Volume Viewing Algorithms: Survey

Roni Yagel

Department of Computer and Information Science
The Ohio State University

Abstract

The task of the rendering process is to display the primitives used to represent the 3D volumetric scene onto a 2D screen. Rendering is composed of a viewing process which is the subject of this paper, and the shading process. The projection process determines, for each screen pixel, which objects are seen by the sight ray cast from this pixel into the scene. The viewing algorithm is heavily dependent on the display primitives used to represent the volume and whether volume rendering or surface rendering are employed. Conventional viewing algorithms and graphics engines can be utilized to display geometric primitives, typically employing surface rendering. However, when volume primitives are displayed directly, a special volume viewing algorithm should be employed. This algorithm should capture the contents of the voxels on the surface as well as the inside of the volumetric object being visualized. This paper surveys and compares previous work in the field of direct volume viewing. We describe methods for accelerating volume viewing, parallel volume viewing algorithms, and special purpose hardware. We conclude by surveying recent method for rendering intermixed volume-and-surface datasets and algorithms for volume viewing of various types of irregular grids.

1. Introduction

The simplest way to implement viewing is to traverse all the volume regarding each voxel as a 3D point that is transformed by the viewing matrix and then projected onto a Z-buffer and drawn onto the screen. Some methods have been suggested in order to reduce the amount of computations needed in the transformation by exploiting the spatial coherency between voxels. These methods are described in more details in Section 3.1.

The *back-to-front* (BTF) algorithm is essentially the same as the Z-buffer method with one exception that is based on the observation that the voxel array is presorted in a fashion that allows scanning of its component in an order of decreasing or increasing distance from the observer. Exploiting this presortedness of the voxel arrays, traversal of the volume in the BTF algorithm is done in order of decreasing distance to the observer. This avoids the need for a Z-buffer for hidden voxel removal considerations by applying the painter's algorithm by simply drawing the current voxel on top of previously drawn voxels or by compositing the current voxel with the screen value [Farrell, Zappulla, and Yang 1984, Frieder, Gordon, and Reynolds 1985].

The *front-to-back* (FTB) algorithm is essentially the same as BTF only that now the voxels are traversed in increasing distance order. Front-to-back has the potential of a more efficient implementation by employing a dynamic data structure for screen representation [Reynolds, Gordon, and Chen 1987] in which only un-lit pixels are processed and newly-lit pixels are efficiently removed from the data structure. It should be observed that while in the basic Z-buffer method it is impossible to support the rendition semitransparent materials since voxels are mapped to the screen in arbitrary order. Compositing is based on a computation that simulates the passage of light through several materials. In this computation the

order of materials is crucial. Therefore, translucency can easily be realized in both BTF and FTB in which objects are mapped to the screen in the order in which the light traverses the scene.

Another method of volumetric projection is based on first transforming each slice from voxel-space to pixel-space using 3D affine transformation (shearing) [Hanrahan 1990] [Schroeder and Salem 1991] and then projecting it to the screen in a FTB fashion, blending it with the projection formed by previous slices [Drebin, Carpenter, and Hanrahan 1988]. Since transformations are destructive in terms of preserving data continuity, triangular or bicubic sampling filter are used in each sampling operation.

Westover [Westover 1989, Westover 1990] has introduced the *splatting* technique in which each voxel is transformed into screen space and then shaded. Blurring, based on 2D lookup tables is performed to obtain a set of points (footprint) that spreads the voxels energy across multiple pixels. These are then composited with the image array. We have described [Sobierajski et al. 1990] a simplified splatting for interactive volume viewing in which only voxels comprising the object's surface are maintained. Rendering is based on the usage of a powerful transformation engine that is fed with multiple points per voxel. Additional speedup is gained by culling voxels that have a normal pointing away from the observer and by adaptive refinement of image quality.

The *ray casting algorithm* casts a ray from each pixel on the screen into the volume data along the viewing vector until it accumulates an opaque value [Bakalash and Kaufman 1989, Farrell, Yang, and Zappulla 1985, Tuy and Tuy 1984, Upson and Keeler 1988].

The simplest method for implementing resampling performs zero order interpolation to locate the nearest voxel while stepping along the discrete ray representation which is generated by a 3D line algorithm [Schlusselberg, Smith, and Woodward 1986]. Alternatively, the volume is sampled at the intersection points of the ray and the voxel faces, its value is interpolated, and then composited [Upson and Keeler 1988]. A more precise algorithm uses higher order interpolation to estimate the appropriate value at evenly spaced sample points along the ray [Kajiya and Von Herzen 1984, Levoy 1988, Sabella 1988].

Ray casting methods, in which only primary rays are traced, have been applied to volumetric datasets, such as those arising in biomedical imaging and scientific visualization applications (e.g., [Drebin, Carpenter, and Hanrahan 1988, Upson and Keeler 1988]). Greene [Greene 1989] has applied ray casting to estimate the illumination of a voxel model of stochastically growing plants. Levoy [Levoy 1990b, Levoy 1990] has used the term ray tracing of volume data to refer to ray casting and compositing of even-spaced samples along the primary viewing rays.

Since ray casting follows only primary rays, it does not directly support the simulation of light phenomena such as reflection, shadows, and refraction. As an alternative we have developed the 3D raster ray tracer (RRT) [Yagel, Kaufman, and Zhang 1991, Yagel, Cohen, and Kaufman 1992] that recursively considers both primary and secondary rays and thus can create ''photorealistic'' images. It exploits the voxel representation for the uniform representation and ray tracing of sampled and computed volumetric datasets, traditional geometric scenes, or an intermixing thereof. For example, a scalpel superimposed on a CT image or radiation beams superimposed on a scanned tumor. Our approach operates in two major phases: a preprocessing voxelization phase and a discrete ray tracing phase. In the voxelization phase, the volumetric data set is reconstructed and stored in a 3D raster. The geometric model is digitized, using incremental 3D scan-conversion algorithms [Cohen and Kaufman 1990, Cohen and Kaufman 1991, Kaufman 1987a, Kaufman 1987b, Mokrzycki 1988]. These algorithms convert the continuous representation of the model into a discrete representation that is intermixed with the sampled data within the 3D raster. In the second phase, a variation of the conventional ray tracing algorithm is employed. In conventional ray tracing algorithms analytical rays, searching for the closest intersection, are intersected with the object list However, in our approach 3D *discrete* rays, searching for the first surface voxels, are traversed through the 3D raster. Encountering a non-transparent voxel represents a ray-surface hit.

Compared to conventional ray tracers, the 3D raster ray tracer (RRT) enjoys some obvious advantages. In conventional ray tracing, computation time grows with the number of objects [Fujimoto, Tanata, and Iwata 1986], because in crowded scenes a ray may pierce a substantial number of objects and there is a considerable probability that a cell contains more than one object. Moreover, conventional ray tracers are extremely sensitive to the type of objects comprising the scene; intersection calculation between a ray

and a parametric surface is significantly more complex than intersecting the ray with a sphere or a polygon. In contrast, RRT completely eliminates the computationally expensive ray-object intersections calculation, and instead relies solely on a fast discrete ray traversal mechanism and a single simple type of object – the voxel. Consequently, RRT is practically independent of the number of objects in the scene or the object's complexity or type. RRT performance, however, is sensitive to resolution of the 3D raster. Therefore, for a given resolution, ray tracing time is nearly constant and can even decrease as the number of objects in the scene increases, as less stepping is necessary before an object is encountered.

Any change to a view-dependent parameter requires a traditional ray tracer to execute a full fledged rendering that consists of computing many view-independent attributes such as surface normal, texture color, and light source visibility and illumination. In contrast, since RRT maintains information regarding each discrete point, the view-independent attributes can be precomputed during the voxelization phase and stored within each voxel. Those attributes are readily accessible for multiple rendering of the fixed scene in which viewing, lighting, and shading parameters change.

Ray tracing CSG (Constructive Solid Geometry) models is traditionally achieved by converting the CSG-tree into boundary representation (B-rep) in an extremely time consuming process. Alternatively, the direct CSG-rendering approach delays the computation of the Boolean operations until the rendering stage. However, the phenomena by which processing time increases more rapidly than the complexity of the model remains the major hurdle confronting this approach. In contrast, as the spatial enumeration of the 3D raster lends itself to voxel-by-voxel Boolean operation, RRT can easily support ray tracing of CSG models that are efficiently synthesized and constructed during the voxelization phase.

We now turn to classify and compare existing volume viewing algorithms. In Section 3 we survey recent advances in acceleration techniques for forward viewing (Section 3.1), backward viewing (section 3.2) and hybrid viewing (Section 3.3). Section 4 is devoted to a topic that is gaining much popularity in recent years: parallel algorithms for volume viewing. In Section 5 we briefly list special purpose hardware that was developed to cope with volumes. We conclude with sections 6 and 7 that deal with viewing volume intermixed with surfaces (Section 6) and volume viewing of irregular grids.

2. Classification of Volume Viewing Methods

Projection methods differ in several aspects which can be used for a their classification in various ways. First, we have to observe whether the algorithm traverses the volume and projects its components onto the screen (called also *forward*, *object-order*, or *voxel-space projection*) [Frieder, Gordon, and Reynolds 1985, Gordon and Reynolds 1985, Westover 1989], does it traverse the pixels and solve the visibility problem for each one by shooting a sight ray into the scene (called also *backward*, *image-order*, or *pixel-space projection*) [Kaufman and Bakalash 1988, Levoy 1988, Sabella 1988, Tuy and Tuy 1984, Upson and Keeler 1988, Yagel 1991b, Yagel and Kaufman 1992], or does it perform some kind of a hybrid traversal [Jackel and Strasser 1988, Upson and Keeler 1988].

Volume rendering algorithms can also be classified according to the partial voxel occupancy they support. Most algorithms [Gordon and Reynolds 1985, Herman and Liu 1979, Reynolds, Gordon, and Chen 1987, Tuy and Tuy 1984, Yagel 1991b, Yagel and Kaufman 1992] assume uniform (binary) occupancy, that is, a voxel is either fully occupied by some object or it is devoid of any object presence. In contrast to uniform voxel occupancy, methods based on partial voxel occupancy utilize intermediate voxel values to represent partial voxel occupancy by objects of homogeneous material. This provides a mechanism for the display of objects that are smaller than the acquisition grid or that are not aligned with it. Partial volume occupancy can be used to estimate occupancy fractions for each of a set of materials that might be present in a voxel [Drebin, Carpenter, and Hanrahan 1988] . Partial volume occupancy is also assumed whenever gray-level gradient [Hoehne and Bernstein 1986a] is used as a measure for the surface inclination. That is, voxel values in the neighborhood of a surface voxel are assumed to reflect the relative average of the various surface types in them. As a side note we remark that this observation can be used for antialiased voxelization of geometric objects. According to this idea, voxels intersected by the geometric object are given, at the time of voxelization, the object value as well as a partial-occupancy

value. At rendering time the object value represents its material while the partial occupancy value is used for gray-level normal estimation.

Volume rendering methods differ also in the way they regard the material of the voxels. Some methods regarded all materials as opaque [Goldwasser 1986, Gordon and Reynolds 1985, Hoehne et al. 1990, Reynolds, Gordon, and Chen 1987, Schlusselberg, Smith, and Woodward 1986, Trousset and Schmitt 1987, Tuy and Tuy 1984] while others allow each voxel to have an opacity attribute [Drebin, Carpenter, and Hanrahan 1988, Levoy 1988, Sabella 1988, Upson and Keeler 1988, Westover 1989, Yagel 1991b, Yagel and Kaufman 1992]. Supporting variable opacities models the appearance of semi-transparent jell and requires composition of multiple voxels along each sight ray.

Yet another aspect of distinction between rendering methods is the number of materials they support. Early methods supported scenes consisting of binary-valued voxels while more recent methods usually support multi-valued voxels. In the first case objects are represented by occupied voxels while the background is represented by void voxels [Frieder, Gordon, and Reynolds 1985, Herman and Liu 1979, Reynolds, Gordon, and Chen 1987, Tuy and Tuy 1984]. In the latter approach, multi-valued voxels are used to represent objects of nonhomogeneous material [Goldwasser 1986, Hoehne and Bernstein 1986b, Schlusselberg, Smith, and Woodward 1986, Trousset and Schmitt 1987]. It should be observed that given a set of voxels having multiple values we can either regard them as fully occupied voxels of various materials (i.e., each value represents a different material) or we can regard the voxel value as an indicator of partial occupancy by a single material, however we can not have both. In order to overcome this limitation, some researchers adopt the multiple-material approach as a basis for a classification process that attaches a material-label to each voxel. Once each voxel has a material label, these researchers regard the

Table 1: Summary of volume rendering methods.

	Pixel vs. Voxel	Occupancy	Opaque vs. Translucent	Number of Materials	Value Variation
[Hoehne et al. 1990], [Kaufman and Bakalash 1988], [Yagel 1992]	Pixel	Uniform	Translucent	Multiple	Zero
[Tuy and Tuy 1984]	Pixel	Uniform	Opaque	Binary	Zero
[Levoy 1988], [Yagel 1991]	Pixel	Uniform	Translucent	Multiple	Trilinear
[Frieder, Gordon, and Reynolds 1985], [Farrell, Zappulla, and Yang 1984], [Goldwasser 1986], [Schlusselberg, Smith, and Woodward 1986], [Meagher 1982]	Voxel	Uniform	Opaque	Multiple	Zero
[Westover 1990]	Voxel	Partial	Translucent	Multiple	Gausian
[Drebin, Carpenter, and Hanrahan 1988]	Voxel	Partial	Translucent	Multiple	BiCubic
[Upson and Keeler 1988]	Hybrid	Uniform	Translucent	Multiple	Trilinear
[Jackel Strasser 1988]	Hybrid	Uniform	Opaque	Multiple	Zero

original voxel values as partial occupancy indicators for the labeled material [Drebin, Carpenter, and Hanrahan 1988].

Finally, volume rendering algorithms can also be classified according to whether they assume constant value across the voxel extent [Kaufman and Bakalash 1988] or do they assume (trilinear) variation of the voxel value [Levoy 1988]. Table 1 summarizes the behavior of the major viewing algorithms according to the classification topics described above.

A severe problem in the *voxel-space projection* is that at some viewing points, holes might appear in the scene. To solve this problem one can regard each voxel in our implementation as a group of points (depending on the viewpoint) [Sobierajski et al. 1993] or maintain a ratio of $1{:}\sqrt{3}$ between a voxel and a pixel [Cline et al.]. Another solution is based on a hybrid of voxel-space and pixel-space projections that is based on traversing the volume in a BTF fashion but computing pixel colors by intersecting the voxel with a scan line (plane) and then integrating the colors in the resulting polygon [Upson and Keeler 1988]. Since this computation is relatively time consuming it is more suitable to small datasets. It is also possible to apply to each voxel a blurring function to obtain a 2D footprint that spreads the sample's energy onto multiple image pixels which are latter composed into the image [Westover 1990]. A major disadvantage in the splatting approach is that it tends to blur the edges of objects and reduce the image contrast. Another deficiency in the *voxel-space projection* method is that it must traverse and project all the voxels in the scene. We have suggested the use of a normal based culling in order to reduce (possibly by half) the amount of processed voxels [Sobierajski et al. 1993]. On the other hand, since voxel-space projection operates in object-space, it is most suitable to various parallelization schemes based on object space subdivision [Goldwasser and Reynolds 1987, Ohashi, Uchiki, and Tokoro 1985, Westover 1989]. For more information on hardware implementation of volume viewing, the interested reader is referred to [Kaufman et al. 1990, Schreiter and Zimmerman 1988] for more details.

The main disadvantages of the *pixel-space projection* scheme are aliasing (specially when assuming uniform value across voxel extent) and the difficulty to parallelize it. While the computation involved in tracing rays can be performed in parallel, memory becomes the bottleneck. Since rays traverse the volume in arbitrary directions it seems to be no way to distribute voxels between memory modules to guarantee contention free access [Levoy 1989]. Yagel [Yagel 1991a, Yagel 1991b] have proposed a a parallel implementation of ray-casting on a special purpose architecture that is based on slice-wise memory modularization and limited data re-orientation operations.

3. Methods for Accelerating Volume Viewing

Either forward projection or backward projection requires the scanning of the volume buffer which is a large buffer of size proportional to the cubic of the resolution. Consequently, volume rendering algorithms are time-consuming algorithms. This paper focuses on techniques for expediting these algorithms.

3.1. Expediting Forward Viewing

The Z-buffer projection algorithm, although surprisingly simple, is inherently very inefficient and when naively implemented, produces low quality images. The inefficiency attribute of this method is rooted in the N^3 vector-by-matrix multiplications it calculates and the N^3 accesses to the Z-buffer it requires. Inferior image quality is caused by this method's inability to support compositing of semitransparent materials, due to the arbitrary order in which voxels are transformed. In addition, transforming a set of discrete points is a source for various sampling artifacts such as holes and jaggies.

Some methods have been suggested to reduce the amount of computations needed for the transformation by exploiting the spatial coherency between voxels. These methods are: recursive 'divide and conquer' [Goldwasser 1986, Meagher 1982], precalculated tables [Frieder, Gordon, and Reynolds 1985], and incremental transformation [Kaufman 1989, Machiraju, Yagel, and Schwiebert 1992].

The first method exploits coherency in voxel space by representing the 3D volume by an octree. A group of neighboring voxels having the same value (or similar, up to a threshold value) may, under some restrictions, be grouped into a uniform cubic subvolume. This aggregate of voxels can be transformed and rendered as a uniform unit instead of processing each of its voxels. In addition, since each octree node has eight equally-sized octants, given the transformation of the parent node, the transformation of its sub-octants can be efficiently computed. If we look at the 2D example of quadtrees and we denote the transformation of the two vertices $[x,y]$ and $[x+d,y+d]$ (where d is the quadrant size) by $[\bar{x},\bar{y}]$ $[\bar{x_d},\bar{y_d}]$ then the center point is transformed to: $[\bar{x}+(\bar{x_d}-\bar{x})/2,\ \bar{y}+(\bar{y_d}-\bar{y})/2]$ a computation that is much more efficient than the straightforward transformation. This method requires, in 3D, three divisions and six additions per coordinate transformation.

The *table-driven transformation* method is based on the observation that volume transformation involves the multiplication of the matrix elements with integer values which are always in the range $[1..N]$, where N is the volume resolution. Therefore, in a short preprocessing stage each matrix element t_{ij} is stored in table $tab_{ij}[N]$ such that $tab_{ij}[k] = t_{ij} * k,\ \ 1 \le k < N$. During the transformation stage, coordinate by matrix multiplication is replaced by table lookup. This method requires, in 3D, nine table lookup operations and nine additions, per coordinate transformation.

Finally, the *incremental transformation* method is based on the observation that the transformation of a voxel $[x+1,y,z]$ can be incrementally computed given the transformed vector $[\bar{x},\ \bar{y},\ \bar{z}]$ of the voxel at $[x,\ y,\ z]$. This is because

$$[x+1,\ y,\ z] * T = [x,\ y,\ z] * T + [1,0,0] * T = [\bar{x},\ \bar{y},\ \bar{z}] + [t_{11},\ t_{12},\ t_{13}]$$

To begin the incremental process we need one matrix by vector multiplication to compute the updated position of the first grid point. The remaining grid points are incrementally transformed, requiring three additions per coordinate. However, to employ this approach, *all* volume elements, including the empty ones, have to be transformed. This approach is therefore more suitable to parallel architecture where it is desired to keep the computation pipeline busy [Machiraju, Yagel, and Schwiebert 1992]. This approach is especially attractive for vector processors since the transformations of the set of voxels $[x+1,1,z]$, $[x+1,1,z]$,...,$[x+1,N,z]$, called *beam* and denoted by $[x+1,\ 1..N,\ z]$, can be computed from the transformation of the vector $[x,\ 1..N,\ z]$ by adding, to each element in this vector, the same three constants: $(t_{11},\ t_{12},\ t_{13})$.

So far we have been looking at methods that ease the computation burden associated with the transformation. However, consulting the Z-buffer N^3 times is also a source of significant slow down. The *back-to-front* (BTF) algorithm is essentially the same as the Z-buffer method with one exception – the order in which voxels are scanned. It is based on the observation that the voxel array is spatially presorted. This attribute allows the renderer to scan the volume in an order of decreasing distance from the observer. By exploiting this presortedness of the voxel arrays, one can draw the volume in a back-to-front order, that is, in order of decreasing distance to the observer. This avoids the need for a Z-buffer for hidden voxel removal by applying the painter's algorithm. That is, the current voxel is simply drawn on top of previously drawn voxels. If compositing is performed, the current voxel is composited with the screen value [Farrell, Zappulla, and Yang 1984, Frieder, Gordon, and Reynolds 1985]. The *front-to-back* (FTB) algorithm is essentially the same as BTF, only that now the voxels are traversed in increasing distance order.

As mentioned above in the basic Z-buffer method it is impossible to support the rendition of semi-transparent materials because voxels are mapped to the screen in an arbitrary order. In contrast, translucency can easily be realized in both BTF and FTB because in these methods objects are mapped to the screen in viewing order.

Another approach to forward projection is based on first transforming the volume from voxel-space to pixel-space by employing a decomposition of the 3D affine transformation into five 1D shearing transformations [Hanrahan 1990]. Then, the transformed voxel is projected onto the screen in an FTB order, which supports the blending of voxels with the projection formed by previous (farther) voxels [Drebin, Carpenter, and Hanrahan 1988]. The major advantage of this approach is its ability (using simple

averaging techniques) to overcome some of the sampling problems causing the production of low quality images. In addition, this approach replaces the 3D transformation by five 1D transformations which require only one floating-point addition each.

Another solution to the image quality problem mentioned above is *splatting* [Westover 1990], in which each voxel is transformed into screen space and then it is shaded. Blurring, based on 2D lookup tables, is performed to obtain a set of points (a cloud) that spreads the voxel's energy across multiple pixels called *footprint*. These are then composited with the image array. However this algorithm which requires extensive filtering is time consuming. We have described [Sobierajski et al. 1990] a simplified approximation to the splatting method for interactive volume viewing in which only voxels comprising the object's surface are maintained. Each voxel is represented by several 3D points (a 3D footprint). Rendering is based on the usage of a contemporary geometry engine that is fed with those multiple points per voxel. Additional speedup is gained by culling voxels that have a normal pointing away from the observer. Finally, adaptive refinement of image quality is also supported: when the volume is manipulated only one point per voxel is rendered, interactively producing a low quality image. When the volume remains stationary and unchanged, for some short period, the rendering system renders the rest of the points to increase image quality.

Another efficient implementation of the splatting algorithm, called *hierarchical splatting* [Laur and Hanrahan 1991] uses a pyramid data structure to hold a multiresolution representation of the volume. For a volume of N^3 resolution the pyramid data structure consists of a sequence of $\log N$ volumes. The first volume contains the original dataset, the next volume in the sequence is half the resolution of the previous one. Each of its voxels contains the average of eight voxels in the higher resolution volume. According to the desired image quality, this algorithm scans the appropriate level of the pyramid in a BTF order. Each element is splatted using the appropriate sized splat. The splats themselves are approximated by polygons which can efficiently be rendered by graphics hardware.

3.2. Expediting Backward Viewing

Backward viewing of volumes, based on casting rays, has three major variations: *parallel (orthographic) ray casting, perspective ray casting,* and *ray tracing*. The first two are variations of *ray casting*, in which only primary rays, that is, rays from the eye through the screen, are followed. These two methods have been widely applied to volumetric datasets, such as those arising in biomedical imaging and scientific visualization applications (e.g., [Drebin, Carpenter, and Hanrahan 1988, Upson and Keeler 1988]). Levoy [Levoy 1990b, Levoy 1990] has used the term *ray tracing* of volume data to refer to ray casting and compositing of even-spaced samples along the primary viewing rays.

Ray casting can further be divided into methods that support only parallel viewing, that is, when the eye is at infinity and all rays are parallel to one viewing vector. This viewing scheme is used in applications that could not benefit from perspective distortion such as biomedicine. Alternatively, ray casting can be implemented to support also perspective viewing.

Since ray casting follows only primary rays, it does not directly support the simulation of light phenomena such as reflection, shadows, and refraction. As an alternative we have developed the *3D raster ray tracer* (RRT) [Yagel, Cohen, and Kaufman 1992] that recursively considers both primary and secondary rays and thus can create ''photorealistic'' images. It exploits the voxel representation for the uniform representation and ray tracing of sampled and computed volumetric datasets, traditional geometric scenes, or intermixing thereof.

The examination of existing methods for speeding up the process of ray casting reveals that most of them rely on one or more of the following principles: (1) pixel-space coherency (2) object-space coherency (3) inter-ray coherency and (4) space-leaping. We now turn to describe each of those in more detail.

1. Pixel-space coherency

There is a high coherency between pixels in image space. That is, it is highly probable that between

two pixels having identical or similar color we will find another pixel having the same (or similar) color. Therefore it is observed that it might be the case that we could avoid sending a ray for such obviously identical pixels.

2. Object-space coherency

   The extension of the pixel-space coherency to 3D states that there is coherency between voxels in object space. Therefore, it is observed that it should be possible to avoid sampling in 3D regions having uniform or similar values.

3. Inter-ray coherency

   There is a great deal of coherency between rays in parallel viewing, that is, all rays, although having different origin, have the same slope. Therefore, the set of steps these rays take when traversing the volume are similar. We exploit this coherency so as to avoid the computation involved in navigating the ray through voxel space.

4. Space-leaping

   The passage of a ray through the volume is two phased. In the first phase the ray advances through the empty space searching for an object. In the second phase the ray integrates colors and opacities as it penetrates the object (in the case of multiple or concave objects these two phases can repeat). Commonly, the second phase involves one or a few steps, depending on the object's opacity. Since the passage of empty space does not contribute to the final image it is observed that skipping the empty space could provide significant speed up without affecting image quality.

The *adaptive image supersampling*, exploits the pixel-space coherency. It was originally developed for traditional ray-tracing [Bergman et al. 1986] and later adapted to volume rendering [Levoy 1990a]. First, rays are cast from only a subset of the screen pixels (e.g., every other pixel). ''Empty pixels'' residing between pixels with similar value are assigned an interpolated value. In areas of high image gradient additional rays are cast to resolve ambiguities.

van Walsum et al [van Walsum et al. 1991] have used the voxel-space coherency. In his method the ray starts sampling the volume in low frequency (i.e., large steps between sample points). If a large value difference is encountered between two adjacent samples, additional samples are taken between them to resolve ambiguities in these high frequency regions. Recently, this basic idea was extended to efficiently lower the sampling rate in either areas where only small contributions of opacities are made, or in regions where the volume is homogeneous [Danskin and Hanrahan 1992]. This method efficiently detects regions of low presence or low variation by employing a pyramid of volumes that decode the minimum and maximum voxel value in a small neighborhood, as well as the distance between these measures.

The *template-based* method [Yagel 1991b, Yagel and Kaufman 1992] utilizes the inter-ray coherency. Observing that, in parallel viewing, all rays have the same form it was realized that there is no need to reactivate the discrete line algorithm for each ray. Instead, we can compute the *form* of the ray once and store it in a data structure called *ray-template*. All rays can then be generated by following the ray template. The rays, however, differ in the exact positioning of the appropriate portion of the template, an operation that has to be performed very carefully. For this purpose a plane that is parallel to one of the volume faces is chosen to serve as a *base-plane* for the template placement. The image is projected by sliding the template along that plane emitting a ray at each of its pixels. This placement guarantees a complete and uniform tessellation of the volume.

The template-based algorithm starts by finding the base-plane, the image extent, and a ray-template. The base-plane is that one of the three object space planes onto which the image is projected to the largest area. From this plane, and within the projected region of the volume, parallel rays are cast into the volume by repeating the sequence of steps specified by the ray-template. The result of the second phase of the algorithm is a projection of the volume on the base-plane. The third phase of the algorithm transforms the projected image from the base-plane to the screen-plane. The regularity and simplicity of this efficient algorithm make it very attractive for hardware implementation [Yagel 1991a] and for massively parallel computers such as the CM-2 [Schroeder and Stoll 1992].

So far we have seen methods that exploit some type of coherency to expedite volumetric ray casting. However, the most prolific and effective branch of volume rendering acceleration techniques involve the utilization of the fourth principle mentioned above – speeding up ray casting by providing efficient means to traverse the empty space.

The hierarchical representation (e.g., octree) decomposes the volume into uniform regions that can be represented by nodes in a hierarchical data structure. An adjusted ray traversal algorithm skips the (uniform) empty space by maneuvering through the hierarchical data structure [Levoy 1990b, Samet and Webber 1988]. It was also observed that traversing the hierarchical data structure is inefficient compared to the traversal of regular grids. A combination of the advantages of both representations is the *uniform buffer*. The ''uniformity information'' decoded by the octree can be stored in the empty space of a regular 3D raster. That is, voxels in the uniform buffer contain either a data value or information indicating to which size empty octant they belong. Rays which are cast into the volume encounter either a data voxel, or a voxel containing ''uniformity information'' which instructs the ray to perform a leap forward that brings it to the first voxel beyond the uniform region [Cohen and Shefer 1993]. This approach saves the need to perform a tree search for the appropriate neighbor – an operation that is the most time consuming and the major disadvantage in the hierarchical data structure.

When a volume consists of one object surrounded by empty space, a common and simple method to skip most of this empty space uses the well known technique of *bounding-boxes*. The object is surrounded by a tightly fit box (or other easy-to-intersect object such as sphere). Rays are intersected with the bounding object and start their actual volume traversal from this intersection point as opposed to starting from the volume boundary. The PARC (Polygon Assisted Ray Casting) approach [Avila, Sobierajski, and Kaufman 1992] strives to have a better fit by allowing a convex polyhedral envelope to be constructed around the object. PARC utilizes available graphics hardware to render the front faces of the envelope (to determine, for each pixel, the ray entry point) and back faces (to determine the ray exit point). The ray is then traversed from entry to exit point. A ray that does not hit any object is not traversed at all.

It is obvious that the empty space does not have to be sampled – it has only to be crossed as fast as possible. Therefore, we have proposed [Yagel et al. 1991, Yagel, Cohen, and Kaufman 1992] to utilize one fast and crude line algorithm in the empty space (e.g., 3D integer-based 26-connected line algorithm) and another, slower but more accurate (e.g., 6-connected integer or 3D DDA floating point line algorithm), in the vicinity and interior of objects. The effectiveness of this approach depends on its ability to efficiently switch back and forth between the two line algorithm, and its ability to efficiently detect the proximity of occupied voxels. This is achieved by surrounding the occupied voxels by a one-voxel-deep ''cloud'' of flag-voxels, that is, all empty voxels neighboring an occupied voxel are assigned, in a preprocessing stage, a special ''vicinity flag''. A crude ray algorithm is employed to rapidly traverse the empty space until it encounters a vicinity voxel. This flags the need to switch to a more accurate ray traversal algorithm. Encountering later an empty voxel (i.e., unoccupied and not carrying the vicinity flag) can signal a switch back to the rapid traversal of empty space.

The *proximity-clouds* method [Cohen and Shefer 1993, Zuiderveld, Koning, and Viergever 1992] is based on the extension of this idea even further. Instead of having a one-voxel-deep vicinity cloud this method computes, in a preprocessing stage, for each empty voxel, the distance to the closest occupied voxel. When a ray is sent into the volume it can either encounter an occupied voxel, to be handled as usual, or a ''proximity voxel'' carrying the value n. This suggests that the ray can take a n-step leap forward, being assured that there is no object in the skipped span of voxels. The effectiveness of this algorithm is obviously dependent on the ability of the line traversal algorithm to efficiently jump arbitrary number of steps [Cohen and Shefer 1993].

Recently, Yagel and Shi [Yagel and Shi 1993] have reported on a method for speeding up the process of volume rendering a sequence of images. It is based on exploiting coherency between consecutive images to shorten the path rays take through the volume. This is achieved by providing each ray with the information needed to leap over the empty space and commence volume traversal at the vicinity of meaningful data. The algorithm starts by projecting the volume into a C-buffer (Coordinate-buffer) which stores, at each pixel location, the object-space coordinates of the first non empty voxel visible from that

pixel. For each change in the viewing parameters, the C-buffer is transformed accordingly. In the case of rotation the transformed C-buffer goes through a process of eliminating coordinates that possibly became hidden [Gudmundsson and Randen 1990]. The remaining values in the C-buffer serve as an estimate of the point where the new rays should start their volume traversal.

3.3. Progressive Refinement

One practical solution to the rendering time problem is the generation of partial images that are progressively refined as the user interacts with the crude image. Both forward and backward approach can support progressive refinement. In the case of forward viewing this technique is based on a pyramid data structure. First, the smaller volume in the pyramid is rendered using large-footprint splats. Later, higher resolution components of the pyramid are rendered [Laur and Hanrahan 1991].

Providing progressive refinement in backward viewing is achieved by first sampling the screen in low resolution. The regions in the screen where no rays were emitted from receive a value interpolated from some close pixels that were assigned rays. Later more rays are cast and the interpolated value is replaced by the more accurate result [Levoy 1990a]. Additionally, rays that are intended to cover large screen areas can be traced in the lower-resolution components of a pyramid [Levoy and Whitaker 1990].

Not only screen-space resolution can be progressively increased. Sampling rate and stopping criteria can also be refined. An efficient implementation of this technique was reported by Danskin and Hanrahan [Danskin and Hanrahan 1992].

4. Parallel Algorithms

The sheer amount of data that arises even from a not too fine resolution volume representation is enormous. A most obvious solution to reduce the total rendering time is to employ parallel architectures and algorithms.

Some implementations of direct volume rendering have recently been reported for a variety of parallel architectures. A few of them exist on experimental parallel processors, such as the DASH parallel system [Neih and Levoy 1992], the Princeton Engine [Schroeder and Stoll 1992], the G2 processor [Leung and Synott 1992], the PixelPlane 5 processor [Yoo et al. 1992, Yoo et al. 1992], and the UWGSP4 [Jong et al. 1992] processor. Other reported implementations have been conducted on commercially avail able parallel machines such as the AMT DAP [Cameron and Underill 1992], Connection Machine CM-2 [Schroeder and Salem 1991, Schroeder and Stoll 1992], MasPar MP-1 [Vezina, Fletcher, and Robertson 1992, Wittenbrink and Somani 1993], Fujitsu AP- 100 [Corrie and Mackerras 1992], Intel iPSC-2 [Montani, Perego, and Scopingo 1992], CRAY [Elvins 1992, Machiraju, Yagel, and Schwiebert 1992], nCUBE [Elvins 1992], BBN TC2000 [Challinger 1992], Silicon Graphics multiprocessor workstation [Fruhauff 1992, Machiraju, Yagel, and Schwiebert 1992], transputers [Yazdy et al. 1990], IBM Power Visualization System (PVS) [Machiraju and Yagel 1993], and network of Suns [Westover 1990].

The viewing algorithms adopted in the afore-mentioned implementations are many and varied. Both traditional direct volume rendering methods, namely feed-forward and backward-feed as well as an hybrid approach, was adapted for parallel processors. In the next three subsection we describe some of the existing parallel algorithms for volume rendering.

4.1.1. Parallel Forward Viewing Methods

Multiple-transformation methods and splatting have been commonly employed for parallel implementations. By converting the three-dimensional view matrix into a series of one-dimensional shears and scales along the orthogonal axis, both Schroeder and Salem [Schroeder and Stoll 1992] and Vezina et. al [Vezina, Fletcher, and Robertson 1992] implemented feed-forward rendering on SIMD processors. A

one-dimensional shear leads to regular communication along the shear axis and hence the decomposition of the transformation into shears. In [Vezina, Fletcher, and Robertson 1992] the implementation was conducted on a Maspar MP-1. Beams of voxels are provided to a toroidally connected processors. This implementation relies on the efficient interconnection network of the MP-1 for optimal communications performance. In the other implementation [Schroeder and Salem 1991] no efforts were made to perform any explicit distribution or virtualization on the CM-2. Indirect addressing was employed and data exchange was avoided until the composition phase of the algorithm. This implementation however suffers from two disadvantages, namely that more number of one-dimensional resampling passes are required and perspective viewing is not handled.

Splatting is another reconstruction technique which has gained attention in the parallel volume rendering community. The earliest splatting implementation was conducted on a network of SUNs by Westover [Westover 1990]. Voxels are enumerated in a front-to-back or back-to-front order and a subset of these is sent to N^2 processes which are executed on a network of Suns. These processes transform and splat the voxels onto image sheets which are then sent to N compositing processes.

Machiraju and Yagel implement a similar splatting algorithm on a IBM Power Visualization System (PVS) [Machiraju and Yagel 1993]. In this method a computational scheme is utilized which allows very efficient transformations based on incremental vectorized computation. Also, the volume is statically divided into an ordered set of slabs (of slices). Each of the slices in a slab is independently transformed, shaded, splatted and composited with the existing local image. A tree-like algorithm is used to hierarchically combine all these local images to create the final image. Thus, inter-node communication is required only in this final stage. The amount of communication required is low and this implementation scales well with increased number of processors.

In Neumann's implementation on the Pixel Planes 5 architecture [Yoo et al. 1992] graphics processors transform and shade the voxels, while the rendering processors implement splatting and receive the requisite instruction stream from the graphics processors. Each rendering processor is assigned a subsection of the screen. Only voxels which map to that screen area are sent to this renderer. A sorting step is required for generating the splat instructions for appropriate renderers. A graphics processor waits for the a circulating token to reach it before it can send the next slice for rendering. While waiting, it transforms the next assigned slice. One disadvantage of this implementation is the sorting which is required for each of the slices.

4.1.2. Parallel Backward Viewing Methods

Backward-feed or image space methods have received a lot of attention in the parallel volume rendering community. Neih and Levoy's [Neih and Levoy 1992] contribution lies in the development of a hybrid data distribution scheme. The screen is divided into several regions which are again subdivided into tiles. Each node of the DASH multicomputer is assigned a portion of the screen. If a node finishes raycasting and finds another region undone, it grabs the next undone tile in a scanline order. To alleviate non-local access the volume is distributed in a round-robin fashion among the nodes. The efficient caching subsystem of the DASH multiprocessor is relied upon to fetch non-local data.

Montani et. al.'s implementation of a ray-tracer on the Intel iPSC/2 used a purely static data distribution scheme [Montani, Perego, and Scopingo 1992]. The scanlines are divided among clusters of processors in an interleaved fashion. The entire volume data is replicated on all clusters with each node of the cluster getting a slab of slices. Rays are traced by each node with in its subvolume using a ray-dataflow approach, wherein a ray is split into several portions and each portion is traced individually. Later when each node is finished all these portions are combined to obtain the color. Such a scheme can lead to a load balanced and scalable implementation and the data distribution scheme maps well to the Hypercube interconnection network.

Fruhaff's implementation on a multiprocessor Silicon Graphics workstation [Fruhauff 1992] is similar in spirit to Schroeder and Salem's implementation on CM-2. The volume is rotated along the viewing rays and then parallel rays are cast. A dynamic data distribution scheme is used to assign the slices to the

various nodes An efficient incremental transformation method, was used for transforming each slice.

Corrie and Mackerras employed the Fujitsu AP1000 MIMD multiprocessor to implement a ray-caster [Corrie and Mackerras 1992]. A master slave paradigm was used to dynamically distribute square regions of pixels to slave cells. An adaptive distribution scheme is obtained by having the slave cells notify the master when they spend more than their allocated time to render the assigned image. The master then subdivides the image region further and distributes the new sub-regions to idle cells. To support such a dynamic scheme the volume is replicated among clusters of neighboring cells.

The template based viewing method [Yagel 1991b] has been successfully implemented in [Cameron and Underill 1992, Schroeder and Stoll 1992] on SIMD machines. Both implementations are very similar. Parallel rays are traced in a lock step fashion by all nodes of the SIMD node. Each node is mapped to a voxel in both implementations. Shading, interpolation and compositing is done by each processor along the ray. After a set of rays have been completely traced, new rays are traced by conducting shifts along one axis.

Recently, the CellFlow method [Law, Yagel, and Jayasimha 96. , Law and Yagel 1995] , has been implemented on some distributed shared memory architectures. It is based on exploiting coherency between frames in an animation sequence. Once the data is distributed among processors, it is observed that, if the animation is rather smooth, the amount of additional information needed for the next frame is small. Lookahead of possible data needed for the next frame is used to provide effective latency hiding by performing communication in the background.

4.1.3. Parallel Hybrid Methods

Hybrid methods have not drawn the same amount of attention as forward backward feed methods. Only two reported implementations [Machiraju and Yagel 1993, Wittenbrink and Somani 1993] and one recent work [Law and Yagel 96. , Law and Yagel 1996] are known. Wittenbrink and Somani's method is applicable to affine transformations. They decompose the transformation into a series of shears used to determine the address of a voxel in the transformed space. This address is used for interpolating in object space and obtaining the the sample value. The advantage of this address computation scheme is that communication required for interpolation is nearest neighbor and regular. In the final stages another communication step is required for actually rearranging the transformed volume.

In [Machiraju and Yagel 1993], a hybrid method was implemented on the IBM Power Visualization System (PVS). The volume data is subdivided into sub-cubes and assigned to different processors. The transformed extents of these sub-cubes are determined in the image space. These image extents are then traversed in scanline order and interpolated in object space. An efficient inverse incremental transformation is employed to obtain points in object space.

Recently [Law and Yagel 96. , Law and Yagel 1996], the *Active Ray* method, which exploits coherency in object-space, was implemented. The data is divided into cells that are distributed randomly to processors. Rays are intersected with the cell boundaries and are placed in a queue associated with the cell they intersect first. These cells are brought into memory by demand in a front-to-back order. Rays queued at a cell that was brought into the processor's memory are advanced in the cell. They either stop due to opacity accumulation, or are queued at the cell they enter next. This hybrid method benefits from both the advantages of image order (e.g., early ray termination) and object order (e.g., regular space traversal) approaches.

5. Special Purpose Hardware

An obvious way to accelerate volume rendering is to build a special purpose hardware well suited for this task. Several attempts have been launched, however, no one have gained market acceptance and have mostly remained research attempts.

Insight [Meagher 1985] employs an octree data structure which is a hierarchical form that enables the rendering process to traverse only the non-empty voxels. While octrees are suitable for relatively uniform and regular objects such as in solid modeling, they cease to be effective when the application calls for complex objects and/or many colors/densities as in medical imaging. Consequently, all other architectures store the volumetric data as a 3D array, while gaining speedup by employing diverse parallelism mechanisms. PARCUM [Jackel 1985] takes advantage of a memory interleaving scheme that allows parallel retrieval of macro-voxels (e.g., a $3\times3\times3$ group of voxels). Cube [Kaufman and Bakalash 1988] uses a unique skewed memory organization that allows parallel read/write of axial beams. The Voxel Processor [Goldwasser et al. 1989] and $3DP^4$ [Ohashi, Uchiki, and Tokoro 1985] achieve parallelism by dividing the volumetric space into sub-cubes and pipelining the merging of the sub-results in a hierarchical fashion. The Flipping Cube architecture [Yagel 1991b] relies on a modular memory that stores a slice of voxels in each module.

One of the major decisions in hardware design is the rendering algorithm to be used. Cube [Kaufman and Bakalash 1988], PARCUM [Jackel 1985], Flipping Cube [Yagel 1991b], and SCOPE [Uchiki and Tokoro 1985] use backward mapping. In Cube, the cast rays are parallel to the main axis. An extended architecture of Cube, Cube-3, for perspective and arbitrary viewing has also been devised [Kaufman, Bakalash, and Cohen 1990]. Flipping Cube supports real-time parallel (orthographic) rendering from arbitrary direction while SCOPE supports perspective viewing as well. PARCUM uses ray casting in the macro-voxel level and employs a Z-buffer for projecting the macro-voxel itself. Insight [Meagher 1985], $3DP^4$ [Ohashi, Uchiki, and Tokoro 1985], and the Voxel Processor [Goldwasser et al. 1989] use forward mapping that directly transforms the object coordinates into screen space in a back-to-front (BTF) traversal [Frieder, Gordon, and Reynolds 1985] for hidden voxel removal. Insight performs octree based recursive BTF traversal of the scene. The Voxel Processor uses recursive BTF to project sub-cubes and then merges the 2D mini-pictures in BTF fashion. $3DP^4$ traverses beams that are parallel to the screen x axis in BTF order employing a Z-buffer for the projection of the beam itself.

6. Viewing Intermixed Volume and Surfaces

The surface-based approach favors a geometric object representation and thus only supports rendering of analytically-defined surfaces. Therefore, the volumetric data must be first transformed into surface-based description by applying a surface reconstruction algorithm [Fuchs, Kedem, and Uselton 1977, Sunguroff and Greenberg 1978]. The opposite approach is the voxel-based approach in which the voxel is the only primitive used for object representation. Geometric objects are converted from their continuous mathematical definition into a volumetric representation by employing scan conversion algorithms [Kaufman and Shimony 1986, Kaufman 1987b, Kaufman 1988]. This volumetric representation is merged directly with the sampled voxel-based data.

Compromising between these two extremes of surface-only and volume-only approaches, are the point-based approach and the hybrid approach. The point-based approach, extends the surface-based approach by adding a point primitive to the set of geometric objects [Johnson and Mosher 1989]. This primitive consists of a 3D location, a normal value used by a geometry engine for illumination calculation, and some additional data values (e.g., color, opacity, density). When synthetic objects are scan converted into voxel representation the normal value can easily be calculated for each voxel. However, sampled volumetric data do not contain information about the surface inclination. Thus, a pre-processing stage of normal restoration is needed in order to convert each non-empty voxel to a point. This is usually done by examining a close neighborhood of the voxels in order to extract a surface and calculate its normal [Cline et al. 1988, Herman and Udupa 1981, Hoehne and Bernstein 1986a].

Since common volumetric data sets consist of millions of voxels and as contemporary graphic hardware can not cope with such a magnitude of objects, a reduction in the amount of points is desired. This reduction is achieved by feeding the graphic engine only with those points (voxels) which are visible, that is, those voxels comprising the outer layer of the objects. One way to extract these voxels from the volume is by thresholding; voxels belonging to an object are assigned the value '1' while the others are assigned a '0'. The '1' voxels bordering with at least one '0' voxel are those possibly visible [Cline et al. 1988]. A more efficient and accurate method is based on starting from a seed point on the surface and then tracking its neighbors until the entire surface is extracted [Frieder et al. 1985].

It should be noted that the point-based approach makes use of the point primitive only for display purpose, while all the other volumetric manipulations are performed on the volume itself. Some operations require only minor alterations of the point list (e.g., scalpeling), while others require more extensive changes (e.g., cut planes), or even a total rebuilding of the point-list (e.g., change in the translucency parameters). The point primitive can be rendered either by using a geometry engine to render it as a point in 3D space [Cline et al. 1988] or by applying a *splatting technique* that computes the contribution of the projected point to a neighborhood of pixels [Westover 1989].

The fourth approach to the intermixing problem is the hybrid approach which supports the hybrid data model by rendering each part of it separately and then combining the surface rendered and the volume rendered images into a final 2D pixel image. Two flavors of this approach are described, the Z-merging and the ray-merging.

The *Z-merging* algorithm [Kaufman, Yagel, and Cohen 1990] employs two separate (and possibly parallel) rendering processes - one executes (any) volume rendering algorithm on the volumetric data set while the other performs (any) traditional surface-based rendering on the geometric model. Each rendering process produces two buffers consisting of the shaded image along with the corresponding Z-buffer. The two image buffers are combined according to their depth values. It should be noted that after the image is rendered for the first time, the system waits for a change that requires re-rendering and then renders only the necessary model. For example, when a surgeon simulates the positioning of a prosthesis, he can interactively move the (geometric) prosthesis over a static (volumetric) organ since only the geometric model is repeatedly re-rendered. Another flavor of the hybrid approach is the ray-merging method which simultaneously traces rays through both the volumetric data set and the synthetic model producing two vectors of samples. These two vectors are combined to produce the final image value [Goodsell, Mian, and Olson 1989, Levoy 1990].

7. Viewing Irregular Grids

There are many types of grids. We deal only with those composed of cells that are bounded by a set of general simple polygons (i.e., non intersecting, without holes, potential concave, and possibly non-planar). We distinguish between rectilinear, structured, and unstructured grids [Speray and Kennon 1990]. Rectilinear grids are composed of a set of connected cells of rectangular prism (brick) shape. The set of voxels completely tessellates a rectangular cartesian sub-space. Imposing a requirements on the voxels to be homogeneous rectangular prisms yields the regular grids which are very common in biomedical applications when scanning resolutions are unequal in all dimensions. Imposing a restriction on the voxels to be cubical yields the cartesian grids which are the most common volumes.

Structured grid, commonly found in Computational Fluid Dynamic (CFD) applications, result from applying non-linear transformation to a rectilinear grid, yielding a grid composed of hexahedral voxels. A structured grid is not necessarily convex and may have holes, but we assume it is connected. When the grids do not have an implicit neighborhood (connectivity) information, like in the previous grid types, we call it an unstructured grid. In the general case, each cell can be an arbitrary polyhedra, however, in this paper we restrict our discussion to unstructured grids composed of tetrahedral cells.

One major advantage of tetrahedral grids is that the faces of voxels are simple convex and planar polygons (triangles) and the basic cell is always convex. Dealing with the various types of tetrahedra is an order of magnitude easier that figuring out all configurations of even hexahedral cells [Williams 1992a],

not the mention arbitrary polyhedra. Another major advantage of a tetrahedral grid is that other types of grids can be converted to it. Structured grids can very easily be converted into tetrahedral grids. One way is to convert each hexahedra into 5 tetrahedra, however this is not a uniform division which may introduce the cracking problem [Shirley and Tuchman 1990]. We either have to coherently divide each cell according to the way we divided its neighbors or alternatively we could divide the hexahedra uniformly along each axis, yielding 12 tetrahedra , or uniformly along all axes, yielding 24 tetrahedra. General unstructured grids can also be converted into tetrahedral grid by applying some tetrahedration algorithm to the set of grid points. (e.g., a Delaunay triangulation) [Max, Hanrahan, and Crawfis 1990].

The most obvious way to render irregular grids is to re-sample them into a regular grid and render them with available methods. One way of re-sampling is to send rays and save the samples in a 3D buffer. Another possibility is to intersect the irregular grid with the boxes that comprise the regular grid. In each box, compute the size of the volumes (partially) residing inside the box, and compose a weighted sum of their contribution. Note that this time consuming re-sampling process has to be done very carefully in order to maintain data integrity and quality. In many applications, cells vastly vary in their size; maintaining the resolution of the smallest cells may require an impractical regular grid resolution. Finally, the re-sampling process has to be repeated for most changes in the viewing parameters (definitely for zoom-in and probably for eye position also).

Direct rendering methods have been developed to render irregular volumes. These methods can be classified as object order or image order. Object order methods traverses and enumerates all voxels of a solid and determines, for each voxel, the affected pixels on a screen. Image order techniques, on the other hand, scans the screen elements and determine, for each pixel, which voxels of a solid affect it. There exist also some methods that combine the two traversal schemes into a hybrid approach as described later. The algorithm described in this paper belongs to the hybrid category. Since we deal with voxel objects we call the first approach voxel-space, and conversely we call the second pixel-space approach.

7.1. Voxel-Space Methods

Voxel-space methods are also called projective or feed-forward methods. The viewing transformation matrix is applied to all voxels (enumerated in some order), thus providing an intermediate volume which is then projected to a two dimensional screen. These methods rely on three basic operations:

(i) Transformation of the voxel vertices from object to image space.
(ii) Sorting the transformed voxels in depth order.
(iii) Rendering each voxel in a back-to-front or front-to-back order.

Phase (i) implementation is straight forward but phase (ii) poses some major difficulties. In general, it is not always possible to sort even a collection of tetrahedral cells and certainly not arbitrary (possibly concave) polyhedra. Only acyclic meshes and tetrahedral meshes generated by Delaunay triangulation can be depth sorted. Max et al [Max, Hanrahan, and Crawfis 1990] have presented a topological sort for acyclic grids composed of convex polyhedra with planar faces (see also [Williams 1992a]). Phase (iii) can also be implemented various ways. One way is to render only the front faces of the tetrahedra, assigning each vertex with the integrated value from that vertex to the back of the tetrahedra [Shirley and Tuchman 1990]. Alternatively, front and back faces of the tetrahedra are rendered separately and the value is integrated between the far and near points [Max, Hanrahan, and Crawfis 1990]. Finally, splatting can be employed [Williams 1992b]. An alternative not yet explored is the extension of the V-buffer algorithm [Upson and Keeler 1988] to polyhedral cells. In this method each transformed cell is intersected by a plane defined by each screen scan-line. The resulting polygon is divided into spans in which integration takes place. We propose a similar technique that intersects the whole transformed mesh with planes that are parallel to the screen.

The major advantage of the voxel based approach is that many operations can be performed by available graphics hardware. Vertex transformation (i) can be done by graphics hardware as well as some rendering operations [Shirley and Tuchman 1990]. Additionally, voxel-based methods tend to run faster than pixel-based methods. However, this approach suffers from few advantages; first is that the whole

process have to be repeated since unlike rectilinear grids the sorting (phase ii) is view dependent. A more serious difficulty is that the sort operation can be applied only to limited types of grids as mentioned above. We present an algorithm that overcomes this last difficulty.

7.2. Pixel-Space Methods

Pixel-based method, also called, backward-feed methods or ray-casting have been adapted for rendering volumes [Brewster et al. 1984, Levoy 1988]. These methods have been applied to irregular grids [Garrity 1990] in an algorithm that casts ray(s) from each screen-pixel. For each ray the algorithm performs:

(i) Find the first cell the ray intersect.
(ii) Searches between the cell neighbors for the exit point
(iii) Between the entry and exit points, sample and interpolating values, and compose them with the cumulative ray value.
(iv) Unless it is the last voxel along the ray, use the exit point as the entry point to the next cell and return to step (ii).

This approach suffers from several difficulties. First, the calculation of (i) as well as the determination if a cell is last (step (iv)) in the case of non convex grid can be very difficult and time consuming. A possible solution is to embed the boundary cells in a regular space-subdivision grid [Garrity 1990]. A major difficulty in the pixel-space approach that in order to perform step (ii) efficiently neighborhood information is required. For unstructured grid, the algorithm must be preceded, therefore, by a process that associates with each cell information regarding its neighbors. Quality of images is low due to point sampling in both image and object spaces. We may completely miss small cells or cells that fit between rays in perspective viewing. A possible direction for research will explore methods for rendering cylinder-shaped (or cone-shaped in perspective) rays rather than rays that simulate lines. Finally, pixel-space rendering is expected to be slow, specially when phase (ii) involves sampling in arbitrarily shaped voxels.

7.3. Hybrid Methods

Hybrid Methods have been investigated [Upson and Keeler 1988]. In these methods the volumes are traversed in object order. The contribution of a set of voxels to a pixel is then computed in image order. Each transformed cell is intersected by a plane defined by each screen scan-line. The resulting polygon is divided into spans in which integration takes place. This methods was extended to investigated for irregular grids by [Giertsen 1992]. Recently [Yagel et al. 1996], we introduced an efficient method for rendering unstructured grids that is based on incremental slicing and hardware polygon rendering. For a given view direction, the grid vertices are transformed to image space using available graphics hardware. We then incrementally compute the 2D polygon-meshes that result from letting a set of equidistant planes, parallel to the screen plane, intersect (slice) the transformed grid. Finally, we use the graphics hardware to render (interpolate-fill) the polygon-meshes and composite them in a front-to-back order. This method also provides adaptive control and progressive image generation.

The resulting polygon is divided into spans in which integration takes place. As in [6], our approach is also based on incremental slicing; however, our method can employ available rendering hardware to achieve interactive rendering, is not sensitive to image resolution, and supports adaptive and progressive rendering.

Acknowledgments

This work was partially supported by DARPA BAA 92-36, the National Science Foundation CDA 9413962, US Department of Energy DE-PS07-95ID113339, and General Electrics.

8. References

Avila, R., Sobierajski, L. M., and Kaufman, A. E., ‘‘Towards a Comprehensive Volume Vsialization System’’, *Proceedings Visualization ’92*, Boston, MA, 13-20, October 1992.

Bakalash, R. and Kaufman, A., ‘‘MediCube: a 3D Medical Imaging Architecture’’, *Computers & Graphics*, **13**, 2, (1989).

Bergman, L., Fuchs, H., Grant, E., and Spach, S., ‘‘Image Rendering by Adaptive Refinement’’, *Computer Graphics*, **20**, 4, 29-37, (August 1986).

Brewster, L. J., Trivedi, S. S., Tuy, H. K., and Udupa, J. K., ‘‘Interactive Surgical Planning’’, *IEEE Computer Graphics and Applications*, **4**, 3, 31-40, (March 1984).

Cameron, G. and Underill, P. E., ‘‘Rendering Volumetric Medical Image Data on a SIMD Architecture Computer’’, *Proceedings of Third Eurographics Workshop on Rendering*, May 1992.

Challinger, J., " ‘‘Parallel Volume Rendering for Curvilinear Volumes"’’, *Proceedings of the Scalable High Performance Computing Conference*, 14-21, 1992.

Cline, H. E., Lorensen, W. E., Ludke, S., Crawford, C. R., and Teeter, B. C., ‘‘Two Algorithms for the Three-Dimensional Construction of Tomograms’’, *Medical Physics*, **15**, 3, 320-327, (May/June 1988).

Cline, H. E., Ludke, S., Lorensen, W. E., and Teeter, B. C., ‘‘A 3D Medical Imaging Research Workstation’’, *Computer Graphics and Image Processing (submitted)*, .

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, , 280-301, 1990.

Cohen, D. and Kaufman, A., ‘‘3D Discrete Lines: Voxelization Algorithms and Connectivity Control’’, TR 91.05.09, Computer Science, SUNY at Stony Brook, May 1991.

Cohen, D. and Shefer, Z., ‘‘Proximity Clouds - An Acceleration Technique for 3D Grid Traversal’’, Technical Report FC 93-01, Department of Mathematics and Computer Science, Ben Gurion University of the Negev, February 1993.

Corrie, B. and Mackerras, P., ‘‘Parallel Volume Rendering and Data Coherence on the Fujitsu AP100’’, Technical Report TR-CS-92-11, Department of Computer Science, Australian National University, Canberra, ACT, Australia, 1992.

Danskin, J. and Hanrahan, P., ‘‘Fast Algorithms for Volume Ray Tracing’’, *Proceedings of 1992 Workshop on Volume Visualization*, Boston, MA, 91-105, October 1992.

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

Elvins, T. T., ‘‘Volume Rendering on a Distributed Memory Parallel Computer’’, *Proceedings Visualization ’92*, Boston, MA, 93-98, October 1992.

Farrell, E. J., Zappulla, R., and Yang, W. C., ‘‘Color 3D Imaging of Normal and Pathologic Intracranial Structures’’, *IEEE Computer Graphics and Applications*, **4**, 9, 5-17, (September 1984).

Farrell, E. J., Yang, W. C., and Zappulla, R. A., ‘‘Animated 3D CT Imaging’’, *IEEE Computer Graphics*

and Applications, **5**, 12, 26-32, (December 1985).

Frieder, G., Herman, G. T., Meyer, C., and Udupa, J., ‘‘Large Software Problems for Small Computers: An Example from Medical Imaging’’, *IEEE Software*, **2**, 5, 36-47, (September 1985).

Frieder, G., Gordon, D., and Reynolds, R. A., ‘‘Back-to-Front Display of Voxel-Based Objects’’, *IEEE Computer Graphics and Applications*, **5**, 1, 52-60, (January 1985).

Fruhauff, T., ‘‘Volume Rendering on a Multiprocessor Architecture with Shared Memory: A Concurrent Volume Rendering Algorithm’’, *Proceedings of the Third Eurographics Workshop on Scientific Visualization*, Pisa, Italy, April 1992.

Fuchs, H., Kedem, Z. M., and Uselton, S. P., ‘‘Optimal Surface Reconstruction from Planar Contours’’, *Communications of the ACM*, **20**, 10, 693-702, (October 1977).

Fujimoto, A., Tanata, T., and Iwata, K., ‘‘ARTS: Accelerated Ray-Tracing System’’, *IEEE Computer Graphics and Applications*, **6**, 4, 16-26, (April 1986).

Garrity, M. P., ‘‘Raytracing Irregular Volume Data’’, *Computer Graphics*, **24**, 5, 35-40, (November 1990).

Giertsen, C., ‘‘Volume Visualization of Sparse Irregular Meshes’’, *IEEE Computer Graphics & Applications*, **12**, 2, 40-48, (March 1992).

Goldwasser, S. M., ‘‘Rapid Techniques for the Display and Manipulation of 3-D Biomedical Data’’, *Proceedings NCGA’86 Conference*, **II**, 115-149, (May 1986).

Goldwasser, S. M. and Reynolds, R. A., ‘‘Real-Time Display and Manipulation of 3-D Medical Objects: The Voxel Processor Architecture’’, *Computer Vision, Graphics and Image Processing*, **39**, 1, 1-27, (July 1987).

Goldwasser, S. M., Reynolds, R. A., Talton, D. A., and Walsh, E. S., ‘‘High Performance Graphics Processors for Medical Imaging Applications’’, in *Parallel Processing for Computer Vision and Display*, P. M. Dew, R. A. Earnshaw, and T. R. Heywood, (eds.), Addison Wesley, Reading, MA, , 461-470, 1989.

Goodsell, D. S., Mian, S., and Olson, A. J., ‘‘Rendering of Volumetric Data in Molecular Systems’’, *Journal of Molecular Graphics*, **7**, 41-47, (March 1989).

Gordon, D. and Reynolds, R. A., ‘‘Image Space Shading of 3-Dimensional Objects’’, *Computer Graphics and Image Processing*, **29**, 3, 361-376, (March 1985).

Greene, N., ‘‘Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space’’, *Computer Graphics*, **23**, 3, 175-184, (July 1989).

Gudmundsson, B. and Randen, M., ‘‘Incremental Generation of Projections of CT-Volumes’’, *Proceedings of the First Conference on Visualization in Biomedical Computing*, Atlanta, GA, 27-34, May 1990.

Hanrahan, P., ‘‘Three-Pass Affine Transforms for Volume Rendering’’, *Computer Graphics*, **24**, 5, 71-78, (November 1990), Proceedings of San Diego Workshop on Volume Visaulization .

Herman, G. T. and Liu, H. K., ‘‘Three-Dimensional Display of Human Organs from Computed Tomograms’’, *Computer Graphics and Image Processing*, **9**, 1, 1-21, (January 1979).

Herman, G. T. and Udupa, J. K., ‘‘Display of Three Dimensional Discrete Surfaces’’, *Proceedings of the SPIE*, **283**, 90-97, (1981).

Hoehne, K. H. and Bernstein, R., ‘‘Shading 3D-Images from CT Using Grey-Level Gradients’’, *IEEE Transactions on Medical Imaging*, **MI-5**, 1, 45-57, (March 1986).

Hoehne, K. H. and Bernstein, R., ‘‘Shading 3D-Images from CT Using Gray-Level Gradients’’, *IEEE*

Transactions on Medical Imaging, **MI-5**, 1, 45-47, (March 1986).

Hoehne, K. H., Bomans, M., Pommert, A., Riemer, M., Schiers, C., Tiede, U., and Wiebecke, G., ‘‘3D-Visualization of Tomographic Volume Data Using the Generalized Voxel Model’’, *The Visual Computer*, **6**, 1, 28-37, (February 1990).

Jackel, D., ‘‘The Graphics PARCUM System: A 3D Memory Based Computer Architecture for Processing and Display of Solid Models’’, *Computer Graphics Forum*, **4**, 1, 21-32, (January 1985).

Jackel, D. and Strasser, W., ‘‘Reconstructing Solids from Tomographic Scans - The PARCUM II System’’, in *Advances in Computer Graphics Hardware II*, A. A. M. Kuijk and W. Strasser, (eds.), Springer-Verlag, Berlin, , 209-227, 1988.

Johnson, E. R. and Mosher, C. E., ‘‘Integration of Volume Rendering and Geometric Graphics’’, *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, NC, 1-8, May 1989.

Jong, J. M., Park, H. W., Eo, K. S., Kim, M. H., Zhang, P., and Kim, Y., ‘‘UWGSP4: An Imaging and Graphics Superworkstation and its Medical Application’’, *Proceedings of Medical Imaging VI: Image Capture, Formatting and Display*, **1653**, 422-433, (February 1992).

Kajiya, J. T. and Von Herzen, B. P., ‘‘Ray Tracing Volume Densities’’, *Computer Graphics*, **18**, 3, 165-174, (July 1984).

Kaufman, A. and Shimony, E., ‘‘3D Scan-Conversion Algorithms for Voxel-Based Graphics’’, *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, Chapel Hill, NC, 45-75, October 1986.

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

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

Kaufman, A., ‘‘Efficient Algorithms for 3D Scan-Converting Polygons’’, *Computers & Graphics*, **12**, 2, 213-219, (1988).

Kaufman, A. and Bakalash, R., ‘‘Memory and Processing Architecture for 3-D Voxel-Based Imagery’’, *IEEE Computer Graphics & Applications*, **8**, 11, 10-23, (November 1988). " .nr t[3

Kaufman, A., " ‘‘The *voxblt* Engine: A Voxel Frame Buffer Processor"’’, , (1989).

Kaufman, A., Bakalash, R., and Cohen, D., ‘‘Viewing and Rendering Processor for a Volume Visualization System’’, in *Advances in Graphics Hardware IV*, R. L. Grimsdale and W. Strasser, (eds.), Springer-Verlag, Berlin, , 1990.

Kaufman, A., Bakalash, R., Cohen, D., and Yagel, R., ‘‘Architectures for Volume Rendering - A Survey’’, *IEEE Engineering in Medicine and Biology*, **9**, 4, 18-23, (December 1990).

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.), Springer-Verlag, , 217-227, 1990.

Laur, D. and Hanrahan, P., ‘‘Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering’’, *Computer Graphics*, **25**, 4, 285-288, (July 1991).

Law, A., Yagel, R., and Jayasimha, D. N., ‘‘VoxelFlow: A Parallel Volume Rendering Method for Scientific Visualization’’, *ISCA Journal of Computers and Their Applications*, April 1996. .

Law, A. and Yagel, R., ‘‘Exploiting Spatial, Ray, and Frame Coherency for Efficient Parallel Volume Rendering’’, *GRAPHICON’96*, Saint-Petersburg, Russia, July 1996. .

Law, A. and Yagel, R., ‘‘CellFlow: A Parallel Rendering Scheme for Distributed Memory Architectures’’, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and*

Applications (PDPTA '95), Athens GA, 3-12, November 1995.

Law, A. and Yagel, R., ''Multi-Frame Thrashless Ray Casting with Advancing Ray-Front'', *Graphics Interface '96*, Toronto, Canada, May 1996.

Leung, L. T. S. and Synott, W., ''G2: The Design and Realization of an Accelerator for Volume Visualization'', *Proceedings of Visualization in Biomedical Computing*, **1808**, 384-394, (1992).

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

Levoy, M., ''Design for Real-Time High-Quality Volume Rendering Workstation'', *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, NC, 85-92, May 1989.

Levoy, M., ''Volume Rendering by Adaptive Refinement'', *The Visual Computer*, **6**, 1, 2-7, (February 1990).

Levoy, M., ''Efficient Ray Tracing of Volume Data'', *ACM Transactions on Graphics*, **9**, 3, 245-261, (July 1990).

Levoy, M. and Whitaker, R., ''Gaze-Directed Volume Rendering'', *Computer Graphics*, **24**, 2, 217-223, (March 1990).

Levoy, M., ''A Hybrid Ray Tracer for Rendering Polygon and Volume Data'', *IEEE Computer Graphics & Applications*, **10**, 3, 33-40, (March 1990).

Machiraju, R., Yagel, R., and Schwiebert, L., ''Parallel Algorithms for Volume Rendering'', OSU-CISRC-10/92-TR29,, Department of Computer and Information Science, The Ohio State University, October 1992.

Machiraju, R. and Yagel, R., ''Efficient Feed-Forward Volume Rendering Techniques for Vector and Parallel Processors'', OSU-CISRC-4/93-TR14,, Department of Computer and Information Science, The Ohio State University, April 1993.

Max, N., Hanrahan, P., and Crawfis, R., ''Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions'', *Computer Graphics*, **24**, 5, 27-33, (November 1990).

Meagher, D. J., ''Geometric Modeling Using Octree Encoding'', *Computer Graphics and Image Processing*, **19**, 2, 129-147, (June 1982).

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

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

Montani, C., Perego, R., and Scopingo, R., ''Parallel Volume Visualization on a Hypercube Architecture'', *Proceedings of 1992 Workshop on Volume Visualization*, Boston, MA, 9-16, October 1992.

Neih, J. and Levoy, M., ''Volume Rendering on Scalable Shared Memory Architecture'', *Proceedings of 1992 Workshop on Volume Visualization*, Boston, MA, 17-24, October 1992.

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

Reynolds, R. A., Gordon, D., and Chen, L. S., ''A Dynamic Screen Technique for Shaded Graphic Display of Slice-Represented Objects'', *Computer Vision, Graphics and Image Processing*, **38**, 3, 275-298, (June 1987).

Sabella, P., ''A Rendering Algorithm for Visualizing 3D Scalar Fields'', *Computer Graphics*, **22**, 4, 51-58, (August 1988).

Samet, H. and Webber, R. E., ''Hierarchical Data Structures and Algorithms for Computer Graphics, Part

II: Applications'', *IEEE Computer Graphics and Applications*, **8**, 7, 59-75, (July 1988).

Schlusselberg, D. S., Smith, K., and Woodward, D. J., ''Three-Dimensional Display of Medical Image Volumes'', *Proceedings of NCGA'86 Conference*, **III**, 114-123, (May 1986).

Schreiter, D. and Zimmerman, J. B., ''Evaluation of 3D Voxel Rendering Algorithms for Real-Time Interaction on a SIMD Graphic Processor'', *Proceedings of SPIE, Medical Imaging II: Image Data Management and Display*, **914**, Part B, 1291-1298, (February 1988).

Schroeder, P. and Salem, J. B., ''Fast Rotation of Volume Data on Data Parallel Architecture'', *Proceedings of Visualization'91*, San Diego, CA, 50-57, October 1991.

Schroeder, P. and Stoll, G., ''Data Parallel Volume Rendering as Line Drawing'', *Proceedings of 1992 Workshop on Volume Visualization*, Boston, MA, 25-31, October 1992.

Shirley, P. and Tuchman, A., ''A Polygonal Approximation to Direct Scalar Volume Rendering'', *Computer Graphics*, **24**, 5, 63-70, (December 1990).

Sobierajski, L., Cohen, D., Kaufman, A., Yagel, R., and Acker, D., ''Interactive Voxel-Based Surgical Planning System'', TR 90.05.22, Computer Science, SUNY at Stony Brook, May 1990.

Sobierajski, L., Cohen, D., Kaufman, A., Yagel, R., and Acker, D., ''A Fast Display Methods for Volumetric Data'', *The Visual Computer*, **10**, (1993).

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

Sunguroff, A. and Greenberg, D., ''Computer Generated Images for Medical Application'', *Computer Graphics*, **12**, 3, 196-202, (August 1978).

Trousset, Y. and Schmitt, F., ''Active-Ray Tracing for 3D Medical Imaging'', *Proceedings of EUROGRAPHICS '87*, Amsterdam, The Netherlands, 139-150, August 1987.

Tuy, H. K. and Tuy, L. T., ''Direct 2-D Display of 3-D Objects'', *IEEE Computer Graphics & Applications*, **4**, 10, 29-33, (November 1984).

Uchiki, T. and Tokoro, M., ''SCOPE: Solid and Colored Object Projection Environment'', *Transaction of the Institute of Electronics and Communication Engineers of Japan*, **68-D**, 4, 741-748, (in Japanese), (April 1985).

Upson, C. and Keeler, M., ''V-BUFFER: Visible Volume Rendering'', *Computer Graphics*, **22**, 4, 59-64, (August 1988).

van Walsum, T., Hin, A. J. S., Versloot, J., and Post, F. H., ''Efficient Hybrid Rendering of Volume data and Polygons'', *Second Eurographics Workshop on Visualization in Scientific Computing*, Delft, Netherlands, April 1991.

Vezina, G., Fletcher, P., and Robertson, P. K., ''Volume Rendering on the MasPar MP-1'', *Proceedings of 1992 Workshop on Volume Visualization*, Boston, MA, 3-8, October 1992.

Westover, L., ''Interactive Volume Rendering'', *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, NC, 9-16, May 1989.

Westover, L., ''Footprint Evaluation for Volume Rendering'', *Computer Graphics*, **24**, 4, 367-376, (August 1990).

Williams, P. L., ''Visibility Ordering Meshed Polyhedra'', *ACM Transactions on Graphics*, **11**, 2, 103-126, (April 1992).

Williams, P. L., ''Interactive Splatting of Nonrectilinear Volumes'', *Proceedings Visualization '92*, Boston, MA, 37-44, October 1992.

Wittenbrink, C. M. and Somani, A. K., ''2D and 3D Optimal Image Warping'', *Proceedings of Seventh*

International Parallel Processing Symposium, Newport Beach, CA, April 1993.

Yagel, R., Cohen, D., Kaufman, A., and Zhang, Q., ‘‘Volumetric Ray Tracing’’, TR 91.01.09, Computer Science, SUNY at Stony Brook, January 1991.

Yagel, R., ‘‘The Flipping Cube: A Device for Rotating 3D Rasters’’, *6th Eurographics Hardware Workshop*, Vienna, Austria, September 1991.

Yagel, R., ‘‘Efficient Methods for Volume Graphics’’, Doctoral Dissertation, Department of Computer Science, SUNY at Stony Brook, December 1991.

Yagel, R., Kaufman, A., and Zhang, Q., ‘‘Realistic Volume Imaging’’, *Proceedings of Visualization’91*, San Diego, CA, 226-231, October 1991.

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

Yagel, R. and Kaufman, A., ‘‘Template-Based Volume Viewing’’, *Computer Graphics Forum*, **11**, 3, 153-157, (September 1992).

Yagel, R. and Shi, Z., ‘‘Accelerating Volume Animation by Space-Leaping’’, OSU-CISRC-3/93-TR10,, Department of Computer and Information Science, The Ohio State University, March 1993.

Yagel, R., Reed, D. M., Law, A., Shih, P., and Shareef, N., ‘‘Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing’’, *Volume Visualization Workshop*, San Fransisco, CA, September 1996.

Yazdy, F. E., Tyrrell, J., Riley, M., and Winterbottom, N., ‘‘CARVUPP: Computer Assisted Radiological Visualization Using Parallel Processing’’, in *3D Imaging in Medicine: Algorithms, Systems, Applications*, K. H. Hoehne, H. Fuchs, and S. M. Pizer, (eds.), Springer-Verlag, Berlin, , 363-375, June 1990.

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

Zuiderveld, K., Koning, A. H. J., and Viergever, M. A., ‘‘Acceleration of Ray Casting Using 3D Distance Transforms’’, *Proceedings of Visualization in Biomedical Computing 1992*, **1808**, 324-335, (October 1992).