VOLUME VISUALIZATION

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 (1,2,3). Volume data are 3-D 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 2-D slices obtained from Computed Tomography (CT), Magnetic Resonance Imaging (MRI), or confocal microscopy is 3-D reconstructed into a volume model and visualized for diagnosis, study, treatment, or surgery. The same technology is often used with industrial CT for nondestructive inspection of composite materials or mechanical parts. In many computational fields, such as 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* has been expanding, and many traditional geometric computer graphics applications, such as CAD and flight simulation, have exploited the advantages of volume techniques.

Over the years many techniques have been developed to visualize volume data. Because methods for displaying geometric primitives were already well established, most of the early methods involve approximating by geometric primitives a surface contained within the data. When volumetric data are visualized by surface-rendering, a dimension of information is essentially lost. In response to this, volume-rendering techniques were developed that attempt to capture the entire 3-D data in a single 2-D image. Volume rendering conveys 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 and 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) representing the value v of some property of the data at a 3-D location (x, y, z). If the value is simply a 0 or a 1, with a value of 0 indicating background and a value of 1 indicating an object, then the data are called binary data. The data may instead be multivalued, where the value represents 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.

In general, the samples may be taken at purely random locations in space, but in most cases V is isotropic and contains samples taken at regularly spaced intervals along three orthogonal axes. When the spacing between samples along each axis is a constant, but there are different spacing constants for the three axes, V is anisotropic. Because V is defined on a regular grid, a 3-D array (also called a *volume buffer, 3-D raster,* or *cubic frame buffer*) is typically used to store the values, and the element location indicates the position of the sample on the grid. For this reason, V is called 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 (4). In a *rectilinear* grid, the cells are axis-aligned, but grid spacings along the axes are arbitrary. When such a grid has been nonlinearly transformed while preserving the grid topology, the grid becomes *curvilinear.* Usually, the curvilinear grid is called *physical space,* and the rectilinear grid defining the logical organization is called *computational space.* Otherwise, the grid is called *unstructured* or *irregular,* which is a collection of cells whose connectivity has to be explicitly specified. These cells can be of an arbitrary shape, such as tetrahedra, prisms, or hexahedra.

The array V defines only the value of some measured property of the data at discrete locations in space. A function $f(x, y, z)$ is defined over R^3 to describe the value at any continuous location. The function $f(x, y, z) = V(x, y, z)$ if (x, y, z) is a grid-location. Otherwise $f(x, y, z)$ approximates the sample value at a location (x, y, z) by applying some interpolative function to V. The simplest interpolative function is known as *zero-order interpolation,* which is actually just a nearest neighbor function. With this interpolative method, there is a region of constant value around each sample in V. Because the samples in V are regularly spaced, each region has a uniform size and shape. The region of constant value that surrounds each sample is known as a *voxel*. Each voxel is a rectangular cuboid with six faces, twelve edges, and eight corners.

Higher order interpolative functions are also used to define $f(x, y, z)$ between sample points. One common interpolative function is a piecewise function known as *first-order interpolation,* or *trilinear interpolation.* With this interpolative function, it is assumed that the value varies linearly along directions parallel to the major axes. Let the point p lie at location (x_p, y_p, z_p) within the regular hexahedron, known as a *cell,* defined by samples A through H. For simplicity, let the distance between samples in all three directions be 1, with sample A at (0, 0, 0) with a value of v_A, and sample H at (1, 1, 1) with a value of v_H. Then the value v_P, according to trilinear interpolation, is given by

$$\begin{aligned} v_P = {} & v_A(1-x_p)(1-y_p)(1-z_p) + v_E(1-x_p)(1-y_p)z_p \\ & + v_B x_p(1-y_p)(1-z_p) + v_F x_p(1-y_p)z_p \\ & + v_C(1-x_p)y_p(1-z_p) + v_G(1-x_p)y_p z_p \\ & + v_D x_p y_p(1-z_p) + v_H x_p y_p z_p \end{aligned} \tag{1}$$

In general, A is at some location (x_A, y_A, z_A), and H is at (x_H, y_H, z_H). In this case, x_p in Eq. (1) is replaced by $(x_p - x_A)/(x_H - x_A)$, with similar substitutions for y_p and z_p.

SURFACE-RENDERING TECHNIQUE

Several surface-rendering techniques have been developed which approximate, using geometric primitives, a surface contained within volumetric data, which is then rendered by conventional graphics accelerator hardware. A surface is defined by applying a binary segmentation function $S(v)$ to the volumetric data. $S(v)$ equals 1 if the value v is considered part of the object, and equals 0 if the value v is part of the background. Then the surface is the region where $S(v)$ changes from 0 to 1. If a zero-order interpolative function is used, then the surface is simply the set of faces shared by voxels with differing values of $S(v)$. If a higher order interpolative function is used, then the surface passes between sample points according to the interpolative function.

For zero-order interpolative functions, the natural choice for a geometric primitive is the rectangle, because the surface is a set of faces of 3-D rectangles of cuboids, and each face is a rectangle. An early algorithm for displaying human organs from computed tomograms (5) uses the square as the geometric primitive. To simplify the projective calculation and decrease rendering times, the assumption is made that the sample spacing in all three directions is the same. Then a software Z-buffer algorithm is used to project the shaded squares onto the image plane to create the final image.

With continuous interpolative functions, a surface, known as an *isovalued surface* or an *isosurface,* is defined by a single value. Several methods for extracting and rendering isosurfaces have been developed. The Marching Cubes algorithm (6) was developed to approximate an isovalued surface with a triangle mesh. The algorithm breaks down the ways in which a surface can pass through a cell into 256 cases, reduced by symmetry to only 15 topologies. For each of these 15 cases, a generic set of tiny triangles representing the surface is stored in a look-up table. Each cell, through which a surface passes, maps onto one of the 15 cases, and the actual triangle vertex locations are determined by linear interpolation on the cell vertices. A normal value is estimated for each triangle vertex, and standard graphics hardware is utilized to project the triangles, resulting in a smooth, shaded image of the isovalued surface.

When rendering a sufficiently large data set with the Marching Cubes algorithm, millions of triangles are generated. Many of them map to a single pixel when projected onto the image plane. This has led to the development of surface-rendering algorithms that instead use 3-D points as the geometric primitive. One such algorithm is Dividing Cubes (7), which subdivides each cell through which a surface passes into subcells. The number of divisions is selected so that the subcells project onto a single pixel on the image plane. Another algorithm (8), instead of subdividing, uses only one 3-D point per visible surface cell, projecting that point on up to three pixels of the image plane to ensure coverage in the image.

VOLUME-RENDERING TECHNIQUES

Representing a surface contained within a volumetric data set by geometric primitives is usrful in many applications. There are, however, several main drawbacks to this approach. First, geometric primitives approximate only surfaces contained within the original data. Adequate approximations require an excessive amount of geometric primitives. Therefore, a trade-off must be made between accuracy and space requirements. Second, because only a surface representation is used, much of the information contained within the data is lost. Also, amorphous phenomena, such as clouds, fog, and fire are adequately represented by surfaces, and therefore must have a volumetric representation, and must be displayed by volume-rendering techniques.

Volume rendering is the process of creating a 2-D image directly from 3-D volumetric data. Although several of the methods described later render surfaces contained within volumetric data, these methods operate on the actual data samples without the intermediate geometric primitive representations. Volume rendering is achieved with an object-order, an image-order, or a domain-based technique. Object-order volume-rendering techniques use a forward mapping scheme where the volume data are 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 are first transformed into an alternative domain, such as compression, frequency, and wavelet, and then a projection is generated directly from that domain.

Object-Order Techniques

Object-order techniques involve mapping the data samples onto the image plane. One way to accomplish a projection of a surface contained within the volume is to loop through the data samples and project each sample which is part of the object onto the image plane. If an image is produced by projecting all voxels with a nonzero value onto the image plane in an arbitrary order, a correct image is guaranteed. If two voxels project to the same pixel on the image plane, the voxel projected later prevails, even if it is farther from the image plane than the earlier projected voxel. This problem can be solved by traversing the data samples in a *back-to-front* order. For this algorithm, the strict definition of back-to-front is relaxed to require that, if two voxels project to the same pixel on the image plane, the first processed voxel must be farther away from the image plane than the second. This is accomplished by traversing the data plane-by-plane and row-by-row inside each plane. For arbitrary orientations of the data relative to the image plane, some axes are traversed in an increasing order, and others are considered in a decreasing order. Although the relative orientations of the data and the image plane specify whether each axis should be traversed in an increasing or decreasing manner, the ordering of the axes in the traversal is arbitrary.

An alternative to back-to-front projection is a *front-to-back* method in which the voxels are traversed in the order of increasing distance from the image plane. Although a back-to-front method is easier to implement, a front-to-back method has the advantage that once a voxel is projected onto a pixel, other voxels which project to the same pixel are ignored, because they would be hidden by the first voxel. Another advantage of front-to-back projection methods is that, if the axis most parallel to the viewing direction is chosen as the outermost loop of the data traversal, meaningful partial image results are displayed to the user. This allows the user to interact better with the data and possibly terminate the image generation if, for example, an incorrect parameter was selected.

For each voxel, its distance to the image plane could be stored in the pixel to which it maps along with the voxel value. At the end of a data traversal, a 2-D array of depth values, called a Z-buffer, is created, where the value at each pixel in the Z-buffer is the distance to the closest nonempty voxel. Then a 2-D, discrete, postshading technique is applied to the image, resulting in an approximated shaded image. The simplest, yet inaccurate, 2-D, discrete, shading method is known as *depth-only shading* (9), where only the Z-buffer is used and the intensity value stored in each pixel of the output image is inversely proportional to the depth of the corresponding pixel.

A more accurately shaded image is obtained by using a 2-D gradient shading (10) which takes into account the object surface orientation and the distance from the light at each pixel to produce a shaded image. This method evaluates the 2-D gradient at each (x, y) pixel location in the 2-D image with backward difference $D(x, y) - D(x - 1, y)$, a forward difference $D(x + 1, y) - D(x, y)$, or a central difference $\frac{1}{2}[D(x + 1, y) - D(x - 1, y)]$, where $z = D(x, y)$ is the depth stored in the Z-buffer associated with pixel (x, y). Similar equations are used for approximating $\delta z/\delta y$. In general, the central difference is a better approximation of the derivative, but along object edges where, for example, pixels (x, y) and $(x + 1, y)$ belong to two different objects, a backward difference provides a better approximation. A context-sensitive normal estimation method (11) was also developed to provide more accurate normal estimations by detecting image discontinuities.

The previous rendering methods consider primarily binary data samples where a value of 1 indicates the object and a value of 0 indicates the background. Many forms of data acquisition produce data samples with 8, 12, or even more bits of data per sample. If these data samples represent the values at some sample points and the values vary according to some convolution applied to the data samples which can reconstruct the original 3-D signal then a scalar field, which approximates the original 3-D signal, is defined.

In forward mapping algorithms, the original signal is reconstructed by spreading the value at a data sample into space. Westover describes a splatting algorithm (12) for approximating smooth object-ordered volume rendering, in which the value of the data samples represents a density. Each data sample $s = [x_s, y_s, z_s, \rho(s)]$, $s \in V$, has a function C defining its contribution to every point (x, y, z) in the space:

$$C_s(x, y, z) = h_v(x - x_s, y - y_s, z - z_s)\rho(s) \tag{2}$$

where h_v is the volume reconstruction kernel and $\rho(s)$ is the density of sample s located at (x_s, y_s, z_s). Then the contribution of a sample s to an image plane pixel (x, y) is computed by integration:

$$C_s(x, y) = \rho(s) \int_{-\infty}^{\infty} h_v(x - x_s, y - y_s, u)\, du \tag{3}$$

where the u coordinate axis is parellel to the view ray. Because this integral is independent of the sample density and depends only on its (x, y) projected location, a footprint function F is defined as follows:

$$F(x, y) = \int_{-\infty}^{\infty} h_v(x, y, u)\, du \tag{4}$$

where (x, y) is the displacement of an image sample from the center of the sample image plane projection. Then the weight w at each pixel is expressed as

$$w(x, y)_s = F(x - x_s, y - y_s) \tag{5}$$

where (x, y) is the pixel location and (x_s, y_s) is the image plane location of the sample s.

A footprint table is generated by evaluating the integral in Eq. (4) on a grid with a resolution much higher than the image plane resolution. A footprint table for a data sample s is centered on the projected image plane location of s and sampled to determine the weight of the contribution of s to each pixel on the image plane. Then multiplying this weight by $\rho(s)$ gives the contribution of s to each pixel.

Computing a footprint table is difficult because of the integration required. Discrete integration methods are used to approximate the continuous integral, and only one generic footprint table is built for the kernel. For each view, a view-transformed footprint table is created from the generic footprint table in three steps. First, the image plane extent of the reconstruction kernel projection, which is a circle or an ellipse, is determined. Next a mapping is computed between this extent and the extent surrounding the generic footprint table. Finally, the value for each entry in the view-transformed footprint table is determined by mapping the location of the entry to the generic footprint table and sampling.

There are several modifiable parameters in this algorithm which greatly affect image quality. First, the size of the footprint table can be varied. Small footprint tables produce blocky images, whereas large footprint tables smooth out details and require more space. Second, different sampling methods can be used when generating the view-transformed footprint table from the generic footprint table. Using a nearest neighbor approach is fast, but produces aliasing artifacts. On the other hand, bilinear interpolation produces smoother images at the expense of longer rendering times. The third parameter which can be modified is the reconstruction kernel itself. For example, the choice of a cone function, Gaussian function, sync function, or bilinear function affects the final image.

Drebin, Carpenter, and Hanrahan (13) developed a technique for rendering volumes that contain mixtures of materials, such as CT data containing bone, muscle, and flesh. In this method, it is assumed that the scalar field was sampled above the Nyquist frequency or a low-pass filter was used to remove high frequencies before sampling. The volume contains either several scalar fields or one scalar field representing the composition of several materials. If the latter is the case, it is assumed that material is differentiated by the scalar value at each point or by additional information about the composition of each volume element.

The first step in this rendering algorithm is to create new scalar fields from the input data, known as material percentage volumes, each of which is a scalar field representing only one material. Then color and opacity are associated with each material, and composite color and opacity are obtained by linearly combining the color and opacity for each percentage volume. A matte volume, that is, a scalar field on the volume with values ranging between 0 and 1, is used to slice the volume or perform other spatial set operations. Actual rendering of the final composite scalar field is obtained by transforming the volume, so that one axis is perpendicular to the image plane. Then the data are projected plane-by-plane in a back-to-front manner and composited to form the final image.

Image-Order Techniques

Image-order volume rendering techniques are fundamentally different from object-order rendering techniques. Instead of determining how a data sample affects the pixels on the image plane, in an image-order technique, the data samples which contribute to it are determined for each pixel on the image plane.

One of the first image-order, volume-rendering techniques, called *binary ray casting* (14), was developed to generate images of surfaces contained within binary volumetric data without the explicit need for boundary detection and hidden-surface removal. For each pixel on the image plane, a ray is cast from that pixel to determine if it intersects the surface contained within the data. For parallel projections, all rays are parallel to the view direction, whereas, for perspective projections, rays are cast from the eye point according to the view direction and the field of view. If an intersection occurs, the intersection point is shaded, and the resuliting color is placed in the pixel. To determine the first intersection along the ray, a stepping technique is used where the value is determined at regular intervals along the ray until the object is intersected. Data samples with a value of 0 are considered as the background whereas those with a nonzero value are considered part of the object. A zero-order interpolative technique is used, so that the value at a location along the ray is 0 if that location is not in any voxel of the data; otherwise it is the value of the closest data sample.

The previous algorithm deals with the display of surfaces within binary data. A more general algorithm is used to generate surface and composite projections of multivalued data. Instead of traversing a continuous ray and determining the closest data sample for each step with a zero-order interpolative function, a discrete representation of the ray is traversed. This discrete ray is generated by a 3-D Bresenham-like algorithm or a 3-D line scan-conversion (voxelization) algorithm (1,15) (see below). As in the previous algorithms, the data samples, which contribute to each pixel in the image plane must be determined. This is done by casting a ray from each pixel in the direction of the viewing ray. This ray is discretized (voxelized), and the contribution from each voxel along the path is considered when producing the final pixel value. This technique is called *discrete ray casting* (16).

To generate a 3-D discrete ray using a voxelization algorithm, the 3-D discrete topology of 3-D paths has to be understood. There are three types of connected paths: 6-connected, 18-connected, and 26-connected, based on the three adjacency relationships between consecutive voxels along the path. Assuming that a voxel is represented as a box centered at the grid point, two voxels are said to be 6-connected if they share a face; they are 18-connected if they share a face or an edge; and they are 26-connected if they share a face, an edge, or a vertex. A 6-connected path is a sequence of voxels, where, for every consecutive pair of voxels, the two voxels are 6-connected. Similar definitions exist for 18- and 26-connected paths. In discrete ray casting, a ray is discretized into a 6-, 18-, or 26-connected path, and only the voxels along this path are considered when determining the final pixel value. Almost twice as many voxels are contained in 6-connected paths as in 26-connected paths, so that an image created with 26-connected paths requires less computation, but a 26-connected path may miss an intersection that would be detected with a 6-connected path.

To produce a shaded image, the distance to the closest surface intersection is stored at each pixel in the image, and then this image is passed to a 2-D discrete shader, such as those described previously. However, better results are obtained by 3-D discrete shading at the intersection point. One such method, known as *normal-based contextual shading* (17) is employed to estimate the normal for zero-order interpolation. The normal for a face of a voxel on the surface of the object is determined by examining the orientation of that face and the orientation of the four faces on the surface that are edge-connected to that face. Because a face of a voxel has only six possible orientations, the error in the approximated normal can be significant. More accurate results are obtained by a technique known as *gray level shading* (7,18). If the intersection occurs at location (x, y, z) in the data, then the gray-level gradient at that location is approximated by (G_x, G_y, G_z), where G_x is the central difference:

$$G_x = \frac{f(x+1, y, z) - f(x-1, y, z)}{2D_x} \tag{6}$$

with similar equations for G_y and G_z. D_x, D_y, and D_z are the distances between neighboring samples in the x, y, and z directions, respectively. The gradient vector is used as a normal vector for shading calculation, and the intensity value obtained from shading is stored in the image. A normal estimation is performed at every sample point, and this information, along with the light direction and the distance from the pixel, is used to shade the sample point.

Actually, stopping at the first opaque voxel and shading there is only one of many operations which can be performed on the voxels along a discrete path or continuous ray. Instead, the whole ray could be traversed, storing in the pixel the maximum value encountered along the ray, which is capable of revealing some internal parts of the data. Another option is to store the sum (simulating X rays) or the average of all values along the ray. More complex techniques, described later involve defining an opacity and color for each scalar value, and then accumulating intensity along the ray according to some compositing function to reveal 3-D structural information and 3-D internal features.

One disadvantage of zero-order interpolation are the aliasing effects in the image. Higher order interpolation functions are used to create a more accurate image but generally at the cost of algorithmic complexity and computation time. The algorithms described later use higher order interpolation functions.

When creating a composite projection of a data set, there are two important parameters, the color at a sample point and the opacity at that location. An image-order, volume-rendering algorithm developed by Levoy (19) states that, given an array of data samples V, two new arrays V_c and V_α, which define the color and opacity at each grid location, can be generated by preprocessing techniques. Then the interpolation functions $f(x, y, z)$, $f_c(x, y, z)$, and $f_\alpha(x, y, z)$, which specify the sample value, color, and opacity at any location in R^3, are defined, f_c and f_α are often called *transfer functions.*

Generating the array V_c of color values involves a shading operation, such as gray-level shading, at every data sample in V. The gradient vector at any location is computed by partially differentiating the interpolation function with respect to x, y, and z to get each component of the gradient. If the interpolation function is not first derivative continuous, aliasing artifacts occur in the image because of the discontinuous normal vector. A smoother set of gradient vectors is obtained by a central differencing method similar to the one described earlier in this section. Calculating the array V_α is essentially a surface classification operation and requires a mapping from $V(x, y, z)$ to $V_\alpha(x, y, z)$. For example, when an isosurface at some constant value v with an opacity α_v ought to be viewed, $V_\alpha(x, y, z)$ is simply assigned to α_v if $V(x, y, z)$ is v, otherwise $V_\alpha(x, y, z) = 0$. This produces aliasing artifacts, which are reduced by setting $V_\alpha(x, y, z)$ close to α_v if $V(x, y, z)$ is close to v.

Once the $V_c(x, y, z)$ and $V_\alpha(x, y, z)$ arrays are determined, rays are cast from the pixels through these two arrays, sampling at evenly spaced locations. To determine the value at a location, the trilinear interpolative functions f_c and f_α are used. Once these sample points along the ray are computed, a fully opaque background is added in, and then the values in back-to-front order are composited to produce a single color that is placed in the pixel.

To simulate light coming from translucent objects, volumetric data with data samples representing density values are considered a field of density emitters (20). A density emitter is a tiny particle that emits and scatters light. The amount of density emitters in any small region within the volume is proportional to the scalar value in that region. These density emitters are used to correctly model the occlusion of deeper parts of the volume by closer parts, but both shadowing and color variation are ignored because of differences in scattering at different wavelengths. The intensity I of light for a given pixel is calculated according to

$$I = \int_{t_1}^{t_2} e^{-\tau \int_{t_1}^{t} \rho^{\gamma}(\lambda)\, d\lambda} \rho^{\gamma}(t)\, dt \tag{7}$$

In this equation, the ray is traversed from t_1 to t_2, accumulating at each location t the density $\rho^\gamma(t)$ at that location attenuated by the probability

$$e^{-\tau \int_{t_1}^{t} \rho^{\gamma}(\lambda)\, d\lambda}$$

that this light is scattered before reaching the eye. The parameter τ controls the attenuation. Higher values of τ specify a medium which darkens more rapidly. The parameter γ is also modifiable and controls the spread of density values. Low γ values produce a diffuse cloud appearance, whereas higher γ values highlight dense portions of the data.

Krueger (21) showed that the various volume-rendering models can be described as special cases of an underlying transport theory model of the transfer of particles in nonhomogeneous media. The basic idea is that a beam of "virtual" particles is sent through the volume, and the user selects the particle properties and the laws of interaction between the particles and the data. Then the image plane contains the "scattered" virtual particles, and information about the data is obtained from the scattering pattern. For example, if the virtual particles are chosen to have the properties of photons and the laws of interaction are governed by optical laws, then this model becomes a generalized ray tracer. Other virtual particles and interactive laws are used, for example, to identify periodicities and similar hidden symmetries of the data. Using Krueger's transport theory model, the intensity of light I at a pixel is described as follows as a path integral along the view ray:

$$I = \int_{p_{near}}^{p_{far}} Q(p) e^{-\int_{p_{near}}^{p} \sigma_a(p') + \sigma_{pc}(p')\, dp'}\, dp \tag{8}$$

The emission at each point p along the ray is scaled by the optical depth of the eye to produce the final intensity value for a pixel. The optical depth is a function of the total extinction coefficient, which is composed of the absorption coefficient σ_a and the scattering coefficient σ_{sc}. The generalized source $Q(p)$

is defined as

$$Q(p) = q(p) + \sigma_{sc(p)} \int \rho_{sc}(\omega' \to \omega) I(S, \omega')\, d\omega' \qquad (9)$$

This generalized source consists of the emission at a given point $q(p)$, and the incoming intensity along all directions scaled by the scattering phase ρ_{sc}. Typically, a low albedo approximation is used to simplify the calculations, reducing the integral in Eq. (9) to a sum over all light sources.

Domain Volume Rendering

In domain rendering the spatial 3-D data are first transformed into another 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 3-D data volume from a certain view direction is obtained by extracting a 2-D slice perpendicular to that view direction out of the 3-D Fourier spectrum and then inverse Fourier transforming it. This approach obtains the 3-D volume projection directly from the 3-D spectrum of the data and therefore, reduces the computational complexity for volume rendering from $O(N^3)$ to $O(N^2 \log N)$ (22–24). A major problem of frequency-domain volume rendering is that the resulting projection is a line integral along the view direction, which does not exhibit any occlusion and attenuation effects. Totsuka and Levoy (25) proposed a linear approximation to the exponential attenuation (20) 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 data set and, therefore, reduces the storage, computation, and transmission overhead of otherwise large volume data. For example, Ning and Hesselink (26) first applied vector quantization in the spatial domain to compress the volume and then directly rendered the quantized blocks by spatial domain volume rendering. Fowler and Yagel (27) combined differential pulse-code modulation and Huffman coding and developed a lossless volume-compressing algorithm, but their algorithm is not coupled with rendering. Yeo and Liu (28) applied a discrete, cosine-transform compressing technique on overlapping blocks of the data. Chiueh et al. (29) applied 3-D a Hartley transform to extend the JPEG still-image compressing algorithm to compress subcubes of the volume and performed frequency-domain rendering on the subcubes before compositing the resulting subimages in the spatial domain. Then each of the 3-D Fourier coefficient in each subcube is quantized, linearly sequenced through a 3-D zigzag 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 with spatial volume rendering, and improved image quality over conventional frequency-domain rendering techniques.

Wavelet theory (30), rooted in time-frequency analysis, has gained popularity in recent years. A wavelet is a fast decaying function with zero averaging. The attractive features of wavelets are that they have a local property in both the spatial and frequency domain and can be used to fully represent the volumes with a small number of wavelet coefficients. Muraki (31) first applied wavelet transform to volumetric datasets, Gross et al. (32) found an approximate solution for the volume-rendering equation using orthonormal wavelet functions, and Westermann (33) combined volume rendering with wavelet-based compression. All of these algorithms have not focused, however, on the acceleration of volume rendering by wavelets. The greater potential of wavelet domain, based on the elegant multiresolution hierarchy provided by the wavelet transform, is still far from fully utilized for volume rendering.

VOLUME-RENDERING OPTIMIZATIONS

A major drawback of the techniques described previously is the time required to generate a high-quality image. In this section, several volume-rendering optimizations are described that decrease rendering times and, therefore, increase interactivity and productivity. An alternative to speeding up volume rendering is to employ special-purpose hardware accelerators for volume rendering, which are described in the following section.

Object-order volume rendering typically loops through the data, calculating the contribution of each volume sample to pixels on the image plane. This is a costly operation for even moderately sized data sets (e.g., 128 Mbytes for a 512^3 sample dataset, with one byte per sample) and leads to rendering times that are noninteractive. For interaction, it is useful to generate a lower quality image faster. For data sets with binary sample values, bits could be packed into bytes such that each byte represents a $2 \times 2 \times 2$ portion of the data (14). A lower resolution image could be generated by processing the data byte-by-byte. A more general method for decreasing data resolution is to build a pyramidal data structure, which consists of a sequence of log N volumes for an original data set of N^3 data samples. The first volume is the original data set, whereas a lower resolution volume is created by averaging each $2 \times 2 \times 2$ sample group of the previous volume. An efficient implementation of the splatting algorithm, called hierarchical splatting (34), uses such a pyramidal data structure. According to the desired image quality, this algorithm scans the appropriate level of the pyramid in a back-to-front order. Each element is splatted onto the image plane with the appropriately sized splat. The splats themselves are approximated by polygons which are efficiently rendered by graphics hardware. The idea of a pyramid is also used in image-order volume rendering. Actually, Wang and Kaufman (35) have proposed the use of multiresolution hierarchy at arbitrary resolutions.

In discrete ray casting, it is quite computationally expensive to discretize every ray cast from the image plane. Fortunately, this is unnecessary for parallel projections. Because all of the rays are parallel, one ray can be discretized into a 26-connected line and used as a "template" for all other rays. This technique, developed by Yagel and Kaufman (36), is called *template-based volume viewing.* Rays are cast from a *baseplane,* that is, the plane of the volume buffer most parallel to the image plane. This ensures that each data sample contributes at most, once to the final image, and all data samples potentially contribute. Once all of the rays are cast from the base plane, a 2-D warp step is needed, which uses bilinear interpolation to determine the pixel values on the image plane from the ray values calculated on the base plane. This template-based ray casting is extended to support continuous ray casting and to allow for screen space supersampling to improve image quality.

The previous ideas have been extended in an algorithm called shear-warp factorization (37). It is based on an algorithm that factors the viewing transformation into a 3-D shear parallel to the data slices, a projection to form an intermediate but distorted image, and a 2-D warp to form an undistorted final image. The algorithm has been extended in three ways. First, a fast object-order rendering algorithm, based on the factorization algorithms with preprocessing and some loss of image quality, has been developed. Shear-warp factorization has the property that rows of voxels in the volume are aligned with rows of pixels in the intermediate image. Consequently, a scan-line-based algorithm has been constructed that traverses the volume and intermediate image in synchrony, taking advantage of the spatial coherence in both. Spatial data structures based on run-length encoding for both the volume and the intermediate image are used. The second extension is shear-warp factorization for perspective viewing transformations. Third, a data structure for encoding spatial coherence in unclassified volumes (i.e., scalar fields with no precomputed opacity) has been introduced.

One obvious optimization for both discrete and continuous ray casting, which has already been discussed is to limit the sampling to the segment of the ray which intersects the data, because samples outside of the data evaluate to 0 and do not contribute to the pixel value. If the data themselves contain many zero-valued data samples or a segmentation function is applied to the data that evaluates to 0 for many samples, the efficiency of ray casting is greatly enhanced by further limiting the segment of the ray in which samples are taken. One such algorithm is known as *polygon-assisted ray casting* (PARC) (38). This algorithm approximates objects contained within a volume by a crude polyhedral representation. The polyhedral representation is created, so that it completely contains the objects. Using conventional graphics hardware, the polygons are projected twice to create two Z-buffers. The first Z-buffer is the standard closest distance Z-buffer, whereas the second is a farthest distance Z-buffer. Because the object is completely contained within the representation, the two Z-buffer values for a given image plane pixel are used as the starting and ending points of a ray segment on which samples are taken.

The PARC algorithm is part of the *VolVis* volume visualization system (38,39), which provides a multialgorithmic progressive refinement approach for interactivity. By using available graphics hardware, the user can interactively navigate in a polyhedral representation of the data. When the user is satisfied with the placement of the data, light sources, and view, the Z-buffer information is passed to the PARC algorithm, which produces a ray-cast image. In a final step, this image is further refined by continuing to follow the PARC rays which intersected the data according to a volumetric ray-tracing algorithm (40) to generate shadows, reflections, and transparency (see below). The ray-tracing algorithm uses various optimization techniques, including uniform space subdivision and bounding boxes, to increase the efficiency of the secondary rays. Surface rendering and transparency with color and opacity transfer functions are incorporated within a global illumination model.

SPECIAL-PURPOSE, VOLUME-RENDERING HARDWARE

The high computation cost of direct volume rendering makes it difficult for general-purpose sequential computers to deliver the targeted level of performance. This situation is aggravated by the continuing trend towards higher and higher resolution data sets. For example, to render a high-resolution data set 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. To address this challenge, researchers have tried to achieve interactive display rates on supercomputers and massively parallel architectures (41–45). Most algorithms, however, require very little repeated computation on each voxel, and data movement actually accounts for a significant portion of the overall performance overhead. Today's commercial supercomputer memory systems do not have adequate latency and memory bandwidth for efficiently handling large amounts of data. Furthermore, supercomputers seldom contain frame buffers and, because of their high cost, are frequently shared by many users.

Just as the special requirements of traditional computer graphics lead to high-performance polygon engines, volume visualization naturally lends itself to special-purpose volume renderers. This allows for stand-alone visualization environments that help scientists to interactively view their static or dynamic data in real time. Several researchers have proposed special-purpose, volume-rendering architectures (1, Chapter 6; 46–50). Most recent studies have focused on accelerators for ray casting regular data sets. Ray casting offers room for algorithmic improvements while still allowing for high image quality. Recent architectures (51) include VOGUE (52), VIRIM (53), and Cube (54).

Cube has pioneered several hardware architectures. Cube-1, a first generation hardware prototype, was based on a specially interleaved memory organization (48), 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 of n voxels parallel to a main axis. Cube-2 is a single-chip VLSI implementation of Cube-1 (55). To achieve higher performance and to further reduce the critical memory access bottleneck, Cube-3 introduced several new concepts (56,57). 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. Cube-4 (58,59,60) 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. 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 high resolution of 1024^3 16 bit voxels and true real-time performance of 30 frames per second.

VOLUMETRIC GLOBAL ILLUMINATION

Standard volume-rendering techniques typically employ only a local illumination model for shading and, therefore, produce images without global effects. Including a global illumination model within a visualization system has several advantages. First, global effects are often desirable in scientific applications. For example, by placing mirrors in the scene, a single image shows several views of an object in a natural, intuitive manner leading to a better understanding of the 3-D nature of the scene. Also, complex geometric surfaces are often easier to render when represented volumetrically than when represented by high-order functions or geometric primitives, and global effects using ray tracing or radiosity are desirable for such applications, called volume graphics applications (see later).

A 3-D raster ray-tracing (RRT) method (16) produces realistic images of volumetric data with a global illumination model. The RRT algorithm is a discrete, recursive, ray-tracing algorithm similar to the discrete ray-casting algorithm described previously. Discrete primary rays are cast from the image plane through the data to determine pixel values. Secondary rays are recursively spawned when a ray encounters a voxel belonging to an object in the data. To save time, the view-independent parts of the illumination equation are precomputed and added to the voxel color, thereby avoiding calculation of this quantity during the ray tracing. Actually, all view-independent attributes (including normal, texture, antialiasing, and light-source visibility) can be precomputed and stored with each voxel.

A volumetric ray tracer (40) is intended to produce much more accurate, informative images. Such a ray tracer should handle volumetric data as well as classical geometric objects, and strict adherence to the laws of optics is not always desirable. For example, a user may wish to generate an image with no shadows or to view the maximum value along the segment of a ray passing through a volume, instead of the optically correct composited value.

To incorporate both volumetric and geometric objects into one scene, the standard ray-tracing intensity equation is expanded to include volumetric effects. The intensity of light $I_\lambda(x, \vec{\omega})$ for a given wavelength λ, arriving at a position x, from the direction $\vec{\omega}$, is computed by

$$I_\lambda(x, \omega) = I_{v\lambda}(x, x') + \tau_\lambda(x, x') I_{s\lambda}(x', \omega) \tag{10}$$

where x' is the first surface intersection point encountered along the ray $\vec{\omega}$ originating at x. $I_{s\lambda}(x', \vec{\omega})$ is the intensity of light at this surface location and is computed with a standard ray tracing illumination equation. $I_{v\lambda}(x, x')$ is the volumetric contribution to the intensity along the ray from x to x', and $\tau_\lambda(x, x')$ is the attenuation of $I_{s\lambda}(x', \vec{\omega})$ by any intervening volumes. These values are determined by volume-rendering techniques, based on a transport theory model of light propagation (21). The basic idea is similar to classical ray tracing, in that rays are cast from the eye into the scene, and surface shading is performed on the closest surface intersection point. The difference is that shading must be performed for all volumetric data encountered along the ray while traveling to the closest surface intersection point.

The volume ray-tracing algorithm is used to capture specular interactions between objects in a scene. In reality, most scenes are dominated by diffuse interactions, which are not accounted for in the standard ray-tracing illumination model, but are accounted for by a radiosity algorithm for volumetric data (60). In volumetric radiosity, a "voxel" element is defined in addition to the basic "patch" element of classical radiosity. As opposed to previous methods that use participating media to augment geometric scenes (61), this method moves the radiosity equations into volumetric space and renders scenes consisting solely of volumetric data. Each voxel emits absorbs, scatters, reflects, and transmits light. Both isotropic and diffuse emission of light are allowed, where "isotropic" implies directional independence and "diffuse" implies Lambertian reflection (i.e., dependent on normal or gradient). Light is scattered isotropically and is reflected diffusely by a voxel. Light that enters a voxel and is not absorbed, scattered, or reflected by the voxel is transmitted unchanged.

To cope with the high number of voxel interactions required, a hierarchical technique similar to (62) is used. The basic hierarchical concept is that the radiosity contribution from some voxel v_i to another voxel v_j is similar to the radiosity contribution from v_i to v_k if the distance between v_j and v_k is small and the distance between v_i and v_j is large. For each volume a hierarchical radiosity structure is built by combining each subvolume of eight voxels at one level to form one voxel at the next higher level. Then an interative algorithm (63) is used to shoot voxel radiosities, where several factors govern the highest level in the hierarchy at which two voxels can interact. Thse factors include the distance between the two voxels, the radiosity of the shooting voxel, and the reflectance and scattering coefficients of the voxel receiving the radiosity. This hierarchical technique reduces the number of interactions required to converge on a solution by more than four orders of magnitude. After the view-independent radiosities are calculated, a view-dependent image is generated by ray casting, where the final pixel value is determined by compositing radiosity values along the ray.

IRREGULAR GRID RENDERING

All the algorithms discussed previously handle only regular gridded data. Irregular gridded data (4) include curvilinear data and unstructured (scattered) data, where no explicit connectivity is defined between cells (64,65). In general, the most convenient grids for rendering are tetrahedral and hexahedral grids. One disadvantage of hexahedral grids is that the four points on the side of a cell are not necessarily coplanar. Tetrahedral grids have several advantages, including easier interpolation, simple representation (especially for connectivity information because the degree of the connectivity graph is bounded and allows for compact data structural representation), and that any other grid can be interpolated to a tetrahedral grid (with the possible introduction of Steiner points). Among disadvantages of tetrahedral grids is that the size of the data sets grows as cells are decomposed into tetrahedra. Compared with regular grids, operations for irregular grids are more complicated and effective visualization methods are more sophisticated. Shading, interpolation, point location, and the like, are all more difficult (and some even not well defined) for irregular grids. One notable exception is isosurface generation (6), which, even in the case of irregular grids, is fairly simple to compute given suitable interpolative functions. Slicing operations are also simple (4).

Volume rendering of irregular grids is a complex operation, and there are several different approaches to the problem. The simplest but most inefficient is to resample the irregular grid to a regular grid. To achieve the necessary accuracy, a high enough sampling rate has to be used, which in most cases makes the resulting regular grid volume too large for storage and rendering purposes, not to mention the time for the resampling.

Extending simple volumetric point sampling ray casting to irregular grids is a challenge. For ray casting, it is necessary to depth-sort samples along each ray. In the case of irregular grids, it is nontrivial to perform this sorting operation. Garrity (66) proposed a scheme where the cells are convex and connectivity information is available. The actual resampling and shading is also nontrivial and must be carefully considered, taking into account the specific application at hand (67). Simple ray casting is too inefficient, because of the large amount of interpixel and interscan-line coherency in ray casting. Giertsen (68) proposed a sweep-plane approach to ray casting that uses different forms of "caching" to speed up ray casting of irregular grids. More recently, Silva et al. (69) proposed lazy-sweep ray casting. It exploits coherency in the data, and it can handle disconnected and nonconvex irregular grids, with minimal time and memory cost. In a different sweeping technique proposed by Yagel et al. (70), the sweep plane is parallel to the viewing plane (as opposed to perpendicular, as in (68,69). This technique achieves impressive rendering times by exploiting available graphics hardware.

Another approach for rendering irregular grids is the use of object-order projection methods, where the cells are projected onto the screen, one by one, incrementally accumulating their contributions to the final image (64,71–73). One major advantage of these methods is the ability to exploit existing graphics hardware to compute simplified volumetric lighting models to speed up rendering. One problem with this method is generating the ordering for cell projections. In general, such ordering does not even exist and cells have to be partitioned into multiple cells for projection. The partitioning is generally view-dependent, but some types of irregular grids (like delaunay triangulations in space) are acyclic and do not need any partitioning.

VOLUME GRAPHICS

Volume buffer representation is more natural for empirical imagery than for geometric objects, because of its ability to represent interiors and digital samples. Nonetheless, the advantages of volumetric representation have also been attracting traditional surface-based applications that deal with the modeling and rendering of synthetic scenes made of geometric models. The geometric model is *voxelized (3-D scan-converted)* into a set of voxels that "best" approximate the model. Then each of these voxels is stored in the volume buffer together with the voxel's precomputed view-independent attributes. The voxelized model is either binary (15,74–76) or volume sampled (77), which generates alias-free density voxelization of the model. Some surface-based application examples are rendering of fractals (78), hyper textures (79), fur (80), gases (81), and other complex models (82), including CAD models and terrain models for flight simulators (83–85). Furthermore, in many applications involving sampled data, such as medical imaging, the data must be visualized along with synthetic objects that may not be available in digital form, such as scalpels, prosthetic devices, injection needles, radiation beams, and isodose surfaces. These geometric objects are voxelized and then intermixed with the sampled organ in the voxel buffer (86).

Volume graphics (84), an emerging subfield of computer graphics, is concerned with the synthesis, modeling, manipulation, and rendering of volumetric geometric objects, stored in a volume buffer of voxels. Unlike volume visualization, which focuses primarily on sampled and computed datasets, volume graphics is concerned primarily with modeled geometric scenes and commonly with those represented in a regular volume buffer. As an approach, volume graphics can greatly advance the field of 3-D graphics by offering a comprehensive alternative to traditional surface graphics.

Voxelization

An indispensable stage in volume graphics is the synthesis of voxel-represented objects from their geometric representation. This stage, is called *voxelization,* is concerned with converting geometric objects from their continuous geometric representation into a set of voxels that "best" approximates the continuous object. Because this process mimics the scan-conversion process that pixelizes (rasterizes) 2-D geometric objects, it is also called *3-D scan conversion.* In 2-D rasterization the pixels are directly drawn onto the screen to be visualized, and filtering is applied to reduce the aliasing artifacts. However, the voxelization process does not render the voxels but merely generates a database of the discrete digitization of the continuous object.

Intuitively, one would assume that a proper voxelization simply "selects" all voxels which are met (if only partially) by the object body. Although this approach is satisfactory in some cases, the objects it generates are commonly too coarse and include more voxels than necessary (87). However, if the object is too "thin", it does not successfully "separate" both sides of the surface. This is apparent when a voxelized scene is rendered by casting discrete rays. The penetration of the background voxels (which simulate the discrete ray traversal) through the voxelized surface causes a hole in the final image. Another type of error might occur when a 3-D flooding algorithm is employed to fill an object or to measure its volume or other properties. In this case the nonseparability of the surface causes a leakage of the flood through the discrete surface.

Unfortunately, the extension of the 2-D definition of separation to the third dimension and to voxel surfaces is not straightforward because voxelized surfaces cannot be defined as an ordered sequence of voxels and a voxel on the surface does not have a specific number of adjacent surface voxels. Furthermore, there are important topological issues, such as the separation of both sides of a surface which cannot be well defined by employing 2-D terminology. The theory that deals with these topological issues is called *3-D discrete topology.* Later we sketch some basic notions and informal definitions used in this field.

An early technique for digitizing solids was spatial enumeration which employs point or cell classification methods in an exhaustive fashion or by recursive subdivision (88). Subdivision techniques for model decomposition into rectangular subspaces, however, are computationally expensive and thus inappropriate for medium or high-resolution grids. Instead, the voxelization algorithms should follow the same paradigm as the 2-D scan-conversion algorithms. They should be incremental, accurate, use simple arithmetic (preferably integral only), and have complexity not more than linear with the number of voxels generated. The literature of 3-D scan conversion is relatively small. Danielsson (89) and Mokrzycki (90) independently developed similar 3-D curve algorithms where the curve is defined by the intersection of two implicit surfaces. Voxelization algorithms have been developed for 3-D lines (91), 3-D circles, and a variety of surfaces and solids, including polygons, polyhedra, and quadratic objects (15). Efficient algorithms have been developed for voxelizing polygons using an integer-based decision mechanism embedded within a scan-line filling algorithm (76), for parametric curves, surfaces, and volumes using an integer-based forward differencing technique (75), and for quadric objects such as cylinders, spheres, and cones using "weaving" algorithms by which a discrete circle/line sweeps along a discrete circle/line (74).

All of these algorithms have used a straightforward method of sampling in space, called *point sampling* or inary voxelization, which generates topologically and geometrically consistent models, but exhibits object-space aliasing. In point sampling, the continuous object is evaluated at the voxel center, and the value of 0 or 1 is assigned to the voxel. Because of this binary classification of the voxels, the resolution of the 3-D raster ultimately determines the precision of the discrete model. Imprecise modeling results in jagged surfaces, known as *object-space aliasing.* The emphasis in antialiased 3-D voxelization is on producing alias-free 3-D models that are stored in the view-independent volume buffer for various volume graphics manipulation, including but not limited to generating aesthetically pleasing displays. To reduce object-space aliasing, a *volume sampling* technique has been developed (77), which estimates the density contribution of the geometric objects to the voxels. The density of a voxel is attenuated by a filter weight function which is proportional to the distance between the center of the voxel and the geometric primitive.

Because the voxelized geometric objects are represented as volume buffers of density values, they are essentially treated as sampled or simulated volume data sets, and then one of many volume-rendering techniques for image generation is employed. One primary advantage of this approach is that volume rendering or volumetric global illumination carries the smoothness of the volume-sampled objects from object space over into its 2-D projection in image space. Hence, the silhouettes of the objects, reflections, and shadows are smooth. Furthermore, by not performing any geometric ray-object intersections or geometric surface normal calculations, the bulk of the rendering time is saved. In addition, CSG operations between two volume-sampled geometric models are accomplished at the voxel level after voxelization, thereby reducing the original problem of evaluating a CSG tree of such operations down to a fuzzy Boolean operation between pairs of nonbinary voxels (36) (see later). Volume-sampled models are also suitable for intermixing with sampled or simulated data sets, because they are treated uniformly as one common data representation. Furthermore, volume-sampled models lend themselves to alias-free multiresolution hierarchical construction (36).

Volume Graphics Advantages

One of the most appealing attributes of volume graphics is its insensitivity to the complexity of the scene, because all objects have been preconverted into a finite sized volume buffer. Although the performance of the preprocessing voxelization phase is influenced by the scene complexity (15,74–76), rendering performance depends mainly on the constant resolution of the volume buffer, not on the number of objects in the scene. Insensitivity to scene complexity makes the volumetric approach especially attractive for scenes consisting of numerous objects.

In volume graphics, rendering is decoupled from voxelization, and all objects are first converted into one meta object, the voxel, which makes the rendering process insensitive to the complexity of the objects. Thus, volume graphics is particularly attractive for objects difficult to render by conventional graphics systems. Examples of such objects include curved surfaces of high order and fractals (78). Constructive solid models are also hard to render by conventional methods, but are straightforward to render in volumetric representation (see below).

Antialiasing and texture mapping are commonly implemented during the last stage of the conventional rendering pipeline, and their complexity is proportional to object complexity. Solid texturing, which employs a 3-D textural image, also has high complexity proportional to object complexity. In volume graphics, however, antialiasing, textural mapping, and solid texturing are performed only once, during the voxelization stage, where the color is calculated and stored in each voxel. The texture is also stored as a separate volumetric entity which is rendered together with the volumetric object [e.g., (39)].

Anticipating repeated access to the volume buffer (such as in interaction or animation), all viewpoint independent attributes are precomputed during the voxelization stage, stored with the voxel, and are readily accessible to speed up the rendering. For each object voxel, the voxelization algorithm generates its color, texture color, normal vector (for visible voxels), antialiasing information (77), and information concerning the visibility of light sources from that voxel. Actually, the view-independent parts of the illumination equation are also precomputed and stored as part of the voxel value.

Once a volume buffer with precomputed view-independent attributes is available, a rendering algorithm, such as a ray casting or a volumetric ray-tracing algorithm, is engaged. Regardless of the complexity of the scene, running time is approximately the same as for simpler scenes and significantly faster than traditional space-subdivision, ray-tracing methods. Moreover, in spite of the discrete nature of the volume buffer representation, images indistinguishable from those produced by conventional surface-based ray tracing are generated by employing accurate ray tracing (41).

Sampled and simulated data sets are often reconstructed from the acquired sampled or simulated points into a regular grid of voxels and stored in a volume buffer. Such data sets provide for the majority of applications using the volumetric approach. Unlike surface graphics, volume graphics naturally and directly supports the representation, manipulation, and rendering of such data sets and provides the volume buffer medium for intermixing sampled or simulated datasets with geometric objects (86). For compatibility between the sampled/computed data and the voxelized geometric object, the object is volume sampled (77) with the same, but not necessarily the same, density frequency as the acquired or simulated datasets. Volume graphics also naturally supports the rendering of translucent volumetric data sets.

A central feature of volumetric representation is that, unlike surface representation, it represents the inner structures of objects, which can be revealed and explored with the appropriate volumetric manipulation and rendering techniques. Natural and synthetic objects are likely to be solid rather than hollow. The inner structure is easily explored by volume graphics and are supported by surface graphics. Moreover, although translucent objects are represented by surface methods, these methods cannot efficiently support the translucent rendering of volumetric objects or the modeling and rendering of amorphous phenomena (e.g., clouds, fire, smoke) that are volumetric and do not contain tangible surfaces (79–81).

An intrinsic characteristic of volume rasters is that adjacent feature in the scene are also represented by neighboring voxels. Therefore, rasters lend themselves to various meaningful block-based operations which are performed during the voxelization stage. For example, the 3-D counterpart of the *bitblt* operations, termed *voxblt* (voxel block-transfer), supports transfer of cuboidal voxel blocks with a variety of voxel-by-voxel operations between source and destination blocks (92). This property is very useful for CSG. Once a CSG model has been constructed in voxel representation by performing the Boolean operations between two voxelized primitives at the voxel level, it is rendered like any other volume buffer. This makes rendering constructive solid models straightforward.

The spatial presortedness of the volume buffer voxels lends itself to other types of grouping or aggregation of neighboring voxels. Voxels are aggregated into supervoxels in a pyramid-like hierarchy or a 3-D "mip-map" (93,94). For example, in a voxel-based flight simulator, the best resolution is used for takeoff and landing. As the aircraft ascends, fewer and fewer details need to be processed and visualized, and a lower resolution suffices. Furthermore, even in the same view, parts of the terrain close to the observer are rendered at high resolution which diminishes towards the horizon. A hierarchical volume buffer is prepared in advance or on-the-fly by subsampling or averaging the appropriate size neighborhoods of voxels [see also (95)].

Weakness of Volume Graphics

A typical volume buffer occupies a large amount of memory. For example, for a medium resolution of 512^3, two bytes per voxel, the volume buffer consists of 256 Mbytes. However, because computer memories are significantly decreasing in price and increasing in their compactness and speed, such large memories are becoming commonplace. This argument echoes a similar discussion when raster graphics emerged as a technology in the mid-seventies. With the rapid progress in memory price and compactness, it is safe to predict that, as in the case of raster graphics, the memory will soon cease to be a stumbling block for volume graphics.

The extremely large throughput that has to be handled requires special architecture and processing attention [see (1) Chapter 6]. Volume engines, analogous to the currently available polygon engines, are emerging. Because of the presortedness of the volume buffer and the fact that only a simple single type of object has to be handled, volume engines are conceptually simpler to implement than current polygon engines. We predict that, consequently, volume engines will materialize in the near future, with capabilities to synthesize, load, store, manipulate, and render volumetric scenes in real time (e.g., 30 frames/s), configured possibly as accelerators or cosystems to existing geometry engines.

Unlike surface graphics, in volume graphics, the 3-D scene is represented in discrete form. This is the cause of some of the problems of voxel-based graphics, which are similar to those of 2-D rasters (96). The finite resolution of the raster limits the accuracy of some operations, such as volume and area measurements, that are based on voxel counting, and becomes especially apparent when zooming in on the 3-D raster. When naive rendering algorithms are used, holes appear "between" voxels. Nevertheless, this is alleviated in ways similar to those adopted by 2-D raster graphics, such as employing reconstruction techniques, a higher resolution volume buffer, or volume sampling. Manipulation and transformation of the discrete volume are difficult without degrading the image quality or losing some information. Again, these can be alleviated by rendering, similar to the 2-D raster techniques.

Once an object has been voxelized, the voxels comprising the discrete object do not retain any geometric information about the geometric definition of the object. Thus, it is advantageous, when exact measurements are required, to employ conventional modeling where the geometric definition of the object is available. A voxel-based object is only a discrete approximation of the original continuous object where the volume buffer resolution determines the precision of such measurements. On the other hand, several measurement types are more easily computed in voxel space (e.g., mass property, adjacency detection, and volume computation). The lack of geometric information in the voxel may inflict other difficulties, such as surface normal computation. In voxel-based models, a discrete shading method is commonly employed to estimate the normal form a context of voxels. A variety of image-based and object-based methods for normal estimation from volumetric data have been devised [see (1) Chapter 4; (11)] and some have been discussed previously. Partial integration between surface and volume graphics is conceivable as part of an object-based approach in which an auxiliary object table, consisting of the geometric definition and global attributes of each object, is maintained in addition to the volume buffer. Each voxel consists of an index to the object table. This allows exact calculation of normal, exact measurements, and intersection verification for discrete ray tracing (16). The auxiliary geometric information might be useful also for re-voxelizing the scene in case the scene itself changes.

Surface Graphics vs Volume Graphics

Contemporary 3-D graphics has been employing an object-based approach at the expense of maintaining and manipulating a display list of geometric objects and regenerating the frame buffer after every change in the scene or viewing parameters. This approach, termed *surface graphics,* is supported by powerful polygon accelerators, which have flourished in the past decade, making surface graphics the state of the art in 3-D graphics.

Surface graphics strikingly resembles vector graphics that prevailed in the sixties and seventies and employed vector drawing devices. Like vector graphics, surface graphics represents the scene as a set of geometric primitives kept in a display list. In surface graphics, these primitives are transformed, mapped to screen coordinates, and converted by scan-conversion algorithms into a discrete set of pixels. Any change to the scene, viewing parameters, or shading parameters requires that the image generation system repeats this process. Like vector graphics that did not support painting the interior of 2-D objects, surface graphics generates merely the surfaces of 3-D objects and does not support the rendering of their interior.

Instead of a list of geometric objects maintained by surface graphics, volume graphics employs a 3-D volume buffer as a medium for representing and manipulating 3-D scenes. A 3-D scene is discretized earlier in the image generation sequence, and the resulting 3-D discrete form is used as a database of the scene for manipulation and rendering, which in effect decouples discretization from rendering. Furthermore, all objects are converted into one uniform metaobject, the voxel. Each voxel is atomic and represents the information about, at most, one object that resides in that voxel.

Volume graphics offers benefits similar to surface graphics, with several advantages due to decoupling, uniformity, and atomicity features. The rendering phase is view-independent and practically insensitive to scene complexity and object complexity. It supports Boolean and block operations and constructive solid modeling. When 3-D sampled or simulated data is used, volume graphics is also suitable for its representation. Volume graphics is capable of representing amorphous phenomena and both the interior and exterior of 3-D objects. Several weaknesses of volume graphics are related to the discrete nature of the representation. For instance, transformations and shading are performed in discrete space. In addition, this approach requires substantial amounts of storage space and specialized processing.

Table 1 contrasts vector graphics with raster graphics. A primary appeal of raster graphics is that it decouples image generation from screen refresh, thus making the refresh task insensitive to the scene and object complexities. In addition, the raster representation lends itself to block operations, such as windows, *bitblt,* and quadttrees. Raster graphics is also suitable for displaying 2-D sampled digital images and, thus, provides the ideal environment for mixing images with synthetic graphics. Unlike vector graphics, raster graphics presents shaded and textured surfaces and line drawings. These advantages, coupled with advances in hardware and the development of antialiasing methods, have led raster graphics to supersede vector graphics as the primary technology for computer graphics. The main weaknesses of raster graphics are the large memory and processing power required for the frame buffer and the discrete nature of the image. These difficulties delayed the full acceptance of raster graphics until the late seventies when the technology was able to provide cheaper and faster memory and hardware to support the demands of the raster approach. In addition, the discrete nature of rasters makes them less suitable for geometric operations, such as transformations and accurate measurements, and once discretized the notion of objects is lost.

The same appeal that drove the evolution of the computer graphics world from vector graphics to raster graphics, once the memory and processing power became available, is driving a variety of applications from a surface-based approach to a volume-based approach. Naturally, this trend first appeared in applications involving sampled or computed 3-D data, such as 3-D medical imaging and scientific visualization, in which the data sets are in volumetric form. These diverse empirical applications of volume visualization still provide a major driving force for advances in volume graphics. The comparison in Table 1 between vector graphics and raster graphics strikingly resembles a comparison between surface graphics and volume graphics. Actually Table 1 itself is used also to contrast surface graphics and volume graphics.

The progress so far in volume graphics, in computer hardware, and memory systems, coupled with the desire to reveal the inner structures of volumetric objects, suggests that volume visualization and volume graphics may develop into major trends in computer graphics. Just as raster graphics in the seventies superseded vector graphics for visualizing surfaces, volume graphics has the potential to supersede surface graphics for handling and visualizing volumes and for modeling and rendering synthetic scenes composed of surfaces.

Table 1. Comparison Between Vector Graphics and Raster Graphics and Between Surface Graphics and Volume Graphics

2-D	Vector Graphics	Raster Graphics
Memory and processing	+	−
Aliasing	+	−
Transformations	+	−
Objects	+	−
Scene/Object complexity	−	+
Block operations	−	+
Sampled data	−	+
Interior	−	+
3-D	Surface Graphics	Volume Graphics

ACKNOWLEDGMENTS

Special thanks are due to Lisa Sobierajski, Rick Avila, Roni Yagel, Dany Cohen, Sid Wang, Taosong He, Hanspeter Pfister, Claudio Silva, and Lichan Hong who contributed to this work, coauthored related papers (40,84,97) with me, and helped with the *VolVis* software. (*VolVis* is obtained by sending email to: volvis@cs.sunysb.edu.) This work was supported by the National Science Foundation under grants CDA-9303181 and MIP-9527694 and a grant from the Office of Naval Research

BIBLIOGRAPHY

1. A. Kaufman, *Volume Visualization,* Los Alamitos, CA: IEEE Computer Society Press, 1991.
2. A. Kaufman, Volume visualization, *ACM Comput. Surv.,* **28** (1): 165–167, 1996.
3. A. Kaufman, Volume visualization, in A. Tucker, (ed.), *Handbook of Computer Science and Engineering,* Boca Raton, FL: CRC Press, 1996.
4. D. Speray and S. Kennon, Volume probes: Interactive data exploration on arbitrary grids, *Comput. Graphics,* **24** (5): 5–12, 1990.
5. G. T. Herman and H. K. Liu, Three-dimensional display of human organs from computed tomograms, *Comput. Graphics Image Processing,* **9**: 1–21, 1979.

6. W. E. Lorensen and H. E. Cline, Marching cubes: A high resolution 3-D surface construction algorithm, *Comput. Graphics,* **21** (4): 163–170, 1987.
7. H. E. Cline et al., Two algorithms for the three-dimensional reconstruction of tomograms, *Medical Physics,* **15** (3): 320–327, 1988.
8. L. Sobierajski et al., A fast display method for volumetric data, *The Visual Comput.,* **10** (2): 116–124, 1993.
9. G. T. Herman and J. K. Udupa, Display of three dimensional discrete surfaces, *Proc. SPIE,* **283**: 90–97, 1981.
10. D. Gordon and R. A. Reynolds, Image space shading of 3-dimensional objects, *Comput. Vision Graphics Image Processing,* **29**: 361–376, 1985.
11. R. Yagel, D. Cohen, and A. Kaufman, Normal estimation in 3-D discrete space, *The Visual Computer,* **8** (5–6): 278–291, 1992.
12. L. Westover, Footprint evaluation for volume rendering, *Comput. Graphics, Proc. SIGGRAPH,* **24** (4): 144–153, 1990.
13. R. A. Drebin, L. Carpenter, and P. Hanrahan, Volume rendering, *Comput. Graphics, Proc. SIGGRAPH,* **22** (4): 65–74, 1988.
14. H. K. Tuy and L. T. Tuy, Direct 2-D display of 3-D objects, *IEEE Comput. Graphics Appl.,* **4** (10): 29–33, 1984.
15. A. Kaufman and E. Shimony, 3-D scan-conversion algorithms for voxel-based graphics, *Proc. ACM Workshop Interactive 3-D Graphics,* Chapel Hill, NC, October 1986, pp. 45–76.
16. R. Yagel, D. Cohen, and A. Kaufman, Discrete ray tracing, *IEEE Comput. Graphics Appl.,* **12** (5): 19–28, 1992.
17. L. S. Chen et al., Surface shading in the cuberille environment, *IEEE Comput. Graphics Appl.,* **5** (12): 33–43, 1985.
18. K. H. Hoehne and R. Bernstein, Shading 3-D-images from CT using gray-level gradients, *IEEE Trans. Med. Imaging,* **MI-5**: 45–47, 1986.
19. M. Levoy, Display of surfaces from volume data, *Comput. Graphics Appl.,* **8** (5): 29–37, 1988.
20. P. Sabella, A rendering algorithm for visualizing 3-D scalar fields, *Computer Graphics, Proc. SIGGRAPH,* **22** (4): 160–165, 1988.
21. W. Kruger, The application of transport theory to visualization of 3-D scalar data fields, *Comput. Phys.,* 397–406, July/August 1991.
22. S. Dunne, S. Napel, and B. Rutt, Fast reprojection of volume data, *Proc. 1st Conf. Visualization Biomedical Comput.,* Atlanta, GA, 1990, pp. 11–18.
23. M. Levoy, Volume rendering using the Fourier projection-slice theorem, *Graphics Interface '92,* 1992, pp. 61–69.
24. T. Malzbender, Fourier volume rendering, *ACM Trans. Graphics,* **12** (3): 233–250, 1993.
25. T. Totsuka and M. Levoy, Frequency domain volume rendering, *Computer Graphics, Proc. SIGGRAPH,* 1993, pp. 271–278.
26. P. Ning and L. Hesselink, Fast volume rendering of compressed data, *Visualization '93 Proc.,* October 1993, pp. 11–18.
27. J. Fowler and R. Yagel, Lossless compression of volume data, *Proc. Symp. Volume Visualization,* Washington, DC, October 1994, pp. 43–50.
28. B. Yeo and B. Liu, Volume rendering of DCT-based compressed 3-D scalar data, *IEEE Trans. Vis. Comput. Graphics,* **1**: 29–43, 1995.
29. T. Chiueh et al., *Compression Domain Volume Rendering,* Technical Report 94.01.04, Computer Science, SUNY at Stony Brook, January 1994.
30. C. Chui, *An Introduction to Wavelets,* New York: Academic Press, 1992.
31. S. Muraki, Volume data and wavelet transform, *IEEE Comput. Graphics Appl.,* **13** (4): 50–56, 1993.
32. M. H. Gross et al., A new method to approximate the volume rendering equation using wavelet bases and piecewise polynomials, *Comput. Graphics,* **19** (1): 47–62, 1995.
33. R. Westermann, A multiresolution framework for volume rendering, *1994 Symp. Volume Visualization,* Washington, DC, October 1994, pp. 51–58.
34. D. Laur and P. Hanrahan, Hierarchical splatting: A progressive refinement algorithm for volume rendering, *Computer Graphics,* **25** (4): 285–288, 1991.
35. S. Wang and A. Kaufman, Volume-sampled 3-D modeling, *IEEE Comput. Graphics Appl.,* **14** (5): 26–32, 1994.
36. R. Yagel and A. Kaufman, Template-based volume viewing, Comput. Graphics *Forum,* **11** (3): 153–167, 1992.
37. P. Lacroute and M. Levoy, Fast volume rendering using a shear-warp factorization of the viewing transformation, Comput. Graphics, **28** (3): 451–458, 1994.
38. R. Avila, L. Sobierajski, and A. Kaufman, Toward a comprehensive volume visualization system, *Visualization '92 Proc.,* October 1992, pp. 13–20.
39. R. Avila et al., VolVis: A diversified volume visualization system, *Visualization '94 Proc.,* Washington, DC, October 1994, pp. 31–38.
40. L. Sobierajski and A. Kaufman, Volumetric ray tracing, *Volume Visualization Symp. Proc.,* Washington, DC, October 1994, pp. 11–18.
41. P. Schroder and G. Stoll, Data parallel volume rendering as line drawing, *Workshop on Volume Visualization,* Boston, MA, October 1992, pp. 25–32.
42. C. Silva and A. Kaufman, Parallel performance measures for volume ray casting, *Visualization '94 Proc.,* Washington, DC, October 1994, pp. 196–203.
43. C. T. Silva, A. Kaufman, and C. Pavlakos, PVR: High-performance volume rendering, *IEEE Computational Sci. Eng.,* **3** (4): 16–28, 1996.
44. G. Vezina, P. A. Fletcher, and P. K. Robertson, Volume rendering on the MasPar MP-1, *Workshop on Volume Visualization,* Boston, MA, October 1992, pp. 3–8.
45. T. S. Yoo et al., Direct visualization of volume data, *IEEE Comput. Graphics Appl.,* **12** (4): 63–71, 1992.
46. S. M. Goldwasser et al., Physician's workstation with real-time performance, *IEEE Comput. Graphics Appl.,* **5** (12): 44–57, 1985.
47. D. Jackel, The graphics PARCUM system: A 3-D memory based computer architecture for processing and display of solid models, *Comput. Graphics Forum,* **4**: 21–32, 1985.
48. A. Kaufman and R. Bakalash, Memory and processing architecture for 3-D voxel-based imagery, *IEEE Comput. Graphics Appl.,* **8** (6): 10–23, 1988. Also in Japanese, *Nikkei Comput. Graphics,* **3** (30): 148–160, 1989.
49. D. J. Meagher, Applying solids processing methods to medical planning, *Proc. NCGA'85,* Dallas, TX, April 1985, pp. 101–109.
50. T. Ohashi, T. Uchiki, and M. Tokoro, A three-dimensional shaded display method for voxel-based representation, *Proc. EUROGRAPHICS '85,* Nice, France, September 1985, pp. 221–232.
51. J. Hesser et al., Three architectures for volume rendering, *Comput. Graphics Forum,* **14** (3): 111–122, 1995.
52. G. Knittel and W. Strasser, A compact volume rendering accelerator, *Volume Visualization Symp. Proc.,* Washington, DC, October 1994, pp. 67–74.
53. T. Guenther et al., VIRIM: A massively parallel processor for real-time volume visualization in medicine, *Proc. 9th Eurographics Hardware Workshop,* Oslo, Norway, September 1994, pp. 103–108.

54. H. Pfister and A. Kaufman, Cube-4: A scalable architecture for real-time volume rendering, *Volume Visualization Symp. Proc.,* San Francisco, CA, October 1996, pp. 47–54.
55. R. Bakalash et al., An extended volume visualization system for arbitrary parallel projection, *Proc. 1992 Eurographics Workshop Graphics Hardware,* Cambridge, UK, September 1992.
56. H. Pfister, A. Kaufman, and T. Chiueh, Cube-3: A real-time architecture for high-resolution volume visualization, *Volume Visualization Symp. Proc.,* Washington, DC, October 1994, pp. 75–82.
57. H. Pfister, F. Wessels, and A. Kaufman, Sheared interpolation and gradient estimation for real-time volume rendering, *Comput. Graphics,* **19** (5): 667–677, 1995.
58. U. Kanus et al., Implementations of cube-4 on the teramac custom computing machine, *Comput. Graphics,* **21** (2): 1997.
59. H. Pfister, A. Kaufman, and F. Wessels, Toward a scalable architecture for real-time volume rendering, *10th Eurographics Workshop Graphics Hardware Proc.,* Maastricht, The Netherlands, August 1995.
60. L. Sobierajski and A. Kaufman, *Volumetric Radiosity,* Technical Report 94.01.05, Computer Science, SUNY Stony Brook, 1994.
61. H. E. Rushmeier and K. E. Torrance, The zonal method for calculating light intensities in the presence of a participating medium, *Comput. Graphics,* **21** (4): 293–302, 1987.
62. P. Hanrahan, D. Salzman, and L. Aupperle, A rapid hierarchical radiosity algorithm, *Comput. Graphics,* **25** (4): 197–206, 1991.
63. M. F. Cohen et al., A progressive refinement approach to fast radiosity image generation, *Comput. Graphics, Proc SIGGRAPH,* 1988, pp. 75–84.
64. N. Max, P. Hanrahan, and R. Crawfis, Area and volume coherence for efficient visualization of 3-D scalar functions, *Comput. Graphics,* **24** (5): 27–34, 1990.
65. G. M. Nielson, Scattered data modeling, *IEEE Comput. Graphics Appl.,* **13** (1): 60–70, 1993.
66. M. P. Garrity, Raytracing irregular volume data, *Comput. Graphics,* **24** (5): 35–40, 1990.
67. N. Max, Optical models for direct volume rendering, *IEEE Trans. Vis. Comput. Graphics,* **1**: 99–108, 1995.
68. C. Giertsen, Volume visualization of sparse irregular meshes, *IEEE Comput. Graphics Appl.,* **12** (2): 40–48, 1992.
69. C. T. Silva, J. S. B. Mitchell, and A. Kaufman, Fast rendering of irregular grids, *Volume Visualization Symp. Proc.,* San Francisco, CA, October 1996, pp. 15–22.
70. R. Yagel et al., Hardware assisted volume rendering of unstructured grids by incremental slicing, *Volume Visualization Symp. Proc.,* San Francisco, CA, October 1996, pp. 55–62.
71. P. Shirley and H. Neeman, Volume visualization at the center for supercomputing research and development, in C. Upson, (ed.), *Proc. Workshop Volume Visualization,* Chapel Hill, NC, May 1989, pp. 17–20.
72. J. Wilhems and A. vanGelder, A coherent projection approach for direct volume rendering, *Comp. Graphics, SIGGRAPH '91 Proc.,* **25**: 275–284, 1991.
73. P. L. Williams, Interactive splatting of nonrectilinear volumes, *Proc. Visualization '92,* Boston, MA, October 1992, pp. 37–44.
74. D. Cohen and A. Kaufman, Scan conversion algorithms for linear and quadratic objects, in A. Kaufman, (ed.), *Volume Visualization,* Los Alamitos, CA: IEEE Computer Society Press, 1991, pp. 280–301.
75. A. Kaufman, Efficient algorithms for 3-D scan-conversion of parametric curves, surfaces, and volumes, *Comput. Graphics,* **21** (4): 171–179, 1987.
76. A. Kaufman, An algorithm for 3-D scan-conversion of polygons, *Proc. EUROGRAPHICS '87,* Amsterdam, Netherlands, August 1987, pp. 197–208.
77. S. Wang and A. Kaufman, Volume sampled voxelization of geometric primitives, *Visualization '93 Proc.,* San Jose, CA, October 1993, pp. 78–84.
78. V. A. Norton, Generation and rendering of geometric fractals in 3-D, *Comput. Graphics,* **16** (3): 61–67, 1982.
79. K. Perlin and E. M. Hoffert, Hypertexture, *Comput. Graphics,* **23** (3): 253–262, 1989.
80. J. T. Kajiya and T. L. Kay, Rendering fur with three dimensional textures, *Comput. Graphics,* **23** (3): 271–280, 1989.
81. D. S. Ebert and R. E. Parent, Rendering and animation of gaseous phenomena by combining fast volume and scanline A-buffer techniques, *Comput. Graphics,* **24** (4): 357–366, 1990.
82. J. M. Snyder and A. H. Barr, Ray tracing complex models containing surface tessellations, *Comput. Graphics,* **21** (4): 119–128, 1987.
83. D. Cohen–Or et al., A real-time photo-realistic visual flythrough, *IEEE Trans. Vis. Comput. Graphics,* **2** (3): 255–265, 1996.
84. A. Kaufman, D. Cohen, and R. Yagel, Volume graphics, *IEEE Comput.,* **26** (7): 51–64, 1993. Also in Japanese, *Nikkei Comput. Graphics,* **1**, (88): 148–155 & **2**, (89): 130–137, 1994.
85. J. Wright and J. Hsieh, A voxel-based forward projection algorithm for rendering surface and volumetric data, *Proc. Visualization '92,* Boston, MA, October 1992, pp. 340–348.
86. A. Kaufman, R. Yagel, and D. Cohen, Intermixing surface and volume rendering, in K. H. Hoehne, H. Fuchs, and S. M. Pizer, (eds.), *3D Imaging in Medicine: Algorithms, Systems, Applications,* 1990, pp. 217–227.
87. D. Cohen–Or and A. Kaufman, Fundamentals of surface voxelization, *CVGIP: Graphics Models and Image Processing,* **56** (6): 453–461, 1995.
88. Y. T. Lee and A. A. G. Requicha, Algorithms for computing the volume and other integral properties of solids: I-Known methods and open issues; II-A family of algorithms based on representation conversion and cellular approximation, *Commun. ACM,* **25** (9): 635–650, 1982.
89. P. E. Danielsson, Incremental curve generation, *IEEE Trans. Comput.,* **C-19**: 783–793, 1970.
90. W. Mokrzycki, Algorithms of discretization of algebraic spatial curves on homogeneous cubical grids, *Comput. Graphics,* **12** (3/4): 477–487, 1988.
91. D. Cohen–Or and A. Kaufman, 3-D line voxelization and connectivity control, *IEEE Comput. Graphics Appl.,* 1997.
92. A. Kaufman, The voxblt Engine: A voxel frame buffer processor, in A. A. M. Kuijk, (ed.), *Advances in Graphics Hardware III,* Berlin: Springer-Verlag, 1992, pp. 85–102.
93. M. Levoy and R. Whitaker, Gaze-directed volume rendering, *Comput. Graphics, Proc. 1990 Symp. Interactive 3-D Graphics,* **24** (2): 217–223, 1990.
94. G. Sakas and J. Hartig, Interactive visualization of large scalar voxel fields, *Proc. Visualization '92,* Boston, MA, October 1992, pp. 29–36.
95. T. He et al., Voxel-based object simplification, *IEEE Visualization '95 Proc.,* Los Alamitos, CA, October 1995, pp. 296–303.
96. C. M. Eastman, Vector versus raster: A functional comparison of drawing technologies, *IEEE Comput. Graphics Appl.,* **10** (5): 68–80, 1990.
97. A. Kaufman and L. Sobierajski, Continuum volume display, in R. S. Gallagher (ed.), *Comput. Visualization,* Boca Raton, FL: CRC Press, 1994, pp. 171–202.

ARIE E. KAUFMAN
State University of New York at Stony Brook