A Survey of Algorithms for Volume Visualization

T. Todd Elvins
Advanced Scientific Visualization Laboratory
San Diego Supercomputer Center

"*... in 10 years, all rendering will be volume rendering.*" Jim Kajiya at SIGGRAPH '91

Many computer graphics programmers are working in the area of scientific visualization. One of the most interesting and fast-growing areas in scientific visualization is volume visualization. Volume visualization systems are used to create high-quality images from scalar and vector datasets defined on multi-dimensional grids, usually for the purpose of gaining insight into a scientific problem. Most volume visualization techniques are based on one of about five foundation algorithms. These algorithms, and the background necessary to understand them, are described here. Pointers to more detailed descriptions, further reading, and advanced techniques are also given.

Introduction

The following is an introduction to the fast-growing field of volume visualization for the computer graphics programmer. Many computer graphics techniques are used in volume visualization. The computer graphics techniques themselves will not be discussed, but the way these techniques are applied in fundamental volume visualization algorithms will be explained. Advantages and disadvantages of each algorithm will be covered along with a rough idea of the algorithm's space and time requirements, ease of implementation, and the type of data it handles best.

Most volume visualization algorithms follow similar steps. These steps will be discussed before any specific algorithms are covered. The discussion will also explain the terms, procedures, and heuristics most often used in the field of volume visualization. Although there is not complete agreement on a standard volume visualization vocabulary yet, a nomenclature is approaching consensus. Where multiple terms exist for the same concept, alternative terms are listed in parentheses. New procedures and heuristic approaches to short-cut volume visualization's high CPU and large memory requirements are a frequent occurrence. Some of the most popular procedures and approaches will be discussed.

Volume visualization is too large and is growing too fast for complete coverage in this paper. The reader is encouraged to consult the papers listed in the bibliography for information on specific algorithms, scientific applications, commercial projects, and advanced topics. Some of the topics not covered in this paper include: special purpose hardware, methods for handling non-Cartesian data, advanced algorithm optimizations and enhancements, user interfaces, and commercial and public domain implementations.

It should be kept in mind that, although not specifically discussed, animation is critical to the volume visualization process. Without animating rendered volumetric images, the system user will usually have a difficult time discerning three-dimensional information from two-dimensional imagery. Animation is often a straight-forward extension to the topics covered here.

The following introduction to a wide range of volume visualization algorithms will give the reader a solid starting point for getting involved in this relatively new field. Other technical perspectives at the introductory level can be found in [Dreb88] [Levo90e] [Kauf91] [Wilh91a]. A good non-technical introduction is given in [Fren89]. Advanced topics in medical volume visualization are covered in [Hohn90][Levo90c].

Furthering scientific insight

This section introduces the reader to the field of volume visualization as a subfield of scientific visualization and discusses many of the current research areas in both.

Challenges in scientific visualization

Scientific visualization uses computer graphics techniques to help give scientists insight into their data [McCo87] [Brod91]. Insight is usually achieved by extracting scientifically meaningful information from numerical descriptions of complex phenomena through the use of interactive imaging systems. Scientists need these systems not only for their own insight, but also to share their results with their colleagues, the institutions that support the scientist's research, and the general public. Few technical articles are published without captioned data visualizations of some sort.

The most active subfield of scientific visualization is volume visualization. Volume visualization is the process of projecting a multidimensional dataset onto a two-dimensional image plane for the purpose of gaining an understanding of the structure (or lack of structure) contained within the volumetric data. Most often the dataset is defined on a three-dimensional lattice with one or more scalar values, and possibly one or more vector values at each gridpoint on the lattice. Methods for visualizing higher-dimensional grids and/or irregular grids are relatively unknown.

To be useful, volume visualization techniques must offer understandable data representations, quick data manipulation, and reasonably fast rendering. Scientific users should be able to change parameters and see the resultant image instantly. Few present day systems are capable of this type of performance; therefore volume visualization algorithm development and refinement are important areas of study. Understanding the fundamental algorithms is requisite for such endeavors.

Challenges and applications of volume visualization

Many challenges still exist in the field of volume visualization [DeFa89]. The typical size of a volume dataset is several megabytes, sometimes hundreds of megabytes, and gigabyte datasets are just waiting for the hardware that can handle them. Many scientists would like to be able to combine all of the interesting aspects of two or more volumetric datasets into one image. Creating static images from volumes of vector data is also an unsolved problem. Not all volume visualization techniques work well with all types of data and current data-classification tools are rudimentary. Amorphous datasets describing wispy, cloud-like structures are particularly difficult to render. These are just a few of the hurdles yet to be dealt with in the field of volume visualization.

Despite these difficulties, macro and micro scientists are still finding many new ways to use volume visualization. Volume visualization is widely used in the medical field as well as in

geoscience, astrophysics, chemistry, microscopy, mechanical engineering, non-destructive testing, and many other scientific and engineering areas. The most common units of measure recorded in data volumes include density, pressure, temperature, electrostatic charge, and velocity. Multiple values in each of these measures could be stored at every gridpoint in one volume. For example, atmospheric scientists often record thirty or more parameters at every gridpoint (images that attempt to display more than one parameter at a time, however, are difficult to interpret). One need only look around to find new volumes to measure.

Data characteristics

Sources of volume data

Scientists sometimes use volume visualization to compare numerical results derived from empirical experiments with results derived from simulations of the empirical event. Finite-element analysis or computational fluid dynamics programs are often used to simulate events from nature. If an event is too big, too small, too fast, or too slow to record in nature, then only the simulated event data volumes can be studied.

Volume datasets are often acquired by scanning the material-of-interest using Magnetic Resonance Imaging (MRI), Computer-aided Tomography (CT), Positron Emission Tomography (PET), and/or Sonogram machines. Laser scan confocal, and other high power microscopes are also used to acquire data.

Volume data can be generated by *voxelizing* geometric descriptions of objects, sculpting digital blocks of marble, hand-painting in three-dimensions with a wand, or by writing programs that generate interesting volumes using stochastic methods. Some of these methods will be discussed in more detail.

All of these datasets can be treated similarly even though they are generated by diverse means. The amount of structure in the dataset usually determines which volume visualization algorithm will create the most informative images.

Alternatives to single scalar data

Vector data is sometimes visualized one slice at a time with arrows at each gridpoint. The direction of the arrow indicates the direction of the vector, and the color of the arrow usually displays the vector's magnitude, although it could display any other scalar value. Alternatively, animations using streamers, ribbons, tufts, particles, and other time-dependent mechanisms do an adequate job of showing the data characteristics of a collection of slices. Tensor data has also been visualized with some success. Finding effective means for visualizing vector and tensor data is still, however, an open area of research.

Particular data acquisition devices, such as CT and MRI scanners, are good at sampling a specific characteristic of a substance. This characteristic is usually different for each kind of device used. For example, MRI scans show soft tissue tumors that do not show up in CT scans. By registering (spatially aligning) two datasets sampled from the same substance, and then using an algorithm that refers to the most appropriate of the two datasets at every gridpoint, either during rendering or as a post-process compositing step, images can be created that show the most important characteristics of both datasets. This bi-modal data approach has also been applied to comparing simulation-generated data and acquired data. Visualizing registered data is an active area of research.

The algorithms that will be discussed here primarily deal with visualizing scalar data volumes. However, some of the techniques can be extrapolated to vector, tensor, bi-modal, and higher-dimensional data visualization.

Volume characteristics

Voxels and cells

Volumes of data are usually treated as either an array of volume elements (*voxels*) or an array of cells. These two approaches stem from the need to resample the volume between gridpoints during the rendering process. Resampling — requiring interpolation — occurs in almost every volume visualization algorithm. Since the underlying function is not usually known, and it is not known whether the function was sampled above the Nyquist frequency, it is impossible to check the reliability of the interpolation used to find data values between discrete gridpoints. It must be assumed that common interpolation techniques are valid for an image to be considered valid.

The voxel approach dictates that the area around a gridpoint has the same value as the gridpoint. A voxel is therefore a hexahedral area of non-varying value surrounding a central gridpoint. In some algorithms the contribution of the voxel to the image drops as the distance from the center of the region of influence increases. In other algorithms the voxel has constant contribution across the region of influence. The voxel approach has the advantage that no assumptions are made about the behavior of data between gridpoints, ie., only known data values are used for generating an image.

The cell approach views a volume as a collection of hexahedra whose corners are gridpoints and whose value varies between the gridpoints. This technique attempts to estimate values inside the cell by interpolating between the values at the corners of the cell. Trilinear and tricubic are the most commonly used interpolation functions. Images generated using the cell approach appear smoother than those images created with the voxel approach. However, the validity of the cell-based images cannot be verified. The splatting algorithm to be discussed in the section on volume visualizatio algorithms performs data reconstruction in image space to avoid the interpolation issue altogether.

Hereafter, the term *element* will be used when the voxel and cell approaches can be used interchangeably.

Grids and lattices

Not all elements are cubes and not all volumes are defined on *Cartesian* grids. The following explanation of non-Cartesian grids is based on [Sper90] and is in order of increasing generality. On a Cartesian grid, all of the cells are identical axis-aligned cubes.

The next most general lattice is called a *regular* grid where all of the elements are identical axis-aligned rectangular prisms, but the elements are not cubes. Elements on a *rectilinear* grid are not necessarily identical but they are still hexahedra and axis-aligned. Elements on a *structured* grid are non-axis-aligned hexahedra (warped bricks), and a *block structured* grid is a collection of various structured grids sewn together to fill a space. Spherical and curvilinear lattices are examples of structured grids. An *unstructured* volume is made up of polyhedra with no implicit connectivity. Cells can be tetrahedra, hexahedra, prisms, pyramids, etc., and do not necessarily have to have planar faces. A volume of tetrahedral elements is a popular type of unstructured volume because any volume can be decomposed into tetrahedra and tetrahedra have planar faces. A *hybrid* volume is a collection of any of the mentioned grids sewn together to fill a space.

Non-Cartesian grids are beyond the scope of this paper but are mentioned here for completeness and to enforce the notion that much of volume visualization is still unsolved. Several papers on the subject of rendering data defined on non-Cartesian grids can be found in [Engl90].

Common steps in volume visualization algorithms

Volume visualization steps

Many steps in the volume visualization process are common to volume visualization algorithms. (See [Kauf91] for a detailed explanation of all of the steps.) Most of the fundamental volume visualization algorithms include only a subset of the steps listed here.

The initial step in every procedure is data acquisition. The next common step is to put the data slices into a form that can be worked with and then to process each slice so that it covers a good distribution of values, is high in contrast, and is free of noise and out-of-range values. Of course, the same set of processes must be applied to every slice.

Next, the dataset is reconstructed so that the ratio of the dimensions is proportional to the ratio of the dimensions of the measured substance or substance being simulated. This may involve interpolating between values in adjacent slices to construct new slices, replicating existing slices, interpolating to estimate missing values, or scan-converting an irregular grid or non-orthogonal grid onto a Cartesian grid by interpolation. At this point three-dimensional enhancement may be applied, such as a slight blurring of the data values. Next, a data-classification or thresholding is performed. This step will be discussed in detail in the section on data classification.

After data-classification, a mapping operation is applied to map the elements into geometric or display primitives. This is the stage that varies the most between algorithms, as will be shown in the section on volume visualization algorithms. At this point the primitives can be stored, manipulated, intermixed with externally defined primitives, shaded, transformed to screen space, and displayed. The shading and transformation steps can be reordered and shading can be done in one of eleven ways as explained in [Kauf91].

Volume visualization methods

The fundamental volume visualization algorithms described below fall into two categories, *direct volume rendering* (DVR) algorithms and *surface-fitting* (SF) algorithms. DVR algorithms include approaches such as ray-casting, integration methods, splatting, and V-buffer rendering. The two latter methods are sometimes called *projection methods* [Wilh91a][Wilh91b]. DVR methods are characterized by mapping elements directly into screen space without using geometric primitives as an intermediate representation. DVR methods are especially appropriate for creating images from datasets containing amorphous features like clouds, fluids, and gases. One disadvantage of using DVR methods is that the entire dataset must be traversed each time an image is rendered. A low resolution pass or random sampling of the data is sometimes used to create low-quality images quickly for parameter checking. The process of successively increasing the resolution and quality of a DVR image over time is called *progressive refinement.*

SF algorithms (sometimes called feature-extraction or iso-surfacing) typically fit (usually planar) surface primitives such as polygons or patches to constant-value contour surfaces in volumetric datasets. The user begins by choosing a threshold value and then geometric primitives are automatically fit to the high-contrast contours in the volume that match the threshold. Cells whose corner-values are all above the chosen threshold (cell is inside) or all below the threshold (cell is outside) are discarded and have no effect on the final image. Showing just the cells falling *on* the threshold is sometimes useful, but can be a problem. Another consideration is the huge number of surface primitives generated for large volumetric datasets.

The SF approach includes contour-connecting, marching cubes, marching tetrahedra, dividing cubes, and others. SF methods are typically faster than DVR methods since SF methods only traverse the volume once to extract surfaces. After extracting the surfaces, rendering hardware and well-known rendering methods can be used to quickly render the surface primitives each time the user changes a viewing or lighting parameter. Changing the SF threshold value is time consuming because it requires that all of the cells be revisited to extract a new set of surface primitives.

SF methods suffer from several problems such as occasional false positive and negative surface pieces, and incorrect handling of small features and branches in the data. Note that such artifacts could be incorrectly viewed by scientists as features in the data.

Sometimes it is desirable to add geometric objects to a volumetric scene. A good example of this is in radiation treatment planning [Levo90c], where tumors are volume rendered from MRI datasets, and tentative radiation beam paths, defined as geometric cylinders or cones, are added to the scene to ensure that the beam paths only intersect within the tumor. There are at least two ways to mix element data and externally defined geometric primitives in the same object space and then render them to the same image space. The first way is to use a SF algorithm to find the iso-surface approximating the shape and location of the tumor. This geometry is then rendered at the same time and in the same scene as the beam path geometry so that the final image shows both the proper hidden surfaces and shading. The second way is to *voxelize* the beam path geometry [Kauf86]. Voxelizing the beam path cylinders means scan-converting the three-dimensional geometry into either the tumor data volume or a separate "empty" volume. The volume containing the scan-converted beams then has elements that contain a "beam-is-present" value if the cylinder intersects the element and contain another value if the cylinder does not intersect the element. If the cylinders are scan-converted into a separate volume, then corresponding elements from the two volumes are always considered together. When the volume(s) are then rendered using a DVR technique, the cylinders and tumor will appear in the correct positions. Combining volume and geometric primitives is proving to be useful in a number of scientific and medical applications.

Data classification

Data classification is the most difficult function that a volume visualization user has to perform. Data classification means choosing a threshold if a SF algorithm is to be used (see previous section), or choosing color (brightness) and opacity (light attenuation) values to go with each possible range of data values if a DVR algorithm is to be used. A DVR's color table is for mapping data values to meaningful colors. The DVR opacity table is used to expose that part of the volume most interesting to the user and to make transparent those parts that are not interesting.

In DVR each element in a volume contributes color to the final image. The amount of color contributed by an element is determined by the color and opacity of the current element and the colors and opacities of all of the elements between the current element and the image plane. To get a reasonable image, the user must choose the tables with care. Visualization programmers are responsible for ensuring that interpolated values cannot map through the table's transfer functions to non-existent materials or incorrect colors.

Color and opacity tables are usually created by the user after they have explored a slice of data with a probe that prints out data values. Ranges of data values are entered by the user along with preliminary color and opacity values, then a test image is rendered. Based on the test image, the user adjusts the value ranges and the corresponding color and opacity values, then renders a second test image. This process is repeated many times until an acceptable

image is generated. Example values from tables used to render CT data would map bone density values to white/opaque, muscle density values to red/semi-transparent, and fat density values to beige/mostly-transparent. Slight changes in opacity values often have an unexpectedly large impact on the rendered image.

Other factors that come into play in the classification stage are the user's knowledge of the material (or simulated material) being classified, the position of the classified elements in the volume, and the material occupancy of the element. If a user is familiar with the material being classified, the table specification is a matter of typing in some numbers. If the user is not familiar with the material, classification may require extensive data exploration or consultation with a person knowledgeable in the characteristics of the material.

It is sometimes desirable to make outer layers of a substance more transparent than inner layers, even though the layers have the same data value. This is why the position of the element is sometimes taken into account during the rendering process. Material occupancy [Dreb88] is the amount of a material contained in an element. If an element is near the interface between two substances, such as bone and muscle, then the element may contain some of each substance and should make a contribution to the image that reflects contributions from both substances. Most volume visualization algorithms save time by disregarding the negligible effect of partial material occupancy and not implementing the multiple-layers-of-substance feature.

Traversals

Once a scientific dataset has been classified, images are typically created in one of two ways: either using an image-order traversal of the pixels in the image plane or an object-order traversal of the elements in the volumetric dataset. Object-order traversals operate by calculating, for each element, the projection and contribution of that element to the pixels in the image plane. Most SF and some DVR algorithms use object-order traversals and some algorithms use a combination of both types of traversal.

Object-order traversals can proceed through the volume either front-to-back or back-to-front. The advantage of front-to-back traversals is that the elements in the back do not have to be traversed if the elements in the front have already created a completely opaque image. The advantage of back-to-front traversals is that the user can watch the image progressing, seeing the structures in the back that will eventually be hidden by the structures in the front.

Image-order traversals usually proceed in scanline order. Alternatively, pixels can be processed in a random order so that the user can watch the image being refined as the missing pixels are filled in.

Viewing and shading

Most DVR algorithms use primarily orthographic viewing since implementing perspective viewing in DVR is fraught with ray divergence problems. SF algorithms use geometric primitives that allow either parallel or perspective views. Using orthographic views for scientific visualization assures that scientists will not be confused by seeing their data warped by the perspective transformation. When perspective is not used, other depth cues such as animation, depth-fog attenuation, depth-brightness attenuation, and shading are sometimes introduced.

Most DVR and SF algorithms use *gradient shading*. A central difference formula is applied at every gridpoint to find the gradient; the measure of change across adjacent elements. Gradients within a cell can be estimated by interpolating between the gradients at the corners of the cell. The gradient at a point in a volume can be used to approximate the normal to an imaginary surface passing through the point [Lore87]. Most standard graphics shading models can be applied in shading elements once an approximate normal is known. Ambient, diffuse, and sometimes specular lighting components are used in the volume visualization shading process.

Some volume visualization algorithms precalculate dot products of the light source vector and the normal (gradient) vector, then store the result at the gridpoint so it only has to be calculated once. Some other algorithms do heuristic shading with a table lookup. SF algorithms often refer back to the gradient information at a point in the original volume to shade an extracted surface primitive at that point. Gradient shading is usually more accurate than using the normals of the surface primitives. As in other computer graphics applications, shading is an important factor in creating understandable volume images.

Photorealism

There is debate in the volume visualization community about whether objects in volume images should appear to be made from physically plausible materials or not. Materials that do not exist in nature, such as light-emitting gels, may make images difficult to interpret since scientists have no experience with the materials. In [Levo90b] it is argued that rendering volumes so that they are made of non-plausible materials is an error. "Visualization should simulate physically plausible objects — no matter how contrived they may be." Non-plausible objects are not readily interpreted, and it is difficult for scientists to gain insight from the non-intuitive images. In [Wilh91a] a strong case is presented for modeling volumes as light-emitting gels; the mapping to color and opacity is very flexible, and the model more abstractly represents the information content of the volume. Both are good arguments, and the jury is still out on this decision.

Volume visualization algorithms

This section describes five of the most commonly-used fundamental volume visualization algorithms and their close relatives. Descriptions of the algorithms include: what type of data they best handle, their advantages and disadvantages, a rough idea of how well they parallelize, their space and time requirements, a few possible optimizations and enhancements, and references for further reading. Table 1 shows a taxonomy of the algorithms described here and how each algorithm fits into the field of volume visualization.

The terms *voxel* and *cell* are not interchangeable in this section.

Volume Visualization Algorithms		
Surface-fitting	Direct Volume Rendering	
	Projection methods	Image-order methods
Opaque cubes	V-buffer	Ray-casting
(Cuberille)	Splatting	Cell integration
Contour connecting	Pixar slice shearing	Sabella method
Marching cubes		
Dividing cubes		
Marching tetrahedra		

Table 1. Algorithms in the field of volume visualization

Contour-connecting

One of the first-invented methods for generating volume visualization images was called *contour-connecting*, and descendents of the contour-connecting algorithm are still in use in many disciplines. The basic idea of tracing one closed contour in each slice of data and then connecting contours in adjacent slices of data was suggested by [Kepp75] and refined by [Fuch77], [Ekou91], and many others.

Contour-connecting is an object-order SF method that begins by operating on each slice of data individually. After the user has specified a threshold value, a closed-curve contour at this value is found for each data slice. Originally, the contour-tracing step was performed by hand. Image processing techniques now provide methods for automatically generating the contours, but human intervention is sometimes still required for low-contrast data.

Once the slice contours are found, the problem becomes one of finding an optimal tesselation, usually of triangles, connecting the curves in each two adjacent slices. Keppel reduced this problem to finding a path in a directed graph using heuristics [Kepp75], and Fuchs, et al, further specified the problem as that of finding a minimum cost path in a directed toroidal graph [Fuch77]. Both of these solutions find an approximation to the surface passing through high-gradient threshold cells in the data.

The last step is to select viewing, lighting, and rendering parameters, and to pass the strips of triangles to a surface renderer. Contour-connecting suffers from most of the problems endemic to SF algorithms specified in the description of volume visualization methods. Advantages of this approach include the simplicity of the algorithm and the plethora of known surface-rendering methods. The surface-fitting portion of this algorithm is parallelizable since no two adjacent slices are dependent on the result of another pair.

Opaque cube

Another straight-forward volume visualization procedure is called the *opaque cube* or *cuberille* algorithm. It was originated by [Herm79] and has been enhanced and optimized by many others.

This algorithm proceeds in two stages. In the first stage, the user selects a threshold value, and an object-order traversal of the volume is made, searching for cells with corner-values that straddle the threshold value. Six polygons, one for each face of the cell, are generated for each such cell. Stage two passes the geometric descriptions of these polygons to a surface renderer for image generation. If multiple thresholds are chosen, then each corresponding set of cell polygons can be rendered in a different color (and with a different opacity) to differentiate them.

The opaque, or semi-opaque, cubes used to represent the iso-value contour surface, will cause the surface to appear blocky in the rendered image. The blocky appearance of the approximated iso-surface can be reduced by using gradient shading in the rendering step.

Opaque cube algorithms suffer from most of the deficiencies of SF algorithms mentioned in the description of volume visualization methods and are especially bad at showing small features of the data. The algorithms are, however, simple to implement and fast at finding and rendering surfaces.

If the cells are traversed back-to-front, the cell's surface primitives can be rendered into an image buffer as they are generated. Surface primitives in the front will occlude primitives in the back as in the painter's algorithm. Note also that step one of the cuberille algorithm can be parallelized because all of the cells can be tested and tesselated independently.

Marching cubes

In [Wyvi86] table-based cell surface-fitting is performed using polygons that are eventually tesselated into triangles. In [Lore87] a table-based surface-fitting procedure is described that fits up to four triangles to each cell. The latter algorithm goes by the name *marching cubes*. The marching cubes algorithm has been widely implemented and proceeds by reading four data slices into memory, finding the gradients at all of the interior gridpoints, marching through all of the interior cells, and then fitting small triangles within each cell through which a threshold-value surface passes. These triangles are then passed to a rendering program that maps them to image space.

Before the algorithm begins, the user specifies a threshold value. The algorithm then loops on each successive group of four adjacent data slices. The slices are read into memory, gradients are calculated, and each cell between the middle two slices is scanned to find if its corner-values straddle the threshold value. Non-straddling cells are disregarded.

Cells that do straddle the threshold are more closely examined. The eight corners of the cube are numbered one through eight and valued "1" if they are above the threshold and "0" if they are below the threshold. The eight values are then put in eight consecutive bit locations (0-7) to form an eight bit byte. This byte is treated as an index into a precomputed edge intersection table. The edge intersection lookup function returns 12 booleans, indicating which of the 12 edges of the cell are intersected by the iso-surface. Interpolation is used to find where the edge is intersected by the iso-surface. If it is assumed that each edge can only be intersected once, then four triangles are sufficient to show the path of the iso-surface through the cell. There are exactly 256 ways that four or less triangles can be fit to a cell, and the number of cases can be reduced to 15 by reflection and rotation. Groups of three cell-edge intersection points are grouped to form triangles. Gradients at the intersection points are found by interpolating between the gradients at the corners of the cell. The interpolated gradients are stored with the triangles and later used for shading.

Marching cubes sometimes connects the wrong set of three points while generating triangles, resulting in false positive and negative triangles in the iso-surface. One way to reduce ambiguous point-connecting situations is to break up each cell into five [Shir90], six, or 24 tetrahedra and do table-based edge intersection tests with the tetrahedra. Since a tetrahedron has only six edges, two triangles are sufficient to show the iso-surfaces inside the tetrahedral cell. The *marching tetrahedra* algorithm generates more triangles than the marching cubes algorithm, so more processing and memory are required. An alternative means for reducing the ambiguous point-connecting situation is described by Nielson, et al, in [Niel91].

Lorensen and Cline soon realized that the size of the generated triangles, when rendered and projected, is often smaller than the size of a pixel. A new algorithm was invented to take advantage of this observation. *Dividing cubes* [Clin88] begins by traversing each cell in the volume. When a cell is encountered with corner-values that straddle the threshold, the cell is projected into screen space to find if it projects into an area larger than a pixel. If it does, the cell is divided into subcells, each of which is rendered as a *surface point*, otherwise the entire cell is rendered as a surface point. Surface points are composed of a value, a location in object space, and a calculated gradient for shading. No intermediate surface primitives are used in the dividing cubes algorithm. Surface points are rendered into the image buffer using a standard computer graphics hidden-surface algorithm such as painter's or z-buffer. Rendering the surface points in a pseudo-random order allows the user to see the quality of the image gradually improve.

Rendering surface points instead of surface primitives saves a great deal of time, especially when surface-rendering hardware is not available. A hardware implementation of the dividing cubes algorithm is described in [Clin90].

All three of the SF techniques described in this section have the same advantages and disadvantages (to some degree) as the generic SF algorithm described in the section on volume visualization methods. Like the opaque cube methods, all three are also parallelizable at the cell level.

Ray-casting

The most often used volume visualization algorithm for the production of high-quality images is *ray-casting*. Ray-casting conducts an image-order traversal of the image plane pixels, finding a color and opacity for each. A ray is fired from each pixel

through the data volume. The opacities and shaded colors encountered along the ray are summed to find the opacity and color of the pixel. Ray-casting differs from *ray-tracing* in that in ray-tracing, rays are bounced when they hit reflective objects. In ray-casting, rays continue in a straight line until the opacity encountered by the ray sums to unity or the ray exits the rear of the volume. No shadows or reflections are generated in ray-casting. Descriptions of ray-casting approaches appear in [Tuy84] [Levo88] [Upso88] [Levo90a] [Levo90d] and in several papers in [Upso89a]. Numerous enhancements, optimizations, and hybrid methods are also described in the literature. One interesting implementation takes the RenderMan approach of encapsulating ray-casting routines into a library [Mont90] so that the user can write her/his own renderers. The ray-casting algorithm was used to produce Figure 1, Dolphin Head, and the image on the back cover, top right, Simulated Electromagnetic Sounding.

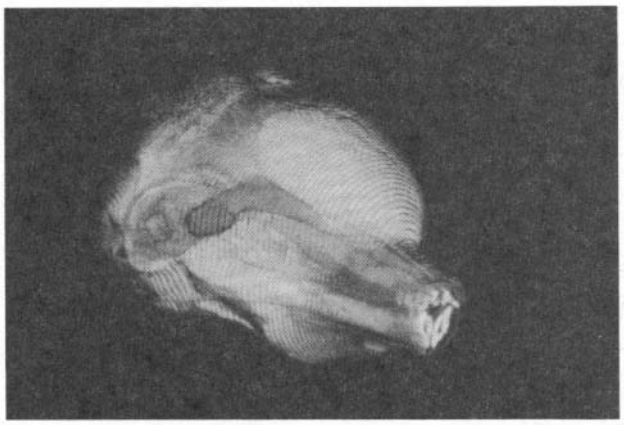

Figure 1: Dolphin Head

The first step in the ray-casting algorithm is for the user to set up data-classification tables as described in the section on data classification above. The user also has to specify viewing and lighting information for the scene. The ray-caster then starts firing rays. When a ray intersects a cell between gridpoints, an interpolation may be performed to find an in-between value for the intersection point. Alternatively, the value stored at the nearest neighboring gridpoint may be used, as in voxel-based rendering. A transfer function is then invoked to find the color and opacity the user has specified for this data value. The color tuple is gradient shaded, and the opacity is attenuated by the normalized magnitude of the gradient. Because the magnitude of the gradient is usually a good indication of the strength of an iso-surface within a cell, the result is a large color contribution to the pixel when the ray encounters a new substance in the volume.

The color tuple and opacity values are added to the pixel using a weighting formula and a step is taken along the ray to the next resample point. The resample points contributing earlier are weighted more heavily than later contributing resample points. The summing continues until the stored opacity value equals unity or the ray exits the volume. If the opacity has not reached unity as the ray exits the back of the volume, the accumulated color tuple is multiplied by the partial opacity.

Ray-casting inherits most of the advantages and disadvantages of DVR methods. Ray-casting is CPU-intensive, but the images show the entire dataset, not just a collection of thin surfaces as in SF. Ray-casting can be parallelized at the pixel level since rays from all of the pixels in the image plane can be cast independently.

In [Sabe88], Sabella describes an alternative ray-casting approach. Voxels are modeled as light-emitting particles, and four values are accumulated as rays are cast into this field: attenuated light intensity, maximum light value, distance of maximum value along the ray, and the centroid of voxel-value along the ray. Color is calculated in hue, saturation, lightness tuples instead of red, green, blue tuples. The hue component is set in relation to the peak along the ray, saturation is used for depth queuing by casting a fog in front of data in the rear of the volume, and lightness is set as a function of the light intensity. This approach is also CPU-intensive, but it generates high-quality glowing images.

Splatting

A recently developed DVR algorithm is called *splatting* [West90]. Splatting performs a front-to-back object-order traversal of the voxels in the volumetric dataset. Each voxel's contribution to the image is calculated and composited using a series of table lookups. The procedure is called splatting because it can be likened to throwing a snowball (voxel) at a glass plate. The snow contribution at the center of impact will be high and the contribution will drop off further away from the center of impact. The splatting algorithm was used to produce Figure 2, Dolphin Skull, and Figure 3, Human Head.

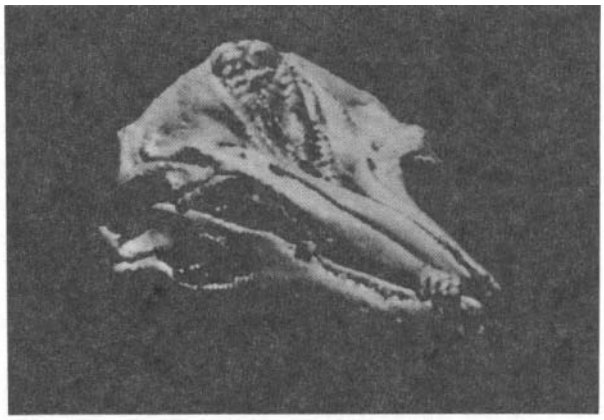

Figure 2: Dolphin Skull

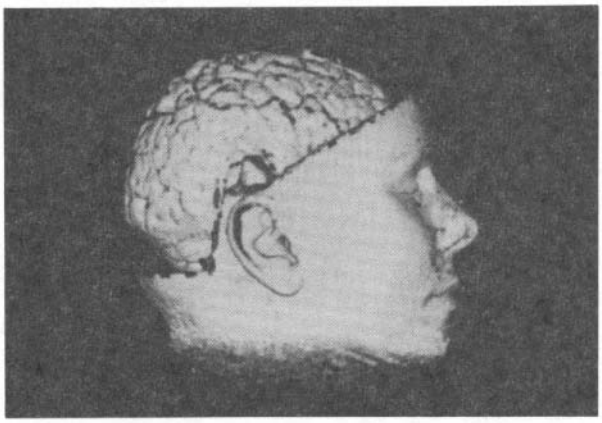

Figure 3: Human Head

The first step in the splatting algorithm is to determine in what order to traverse the volume. The face of the volume and corner of the face closest to the image plane can be found by passing the coordinates of the corners of the volume through the viewing matrix. Voxels are splatted according to their distance from the image plane, with the closest voxel being splatted first. The voxels in one slice are all splatted before the next slice is started. The value at each voxel is classified according to user-specified color and opacity transfer functions. The resulting red, green, and blue tuple is shaded using gradient shading, and the opacity is attenuated by the normalized strength of the gradient.

The next step finds the contribution of the tuple (and opacity) to an image buffer by projecting the voxel into image space. A round filter called a *reconstruction kernel* is used to find the pixel extent of this contribution. For orthogonal viewing, a circular kernel (usually a Gaussian) can be calculated once and used for all of the voxels. However, for perspective viewing, a new oblique kernel must be calculated for every voxel. The projection of the kernel into the image buffer is called a *footprint*. The size of the footprint is made proportional to the size of the volume and the size of the image to be generated, so that a small volume can fill a large image. Once the footprint is calculated, its center is placed at the center of the voxel's projection in the image buffer. Note that there is not necessarily going to be a pixel center at this location.

The shaded color tuple and opacity values are blended into the image buffer at every pixel that falls within the area covered by the circular Gaussian footprint. Before a tuple is blended with a pixel, it is attenuated by the value of the Gaussian footprint at that particular pixel center. This has the effect of making voxel contributions higher when near the center of the projection and lower when far from the center. After all of the voxels have been splatted into the image buffer, the image is complete. When the opacity at a pixel in the image buffer reaches unity, then no further splats will have an effect on it.

Although the splatting technique operates differently than either surface-fitting or ray-casting, it produces high-quality images that are similar to the images produced by the other algorithms. Splatting has most of the advantages and disadvantages of DVR algorithms. One added advantage is that the user can watch the image grow more refined one slice at a time instead of one pixel at a time, as in ray-casting. Some splatting optimizations are given in [Hanr91].

A similar and earlier method called the *V-buffer* algorithm [Upso88] is cell-based rather than voxel-based but uses a similar front-to-back object-order traversal and projection technique. The V-buffer technique traverses a many-point path within each cell, interpolating between cell corner-values and projecting each interpolated value into an image buffer after the value has been color-mapped and shaded.

The splatting and V-buffer approaches are parallelizable at the voxel level. Image contributions can be computed on separate processors so long as the contributions are composited in the correct order. Contributions from the voxels closest to the image plane get composited first.

Pixar slice shearing

The volume slice shearing algorithm described in [Dreb88] takes advantage of the unique hardware capabilities of the Pixar Image Computer. This algorithm is rarely used on non-Pixar hardware. The image on the back cover, top left, Three Time Steps from a Cosmic Jet Simulation, was produced on a Pixar Image Computer.

Ethical issues

Getting all of the bugs out of a volume visualization implementation is a serious matter. For example, lingering flaws in an implementation could produce images that would lead to an incorrect medical diagnosis. Errors in images could have serious ramifications in radiation treatment planning, surgical planning, drug design, and other medical, research, and engineering disciplines. A standard means for validating algorithm implementations and the images they produce would be invaluable to the visualization community.

Future work

As mentioned before, methods for visualization of vector, multi-modal, multi-variate, higher-dimensional, and non-Cartesian volume data are still largely unknown. Adaptation of volume visualization algorithms for both parallel architectures [Elvi92] and network-based environments [Elvi91b][Merc92] toward real-time rendering is also a pressing problem.

There is a need for intuitive user interfaces and semi-automatic data-classification tools, so that scientists can quickly learn and use powerful volume visualization systems. Also, systems should be improved to render volumes in the context of the source of the data instead of rendering volumes floating in black space. For example, CT data could be rendered combined with anatomical three-dimensional models or photographs, and simulation data could be composited with data or images acquired from the actual event.

Hardware implementations of DVR, and possibly SF, algorithms will be available in the near future. These will allow real-time volume visualization systems to be developed and make unsolved volume visualization problems an area for concentration.

Summary

Many of the fundamental concepts of volume visualization have been explained along with some of the pioneering algorithms. Many problems of handling non-Cartesian grids, multivariate and higher-dimensional data are still unsolved. Methods for semi-automatic data-classification would be extremely helpful. Scientists need powerful tools accessed through easy-to-use interfaces. Animation of volume-visualized data is also critical to the data analysis process.

For technical descriptions of some commercial and public domain volume visualization systems see [Stow89] [Upso89b] [Brit90] [Dyer90].

Bibliography

[Brit90] Brittain, D.L., Aller, J., Wilson, M., and Wang, S.C.,"Design of an End-user Data Visualization System," *Proceedings of the IEEE Visualization '90 Conference,* IEEE Computer Society Press, October 1990, 323–328.

[Brod91] Brodlie, K.W., Carpenter, L.A., Earnshaw, R.A., Gallop, J.R., Hubbold, R.J., Mumford, C.D., Osland, C.D., and Quarendon, P. (eds.), *Scientific Visualization - Techniques and Applications,* Springer Verlag Press, 1991.

[Clin88] Cline, H.E., Lorensen, W.E., Ludke, S., Crawford, C.R., and Teeter, B.C., "Two Algorithms for Three-dimensional Reconstruction of Tomograms," *Medical Physics*, Volume 15, Number 3, May/June 1988, 320–327.

[Clin90] Cline, H.E., Ludke, S., Lorensen, W.E., and Teeter, B.C., "A 3D Medical Imaging Research Workstation," *Volume Visualization Algorithms and Architectures*, ACM SIGGRAPH '90 Course Notes, Course Number 11, ACM Press, August 1990, 243–255.

[DeFa89] DeFanti, T.S. and Brown, M.D., "Visualization Expanding Scientific and Engineering Research Opportunities," *Computer*, Volume 22, Number 8, August 1989, 12–25.

[Dreb88] Drebin, R., Carpenter, L., and Hanrahan, P., "Volume Rendering," *Computer Graphics*, Volume 22, Number 4, August 1988, 65–74.

[Dyer90] Dyer, D.S., "A Dataflow Toolkit for Visualization," *IEEE Computer Graphics and Applications*, Volume 10, Number 4, July 1990, 60–69.

[Ekou91] Ekoule, A.B., Peyrin, F.C., and Odet, C.L., "A Triangulation Algorithm from Arbitrary Shaped Multiple Planar Contours," *ACM Transactions on Graphics*, Volume 10, Number 2, April 1991, 182–199.

[Elvi91a] Elvins, T.T., "San Diego Workshop on Volume Visualization Report," *Computer Graphics*, Volume 25, Number 5, October 1991, 264.

[Elvi91b] Elvins, T.T. and Nadeau, D.R., "NetV: An Experimental Network-based Volume Visualization System," *Proceedings of the IEEE Visualization '91 Conference,* IEEE Computer Society Press, October 1991, 239–245.

[Elvi92] Elvins, T.T. and Nadeau, D.R., "Scientific Visualization in a

Network Computing Environment," *Eurographics UK Conference Proceedings*, April 1992, To appear.
[Engl90] England, N. (ed.), "San Diego Workshop on Volume Visualization, Conference Proceedings," *Computer Graphics*, Volume 24, Number 5, November 1990.
[Fren89] Frenkel, K.A., *"Volume Rendering,"Communications of the ACM*, Volume 32, Number 4, April 1989, 426–435.
[Fuch77] Fuchs, H., Kedem, Z.M., and Uselton, S.P., "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, Volume 20, Number 10, October 1977, 693–702.
[Hanr91] Hanrahan, P. and Laur, D., "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics*, Volume 25, Number 4, August 1991, 285–288.
[Herm79] Herman, G.T. and Liu, H.K., "Three-dimensional display of Human Organs from Computed Tomograms," *Computer Graphics and Image Processing*," Volume 9, Number 1, January 1979, 1–21.
[Hohn90] Hohne, K.H., Fuchs, H., and Pizer, S.M. (eds.), *3D Imaging in Medicine, Algorithms, Systems, Applications*," Springer Verlag Press, 1990.
[Kauf86] Kaufman, A. and Simony E., "Scan-conversion Algorithms for Voxel-base Graphics," *Proceedings, ACM Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October 1986, 45–75.
[Kauf91] Kaufman, A., "Introduction to Volume Visualization," *Volume Visualization*, A. Kaufman (ed.), IEEE Computer Society Press, 1991, 1–18.
[Kepp75] Keppel, E.,"Approximating Complex Surfaces by Triangulation of Contour Lines," *IBM Journal of Research and Development*, Volume 19, Number 1, January 1975, 2–11.
[Levo88] Levoy, M.,"Display of Surfaces from Volume Data," *IEEE Computer graphics and Applications*, Volume 8, Number 3, March 1988, 29–37.
[Levo90a] Levoy, M., "Volume Rendering, A Hybrid Ray Tracer for Rendering Polygon and Volume Data," *IEEE Computer graphics and Applications*, Volume 10, Number 2, March 1990, 33–40.
[Levo90b] Levoy, M., "Volume Visualization: A Look Back, A Look Ahead," Keynote address given at San Diego Workshop on Volume Visualization, unpublished, December 1990.
[Levo90c] Levoy, M., Fuchs, H., Pizer, S.M., Rosenman, J., Chaney, E.L., Sherouse, G.W., Interrante, V., and Kiel, J., "Volume Rendering in Radiation Treatment Planning," *Proceedings of the First Conference on Visualization in Biomedical Computing*, May 1990.
[Levo90d] Levoy, M., "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, Volume 9, Number 3, July 1990, 245–261.
[Levo90e] Levoy, M., "A Taxonomy of Volume Visualization Algorithms," *Volume Visualization Algorithms and Architectures*, ACM SIGGRAPH '90 Course Notes, Course Number 11, ACM Press, August 1990, 6–12.
[Lore87] Lorensen, W.E. and Cline, H.E., "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Volume 21, Number 4, July 1987, 163–169.
[McCo87] McCormick, B., DeFanti, T., and Brown, M., "Visualization in Scientific Computing," *Computer Graphics*, Volume 21, Number 6, November 1987.
[Merc92] Mercurio, P.J., Elvins, T.T., Fall, K.R., Cohen, P.S., Young, S.J., and Ellisman, M.H., "The Distributed Laboratory: An Interactive Visualization Environment for Electron Microscopy and Three-dimensional Imaging," *Communications of the ACM*, ACM Press, Volume 35, Number 6, June 1992, to appear.
[Mont90] Montine, J., "A Procedural Interface for Volume Rendering," *Proceedings of the IEEE Visualization '90 Conference*, IEEE Computer Society Press, October 1990, 36–41.
[Niel91] Nielson, G.M. and Hamann, B., "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes," *Proceedings of the IEEE Visualization '91 Conference*, IEEE Computer Society Press, October 1991, 83–91.
[Sabe88] Sabella, P., "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics*, Volume 22, Number 4, July 1988, 51–58.
[Shir90] Shirley, P. and Tuckman, A., "A Polygonal Approximation to Direct Scalar Volume Rendering," *Computer Graphics*, Volume 24, Number 5, November 1990, 63–70.
[Sper90] Speray, D. and Kennon, S., "Volume Probes: Interactive Data Exploration on Arbitrary Grids," *Computer Graphics*, Volume 24, Number 5, November 1990, 5–12.
[Stow89] Stowell, A., "New X Window Packages," *NCSA access*, Volume 3, Number 4, July-August 1989, 15–16.
[Tuy84] Tuy, H.K., and Tuy, L.t., "Direct 2-D Display of 3-D Objects," *IEEE Computer Graphics and Applications*, Volume 4, Number 10, October 1984, 29–33.
[Upso88] Upson, C. and Keeler, M.,"The V-Buffer: Visible Volume Rendering," *Computer Graphics*, Volume 22, Number 4, July 1990, 59–64.
[Upso89a] Upson, C. (ed.), *Chapel Hill Workshop on Volume Visualization, Conference Proceedings*, Department of Computer Science, University of North Carolina, Chapel Hill, NC, May 1989.
[Upso89b] Upson, C., Faulhaber, T., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., and van Dam, A., "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, Volume 9, Number 4, July 1989, 30–42.
[West90] Westover, L., "Footprint Evaluation for Volume Rendering," *Computer Graphics*, Volume 24, Number 4, August 1990, 367–376.
[Wilh91a] Wilhelms, J., "Decisions in Volume Rendering," *State of the Art in Volume Visualization*, ACM SIGGRAPH '91 Course Notes, Course Number 8, ACM Press, August 1991, I.1–I.11.
[Wilh91b] Wilhelms, J. and Van Gelder, A., "A Coherent Projection Approach for Direct Volume Rendering," *Computer Graphics*, Volume 25, Number 4, August 1991, 275–284.
[Wyvi86] Wyvill, G., McPheeters, C., and Wyvill, B., "Data Structure for Soft Objects," *The Visual Computer*, Volume 2, Number 4, August 1986, 227–234.

Acknowledgements

This work was supported by the National Science Foundation under grant ASC8902825 to the San Diego Supercomputer Center. Additional support was provided by the State of California. Special thanks to the director of the SDSC Visualization group, Mike Bailey, and the rest of the SDSC viskids, Phil Mercurio, Dave Nadeau, Jim McLeod, Kevin Landel, and Charlotte Smart.

About the figures in this article

Figure 1. Dolphin Head. Visualization by T. Todd Elvins, software "voxvu," from Sun Microsystems.

Figure 2. Dolphin Skull. Visualization by T. Todd Elvins with his software, "parv." The data, 230 slice of CT scan data of a dolphin head, was acquired by Ted Cranford, University of California, Santa Cruz, who is studying focused echo location capabilities in marine animals.

Figure 3. Human Head. Visualization by T. Todd Elvins with his software, "parv." This image comes from 109 slices of MRI scan data of a human head with a portion of the skull manually removed from the dataset. The original data is from Siemens Imaging in New Jersey.

The other images referred to in this article are featured on the back cover. More information about these images is in the "About the Cover Images" section.

About the author

T. Todd Elvins is an associate staff visualization programmer at the San Diego Supercomputer Center (SDSC) in San Diego, California, where he works in a group of software engineers and animators who research new computer graphics techniques that allow scientists to gain greater insight into a broad variety of scientific problems. He was involved in the design and implementation of the SDSC Advanced Scientific Visualization Laboratory and has participated in numerous collaborative visualization projects with SDSC scientists.

The author can be contacted at:
San Diego Supercomputer Center
P.O. Box 85608
San Diego, CA 92186-9784 USA
Phone: (619) 534-5128 FAX: (619) 534-5113
E-mail: todd@sdsc.edu