Introduction to Volume Graphics

Arie E. Kaufman

Center for Visual Computing
and Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
ari@cs.sunysb.edu
http://www.cs.sunysb.edu/~ari

Abstract

This paper is a survey of volume graphics. It includes an introduction to volumetric data and to volume modeling techniques, such as voxelization, texture mapping, amorphous phenomena, block operations, constructive solid modeling, and volume sculpting. A comparison between surface graphics and volume graphics is given, along with a consideration of volume graphics advantages and weaknesses. The paper concludes with a discussion on special-purpose volume rendering hardware.

1. Introduction

Volume data are 3D entities that may have information inside them, might not consist of surfaces and edges, or might be too voluminous to be represented geometrically. Volume visualization is a method of extracting meaningful information from volumetric data using interactive graphics and imaging, and it is concerned with volume data representation, modeling, manipulation, and rendering [36, 41, 42]. Volume data are obtained by sampling, simulation, or modeling techniques. For example, a sequence of 2D slices obtained from Magnetic Resonance Imaging (MRI) or Computed Tomography (CT) is 3D reconstructed into a volume model and visualized for diagnostic purposes or for planning of treatment or surgery. The same technology is often used with industrial CT for non-destructive inspection of composite materials or mechanical parts. Similarly, confocal microscopes produce data which is visualized to study the morphology of biological structures. In many computational fields, such as in computational fluid dynamics, the results of simulation typically running on a supercomputer are often visualized as volume data for analysis and verification. Recently, many traditional geometric computer graphics applications, such as CAD and simulation, have exploited the advantages of volume techniques called *volume graphics* for modeling, manipulation, and visualization.

Volume graphics [38], which is 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 that are represented in a regular volume buffer. As an approach, volume graphics has the potential to greatly advance the field of 3D graphics by offering a comprehensive alternative to traditional surface graphics.

We begin in Section 2 with an introduction to volumetric data. In the following sections we describe the volumetric approach to several common volume graphics modeling techniques. We describe the generation of object primitives by voxelization (Section 3), fundamentals of 3D discrete topology (Section 4), binary voxelization (Section 5), 3D antialiasing and multivalued voxelization (Section 6), texture and photo mapping and solid-texturing (Section 7), modeling of amorphous phenomena (Section

8), modeling by block operations and constructive solid modeling (Section 9), and volume sculpting (Section 10). Then, volume graphics is contrasted with surface graphics (Section 11), and the corresponding advantages (Section 12) and disadvantages (Section 13) are discussed. In Section 14 we describe special-purpose volume rendering hardware.

2. Volumetric Data

Volumetric data is typically a set S of samples (x, y, z, v), representing the value v of some property of the data, at a 3D location (x, y, z). If the value is simply a 0 or 1, with a value of 0 indicating background and a value of 1 indicating the object, then the data is referred to as binary data. The data may instead be multivalued, with the value representing some measurable property of the data, including, for example, color, density, heat or pressure. The value v may even be a vector, representing, for example, velocity at each location.

In general, samples may be taken at purely random locations in space, but in most cases the set S is isotropic containing samples taken at regularly spaced intervals along three orthogonal axes. When the spacing between samples along each axis is a constant, but there may be three different spacing constants for the three axes, then set S is anisotropic. Since the set of samples is defined on a regular grid, a 3D array (called also *volume buffer, cubic frame buffer, 3D raster*) is typically used to store the values, with the element location indicating position of the sample on the grid. For this reason, the set S will be referred to as the array of values $S(x, y, z)$, which is defined only at grid locations. Alternatively, either rectilinear, curvilinear (structured), or unstructured grids, are employed (e.g., [71]). In a *rectilinear* grid the cells are axis-aligned, but grid spacings along the axes are arbitrary. When such a grid has been non-linearly transformed while preserving the grid topology, the grid becomes *curvilinear*. Usually, the rectilinear grid defining the logical organization is called *computational space,* and the curvilinear grid is called *physical space*. Otherwise the grid is called *unstructured* or *irregular*. An unstructured or irregular volume data is a collection of cells whose connectivity has to be specified explicitly. These cells can be of an arbitrary shape such as tetrahedra, hexahedra, or prisms.

The array S only defines the value of some measured property of the data at discrete locations in space. A function $f(x, y, z)$ may be defined over R^3 in order to describe the value at any continuous location. The function $f(x, y, z) = S(x, y, z)$ 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 interpolation function to S. There are many possible interpolation functions. The simplest interpolation function is known as *zero-order interpolation*, which is actually just a nearest-neighbor function. The value at any location in R^3 is simply the value of the closest sample to that location. With this interpolation method there is a region of constant value around each sample in S. Since the samples in S are regularly spaced, each region is of uniform size and shape. The region of constant value that surrounds each sample is known as a *voxel* with each voxel being a rectangular cuboid having six faces, twelve edges, and eight corners.

Higher-order interpolation functions can also be used to define $f(x, y, z)$ between sample points. One common interpolation function is a piecewise function known as *first-order interpolation*, or *trilinear interpolation*. With this interpolation function, the value is assumed to vary linearly along directions parallel to one of 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. The value v_P, according to trilinear interpolation, is then:

$$v_P = v_A\,(1-x_p)(1-y_p)(1-z_p) + v_E\,(1-x_p)(1-y_p)\,z_p\; + \tag{1}$$

$$v_B\;\; x_p\,(1-y_p)(1-z_p) + v_F\;x_p\,(1-y_p)\,z_p\; +$$

$$v_C\,(1-x_p)\,y_p\,(1-z_p) + v_G\,(1-x_p)\,y_p\;\; z_p\; +$$

$$v_D\;x_p\,y_p\,(1-z_p) + v_H\;x_p\,y_p\;\; z_p$$

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 Equation 1 would be replaced by $\frac{(x_p - x_A)}{(x_H - x_A)}$, with similar substitutions made for y_p and z_p.

Over the years many techniques have been developed to visualize 3D data. Since methods for displaying geometric primitives were already well-established, most of the early methods involve approximating a surface contained within the data using geometric primitives [4, 47]. When volumetric data are visualized using a surface rendering technique, a dimension of information is essentially lost. In response to this, volume rendering techniques were developed that attempt to capture the entire 3D data in a single 2D image [12, 36, 45, 62, 79, 86]. Volume rendering convey more information than surface rendering images, but at the cost of increased algorithm complexity, and consequently increased rendering times. To improve interactivity in volume rendering, many optimization methods as well as several special-purpose volume rendering machines have been developed (see Section 14).

The 3D raster representation seems to be more natural for empirical imagery than for geometric objects, due to its ability to represent interiors and digital samples. Nonetheless, the advantages of this representation are also attracting traditional surface-based applications that deal with the modeling and

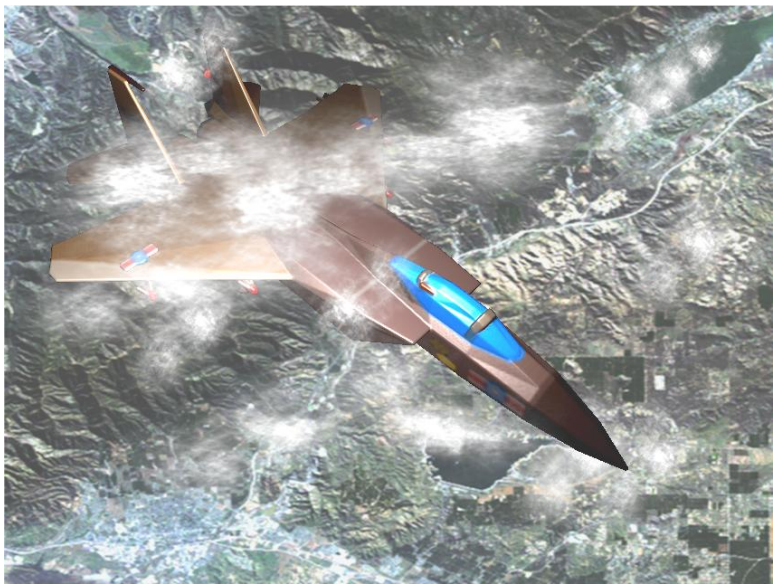

Figure 1: A volume-sampled plane within a volumetric cloud over volumetric model of terrain enhanced with photo mapping of satellite images.

rendering of synthetic scenes made out of geometric models. The geometric model is *voxelized (3D scan-converted)* into a set of voxels that ‘‘best’’ approximate the model. Each of these voxels is then stored in the volume buffer together with the voxel pre-computed view-independent attributes. The voxelized model can be either binary (see [5, 30-32] and Section 5) or volume sampled (see [72, 82] and Section 6) which generates alias-free density voxelization of the model. Some surface-based application examples are the rendering of fractals [51], hyper textures [54], fur [28], gases [15], and other complex models [69] including terrain models for flight simulators (see Figures 1 and 2) [6, 38, 81, 87]. and CAD models (see Figure 3). Furthermore, in many applications involving sampled data, such as medial imaging, the data need to 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 can be voxelized and intermixed with the sampled organ in the voxel buffer [35].

3. Voxelization

An indispensable stage in volume graphics is the synthesis of voxel-represented objects from their geometric representation. This stage, which 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. As this process mimics the scan-conversion process that pixelizes (rasterizes) 2D geometric objects, it is also referred to as *3D scan-conversion*. In 2D 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.

Figure 2: A volumetric model of terrain enhanced with photo mapping of satellite images. The buildings are synthetic voxel models raised on top of the terrain. The voxelized terrain has been mapped with aerial photos during the voxelization stage.

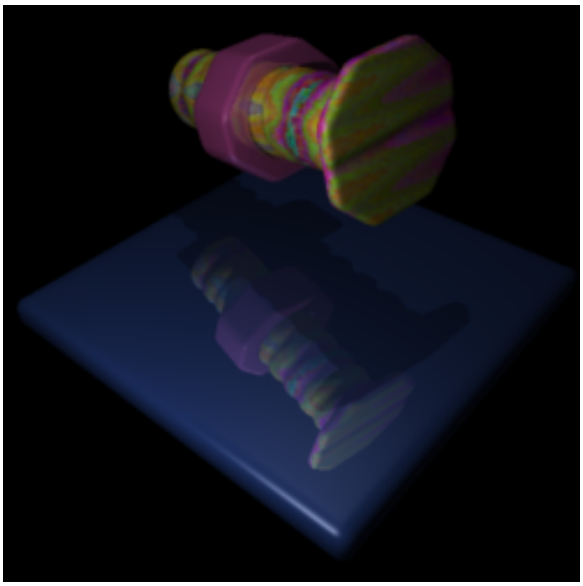

Figure 3: Volume-sampled bolt and nut generated by a sequence of CSG operations on hexagonal, cylindrical, and helix primitives, reflected on a volume-sampled mirror.

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 could be satisfactory in some cases, the objects it generates are commonly too coarse and include more voxels than are necessary. For example, when a 2D curve is rasterized into a connected sequence of pixels, the discrete curve does not ‘‘cover’’ the entire continuous curve, but is connected, concisely and successfully ‘‘separating’’ both ‘‘sides’’ of the curve [7].

One practical meaning of separation is apparent when a voxelized scene is rendered by casting discrete rays from the image plane to the scene. The penetration of the background voxels (which simulate the discrete ray traversal) through the voxelized surface causes the appearance of a hole in the final image of the rendered surface. Another type of error might occur when a 3D flooding algorithm is employed either to fill an object or to measure its volume, surface area, 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 2D definition of separation to the third dimension and voxel surfaces is not straightforward, since 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 2D terminology. The theory that deals with these topological issues is called *3D discrete topology*. We sketch below some basic notions and informal definitions used in this field.

4. Fundamentals of 3D Discrete Topology

The 3D discrete space is a set of integral grid points in 3D Euclidean space defined by their Cartesian coordinates (x, y, z). A voxel is the unit cubic volume centered at the integral grid point. The voxel

value is mapped onto {0,1}: the voxels assigned ‘‘1’’ are called the ‘‘black’’ voxels representing opaque objects, and those assigned ‘‘0’’ are the ‘‘white’’ voxels representing the transparent background. In Section 6 we describe non-binary approaches where the voxel value is mapped onto the interval [0,1] representing either partial coverage, variable densities, or graded opacities. Due to its larger dynamic range of values, this approach supports 3D antialiasing and thus supports higher quality rendering.

Two voxels are *26-adjacent* if they share either a vertex, an edge, or a face (see Figure 4). Every voxel has 26 such adjacent voxels: eight share a vertex (corner) with the center voxel, twelve share an edge, and six share a face. Accordingly, face-sharing voxels are defined as *6-adjacent*, and edge-sharing and face-sharing voxels are defined as *18-adjacent*. The prefix N is used to define the adjacency relation, where $N = 6, 18,$ or 26. A sequence of voxels having the same value (e.g., ‘‘black’’) is called an *N-path* if all consecutive pairs are N-adjacent. A set of voxels W is *N-connected* if there is an N-path between every pair of voxels in W (see Figure 4). An *N-connected component* is a maximal N-connected set.

Given a 2D discrete 8-connected black curve, there are sequences of 8-connected white pixels (8-component) that pass from one side of the black component to its other side without intersecting it. This phenomenon is a discrete disagreement with the continuous case where there is no way of penetrating a closed curve without intersecting it. To avoid such a scenario, it has been the convention to define ‘‘opposite’’ types of connectivity for the white and black sets. ‘‘Opposite’’ types in 2D space are 4 and 8, while in 3D space 6 is ‘‘opposite’’ to 26 or to 18.

Assume that a voxel space, denoted by Σ, includes one subset of ‘‘black’’ voxels S. If $\Sigma - S$ is not N-connected, that is, $\Sigma - S$ consists of at least two white N-connected components, then S is said to be *N-separating* in Σ. Loosely speaking, in 2D, an 8-connected black path that divides the white pixels into two groups is 4-separating and a 4-connected black path that divides the white pixels into two groups is 8-separating. There are no analogous results in 3D space.

Let W be an N-separating surface. A voxel $p \in W$ is said to be an *N-simple voxel* if $W - p$ is still N-separating. An N-separating surface is called *N-minimal* if it does not contain any N-simple voxel. A *cover* of a continuous surface is a set of voxels such that every point of the continuous surface lies in a voxel of the cover. A cover is said to be a *minimal cover* if none of its subsets is also a cover. The cover property is essential in applications that employ space subdivision for fast ray tracing [18]. The subspaces (voxels) which contain objects have to be identified along the traced ray. Note that a cover is

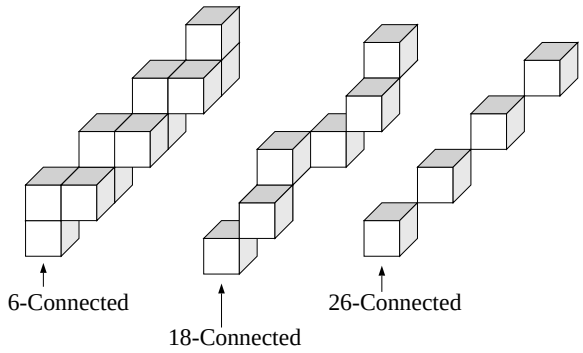

Figure 4: 6-, 18-, and 26-connected paths

not necessarily separating, while on the other hand, as mentioned above, it may include simple voxels. In fact, even a minimal cover is not necessarily N-minimal for any N [7].

5. Binary Voxelization

An early technique for the digitization of solids was spatial enumeration which employs point or cell classification methods in either an exhaustive fashion or by recursive subdivision [44]. However, subdivision techniques for model decomposition into rectangular subspaces are computationally expensive and thus inappropriate for medium or high resolution grids. Instead, objects should be directly voxelized, preferably generating an N-separating, N-minimal, and covering set, where N is application dependent. The voxelization algorithms should follow the same paradigm as the 2D scan-conversion algorithms; they should be incremental, accurate, use simple arithmetic (preferably integer only), and have a complexity that is not more than linear with the number of voxels generated.

The literature of 3D scan-conversion is relatively small. Danielsson [11] and Mokrzycki [49] developed independently similar 3D curve algorithms where the curve is defined by the intersection of two implicit surfaces. Voxelization algorithms have been developed for 3D lines [8], 3D circles, and a variety of surfaces and solids, including polygons, polyhedra, and quadric objects [30]. Efficient algorithms have been developed for voxelizing polygons using an integer-based decision mechanism embedded within a scan-line filling algorithm [31] or with an incremental pre-filtered algorithm [10], for parametric curves, surfaces, and volumes using an integer-based forward differencing technique [32], 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 [5]. Figure 2 consists of a variety of objects (polygons, boxes, cylinders) voxelized using these methods. These pioneering attempts should now be followed by enhanced voxelization algorithms that, in addition to being efficient and accurate, will also adhere to the topological requirements of separation, coverage, and minimality.

6. 3D Antialiasing and Multivalueded Voxelization

The previous section discussed binary voxelization, which generates topologically and geometrically consistent models, but exhibits object space aliasing. These algorithms have used a straightforward method of sampling in space, called *point sampling*. 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 3D raster ultimately determines the precision of the discrete model. Imprecise modeling results in jagged surfaces, known as *object space aliasing* (see Figure 2). In this section, we first present a 3D object-space antialiasing technique. It performs antialiasing once, on a 3D view-independent representation, as part of the modeling stage. Unlike antialiasing of 2D scan-converted graphics, where the main focus is on generating aesthetically pleasing displays, the emphasis in antialiased 3D voxelization is on producing alias-free 3D models that are stored in the view-independent volume buffer for various volume graphics manipulations, including but not limited to the generation of aesthetically pleasing displays (see Figure 1).

To reduce object space aliasing, a *volume sampling* technique has been developed [82], 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. To improve performance, precomputed lookup tables of densities for a predefined set of geometric primitives can be used to select the density value of each voxel. For each voxel visited

by the binary voxelization algorithm, the distance to the predefined primitive is used as an index into a lookup table of densities.

Since voxelized geometric objects are represented as volume rasters of density values, they can essentially be treated as sampled or simulated volume datasets, such as 3D medical imaging datasets, and one of many volume rendering techniques for image generation can be 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 2D projection in image space [84]. Hence, the silhouette of the objects, reflections, and shadows are smooth. 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 non-binary voxels [83] (see Section 9). Volume-sampled models are also suitable for intermixing with sampled or simulated datasets, since they can be treated uniformly as one common data representation. Furthermore, volume-sampled models lend themselves to alias-free multi-resolution hierarchy construction [83].

Further study of filtered voxelization has been conducted by Sramek and Kaufman [72, 74]. The basic idea has been that in voxelization filter design one needs to consider the visualization techniques used (i.e., data interpolation and normal estimation). They showed that a combination of first order filters on voxelization and visualization, with proper parameters, results in rendered images with negligible error in estimating object surface position and normal direction. More specifically, if a trilinear interpolation is used for reconstruction of the continuous volume, with subsequent surface detection by thresholding and normal gradient calculation by central differences, best results are obtained if the density of the voxelized object near its surface is linear along the surface normal direction (e.g., [27, 73]). This linear profile results from convolution of the object with a 1D box filter applied along a direction perpendicular to the surface, called oriented box filter [72]. Furthermore, a Gaussian surface density profile combined with tricubic interpolation and Gabor derivative reconstruction outperforms the linear density profile, but for a sharp increase in the computation time.

A C++ library for filtered voxelization of objects, vxt, has been developed [75]. It provides the user with an extensible set of easy-to-use tools and routines for alias-free voxelization of analytically defined monochromatic and color objects. Thus, resulting volumetric data represent a suitable input for both software and hardware volume rendering systems. The library provides for voxelization of primitive objects; however, when supplemented by a suitable parser, it represents a basis for voxelization of complex models defined in various graphics formats.

7. Texture Mapping

One type of object complexity involves objects that are enhanced with texture mapping, photo mapping, environment mapping, or solid texturing. Texture mapping is commonly implemented during the last stage of the rendering pipeline, and its complexity is proportional to the object complexity. In volume graphics, however, texture mapping is performed during the voxelization stage, and the texture color is stored in each voxel in the volume buffer.

In photo mapping six orthogonal photographs of the real object are projected back onto the voxelized object. Once this mapping is applied, it is stored with the voxels themselves during the voxelization stage, and therefore does not degrade the rendering performance. Texture and photo mapping are also viewpoint independent attributes implying that once the texture is stored as part of the voxel value,

texture mapping need not be repeated. This important feature is exploited, for example, by voxel-based flight simulators (see Figures 1 and 2) and in CAD systems (see Figure 3).

A central feature of volumetric representation is that, unlike surface representation, it is capable of representing inner structures of objects, which can be revealed and explored with appropriate manipulation and rendering techniques. This capability is essential for the exploration of sampled or computed objects. Synthetic objects are also likely to be solid rather than hollow. One method for modeling various solid types is solid texturing, in which a function or a 3D map models the color of the objects in 3D (see Figure 3). During the voxelization phase each voxel belonging to the objects is assigned a value by the texturing function or the 3D map. This value is then stored as part of the voxel information. Again, since this value is view independent, it does not have to be recomputed for every change in the rendering parameters.

8. Amorphous Phenomena

While translucent objects can be represented by surface methods, these methods cannot efficiently support the modeling and rendering of amorphous phenomena (e.g., clouds, fire, smoke) that are volumetric in nature and lack any tangible surfaces. A common modeling and rendering approach is based on a volumetric function that, for any input point in 3D, calculates some object features such as density, reflectivity, or color (see Figure 1). These functions can then be rendered by ray casting, which casts a ray from each pixel into the function domain. Along the passage of the ray, at constant intervals the function is evaluated to yield a sample. All samples along each ray are combined to form the pixel color. Some examples for the use of this or similar techniques are the rendering of fractals [22], hypertextures [54], fur [28], and gases [15].

The process of function evaluation at each sample point in 3D has to be repeated for each image generated. In contrast, the volumetric approach allows the pre-computation of these functions at each grid point of the volume buffer. The resulting volumetric dataset can then be rendered from multiple viewpoints without recomputing the modeling function. As in other volume graphics techniques, accuracy is traded for speed, due to the resolution limit. Instead of accurately computing the function at each sample point, some type of interpolation from the precomputed grid values is employed.

9. Block Operations and Constructive Solid Modeling

The presortedness of the volume buffer naturally lends itself to grouping operations that can be exploited in various ways. For example, by generating multi-resolution volume hierarchy that can support time critical and space critical volume graphics applications can be better supported. The basic idea is similar to that of level-of-detail surface rendering which has recently proliferated [16, 25, 61, 65, 78], in which the perceptual importance of a given object in the scene determines its appropriate level-of-detail representation. One simple approach is the 3D "mip-map" approach [46, 63], where every level of the hierarchy is formed by averaging several voxels from the previous level. A better approach is based on sampling theory, in which an object is modeled with a sequence of alias-free volume buffers at different resolutions using the volume-sampled voxelization approach [23]. To accomplish this, high frequencies that exceed the Nyquist frequency of the corresponding volume buffer are filtered out by applying an ideal low-pass filter (*sinc*) with infinite support. In practice, the ideal filter is approximated by filters with finite support. Low sampling resolution of the volume buffer corresponds to a lower Nyquist frequency, and therefore requires a low-pass filter with wider support for good approximation.

As one moves up the hierarchy, low-pass filters with wider and wider support are applied. Compared to the level-of-detail hierarchy in surface graphics, the multi-resolution volume buffers are easy to generate and to spatially correspond neighboring levels, and are also free of object space aliasing. Furthermore, arbitrary resolutions can be generated, and errors caused by a non-ideal filter do not propagate and accumulate from level to level. Depending on the required speed and accuracy, a variety of low-pass filters (zero order, cubic, Gaussian) can be applied.

An intrinsic characteristic of the volume buffer is that adjacent objects in the scene are also represented by neighboring memory cells. Therefore, rasters lend themselves to various meaningful grouping-based operations, such as *bitblt* in 2D, or *voxblt* in 3D [37]. These include transfer of volume buffer rectangular blocks (cuboids) while supporting voxel-by-voxel operations between source and destination blocks. Block operations add a variety of modeling capabilities which aid in the task of image synthesis and form the basis for the efficient implementation of a 3D ‘‘room manager’’, which is the extension of window management to the third dimension.

Since the volume buffer lends itself to Boolean operations that can be performed on a voxel-by-voxel basis during the voxelization stage, it is advantageous to use CSG as the modeling paradigm. Subtraction, union, and intersection operations between two voxelized objects are accomplished at the voxel level, thereby reducing the original problem of evaluating a CSG tree during rendering time down to a 1D Boolean operation between pairs of voxels during a preprocessing stage.

For two point-sampled binary objects the Boolean operations of CSG or *voxblt* are trivially defined. However, the Boolean operations applied to volume-sampled models are analogous to those of fuzzy set theory (cf. [13]). The volume-sampled model is a density function $d(x)$ over R^3, where d is 1 inside the object, 0 outside the object, and $0 < d < 1$ within the "soft" region of the filtered surface. Some of the common operations, intersection, complement, difference, and union, between two objects A and B are defined as follows:

$$d_{A \bigcap B}(x) \equiv \min (d_A(x), d_B(x)) \tag{16}$$

$$d_{\bar{A}}(x) \equiv 1 - d_A(x) \tag{17}$$

$$d_{A-B}(x) \equiv \min (d_A(x), 1 - d_B(x)) \tag{18}$$

$$d_{A \bigcup B}(x) \equiv \max (d_A(x), d_B(x)) \tag{19}$$

The only law of set theory that is no longer true is the excluded-middle law (i.e., $A \bigcap \bar{A} \neq \phi$ and $A \bigcup \bar{A} \neq Universe$). The use of the min and max functions causes discontinuity at the region where the soft regions of the two objects meet, since the density value at each location in the region is determined solely by one of the two overlapping objects.

Complex geometric models can be generated by performing the CSG operations in Equations 16-19 between volume-sampled primitives. Volume-sampled models can also function as matte volumes [12] for various matting operations, such as performing cut-aways and merging multiple volumes into a single volume using the union operation. However, in order to preserve continuity on the cut-away boundaries between the material and the empty space, one should use an alternative set of Boolean operators based on algebraic sum and algebraic product [13, 20] :

$$d_{A \bigcap B}(x) \equiv d_A(x)\ d_B(x) \tag{20}$$

$$d_{\bar{A}}(x) \equiv 1 - d_A(x) \tag{21}$$

$$d_{A-B}(x) \equiv d_A(x) - d_A(x)\, d_B(x) \tag{22}$$

$$d_{A \bigcup B}(x) \equiv d_A(x) + d_B(x) - d_A(x)\, d_B(x) \tag{23}$$

Unlike the min and max operators, algebraic sum and product operators result in $A \bigcup A \neq A$, which is undesirable. A consequence, for example, is that during modeling via sweeping, the resulting model is sensitive to the sampling rate of the swept path [83].

Once a CSG model has been constructed in voxel representation, it is rendered in the same way any other volume buffer is. This makes, for example, volumetric ray tracing of constructive solid models straightforward [70] (see Figure 3).

10. Volume Sculpting

Surface-based sculpting has been studied extensively (e.g., [9, 66]), while volume sculpting has been recently introduced for clay or wax-like sculptures [17] and for comprehensive detailed sculpting [85]. The latter approach is a free-form interactive modeling technique based on the metaphor of sculpting and painting a voxel-based solid material, such as a block of marble or wood. There are two motivations for this approach. First, modeling topologically complex and highly-detailed objects are still difficult in most CAD systems. Second, sculpting has shown to be useful in volumetric applications. For example, scientists and physicians often need to explore the inner structures of their simulated or sampled datasets by gradually removing material.

Real-time human interaction could be achieved in this approach, since the actions of sculpting (e.g., carving, sawing) and painting are localized in the volume buffer, a localized rendering can be employed to reproject only those pixels that are affected. Carving is the process of taking a pre-existing volume-sampled tool to chip or chisel the object bit by bit. Since both the object and tool are represented as independent volume buffers, the process of sculpting involves positioning the tool with respect to the object and performing a Boolean subtraction between the two volumes. Sawing is the process of removing a whole chunk of material at once, much like a carpenter sawing off a portion of a wood piece. Unlike carving, sawing requires generating the volume-sampled tool on-the-fly, using a user interface. To prevent object space aliasing and to achieve interactive speed, 3D splatting is employed.

11. Surface Graphics vs. Volume Graphics

Contemporary 3D 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 geometry engines which have flourished in the past decade, making surface graphics the state-of-the-art in 3D 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 the image generation

system to repeat this process. Like vector graphics that did not support painting the interior of 2D objects, surface graphics generates merely the surfaces of 3D 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 3D volume buffer as a medium for the representation and manipulation of 3D scenes. A 3D scene is discretized earlier in the image generation sequence, and the resulting 3D discrete form is used as a database of the scene for manipulation and rendering purposes, which in effect decouples discretization from rendering. Furthermore, all objects are converted into one uniform meta-object − the voxel. Each voxel is atomic and represents the information about at most one object that resides in that voxel.

Volume graphics offers similar benefits to surface graphics, with several advantages that are due to the decoupling, uniformity, and atomicity features. The rendering phase is viewpoint independent and insensitive to scene complexity and object complexity. It supports Boolean and block operations and constructive solid modeling. When 3D sampled or simulated data are used, such as that generated by medical scanners (e.g., CT, MRI) or scientific simulations (e.g., CFD), volume graphic is suitable for their representation too. It is capable of representing amorphous phenomena and both the interior and exterior of 3D objects. These features of volume graphics as compared with surface graphics are discussed in detail in Section 12. 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. These weaknesses are discussed in detail in Section 13.

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 *bitblt* and quadttree. Raster graphics is also suitable for displaying 2D sampled digital images, and thus

Table 1: Comparison between vector graphics and raster graphics and between surface graphics and volume graphics.

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

provides the ideal environment for mixing digital images with synthetic graphic. Unlike vector graphics, raster graphics provides the capability to present shaded and textured surfaces, as well as 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 it requires for the frame buffer, as well as 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 3D data, such as 3D medical imaging and scientific visualization, in which the datasets 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 also used to contrast surface graphics and volume graphics.

12. Volume Graphics Features

One of the most appealing attributes of volume graphics is its insensitivity to the complexity of the scene, since all objects have been pre-converted into a finite size volume buffer. Although the performance of the pre-processing voxelization phase is influenced by the scene complexity [5, 30-32], rendering performance depends mainly on the constant resolution of the volume buffer and not on the number of objects in the scene. Insensitivity to the scene complexity makes the volumetric approach especially attractive for scenes consisting of a large number of 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 that are hard to render using conventional graphics systems. Examples of such objects include curved surfaces of high order and fractals which require expensive computation of an iterative function for each volume unit [51]. Constructive solid models are also hard to render by conventional methods, but are straightforward to render in volumetric representation (see below).

Anti-aliasing 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 3D texture image, has also a high complexity proportional to object complexity. In volume graphics, however, anti-aliasing, texture mapping, and solid texturing are performed only once - during the voxelization stage - where the color is calculated and stored in each voxel. The texture can also be stored as a separate volumetric entity which is rendered together with the volumetric object, as in the *VolVis* software system for volume visualization [1].

The textured objects in Figure 1, 2 and 3 have been assigned texture during the voxelization stage by

mapping each voxel back to the corresponding value on a texture map or solid. Once this mapping is applied, it is stored with the voxels themselves during the voxelization stage, which does not degrade the rendering performance. In addition, texture mapping and photo mapping are also viewpoint independent attributes, implying that once the texture is stored as part of the voxel value, texture mapping need not be repeated.

In anticipation of repeated access to the volume buffer (such as in animation), all viewpoint independent attributes can be precomputed during the voxelization stage, stored with the voxel, and be readily accessible for speeding up the rendering. The voxelization algorithm can generate for each object voxel its color, texture color, normal vector (for visible voxels), antialiasing information [82], and information concerning the visibility of the light sources from that voxel. Actually, the viewpoint independent parts of the illumination equation can also be 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 discrete ray tracing or a volumetric ray tracing algorithm can be engaged. Either ray tracing approach is especially attractive for complex surface scenes and constructive solid models, as well as 3D sampled or computed datasets (see below). Figure 3 shows an example of objects that were ray traced in discrete voxel space. In spite of the complexity of these scenes, volumetric ray tracing time was approximately the same as for much 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 the ones produced by conventional surface-based ray tracing can be generated by employing, accurate ray tracing, auxiliary object information, or screen supersampling techniques.

Sampled datasets, such as in 3D medical imaging (see Figure 3), volume microscopy, and geology, and simulated datasets, such as in computational fluid dynamics, chemistry, and materials simulation are often reconstructed from the acquired sampled or simulated points into a regular grid of voxels and stored in a volume buffer. Such datasets 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 datasets, as well as providing the volume buffer medium for intermixing sampled or simulated datasets with geometric objects [35], as can be seen in Figure 5. For compatibility between the sampled/computed data and the voxelized geometric object, the object can be volume sampled [82] with the same, but not necessarily the same, density frequency as the acquired or simulated datasets. In volume sampling the continuous object is filtered during the voxelization stage generating alias-free 3D density primitives. Volume graphics also naturally supports the rendering of translucent volumetric datasets (see Figures 1 and 5).

A central feature of volumetric representation is that unlike surface representation it is capable of representing inner structures of objects, which can be revealed and explored with the appropriate volumetric manipulation and rendering techniques. Natural objects as well as synthetic objects are likely to be solid rather than hollow. The inner structure is easily explored using volume graphics and cannot be supported by surface graphics (see Figure 5). Moreover, while translucent objects can be 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 in nature and do not contain any tangible surfaces (see Figure 1) [15, 28, 54].

An intrinsic characteristic of rasters is that adjacent objects in the scene are also represented by neighboring voxels. Therefore, rasters lend themselves to various meaningful block-based operations which can be performed during the voxelization stage. For example, the 3D counterpart of the *bitblt*

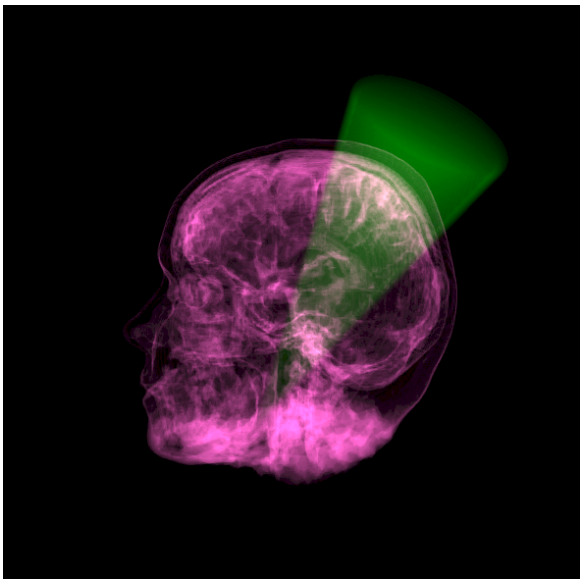

Figure 5: Intermixing of a volume-sampled cone with an MRI head using a union operation.

operations, termed *voxblt* (voxel block-transfer), can support transfer of cuboidal voxel blocks with a variety of voxel-by-voxel operations between source and destination blocks [37]. This property is very useful for *voxblt* and CSG. Once a CSG model has been constructed in voxel representation, it is rendered like any other volume buffer. This makes rendering of constructive solid models straightforward.

The spatial presortedness of the volume buffer voxels lends itself to other types of grouping or aggregation of neighboring voxels. For example, the terrain image shown in Figure 2 was generated by the voxel-based Hughes Aircraft Co. flight simulator [87]. It simulates a flight over voxel-represented terrain enhanced with satellite or aerial photo mapping with additional synthetic raised objects, such as buildings, trees, vehicles, aircraft, clouds and the like. Since the information below the terrain surface is invisible, terrain voxels can be actually represented as tall cuboids extending from sea level to the terrain height. The raised and moving objects, however, have to be represented in a more conventional voxel-based form.

Similarly, voxels can be aggregated into super-voxels in a pyramid-like hierarchy. For example, in a voxel-based flight simulator, the best resolution can be 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 deceases towards the horizon. A hierarchical volume buffer can be prepared in advance or on-the-fly by subsampling or averaging the appropriate size neighborhoods of voxels (see also [23]).

13. Weaknesses 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 256M bytes. However, since 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, memory will soon cease to be a stumbling block for volume graphics.

The extremely large throughput that has to be handled requires a special architecture and processing attention (see Section 14 and [36] Chapter 6). *Volume engines*, analogous to the currently available geometry (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 geometry engines (see Section 14). 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/sec), configured as accelerators or co-systems to existing geometry engines.

Unlike surface graphics, in volume graphics the 3D scene is represented in discrete form. This is the source of many of the problems of voxel-based graphics, which are similar to those of 2D rasters [14]. The finite resolution of the raster poses a limit on the accuracy of some operations, such as volume and area measurements, that are based on voxel counting.

Since the discrete data is sampled during rendering, a low resolution volume yields high aliasing artifacts. This becomes especially apparent when zooming in on the 3D raster. When naive rendering algorithms are used, holes may appear "between" voxels. Nevertheless, this can be alleviated in ways similar to those adopted by 2D raster graphics, such as employing either reconstruction techniques, a higher-resolution volume buffer, or volume sampling.

Manipulation and transformation of the discrete volume are difficult to achieve without degrading the image quality or losing some information. Rotation of rasters by angles other than 90 degrees is especially problematic since a sequence of consecutive rotations will distort the image. Again, these can be alleviated in ways similar to the 2D raster techniques.

Once an object has been voxelized, the voxels comprising the discrete object do not retain any geometric information regarding the geometric definition of the object. Thus, it is advantageous, when exact measurements are required (e.g., distance, area), 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 from a context of voxels. A variety of image-based and object-based methods for normal estimation from volumetric data has been devised (see [90], [36, Chapter 4]) and some have been discussed above. Most methods are based on fitting some type of a surface primitive to a small neighborhood of voxels.

A 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 [89]. The auxiliary geometric information might be useful also for re-voxelizing

the scene in case of a change in the scene itself.

14. Special-Purpose Volume Rendering Hardware

The high computational cost of direct volume rendering makes it difficult for sequential implementations and general-purpose computers to deliver the targeted level of performance. This situation is aggravated by the continuing trend towards higher and higher resolution datasets. For example, to render a dataset of 1024^3 16-bit voxels at 30 frames per second requires 2 GBytes of storage, a memory transfer rate of 60 GBytes per second and approximately 300 billion instructions per second, assuming only 10 instructions per voxel per projection. To address this challenge, researchers have tried to achieve interactive display rates on supercomputers and massively parallel architectures [50, 64, 67, 68, 80, 91]. However, most algorithms 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, nor will they in the near future, adequate latency and memory bandwidth for efficiently transferring the required large amounts of data. Furthermore, supercomputers seldom contain frame buffers and, due to their high cost, are frequently shared by many users.

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

Several researchers have proposed special-purpose volume rendering architectures [36, Chapter 6] [19, 26, 34, 48, 52, 76, 77, 88]. Most recent research has focused on accelerators for ray-casting of regular datasets. Ray-casting offers room for algorithmic improvements while still allowing for high image quality. Recent architectures [24] include VOGUE, VIRIM, and Cube.

VOGUE [43], a modular add-on accelerator, is estimated to achieve 2.5 frames per second for 256^3 datasets. For each pixel a ray is defined by the host computer and sent to the accelerator. The VOGUE module autonomously processes the complete ray, consisting of evenly spaced resampling locations, and returns the final pixel color of that ray to the host. Several VOGUE modules can be combined to yield higher performance implementations. For example, to achieve 20 projections per second of 512^3 datasets requires 64 boards and a 5.2 GB per second ring-connected cubic network.

VIRIM [21] is a flexible and programmable ray-casting engine. The hardware consists of two separate units, the first being responsible for 3D resampling of the volume using lookup tables to implement different interpolation schemes. The second unit performs the ray-casting through the resampled dataset according to user programmable lighting and viewing parameters. The underlying ray-casting model allows for arbitrary parallel and perspective projections and shadows. An existing hardware implementation for the visualization of $256 \times 256 \times 128$ datasets at 10 frames per second requires 16 processing boards.

The Cube project aims at the realization of high-performance volume rendering systems for large datasets and pioneered several hardware architectures. Cube-1, a first generation hardware prototype,

was based on a specially interleaved memory organization [33], which has also been used in all subsequent generations of the Cube architecture. This interleaving of the n^3 voxel enables conflict-free access to any ray parallel to a main axis of n voxels. A fully operational printed circuit board (PCB) implementation of Cube-1 is capable of generating orthographic projections of 16^3 datasets from a finite number of predetermined directions in real-time. Cube-2 was a single-chip VLSI implementation of this prototype [3].

To achieve higher performance and to further reduce the critical memory access bottleneck, Cube-3 introduced several new concepts [55-57]. 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. Estimated performance for arbitrary parallel and perspective projections is 30 frames per second for 512^3 datasets. Cube-4 [29, 58, 59] has only simple and local interconnections, thereby allowing for easy scalability of performance. Instead of processing individual rays, Cube-4 manipulates a group of rays at a time. As a result, the rendering pipeline is directly connected to the memory. Accumulating compositors replace the binary compositing tree. A pixel-bus collects and aligns the pixel output from the compositors. Cube-4 is easily scalable to very high resolution of 1024^3 16-bit voxels and true real-time performance implementations of 30 frames per second.

Enhancing the Cube-4 architecture, Mitsubishi Electric has derived EM-Cube (Enhanced Memory Cube-4). A system based on EM-Cube consists of a PCI card with four volume rendering chips, four 64Mbit SDRAMs to hold the volume data, and four SRAMs to capture the rendered image [53]. The primary innovation of EM-Cube is the block-skewed memory, where the volume memory is organized in subcubes (blocks) in such a way that all the voxels of a block are stored linearly in the same DRAM page. EM-Cube has been further developed into a commercial product where a volume rendering chip, called vg500, has been developed by Mitsubishi. It computes 500 million interpolated, Phong-illuminated, composited samples per second. The vg500 is the heart of a VolumePro PC card consisting of one vg500 and configurable standard SDRAM memory architectures. The first generation, available in 1999, supports rendering of a rectangular data set up to 256x256x256 12-bit voxels, in real-time 30 frames/sec [60].

Simultaneously, Japan Radio Co. has enhanced Cube-4 and developed a special-purpose architecture U-Cube. U-Cube is specifically designed for real-time volume rendering of 3D ultrasound data.

The choice of whether one adopts a general-purpose or a special-purpose solution to volume rendering depends upon the circumstances. If maximum flexibility is required, general-purpose appears to be the best way to proceed. However, an important feature of graphics accelerators is that they are integrated into a much larger environment where software can shape the form of input and output data, thereby providing the additional flexibility that is needed. A good example is the relationship between the needs of conventional computer graphics and special-purpose graphics hardware. Nobody would dispute the necessity for polygon graphics acceleration despite its obvious limitations. The same argument can be made for special-purpose volume rendering architectures.

15. Conclusions

The important concepts and computational methods of volume graphics have been presented. Although volumetric representations and visualization techniques seem more natural for sampled or computed data sets, their advantages are also attracting traditional geometric-based applications. This trend

implies an expanding role for volume visualization, and it has thus the potential to revolutionize the field of computer graphics, by providing an alternative to surface graphics, called volume graphics. We have introduced recent trends in volume visualization that brought about the emergence of volume graphics. Volume graphics has advantages over surface graphics by being viewpoint independent, insensitive to scene and object complexity, and lending itself to the realization of block operations, CSG modeling, and hierarchical representation. It is suitable for the representation of sampled or simulated datasets and their intermixing with geometric objects, and it supports the visualization of internal structures. The problems associated with the volume buffer representation, such as memory size, processing time, aliasing, and lack of geometric representation, echo problems encountered when raster graphics emerged as an alternative technology to vector graphics and can be alleviated in similar ways.

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 as well as for modeling and rendering synthetic scenes composed of surfaces.

Acknowledgments

Special thanks are due to Lisa Sobierajski, Rick Avila, Roni Yagel, Dany Cohen, Sid Wang, Taosong He, Hanspeter Pfister, and Lichan Hong who contributed to this paper, co-authored with me related papers [2, 38-40], and helped with the *VolVis* software. (*VolVis* can be obtained by sending email to: volvis@cs.sunysb.edu.) This work has been supported by the National Science Foundation under grant MIP-9527694 and a grant from the Office of Naval Research N000149710402. The MRI head data in Figure 5 is courtesy of Siemens Medical Systems, Inc., Iselin, NJ. Figure 2 is courtesy of Hughes Aircraft Company, Long Beach, CA. This image has been voxelized using voxelization algorithms, a voxel-based modeler, and a photo-mapper developed at Stony Brook Visualization Lab.

16. References

1. Avila, R., Sobierajski, L. and Kaufman, A., ‘‘Towards a Comprehensive Volume Visualization System’’, *Visualization ’92 Proceedings*, October 1992, 13-20.
2. Avila, R., He, T., Hong, L., Kaufman, A., Pfister, H., Silva, C., Sobierajski, L. and Wang, S., ‘‘VolVis: A Diversified Volume Visualization System’’, *Visualization ’94 Proceedings*, Washington, DC, October 1994, 31-38.
3. Bakalash, R., Kaufman, A., Pacheco, R. and Pfister, H., ‘‘An Extended Volume Visualization System for Arbitrary Parallel Projection’’, *Proceedings of the 1992 Eurographics Workshop on Graphics Hardware*, Cambridge, UK, September 1992.
4. Cline, H. E., Lorensen, W. E., Ludke, S., Crawford, C. R. and Teeter, B. C., ‘‘Two Algorithms for the Three-Dimensional Reconstruction of Tomograms’’, *Medical Physics*, **15**, 3 (May/June 1988), 320-327.
5. Cohen, D. and Kaufman, A., ‘‘Scan Conversion Algorithms for Linear and Quadratic Objects’’, in *Volume Visualization*, A. Kaufman, (ed.), IEEE Computer Society Press, Los Alamitos, CA, 1991, 280-301.
6. Cohen, D. and Shaked, A., ‘‘Photo-Realistic Imaging of Digital Terrain’’, *Computer Graphics Forum*, **12**, 3 (September 1993), 363-374.

7. Cohen-Or, D. and Kaufman, A., ‘‘Fundamentals of Surface Voxelization’’, *CVGIP: Graphics Models and Image Processing*, **56**, 6 (November 1995), 453-461.

8. Cohen-Or, D. and Kaufman, A., ‘‘3D Line Voxelization and Connectivity Control’’, *IEEE Computer Graphics & Applications*, 1997.

9. Coquillart, S., ‘‘Extended Free-Form Deformation: A Sculpturing Tool for 3D Geometric Modeling’’, *Computer Graphics*, **24**, 4 (August 1990), 187-196.

10. Dachille, F. and Kaufman, A., ‘‘Incremental Triangle Voxelization’’, submitted for publication, 1999.

11. Danielsson, P. E., ‘‘Incremental Curve Generation’’, *IEEE Transactions on Computers*, **C-19**, (1970), 783-793.

12. Drebin, R. A., Carpenter, L. and Hanrahan, P., ‘‘Volume Rendering’’, *Computer Graphics (Proc. SIGGRAPH)*, **22**, 4 (August 1988), 65-74.

13. Dubois, D. and Prade, H., *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, 1980.

14. Eastman, C. M., ‘‘Vector versus Raster: A Functional Comarison of Drawing Technologies’’, *IEEE Computer Graphics & Applications*, **10**, 5 (September 1990), 68-80.

15. Ebert, D. S. and Parent, R. E., ‘‘Rendering and Animation of Gaseous Phenomena by Combining Fast Volume and Scanline A-buffer Techniques’’, *Computer Graphics*, **24**, 4 (August 1990), 357-366.

16. Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M. and Stuetzle, W., ‘‘Multiresolution Analysis of Arbitrary Meshes’’, *SIGGRAPH’95 Conference Proceedings*, August 1995, 173-182.

17. Galyean, T. A. and Hughes, J. F., ‘‘Sculpting: An Interactive Volumetric Medeling Technique’’, *Computer Graphics*, **25**, 4 (July 1991), 267-274.

18. Glassner, A. S., ‘‘Space Subdivision for Fast Ray Tracing’’, *IEEE Computer Graphics and Applications*, **4**, 10 (October 1984), 15-22.

19. Goldwasser, S. M., Reynolds, R. A., Bapty, T., Baraff, D., Summers, J., Talton, D. A. and Walsh, E., ‘‘Physician’s Workstation with Real-Time Performance’’, *IEEE Computer Graphics & Applications*, **5**, 12 (December 1985), 44-57.

20. Goodman, J. R. and Sequin, C. H., ‘‘Hypertree: A Multiprocessor Interconnection Topology’’, *IEEE Transactions on Computers*, **C-30**, 12 (December 1981), 923-933.

21. Guenther, T., Poliwoda, C., Reinhard, C., Hesser, J., Maenner, R., Meinzer, H. and Baur, H., ‘‘VIRIM: A Mssivealy Parallel Processor for Real-Time Volume Visualization in Medicine’’, *Proceedings of the 9th Eurographics Hardware Workshop*, Oslo, Norway, September 1994, 103-108.

22. Hart, J. C., Sandin, D. J. and Kauffman, L. H., ‘‘Ray Tracing Deterministic 3-D Fractals’’, *Computer Graphics*, **23**, 3 (July 1989), 289-296.

23. He, T., Hong, L., Kaufman, A., Varshney, A. and Wang, S., ‘‘Voxel-Based Object Simplification’’, *IEEE Visualization ’95 Proceedings*, Los Alamitos, CA, October 1995, 296-303.

24. Hesser, J., Maenner, R., Knittel, G., Strasser, W., Pfister, H. and Kaufman, A., ‘‘Three Architectures for Volume Rendering’’, *Computer Graphics Forum*, **14**, 3 (August 1995), 111-122.

25. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W., ‘‘Mesh Optimization’’, *Computer Graphics (SIGGRAPH ’93 Proceedings)*, **27**, (August 1993), 19-26.

26. Jackel, D., ‘‘The Graphics PARCUM System: A 3D Memory Based Computer Architecture for Processing and Display of Solid Models’’, *Computer Graphics Forum*, **4**, (1985), 21-32.

27. Jones, M. W., ‘‘The Production of Volume Data from Triangular Meshes using Voxelisation’’, *Computer Graphics Forum*, **14**, 5 (December 1996), 311-318.

28. Kajiya, J. T. and Kay, T. L., ‘‘Rendering Fur with Three Dimensional Textures’’, *Computer Graphics*, **23**, 3 (July 1989), 271-280.

29. Kanus, U., Meissner, M., Strasser, W., Pfister, H., Kaufman, A., Amerson, R., Carter, R. J., Culbertson, B., Kuekes, P. and Snider, G., ‘‘Implementations of Cube-4 on the Teramac Custom Computing Machine’’, *Computers & Graphics*, **21**, 2 (1997), .

30. Kaufman, A. and Shimony, E., ‘‘3D Scan-Conversion Algorithms for Voxel-Based Graphics’’, *Proc. ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October 1986, 45-76.

31. Kaufman, A., ‘‘An Algorithm for 3D Scan-Conversion of Polygons’’, *Proc. EUROGRAPHICS’87*, Amsterdam, Netherlands, August 1987, 197-208.

32. Kaufman, A., ‘‘Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes’’, *Computer Graphics*, **21**, 4 (July 1987), 171-179.

33. Kaufman, A. and Bakalash, R., ‘‘Memory and Processing Architecture for 3-D Voxel-Based Imagery’’, *IEEE Computer Graphics & Applications*, **8**, 6 (November 1988), 10-23. Also in Japanese, *Nikkei Computer Graphics*, 3, 30, March 1989, pp. 148-160.

34. Kaufman, A. and Bakalash, R., ‘‘CUBE - An Architecture Based on a 3-D Voxel Map’’, in *Theoretical Foundations of Computer Graphics and CAD*, R. A. Earnshaw, (ed.), Springer-Verlag, 1988, 689-701.

35. Kaufman, A., Yagel, R. and Cohen, D., ‘‘Intermixing Surface and Volume Rendering’’, in *3D Imaging in Medicine: Algorithms, Systems, Applications*, K. H. Hoehne, H. Fuchs and S. M. Pizer, (eds.), June 1990, 217-227.

36. Kaufman, A., *Volume Visualization*, IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1991.

37. Kaufman, A., ‘‘The *voxblt* Engine: A Voxel Frame Buffer Processor’’, in *Advances in Graphics Hardware III*, A. A. M. Kuijk, (ed.), Springer-Verlag, Berlin, 1992, 85-102.

38. Kaufman, A., Cohen, D. and Yagel, R., ‘‘Volume Graphics’’, *IEEE Computer*, **26**, 7 (July 1993), 51-64. Also in Japanese, *Nikkei Computer Graphics*, 1, No. 88, 148-155 & 2, No. 89, 130-137, 1994.

39. Kaufman, A., Yagel, R. and Cohen, D., ‘‘Modeling in Volume Graphics’’, in *Modeling in Computer Graphics*, B. Falcidieno and T. L. Kunii, (eds.), Springer-Verlag, June 1993, 441-454.

40. Kaufman, A. and Sobierajski, L., ‘‘Continuum Volume Display’’, in *Computer Visualization*, R. S. Gallagher, (ed.), CRC Press, Boca Raton, FL, 1994, 171-202.

41. Kaufman, A., ‘‘Volume Visualization’’, *ACM Computing Surveys*, **28**, 1 (1996), 165-167.

42. Kaufman, A., ‘‘Volume Visualization’’, in *Handbook of Computer Science and Engineering*, A. Tucker, (ed.), CRC Press, 1996.

43. Knittel, G. and Strasser, W., ‘‘A Compact Volume Rendering Accelerator’’, *Volume Visualization Symposium Proceedings*, Washington, DC, October 1994, 67-74.

44. Lee, Y. T. and Requicha, A. A. G., ‘‘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’’, *Communications of the ACM*, **25**, 9

(September 1982), 635-650.

45. Levoy, M., ‘‘Display of Surfaces from Volume Data’’, *Computer Graphics and Applications*, **8**, 5 (May 1988), 29-37.

46. Levoy, M. and Whitaker, R., ‘‘Gaze-Directed Volume Rendering’’, *Computer Graphics (Proc. 1990 Symposium on Interactive 3D Graphics)*, **24**, 2 (March 1990), 217-223.

47. Lorensen, W. E. and Cline, H. E., ‘‘Marching Cubes: A High Resolution 3D Surface Construction Algorithm’’, *Computer Graphics*, **21**, 4 (July 1987), 163-170.

48. Meagher, D. J., ‘‘Applying Solids Processing Methods to Medical Planning’’, *Proceedings NCGA’85*, Dallas, TX, April 1985, 101-109.

49. Mokrzycki, W., ‘‘Algorithms of Discretization of Algebraic Spatial Curves on Homogeneous Cubical Grids’’, *Computers & Graphics*, **12**, 3/4 (1988), 477-487.

50. Molnar, S., Eyles, J. and Poulton, J., ‘‘PixelFlow: High-Speed Rendering Using Image Composition’’, *Computer Graphics*, **26**, 2 (July 1992), 231-240.

51. Norton, V. A., ‘‘Generation and Rendering of Geometric Fractals in 3-D’’, *Computer Graphics*, **16**, 3 (1982), 61-67.

52. Ohashi, T., Uchiki, T. and Tokoro, M., ‘‘A Three-Dimensional Shaded Display Method for Voxel-Based Representation’’, *Proceedings EUROGRAPHICS ’85*, Nice, France, September 1985, 221-232.

53. Osborne, R., Pfister, H., Lauer, H., McKenzie, N., Gibson, S., Hiatt, W. and Ohkami, H., ‘‘EM-Cube: an Architecture for Low-Cost Real-Time Volume Rendering’’, *Proceedings of the SIGGRAPH/Eurographics Hardware Workshop*, 1997, 131-138.

54. Perlin, K. and Hoffert, E. M., ‘‘Hypertexture’’, *Computer Graphics*, **23**, 3 (July 1989), 253-262.

55. Pfister, H., Wessels, F. and Kaufman, A., ‘‘Sheared Interpolation and Gradient Estimation for Real-Time Volume Rendering’’, *9th Eurographics Workshop on Graphics Hardware Proceedings*, Oslo, Norway, September 1994.

56. Pfister, H., Kaufman, A. and Chiueh, T., ‘‘Cube-3: A Real-Time Architecture for High-resolution Volume Visualization’’, *Volume Visualization Symposium Proceedings*, Washington, DC, October 1994, 75-82.

57. Pfister, H., Wessels, F. and Kaufman, A., ‘‘Sheared Interpolation and Gradient Estimation for Real-Time Volume Rendering’’, *Computers & Graphics*, **19**, 5 (September 1995), 667-677.

58. Pfister, H., Kaufman, A. and Wessels, F., ‘‘Towards a Scalable Architecture for Real-Time Volume Rendering’’, *10th Eurographics Workshop on Graphics Hardware Proceedings*, Maastricht, The Netherlands, August 1995.

59. Pfister, H. and Kaufman, A., ‘‘Cube-4: A Scalable Architecture for Real-Time Volume Rendering’’, *Volume Visualization Symposium Proceedings*, San Francisco, CA, October 1996, 47-54.

60. Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H. and Seiler, L., ‘‘The VolumePro Real-Time Ray-Casting System’’, *Proceedings SIGGRAPH’99*, August 1999.

61. Rossignac, J. and Borrel, P., ‘‘Multi-Resolution 3D Approximations for Rendering Complex Scenes’’, in *Modeling in Computer Graphics*, B. Falcidieno and T. L. Kunni, (eds.), Springer-Verlag, 1993, 455-465.

62. Sabella, P., ‘‘A Rendering Algorithm for Visualizing 3D Scalar Fields’’, *Computer Graphics (Proc. SIGGRAPH)*, **22**, 4 (August 1988), 160-165.

63. Sakas, G. and Hartig, J., ‘‘Interactive Visualization of Large Scalar Voxel Fields’’, *Proceedings Visualization ’92*, Boston, MA, October 1992, 29-36.

64. Schroder, P. and Stoll, G., ‘‘Data Parallel Volume Rendering as Line Drawing’’, *Workshop on Volume Visualization*, Boston, MA, October 1992, 25-32.

65. Schroeder, W. J., Zarge, J. A. and Lorensen, W. E., ‘‘Decimation of Triangle Meshes’’, *Computer Graphics*, **26**, 2 (July 26-31 1992), 65-70.

66. Sederberg, T. W. and Parry, S. R., ‘‘Free-Form Deformation of Solid Geometry Models’’, *Computer Graphics*, **20**, 4 (August 1986), 151-160.

67. Silva, C. and Kaufman, A., ‘‘Parallel Performance Measures for Volume Ray Casting’’, *Visualization ’94 Proceedings*, Washington, DC, October 1994, 196-203.

68. Silva, C. T., Kaufman, A. and Pavlakos, C., ‘‘PVR: High-Performance Volume Rendering’’, *IEEE Computational Science & Engineering*, **3**, 4 (December 1996), 16-28.

69. Snyder, J. M. and Barr, A. H., ‘‘Ray Tracing Complex Models Containing Surface Tessellations’’, *Computer Graphics*, **21**, 4 (July 1987), 119-128.

70. Sobierajski, L. and Kaufman, A., ‘‘Volumetric Ray Tracing’’, *Volume Visualization Symposium Proceedings*, Washington, DC, October 1994, 11-18.

71. Speray, D. and Kennon, S., ‘‘Volume Probes: Interactive Data Exploration on Arbitrary Grids’’, *Computer Graphics*, **24**, 5 (November 1990), 5-12.

72. Sramek, M. and Kaufman, A., ‘‘Object Voxelization by Filtering’’, *ACM/IEEE Volume Visualization ’98 Symposium Proceedings*, October 1998, 111-118.

73. Sramek, M., ‘‘Visualization of Volumetric Data by Ray Tracing’’, *Proceedings Symposium on Volume Visualization*, Austria, 1998.

74. Sramek, M. and Kaufman, A., ‘‘Alias-free Voxelization of Geometric Objects’’, *IEEE Transactions on Visualization and Computer Graphics*, 1999.

75. Sramek, M. and Kaufman, A., ‘‘vxt: a C++ Class Library for Object Voxelization’’, *International Workshop on Volume Graphics*, Swansea, UK, March 1999, 295-306.

76. Stytz, M. R., Frieder, G. and Frieder, O., ‘‘Three-Dimensional Medical Imaging: Algorithms and Computer Systems’’, *ACM Computing Surveys*, December 1991, 421-499.

77. Stytz, M. R. and Frieder, O., ‘‘Computer Systems for Three-Dimensional Diagnostic Imaging: An Examination of the State of the Art’’, *Critical Reviews in Biomedical Engineering*, August 1991, 1-46.

78. Turk, G., ‘‘Re-Tiling Polygonal Surfaces’’, *Computer Graphics*, **26**, 2 (July 26-31 1992), 55-64.

79. Upson, C. and Keeler, M., ‘‘V-BUFFER: Visible Volume Rendering’’, *Computer Graphics (Proc. SIGGRAPH)*, 1988, 59-64.

80. Vezina, G., Fletcher, P. A. and Robertson, P. K., ‘‘Volume Rendering on the MasPar MP-1’’, *Workshop on Volume Visualization*, Boston, MA, October 1992, 3-8.

81. Wan, M., Qu, H. and Kaufman, A., ‘‘Virtual Flythrough over a Voxel-Based Terrain’’, *IEEE Virtual Reality Conference Proceedings*, March 1999, 53-60.

82. Wang, S. and Kaufman, A., ‘‘Volume Sampled Voxelization of Geometric Primitives’’, *Visualization ’93 Proceedings*, San Jose, CA, October 1993, 78-84.

83. Wang, S. and Kaufman, A., ‘‘Volume-Sampled 3D Modeling’’, *IEEE Computer Graphics & Applications*, **14**, 5 (September 1994), 26-32.

84. Wang, S. and Kaufman, A., ‘‘3D Antialiasing’’, Technical Report 94.01.03, Computer Science, SUNY Stony Brook, January 1994.

85. Wang, S. and Kaufman, A., ‘‘Volume Sculpting’’, *ACM Symposium on Interactive 3D Graphics*, Monterey, CA, April 1995, 151-156.

86. Westover, L., ‘‘Footprint Evaluation for Volume Rendering’’, *Computer Graphics (Proc. SIGGRAPH)*, **24**, 4 (August 1990), 144-153.

87. Wright, J. and Hsieh, J., ‘‘A Voxel-Based, Forward Projection Algorithm for Rendering Surface and Volumetric Data’’, *Proceedings Visualization ’92*, Boston, MA, October 1992, 340-348.

88. Yagel, R. and Kaufman, A., ‘‘The Flipping Cube Architecture’’, Tech. Rep. 91.07.26, Computer Science, SUNY at Stony Brook, July 1991.

89. Yagel, R., Cohen, D. and Kaufman, A., ‘‘Discrete Ray Tracing’’, *IEEE Computer Graphics & Applications*, **12**, 5 (September 1992), 19-28.

90. Yagel, R., Cohen, D. and Kaufman, A., ‘‘Normal Estimation in 3D Discrete Space’’, *The Visual Computer*, June 1992, 278-291.

91. Yoo, T. S., Neumann, U., Fuchs, H., Pizer, S. M., Cullip, T., Rhoades, J. and Whitaker, R., ‘‘Direct Visualization of Volume Data’’, *IEEE Computer Graphics & Applications*, **12**, 4 (July 1992), 63-71.