *Proc. of SIGGRAPH '94*, pp. 451-458, 1994.

[121] P. Lacroute, "Analysis of a Parallel Volume Rendering System Based on the Shear-Warp Factorization", *IEEE Trans. of Visualization and Computer Graphics*, vol. 2, no. 3, pp. 218-231, 1996.

[122] E. LaMar, B. Hamann, and K. Joy, "Multiresolution Techniques for Interactive Texture-Based Volume Visualization," *Proc. of IEEE Visualization'99,* pp. 355-361, 1999.

[123] E. Larsen, and D. McAllister, "Fast Matrix Multiplies using Graphics Hardware," *Supercomputing'01,* 2001.

[124] D. Laur, and P. Hanrahan, "Hierarchical splatting: A progressive refinement algorithm for volume rendering," *Proc. of SIGGRAPH'91*, pp. 285-288, 1991.

[125] R. Lee, and I. Ihm, "On enhancing the speed of splatting using both object- and image space coherence," *Graphical Models and Image Processing*, vol. 62, no. 4, pp 263-282, 2000.

[126] J. Leven, J. Corso, S. Kumar and J. Cohen, "Interactive visualization of unstructured grids using hierarchical 3D textures," *Proc. of Symposium on Volume Visualization and Graphics'02,* pp 33-40, 2002.

[127] M. Levoy, "Display of surfaces from volume data," *IEEE Comp. Graph. & Appl.*, vol. 8, no. 5, pp. 29-37, 1988.

[128] M. Levoy, "Efficient ray tracing of volume data," *ACM Trans. Comp. Graph.*, vol. 9, no. 3, pp. 245-261, 1990.

[129] M. Levoy, and P. Hanrahan, "Light field rendering," *Proc. of SIGGRAPH '96*, pp. 31-42, 1996.

[130] W. Li, and A. Kaufman, "Accelerating Volume Rendering with Texture Hulls," *IEEE/Siggraph Symposium on Volume Visualization and Graphics 2002 (VolVis'02)*, pp. 115-122, 2002.

[131] W. Li, and A. Kaufman, "Texture Partitioning and Packing for Accelerating Texture-based Volume Rendering," *Graphics Interface'03,* 2003.

[132] W. Li, X. Wei, and A. Kaufman, "Implementing Lattice Boltzmann computation on graphics hardware," (to appear) *The Visual Computer*, 2003.

[133] P. Li, S. Whitman., R. Mendoza, and J. Tsiao, "Prefix - A Parallel Splattting Volume Rendering System for Distributed Visualization," *Proc. of Parallel Rendering Symposium'97*, 1997.

[134] B. Lichtenbelt, R. Crane, and S. Naqvi, *Volume Rendering,* Prentice-Hall, 1998.

[135] Y. Livnat, H. Shen, and C. Johnson, "A near optimal isosurface extraction algorithm for structured and unstructured grids," *IEEE Trans. on Vis. and Comp. Graph.* Vol. 2, no. 1, pp. 73-84, 1996.

[136] E. Lorensen, and H. Cline, "Marching cubes: a high resolution 3D surface construction algorithm," *Proc. of SIGGRAPH'87*, pp. 163-169, 1987.

[137] A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen, "Non-Photorealistic Volume Rendering Using Stippling Techniques," *Proc. of IEEE Visualization'02*, pp. 211-217, 2002.

[138] A. Lu, C. Morris, J. Taylor, D. Ebert, P. Rheingans, C. Hansen, and M. Hartner, "Illustrative Interactive Stipple Rendering," *IEEE Transactions on Visualization and Computer Graphics,* 2003.

[139] E. Lum, and K. Ma, "Hardware-Accelerated Parallel Non-Photorealistic Volume Rendering," *International Symposium*

on Nonphotorealistic Rendering and Animation'02, 2002.

[140] E. Lum, and K. Ma, "Nonphotorealistic Rendering using Watercolor Inspired Textures and Illumination," *Pacific Graphics'01*, 2001

[141] E. Lum, K. Ma, and J. Clyne, "Texture hardware assisted rendering of time-varying volume data," *Proc. of IEEE Visualization'01*, pp. 262-270, 2001.

[142] F. Ma, W. Wang, W. Tsang, Z. Tang, and S. Xia, "Probabilistic Segmentation of Volume Data for Visualization Using SOM-PNN Classifier", *Symposium on Volume Visualization'98*, pp. 71-78, 1998.

[143] K. Ma, and S. Parker, "Massively parallel software rendering for visualizing large-scale datasets," *IEEE Computer Graphics & Applications*, pp. 72-83, 2001.

[144] K. Ma., "Parallel Volume Ray-Casting for Unstructured-Grid Data on Distributed-Memory Architectures", *Proc. of Parallel Rendering Symposium'95*, pp. 23-30, 1995.

[145] K. Ma, and T. Crockett, "A Scalable Parallel Cell-Projection Volume Rendering Algorithm for Three-Dimensional Unstructured Data", *Proc. of Parallel Rendering Symposium'97*, 1997.

[146] R. Machiraju, A. Gaddipati, and R. Yagel, "Detection and enhancement of scale cohererent structures using wavelet transform products," *Proc. of the Technical Conference on Wavelets in Signal and Image Processing V*, SPIE Annual Meeting, pp. 458-469, 1997.

[147] R. Machiraju, and R. Yagel, "Efficient Feed-Forward Volume Rendering Techniques for Vector and Parallel Processors," *SUPERCOMPUTING'93*, pp. 699-708, 1993.

[148] R. Machiraju, and R. Yagel, "Reconstruction Error and Control: A Sampling Theory Approach," *IEEE Transactions on Visualization and Graphics*, vol. 2, no. 3, December 1996.

[149] T. Malzbender, and F. Kitson, "A Fourier Technique for Volume Rendering," *Focus on Scientific Visualization*, pp. 305-316, 1991.

[150] J. Marks, B. Andalman, P.A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Rum, et.al, "Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation," *Proc. of SIGGRAPH'97*, pp. 389-400, 1997.

[151] S. Marschner, and R. Lobb, "An evaluation of reconstruction filters for volume rendering," *Proc. of IEEE Visualization'94,* pp. 100-107, 1994.

[152] N. Max, "Optical models for direct volume rendering," *IEEE Trans. Vis. and Comp. Graph.,* vol. 1, no. 2, pp. 99-108, 1995.

[153] N. Max, P. Hanrahan, and R. Crawfis, "Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions," *Computer Graphics*, vol. 24, no. 5, pp. 27-33, 1990.

[154] L. McMillan, and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. of SIGGRAPH '95*, pp. 39-46, 1995.

[155] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129-147, 1982.

[156] D. Meagher, "Applying Solids Processing Methods to Medical Planning," *Proc. of NCGA'85*, pp. 101-109, 1985.

[157] H. Meinzer, K. Meetz, D. Scheppelmann, U. Engelmann, and H. Baur, "The Heidelberg Raytracing Model," *IEEE Computer Graphics & Applications,* vol. 11, no. 6, pp. 34-43, 1991.

[158] M. Meißner, and S. Guthe, "Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators," *Graphics Interface'02,* 2002.

[159] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis, "A practical comparison of popular volume rendering algorithms," *Symposium on Volume Visualization and Graphics 2000*, pp. 81-90, 2000.

[160] M. Meißner, M. Doggett, U. Kanus, and J. Hirche, "Efficient Space Leaping for Ray casting Architectures", *Proc. of the 2nd Workshop on Volume Graphics*, 2001.

[161] M. Meißner, U. Hoffmann, and W. Straßer, "Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extension," *Proc. of IEEE Visualization'99*, 1999.

[162] M. Meißner, U. Kanus, and W. Straßer, "VIZARD II: A PCI-card for Real-Time Volume Rendering", *Proc. of Siggraph/ Eurographics Workshop on Graphics Hardware'98,* pp. 61-67, 1998.

[163] M. Meißner, U. Kanus, G. Wetekam, J. Hirche, A. Ehlert, W. Straßer, M. Doggett, and R. Proksa "A Reconfigurable Interactive Volume Rendering System," *Proc. of SIGGRAPH/ Eurographics Workshop on Graphics Hardware'02,* 2002.

[164] J. Ming, R. Machiraju, and D. Thompson, "A Novel Approach to Vortex Core Detection," *Proc. of VisSym'02*, pp.217-225, 2002.

[165] D. Mitchell, and A. Netravali, "Reconstruction filters in computer graphics," *Proc. of SIGGRAPH '88*, pp. 221- 228, 1988.

[166] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, "A comparison of normal estimation schemes," *Proc. of IEEE Visualization'97*, pp. 19-26, 1997.

[167] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, "Evaluation and Design of Filters Using a Taylor Series Expansion", *IEEE Transactions on Visualization and Computer Graphics,* vol. 3, no. 2, pp. 184-199, 1997.

[168] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs, "A Sorting Classification of Parallel Rendering," *IEEE Computer Graphics and Applications*, vol. 14, no. 4, pp. 23-32, 1994.

[169] C. Montani, R. Scateni, and R. Scopigno, "Discretized marching cubes," *Proc. of IEEE Visualization '94*, pp. 281-287, 1994.

[170] C. Montani, R. Perego, and R. Scopigno, "Parallel Volume Visualization on a Hypercube Architecture", *Proc. of Volume Visualization Symposium'92*, pp. 9-16, 1992.

[171] B. Mora, J. Jessel, and R. Caubet, "A new object-order ray-casting algorithm," *Proc. of IEEE Visualization'02*, pp. 203-210, 2002.

[172] C. Morris, and D. Ebert, "Direct volume rendering of photographic volumes using multi-dimensional color-based transfer functions," *EUROGRAPHICS IEEE TCVG Symp. on Visualization'02*, pp. 115-124, 2002.

[173] K. Mueller, and R. Crawfis, "Eliminating Popping Artifacts in Sheet Buffer-Based Splatting," *Proc. of IEEE Visualization'98,* pp. 239-245, 1998.

[174] K. Mueller, and R. Yagel, "Fast perspective volume rendering with splatting by using a ray-driven approach," *Proc. of IEEE Visualization'96*, pp. 65-72, 1996.

[175] K. Mueller, and R. Yagel, "Rapid 3D cone-beam reconstruction with the Algebraic Reconstruction Technique (ART) by using texture mapping hardware," vol. 19, no. 12, pp. 1227-1237, *IEEE Transactions on Medical Imaging*, 2000.

[176] K. Mueller, M. Chen, and A. Kaufman (Eds.) *Volume Graphics'01*, Springer: London, 2001.

[177] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "High-quality splatting on rectilinear grids with efficient culling of occluded voxels," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 2, pp. 116-134, 1999.

[178] K. Mueller, N. Shareef, J. Huang, and R. Crawfis, "IBR Assisted Volume Rendering," *Proc. of IEEE Visualization'99*, pp. 5-8, 1999.

[179] K. Mueller, T. Moeller, J.E. Swan, R. Crawfis, N. Shareef, and R. Yagel, "Splatting errors and antialiasing," *IEEE Transactions on Visualization and Computer Graphics*, vol.

4, no. 2, pp. 178-191, 1998.

[180] K. Mueller, T. Möller, and R. Crawfis, "Splatting without the blur," *Proc. of IEEE Visualization'99*, pp. 363-371, 1999.

[181] S. Muraki, "Volume data and wavelet transform," *IEEE Comput. Graphics Appl.,* vol. 13, no. 4, pp. 50-56, 1993.

[182] N. Neophytou, and K. Mueller, "Space-time points: 4D Splatting on effcient grids," *Symposium on Volume Visualization and Graphics'02*, pp. 97-106, 2002.

[183] N. Neophytou, and K. Mueller, "Post-convolved splatting," *Joint Eurographics-IEEE TCVG Symposium on Visualization'03,* 2003.

[184] J. Nieh, and M. Levoy, "Volume Rendering on Scalable Shared- Memory MIMD Architectures," *Proc. of Volume Visualization Symposium*, pp. 17-24, 1992.

[185] G. Nielson, and B. Hamann, "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes," *Proc. of IEEE Visualization '91*, pp. 29-38, 1991.

[186] M. Nielson, "Scattered Data Modeling," *IEEE Computer Graphics and Applications*, vol. 13, no. 1. pp. 60-70, January 1993.

[187] P. Ning, and L. Hesselink, "Fast volume rendering of compressed data," *Proc. of IEEE Visualization'93*, pp. 11-18. 1993.

[188] P. Ning, and L. Hesselink, "Vector quantization for volume rendering," *Proc. of IEEE Visualization'92*, pp. 69-74, 1992.

[189] H. Noordmans, H. Voort, and A. Smeulders, "Spectral Volume Rendering," *IEEE Transactions on Visualization and Computer Graphics* vol. 6, no. 3, pp. 196-207, 2000.

[190] L. Novins, F. X. Sillion, and D. P. Greenberg, "An efficient method for volume rendering using perspective projection," *Computer Graphics*, vol. 24, no. 5, pp. 95-100, 1990.

[191] M. Nulkar, and K. Mueller, "Splatting with shadows," *International Workshop on Volume Graphics '01*, 2001.

[192] T. Ohashi, T. Uchiki, and M. Tokyo, "A Three-Dimensional Shaded Display Method for Voxel-Based Representation," *Proc. of EUROGRAPHICS'85*, pp. 221-232, 1985.

[193] R. Osborne, H. Pfister, H. Lauer, T. Ohkami, N. McKenzie, S. Gibson, and W. Hiatt, "EM-Cube: an architecture for low-cost real-time volume rendering," *Proc. of Eurographics Hardware Rendering Workshop'97*, pp. 131-138, 1997.

[194] S. Parker, M. Parker. Y. Livnat, P. Sloan, C. Hansen, and P. Shirley, "Interactive Ray Tracing for Volume Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 3, pp. 238-250, 1999.

[195] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan, "Interactive ray tracing for isosurface rendering," *Proc. of IEEE Visualization'98*, pp. 233-238, 1998.

[196] V. Pascucci, and K. Cole-McLaughlin, "Efficient computation of the topology of level sets," *Proc. of IEEE Visualization'02*, pp. 187-194, 2002.

[197] S. Peercy, "Linear Color Representations for Full Spectral Rendering," *Computer Graphics*, vol. 27, no.3, pp. 191-198, 1993.

[198] V. Pekar, R. Wiemker, and D. Hempel, "Fast detection of meaningful isosurfaces for volume data visualization," *Proc. of IEEE Visualization'01*, pp. 223-230, 2001.

[199] H. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Avila, K. Martin, R. Machiraju, and J. Lee, "The transfer function bake-off," *Proc. of IEEE Computer Graphics & Applications,* vol. 21, no. 3, pp. 16-22, 2001.

[200] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, "The VolumePro real-time raycasting system," *Proc. of SIGGRAPH'99,* pp. 251-260, 1999.

[201] H. Pfister, M. Zwicker, J. Baar, and M. Gross, "Surfels: surface elements as rendering primitives," *Proc. of SIGGRAPH'00,* pp, 335-342, 2000.

[202] H. Pfister, A. Kaufman, and F. Wessels, "Towards a Scalable Architecture for Real-Time Volume Rendering," *Proc. of 10th Eurographics Workshop on Graphics Hardware'95*, 1995.

[203] H. Pfister, A. Kaufman, and T. Chiueh, "Cube-3: A Real-Time Architecture for High-resolution Volume Visualization," *Symposium of Volume Visualization'94*, pp. 75-82, 1994.

[204] H. Pfister, and A. Kaufman, "Cube-4: A Scalable Architecture for Real-Time Volume Rendering," *Proc. of Volume Visualization Symposium'96*, pp. 47-54, 1996.

[205] H. Pfister, F. Wessels, and A. Kaufman, "Sheared Interpolation and Gradient Estimation for Real-Time Volume Rendering," *Computer & Graphics*, vol. 19, no. 5, pp. 667-677, 1995.

[206] H. Pfister, F. Wessels, and A. Kaufman, "Sheared Interpolation and Gradient Estimation for Real-Time Volume Rendering," *Proc. of 9th Eurographics Workshop on Graphics Hardware'94*, 1994.

[207] T. Porter and T. Duff, "Compositing digital images," *Computer Graphics (Proc. Siggraph'84),* pp. 253-259, 1984.

[208] R. Reynolds, D. Gordon, and L. Chen, "A dynamic screen technique for shaded graphics display of slice-represented objects," *Computer Graphics and Image Processing*, vol. 38, pp. 275-298, 1987.

[209] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage-rasterization" *Proc. of SIGGRAPH/Eurographics Workshop on Graphics Hardware'00*, pp. 109-118, 2000.

[210] S. Roettger, and T. Ertl, "A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids," *Proc. of VolVis '02*, pp. 23-28, 2002.

[211] S. Roettger, M. Kraus, and T. Ertl, "Hardware-Accelerated Volume and Isosurface Rendering Based On Cell-Projection," *Proc. of IEEE Visualization '00*, pp. 109-116, 2000.

[212] J. Rossignac, "Considerations on the interactive rendering of four-dimensional volumes," *Chapel Hill Workshop on Volume Visualization*, pp. 67-76. 1989.

[213] H. Rushmeier, and E. Torrance, "The Zonal Method For Calculating Light Intensities in the Presence of a Participating Medium," *Computer Graphics*, vol. 21, no. 4, pp. 293-302, July 1987.

[214] S. Rusinkiewicz, and M. Levoy, "QSplat: A Multiresolution Point Rendering System for Large Meshes", *Proc. of SIGGRAPH'00*, 2000.

[215] P. Sabella, "A rendering algorithm for visualizing 3D scalar fields," *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4, pp. 51-58, 1988.

[216] M. Salisbury, C. Anderson, D. Lischinski, and D. Salesin, "Scale-dependent reproduction of pen-and-ink illustrations," *Proc, of SIGGRAPH'96*, pp. 461-468, 1996.

[217] M. Salisbury, M. Wong, J. Hughes, and D. Salesin, "Orientable textures for image-based pen-and-ink illustration," *Proc. of SIGGRAPH'97*, pp. 401-406, 1997.

[218] H. Samet, *Application of Spatial Data Structures*. Reading: Addison-Wesley, 1990.

[219] K. Sano, H. Kitajima, H. Kobayashi, and T. Nakamura, "Parallel Processing of the Shear-Warp Factorization with the Binary-Swap Method on a Distributed-Memory Multiprocessor System," *Proc. of Parallel Rendering Symposium'97*, 1997.

[220] W. Schroeder, J. Zarge, and W. Lorensen, "Decimation of triangle meshes," *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2, pp.65-70, 1992.

[221] J. Shade, S. Gortler, Li-Wei He, and R. Szeliski, "Layered

Depth Images," *Proc. of SIGGRAPH '98*, pp. 231-242, 1998.
[222] N. Shareef, D. Wang, and R. Yagel, "Segmentation of Medical Images Using LEGION," *IEEE Transactions on Medical Imaging*, vol. 18, no. 1, 1999.
[223] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill, "Octree-based decimation of marching cubes surfaces," *Proc of IEEE Visual Conf.*, pp.335-342, 1996.
[224] H. Shen, C. Hansen, Y. Livnat, and C. Johnson, "Isosurfacing in span space with utmost efficiency (ISSUE)," *Proc. of IEEE Visualization'96,* pp.287-294., 1996.
[225] H. Shen, L. Chiang, and K. Ma, "A fast volume rendering algorithm for time-varying fields using a time-space partitioning tree," *Proc. of IEEE Visualization'99*, pp. 371-377, 1999.
[226] H. Shen and C. Johnson, "Differential volume rendering: A fast volume visualization technique for flow animation," *Proc. Visualization' 94*, pp. 180-187, 1994.
[227] Y. Shinagawa, and T. Kunii, "Constructing a Reeb Graph Automatically from Cross Sections," *IEEE Computer Graphics and Applications*, vol. 11, no. 6, pp.45-51, 1991.
[228] P. Shirley, and A. Tuchman, "A polygonal approximation to direct scalar volume rendering," *Computer Graphics*, vol. 24, no. 5, pp. 63-70, 1990.
[229] J. Sijbers, P. Scheunders, M. Verhoye, A. Linden, D. Dyck, and E. Raman, "Watershed-based segmentation of 3D MR data for volume quantization," *Magnetic Resonance Imaging*, vol. 15, pp. 679-688, 1997.
[230] C. Silva, A. Kaufman, and C. Pavlakos, "PVR: High-performance Volume Rendering," *IEEE Computational Science and Engineering*, pp. 18-28, Winter 1996
[231] C. Silva, and J. Mitchell, "The lazy sweep ray casting algorithm for rendering irregular grids," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 2, April-June 1997.
[232] C. Silva, J. Mitchell, and P. Williams, "An exact interactive time visibility ordering algorithm for polyhedral cell complexes," *Volume Visualization Symposium'98*, pp. 87-94, October 1998.
[233] D. Silver, and X. Wang, "Tracking Scalar Features in Unstructured Datasets," *Proc. of IEEE Visualization'98,* pp. 79-86, 1998.
[234] D. Silver, and X. Wang, "Tracking and Visualizing Turbulent 3D features," *IEEE Transactions on Visualization and Computer Graphics,* vol. 3, no. 2, 1997.
[235] L. Sobierajski, D. Cohen, A. Kaufman, R. Yagel, D. Acker, "A fast display method for volumetric data," *Visual Computer*, vol 10, no. 2, pp. 116-124, 1993.
[236] L. Sobierajski, and A. Kaufman, "Volumetric raytracing," *Symposium on Volume Visualization'94*, pp. 11-18, 1994.
[237] L. Sobierajski, and R. Avila, "A Hardware Acceleration Method for Volumetric Ray Tracing," *Proc. of IEEE Visualization'95*, pp. 27-35, 1995.
[238] L. Sobierajski, D. Cohen, A. Kaufman, R. Yagel, and D. Acker, "A fast display method for volumetric data," *Visual Comut.,* vol. 10, no. 2, pp. 116-124, 1993.
[239] B. Sohn, C. Bajaj, and V. Siddavanahalli, "Feature based volumetric video compression for interactive playback," *VolVis'02*, pp. 89-96, 2002.
[240] D. Spearey and S. Kennon, "Volume Probes: Interactive data exploration on arbitrary grids," *Computer Graphics*, 25, 5, pp. 5-12. 1990.
[241] M. Sramek, and A. Kaufman, "Fast Ray-tracing of Rectilinear Volume Data Using Distance Transforms," *IEEE Transactions on Visualization and Computer Graphics* vol. 3, no. 6, pp. 236-252, 2000.
[242] B. Stander, and J. Hart. "A Lipschitz method for accelerated volume rendering," *Proc. of the Symposium on Volume Visualization'94,* pp. 107-114, 1994.
[243] C. Stein, B. Becker, and N. Max. "Sorting and hardware assisted rendering for volume visualization," *Symposium on Volume Visualization'94*, pp. 83-90, 1994.
[244] A. Stompel, E. Lum, and K. Ma, "Feature-Enhanced Visualization of Multidimensional, Multivariate Volume Data Using Non-photorealistic Rendering Techniques", *Proc. of Pacific Graphics'02*, pp. 394-402, 2002.
[245] M. Stytz, and O. Frieder, "Computer Systems for Three-Dimensional Diagnostic Imaging: An Examination of the State of the Art," *Critical Reviews in Biomedical Engineering*, pp. 1-46, 1991.
[246] M. Stytz, G. Frieder, and O. Frieder, "Three-Dimensional Medical Imaging: Algorithms and Computer Systems," *ACM Computing Surveys*, pp. 421-499, 1991.
[247] P. Sutton, and C. Hansen, "Isosurface extraction in time-varying fields using a temporal branch-on-need tree (T-BON)," *Proc. of IEEE Visualization'99*, pp. 147-153, 1999.
[248] E. Swan, K. Mueller, T. Moller, N. Shareef, R. Crawfis, and R. Yagel, "An Anti-Aliasing Technique For Splatting," *Proc. of IEEE Visualization'97*, pp. 197-204, 1997.
[249] J. Sweeney, and K. Mueller, "Shear-Warp Deluxe: The Shear-Warp algorithm revisited," *Joint Eurographics - IEEE TCVG Symposium on Visualization'02*, pp. 95-104, 2002.
[250] S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda, "Algorithms for Extracting Correct Critical Points and Constructing Topological Graphs from Discrete Geographical Elevation Data," *Computer Graphics Forum*, vol. 14, no. 3, pp. 181-192, 1995.
[251] S. Tenginakai, J. Lee, and R. Machiraju, "Salient iso-surface detection with model-independent statistical signatures," *Proc. of IEEE Visualization'01*, pp. 231-238, 2001.
[252] T. Theußl, H. Hauser, and M. Gröller, "Mastering Windows: Improving Reconstruction," *Proc. of IEEE Symposium on Volume Visualization'00*, 2000.
[253] T. Theußl, T. Möller, and E. Gröller, "Optimal regular volume sampling," *Proc. of IEEE Visualization'01*, 2001.
[254] P. Thévenaz, T. Blu, and M. Unser, "Interpolation Revisited," *IEEE Transactions on Medical Imaging*, vol. 19, no. 7, pp. 739-758, 2000.
[255] U. Tiede, T. Schiemann, and K. Hoehne, "High quality rendering of attributed volume data," *Proc. of IEEE Visualization'98*, pp. 255-262, 1998.
[256] T. Totsuka, and M. Levoy, "Frequency domain volume rendering," *Proc. of SIGGRAPH'93,* pp. 271-278, 1993.
[257] S. Treavett, and M. Chen, "Pen-and-ink rendering in volume visualization," *Proc. of IEEE Visualization'00*, pp. 203-210, 2000.
[258] H. Tuy, and L. Tuy, "Direct 2D display of 3D objects," *IEEE Computer Graphics & Applications*, vol. 4, no. 10, pp. 29-33, 1984.
[259] J. Udupa, and D. Odhner, "Shell Rendering," *IEEE Computer Graphics and Applications*, vol. 13, no. 6, pp. 58-67, 1993.
[260] S. Uselton, "Volume rendering for computational fluid dynamics: Initial results," *Tech Report RNR-91-026,* Nasa Ames Research Center, 1991.
[261] G. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30-44, 1991.
[262] M. Wan, Q. Tang, A. Kaufman, Z. Liang, and M. Wax, "Volume Rendering Based Interactive Navigation within the Human Colon," *Proc. of IEEE Visualization'99,* pp.397-400, 1999.
[263] C. Weigle, and D. Banks, "Extracting iso-valued features in 4-dimensional datasets," *Proc. of IEEE Visualization'98*, pp. 103-110, 1998.

[264] D. Weiskopf, K. Engel, and T. Ertl. “Volume Clipping via Per-Fragment Operations in Texture-Based Volume Visualization,” *Proc. of IEEE Visualization’02*, pp. 93-100, 2002.

[265] R. Westermann, “A multiresolution framework for volume rendering,” *Proc. of Symp. on Volume Visualization’94*, pp. 51-58, 1994.

[266] R. Westermann, “Compression domain rendering of time-resolved volume data,” *Proc. of IEEE Visualization’95*, pp. 168-174, 1995.

[267] R. Westermann, and T. Ertl, “Efficiently using graphics hardware in volume rendering applications,” *Proc. of SIGGRAPH’99*, pp.169-177, 1999.

[268] L. Westover, “Footprint evaluation for volume rendering,” *Proc. of SIGGRAPH’90*, pp. 367-376, 1990.

[269] L. Westover, “Interactive volume rendering,” *Chapel Hill Volume Visualization Workshop*, pp. 9-16, 1989.

[270] L. Westover, “SPLATTING: A parallel, feed-forward volume rendering algorithm,” *PhD Dissert.* UNC-Chapel Hill, 1991.

[271] S. Whitman, “A Task Adaptive Parallel Graphics Renderer”, *Proc. of Parallel Rendering Symposium’93*, pp. 27-34, 1993.

[272] J. Wijk, “Spot Noise-Texture Synthesis for Data Visualization,” *Proc. of SIGGRAPH’91*, vol. 25, no. 4, pp. 309-318, July 1991.

[273] J. Wilhelms, and A. Gelder, “A Coherent Projection Approach for Direct Volume Rendering,” *Proc. of SIGGRAPH'91*, vol. 25, no. 4, pp. 275-284, 1991.

[274] J. Wilhelms, and A. Gelder, “Octrees for Faster Isosurface Generation,” *ACM Transactions on Graphics*, vol. 11, no. 3, pp. 201-227, 1992.

[275] P. Williams, “Interactive splatting of nonrectilinear volumes,” *Proc. of IEEE Visualization’92,* pp. 37-44, 1992.

[276] P. Williams, “Visibility ordering meshed polyhedra,” *ACM Transaction on Graphics*, vol. 11, no.2, pp.103-125, 1992.

[277] G. Winkenbach, and D. Salesin, “Computer-generated pen-and-ink illustration,” *Proc. of SIGGRAPH’94*, pp. 91-100, 1994.

[278] G. Winkenbach, and D. Salesin, “Rendering parametric surfaces in pen and ink,” *Proc. of SIGGRAPH’96*, pp. 469-476, 1996.

[279] T. Wischgoll, and G. Scheuermann, “Detection and Visualization of Closed Streamlines in Planar Flows,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 2, pp. 165-172. 2001.

[280] C. Wittenbrink, T. Malzbender, and M. Goss, “Opacity-weighted color interpolation for volume sampling,” *Symposium on Volume Visualization’98*, pp. 135-142, 1998.

[281] G. Wolberg, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

[282] Y. Wu, V. Bhatia, H. Lauer, and L. Seiler, “Shear-Image Ray Casting Volume Rendering,” *ACM SIGGRAPH Symposium on Interactive 3D Graphics’03,* 2003.

[283] F. Xu, and K. Mueller, “A Unified Framework for Rapid 3D Computed Tomography on Commodity GPUs,” (to appear) *Proc. of IEEE Medical Imaging Conference’03,* 2003.

[284] R. Yagel, and A. Kaufman, "Template-Based Volume Viewing," *Computer Graphics Forum*, vol. 11, no. 3, pp. 153-167, *Proc. of EUROGRAPHICS'92*, 1992.

[285] R. Yagel, D. Reed, A. Law, P.-W. Shih, and N. Shareef, “Hardware assisted volume rendering of unstructured grids by incremental slicing,” *Volume Visualization Symposium’96*, pp. 55-62. 1996.

[286] R. Yagel, and Z. Shi, "Accelerating Volume Animation by Space-Leaping," *Proc. of IEEE Visualization'93*, pp. 62-69, 1993.

[287] R. Yagel, and A. Kaufman, “The Flipping Cube Architecture,” *Tech. Rep.* 91.07.26, Computer Science, SUNY at Stony Brook, 1991.

[288] B. Yeo, and B. Liu, “Volume rendering of DCT-based compressed 3D scalar data,” *IEEE Trans. Visualization Comput. Graphics,* vol. 1, no. 1, pp. 29-43, 1995.

[289] H. Zhang, D. Manocha, T. Hudson, and K. Hoff, “Visibility culling using hierarchical occlusion maps,” *Proc. of SIGGRAPH’97*, pp. 77-88, 1997.

[290] C. Zhang, and R. Crawfis, “Volumetric Shadows Using Splatting,” *Proc. of IEEE Visualization’02*, pp. 85-92, 2002.

[291] K. Zuiderveld, A. Koning, and M. Viergever, “Acceleration of ray-casting using 3D distance transforms,” *Visualization in Biomedical Computing’92*, pp. 324-335, 1992.

[292] M. Zwicker, H. Pfister, J. Baar, and M. Gross, “Surface splatting,” *Proc. of SIGGRAPH’01*, pp. 371-378, 2001.

[293] M. Zwicker, H. Pfister, J. Baar, and M. Gross, “EWA Volume Splatting,” *Proc. of IEEE Visualization’01*, 2001.