Malaysian Journal of Computer Science, Vol. 12 No. 2, December 1999, pp. 19-26

HYBRID SHEAR-WARP RENDERING

M. Nordin Zakaria
Multimedia Department
Universiti Putra Malaysia
43400 UPM Serdang, Selangor, Malaysia
email: nordinz@fsas.upm.edu.my

N. Selvanathan
Faculty of Computer Science and
Information Technology
Universiti Malaya
email: selva@fsktm.um.edu.my

ABSTRACT

Shear-warp rendering is a fast and efficient method for visualizing a volume of sampled data based on a factorization of the viewing transformation into a shear and a warp. In shear-warp rendering, the volume is resampled, composited and warped to obtain the final image. Many applications, however, require a mixture of polygonal and volumetric data to be rendered together in a single image. This paper describes a new approach for extending the shear-warp rendering to simultaneously handle polygonal objects. A data structure, the zlist-buffe, is presented. It is basically a multilayered z-buffer. With the zlist-buffer, an object-based scan conversion of polygons requires only a simple modification of the standard polygon scan-conversion algorithm. This paper shows how the scan conversion can be integrated with shear-warp rendering of run-length encoded volume data to obtain quality images in real time. The utility and performance of the approach using a number of test renderings is also discussed.

Keywords: ***Volume Rendering, Polygon Rendering, Shear-Warp Factorization***

1.0 INTRODUCTION

Integrated rendering of polygon and volume data finds applications in numerous problems. Rhinosurgeons need to visualize the air flow in the human nose backdrop against the physical anatomy. Engineers need to be able to visualize material defects recorded using ultrasound together with CAD-generated geometrical objects. Scanned data of a patient's anatomy can be converted to a mathematical solid model of the bone or joint that needs repair. Doctors can use these models as visualisation tools and as dimensional references when designing CAD models of implants. Furthermore, virtual reality for medical simulation and training requires mixtures of geometric and sampled data to be rendered together in order to achieve effective photorealism.

A number of approaches have been described for integrated polygon-volume rendering. Two classes of approaches can be identified:

1) transforming the data to a uniform representation, i.e. from geometric to volumetric representation or vice versa.
2) preserving the data in its original representation, and merging the results of one or more rendering processes.

A common example of the first approach is voxelization [8, 9], a process of scan-conversion from geometric representation into volumetric form. The reverse process, converting from volumetric data to geometric representation typically employs a surface-extraction algorithm such as the Marching Cube algorithm [16, 17]. Both approaches require binary classification of the volume data, and are subject to aliasing, in the form of spurious, disconnected or missing surfaces or features.

The second approach, commonly known as hybrid rendering, preserves the original representations, hence avoiding conversion artifacts. Z-merging [10] is one instance of such approach. The algorithm creates two z-buffer maps from separate renderings of volume and geometric data. A single image is then formed by depth-sorting and compositing the two separately rendered images. This method cannot correctly render translucent polygons interpersed with the volume data.

Ray-merging is another of the hybrid rendering approach. Goodsell, Mian and Olson [6] first render geometric data with z-buffering. Rays are then casted into the volume data, and the volume is sampled at regular interval along the rays. For each ray, using the z-buffer, samples are partitioned into a sequence for which each element is closer than the rendered geometric data, and another sequence for which each element is further. Merging is then done in a manner which is conceptually similar to that in z-merging. And as is z-merging, it fails to account for translucency.

Levoy [15] casts rays into the volume. The volume is sampled at regular interval along the ray. In Levoy's approach, however, all intersections between the ray and the polygons in the environment are independently computed and shaded. Samples due to volume sampling and that due to ray-polygon intersection are then depth-sorted together and composited to compute the value of a single pixel in the viewing plane. As in any conventional ray-caster, due to the need for ray traversal, random access to volumetric data, and intersection calculation with surfaces, rendering time is slow and increases with the complexity of the geometry. Sobierajski and Kaufman [19] extend the concept of volumetric ray casting to ray tracing, using secondary rays. A common ray is used in the computation of intersection with both geometric and

volumetric data. Intersections along rays are resolved by type, and contributions due to the intersection computed accordingly. While hybrid ray tracing produces the best images, being capable of simulation of phenomena normally associated with conventional ray tracing, it lacks fast execution, especially with complex geometries.

Ebert et al [4] uses a scanline a-buffer rendering algorithm for the surface-defined objects in the hybrid scene. An a-buffer is first created for a scanline, containing a list for each pixel of all the fragments that partially or fully cover the pixel. Then, if a volume is active for a pixel, the extent of volume rendering needed is determined. Actual volume rendering follows, creating a-buffer fragments for separate sections of the volume, terminating when full coverage of the pixel by volume or surface-defined elements is achieved. The volume a-buffer fragments are then sorted into the a-buffer fragment list based on their average z-depth values and the entire list is together rendered to produce the final color of the pixel.

Rendering an intermixture of polygons and volumetric object is relatively straightforward using texture-mapping based algorithm. The method described in the SIGGRAPH 1997 course on advanced rendering with OpenGL [22] is capable of rendering both opaque and transparent geometric primitives embedded within the volume. However, texture-mapping based volume rendering works efficiently and robustly only in the presence of 3D texture-mapping hardware. Furthermore, the method does not easily lend to a flexible choice of sophisticated shading or classification scheme.

An inherent feature in many of the hybrid rendering approaches is the creation and composition of depth-sorted lists of volume and polygon samples. For the algorithm to be introduced in this paper, an instance of how such list can be created and utilized for hybrid rendering based on the shear-warp factorization is shown. The focus is on rendering in the context of parallel-projection, the de facto viewing scheme as far as volume rendering is concerned. As for geometry or surface, this paper focuses on simple, convex polygon. More complex geometry can be simplified using a tessellation algorithm [18].

The rest of this paper is organized as follows: Section 2 discusses the motivation for using the shear-warp algorithm as the foundation of a new hybrid-rendering algorithm. Section 2.1 discusses the concept of shear-warp volume rendering. Section 3 discusses the zlist-buffer data structure, section 4 discusses polygon scan-conversion using the zlist-buffer data structure. Section 5 discusses how the volume samples are composited along with the zlists in the zlist-buffer, and finally sections 6 and 7 present the results and conclusion.

2.0 SHEAR-WARP VOLUME RENDERING

A number of approaches have been proposed for volume rendering. The more dominant approaches can be categorized into the following classes: ray tracing [14], splatting [13], cell-projection [20], shear-warp [11], and texture-based algorithms [1]. In designing a hybrid rendering algorithm, the “pure” volume rendering algorithm that was used as the base for our design must be fast, and has reasonable image quality. By fast, it is meant that it must perform within a few seconds for a typical medical 128 x 128 x 128 medical volume data on a desktop PC with no graphics accelerator. As for the image quality, needless to say, the rendering algorithm must generate image, which for the general case is relatively free of artefacts.

The shear-warp volume-rendering algorithm was chosen. VolPack [21], a volume rendering engine based on the algorithm, proves to be good enough for the purpose, requiring no exceptional hardware support, and with rendering performance within expectation. The freely available source code serves as a useful archive for reusable codes in the task of building an experimental hybrid visualization system.

2.1 Algorithm

The rendering problem in volume rendering involves the projection of a volume data onto a 2D viewing or image plane. The orientation of the viewing plane depends on the viewing vector, and this in turn depends on the viewing transformation. In shear-warp volume rendering, the viewing transformation is factored into a shear and a warp. To simplify the factorization, the volume must be transposed so that the viewing vector is most parallel with the third axis of the volume object co-ordinate system. The transposition transforms the volume object co-ordinate system to what is called the volume standard object co-ordinate system. Both co-ordinate systems are right-handed. A permutation matrix is used for the transposition [12]. Fig. 1 depicts a volume object co-ordinate system. The axis of co-ordinate system shown are labelled x, y, and z. For the standard object co-ordinate system, they are labelled i, j and k respectively.

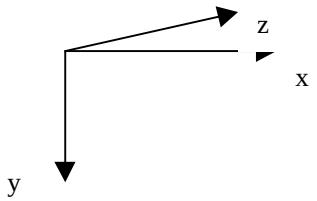


Fig. 1: Volume object co-ordinate system

The primary advantage of the shear-warp factorization is that it simplifies the resampling required during the projection. The factorization also enables the formulation of the rendering algorithm as an image space technique capable of utilizing both images and objects space

coherence. The algorithm uses an efficient scanline-order traversal to stream through the data structures in storage order, thereby reducing overhead incurred by accessing the data structures. Runlength encoding of both the volume and the image enables the traversal to skip over transparent voxels and opaque pixels [11].

Fig. 2 shows the basic concept behind a shear-warp volume-rendering algorithm. Within the standard object coordinate system, the viewing vector is most parallel to the third axis, but it needs not be perpendicular to the slices orthogonal to the third axis. However, resampling volume samples along a viewing ray is simplest when the ray is perpendicular to the slices. The shear factor in the shear-warp factorization creates this situation; it causes the viewing vector to become perpendicular to the slices. The shear is implemented by translating and resampling each slice of the transposed volume data. Projection is then trivial. If the k-component of the viewing vector is positive, then the resampled slices are composited together in a front-to-back order using the "over" to form an intermediate image. Otherwise, the same operation still takes place, but with the composition order being back-to-front.

The intermediate image will appear distorted. This is because the shear transformation is only a factor of the entire viewing transformation. The remaining factor is a general affine warp, and it can be implemented as a 2D warp.

Full details on the shear-warp factorization and rendering, including the mathematical derivations, can be found in the study by Lacroute [12].

3.0 THE ZLIST DATA STRUCTURE

Volume rendering basically assumes a volume data to be a colored semi-transparent gel-like object. Samples along viewing rays are composited together to obtain the color and opacity of pixels on the image plane. Surfaces are not explicitly defined or detected. Instead, they can be understood as a natural byproduct of the stepwise accumulation of color and opacity along a ray [14]. To integrate polygon rendering into a volume rendering algorithm such that polygons are allowed to be transparent, it is apparent that samples collected along a ray include not just samples from the volume, but also samples from the polygons. The samples can then be composited onto pixels on the image plane in a depth-sorted manner.

Using a ray tracer, one could use independent rays to collect samples from the volume, and samples from the polygons [15], or one could use a common ray [19]. Since the shear-warp volume rendering algorithm traverses through the volume in storage order, projecting volume samples onto the image plane, an object-order polygon rendering scan-conversion is a natural candidate for collecting samples from polygons. Instead of rendering into a color buffer and a zbuffer, however, the polygon scan-conversion will render polygons into what we call a zlist-buffer.

A zlist-buffer is simply an array of pixels whereby each pixel stores a list of z values. Each list is called a zlist. Hence, in a zlist-buffer, each pixel may store more than one z values. Since, in a hybrid rendering, for the general case, it is not possible to tell the number of polygon samples lying along a viewing ray, each zlist in a zlist-buffer is appropriately a linked list.

The number of zlists in the zlist-buffer is then equal to the size of the intermediate image. The size of each zlist is at most the number of geometric points falling upon the respective pixel. The structure of the zlist-buffer may be diagrammatically viewed as in Fig. 3.

It should be pointed out that the notion of a zlist-buffer is not a new idea. Similar structure has been reported [7, 2]. Zlist has been used for image-based rendering [7] and transparency [2]. Here, in this paper, we are using it in the context of hybrid volume rendering. In all reports we have come across, zlist has always been depth-sorted, and stored together in a two-dimensional array to form a sparse volume.

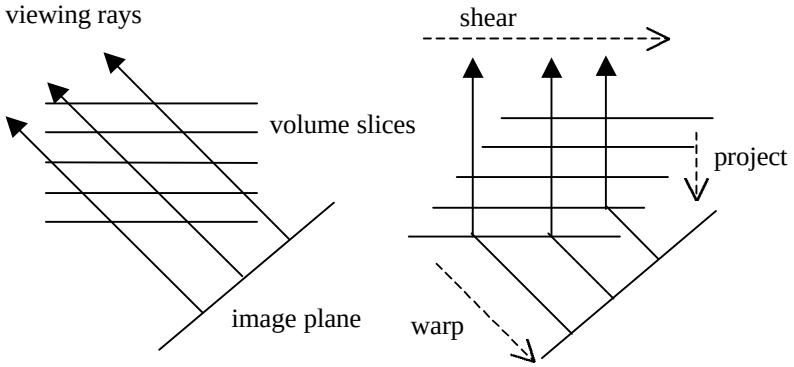


Fig. 2: Shear-Warp Volume Rendering

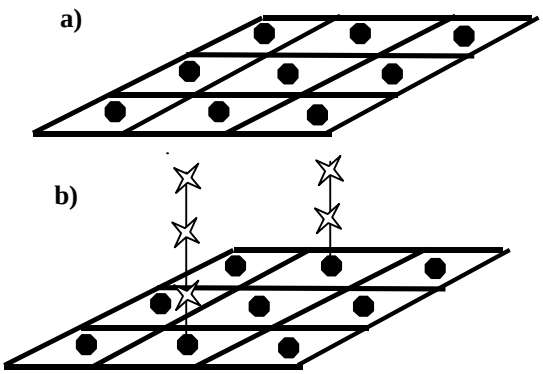


Fig. 3 a) A z-buffer and b) a zlist-buffer. In a) each blotted point indicates a z value, while in b) each blotted point indicates the start of a zlist. Each star in b) indicates a zlist node.

We render geometric objects into the zlist buffer before the composition of the sheared volume slices. Each polygon lying within the volume must be transposed and sheared before the rendering. The same transposition and shear that applies to the volume slices due to the shear-warp factorization applies to the polygons. In other words, the polygons must be transformed to the volume sheared object co-ordinate.

In the implementation, each node in each zlist stores the color and opacity of the point of the polygon or geometry projected onto the corresponding pixel, in addition to its depth value. Aside from the list of nodes itself, we also associate a variable for each zlist to store the z value of the last node in the list. As will be detailed out in the next section, the variable, called z-flag, ensures that a zlist stops growing once the total accumulative opacity in it reaches a maximum opacity threshold [14]. The structure of a zlist node is summarized in the following conceptual representation:

```
ZlistNode =
        Color:  32 bit integer
        Opacity: 8 bit integer
        Z: double
        Next: pointer to ZListNode

Zlist =
        z_flag:  double
        start: pointer to ZlistNode

ZlistBuffer =
        Zlist[0..numpixels – 1]: array of zlist
```

4.0 SCAN-CONVERTING THE POLYGONS

In the standard scanline polygon scan-conversion algorithm [5], two primary data structures are used to render the polygon: an edge table (ET) and an active edge table (AET). The ET is a list of buckets, one bucket per scanline. Each bucket contains a list of polygon edges for which the minimum y values is equal to the scanline number of the corresponding bucket. The AET is a list of edges currently being scan-converted. For each scanline, the algorithm rasterizes the pixels between pairs of edges from the AET, and then update the values – the current x, current normal or color, and the current z or depth value – in each edge in the AET.

The zlist data structure requires a simple modification of the rasterization phase. Instead of writing into a color-buffer and a z-buffer, it is written into a zlist-buffer. This step is done before volume composition. Hence, one other way of viewing the zlist-buffer is a multilayer hybrid buffer.

The composition step of the shear-warp volume rendering algorithm takes place in sheared object coordinate [12]. If the k-component of the viewing vector in sheared object coordinate is negative, a reverse-viewing order is implied, requiring a back-to-front composition of slices [12]. In this case, values are inserted into the zlist-buffer, such that a decreasing order of z-values in respective zlist is maintained. If, on the other hand, the k-component is positive and hence requires a front-to-back composition of slices, values are inserted into the zlist-buffer, such that an increasing order of z-values in respective zlist is maintained.

The z-flag of a zlist decides if a new node can be added to the zlist. Let us assume a normal front-to-back viewing. The z-flag of each zlist in the zlist-buffer is first initialized to the maximum possible floating value. During the rasterization, a new node is added to the zlist only if the z value of the prospective node is less than the z-flag. Each time a new zlist node is appended or inserted into the zlist, it traverses through the zlist from the head node onward, accumulating opacity, until the accumulated opacity reaches the maximum opacity threshold or until the end of the list is reached. If the accumulated opacity has reached maximum opacity threshold when exiting from the traversal, nodes beyond the last node visited will be destroyed. We also set the z-flag to the z value of the last node visited. If the accumulated opacity do not reached the threshold by the end of the zlist, we simply reset the z-flag of the zlist to the maximum floating value again.

A similar procedure occurs in the case of reverse-order viewing. The differences are that in this case, we initialize the z-flag of each zlist to the minimum possible negative floating value, and we insert or append a new node into a zlist only if the z-value of the node is greater than the z-flag.

In our implementation, we render the polygons before we render the volume. One other way is to interleave the rendering of polygons and volume slices. This has been suggested by Lacroute [12]. The main advantage of this approach is that rendering of polygons is done per slabs, resulting in smaller memory requirement for the zlist-buffer. The disadvantage of the approach is that polygons will have to be clipped in 3D against each slab, hence requiring more processing effort. In our approach, we save a polygon point into a zlist only if the accumulated opacity in the zlist is less than the user- or programmer-defined threshold. Hence, generally, excessive memory usage due to polygons can be checked.

Yet another way is to make the volume rendering itself return zlists. The advantage of using zlists from polygons is that the zlist structure requires no pre-sorting of the polygon. The zlist itself sort each fragment falling upon a pixel. No sorting need to be done for volume slices, on the other hand, as they are already in order.

5.0 COMPOSITING VOLUME AND POLYGON SAMPLES

The shear factor from the shear-warp factorization is implemented by translating and resampling each volume slice into the volume sheared object coordinate. Resampling is needed as individual voxels in the translated slice do not, in general, fit perfectly to individual pixels on the intermediate image plane. Resampling can be implemented using any appropriate interpolation filter. From our experience, bilinear interpolation filter is optimal, as far as speed and image quality is concerned (it is also the interpolation scheme implemented in Lacroute's VolPack engine [21]).

In a "pure" shear-warp volume rendering algorithm, each volume sample is immediately composited into the corresponding pixel in the intermediate image using the "over" operation. In a hybrid version of the algorithm using the zlist-buffer, however, we must check if there is any polygon fragment lying within the volume sample. Suppose the slice index of the current volume slice being processed is k. Let us assume that the viewing vector in sheared object coordinate indicates a normal front-to-back viewing. Then, the presence of any node in the corresponding zlist with z-value, z, such that the integer component of z is equal to k implies that the volume sample does contain polygon fragment (or fragments if there is more than one such node).

If there is no polygon fragment within the volume sample, we proceed as usual. Otherwise, we have to "break up" the voxel samples into smaller fragments depending on the position of the polygon fragment or fragments. The volume and polygon fragments are then composited in sorted order using the "over" operation.

5.1 Opacity Correction

In an object-order volume rendering algorithm, the distance between volume samples is constant in object space, but varies in image space. As the sample spacing changes, so must be the opacities [15].

In our hybrid rendering algorithm, not only must opacities be corrected due to changes in sample spacing; the opacity of each subblocks in a volume sample interspersed by polygon fragment(s) must also be corrected or computed from the corrected opacity of the original sample. The reason is that that spacing used by each subblock is only a fraction of the original spacing used by the original sample.

As is for opacity correction due to changes in sample spacing, the formula that we have used for opacity correction of subfragments of a volume sample is as follows [12]:

$$\alpha_{new} = 1 - (1 - \alpha_{old})^{tn/to}$$

where

α_{new} indicates new or corrected opacity,
α_{old} indicates previous or original opacity,
t_n indicates new thickness of volume sample,
and t_o indicates previous or original thickness of the volume sample.

At this point, a note should be made regarding the way in which we have treated the opacity for each subblock of a volume sample. It is only an approximation (or perhaps, more appropriately termed, a hack).

Let us assume that bilinear interpolation is being used to resample each volume slice into the sheared object space. For a sample point p in the current slice, let O_1, O_2, O_3, O_4 be the opacities of 4 surrounding voxels in the same slice. Let C_1, C_2, C_3, C_4 be the respective colors, and W_1, W_2, W_3, W_4 be the respectives weights of the contribution of the voxels. Using bilinear interpolation, the color and opacity of the point p would then be

$$\text{Color} = W_1*C_1*O_1 + W_2*C_2*O_2 + W_3*C_3*O_3 + W_4*C_4*O_4, \text{ and}$$
$$\text{Opacity} = W_1*O_1 + W_2*O_2 + W_3*O_3 + W_4*O_4$$

If a polygon point exists within the resampled voxel, p, then one needs to find the color and opacity of the fraction of the resampled voxel lying before the polygon point. The primary difficulty in designating a formulae for correcting the color and opacity of a sub-fragment of the voxel lies in that geometric polygons is continuous while voxels are discrete. Ideally, for a particular sample point, p, one finds the approximate location of the polygon point in the surrounding 4 voxels. Let $\text{Corr}(O_i)$ be the corrected value

for opacity O_i. Then, the corrected color and opacity of the voxel fragment lying before the polygon point is

$$\text{Color} = W_1*C_1*\text{Corr}(O_1) + W_2*C_2*\text{Corr}(O_2) + W_3*C_3*\text{Corr}(O_3) + W_4*C_4*\text{Corr}(O_4) \text{ , and}$$
$$\text{Opacity} = W_1*\text{Corr}(O_1) + W_2*\text{Corr}(O_2) + W_3*\text{Corr}(O_3) + W_4*\text{Corr}(O_4)$$

A kludge that work more efficiently and produces acceptable images, however, as testified by the illustrative images in section 6, corrects the opacity of the voxel fragment according to the opacity-correction formulae given above, and rescale the color by mutiplying with the new opacity.

6.0 IMPLEMENTATION AND RESULTS

The zlist-buffering engine is implemented entirely in software and in the C language, reusing codes where applicable from the source code for the Volpack shear-warp volume rendering library [21]. The engine is integrated into a volume visualization system that is hoped will eventually find widespread use as a medical educational tool in the local schools. Considering the expected budgets and available facility of target users, test runs were all conducted on a Linux Pentium 133Mhz machine with 64 megabytes of main memory.

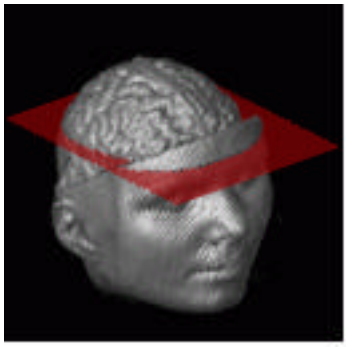
For illustration, since the prime concern has been hybrid volume rendering, and not with any novel volume rendering technique itself, one volume data is used, and it is rendered with an assortment of polygonal objects. The 128 * 128 * 84 volume data, which originated from the Chapel Hill Volume Rendering Test Dataset distributed by the University of North Carolina, is that of a human head, with the brain partially shown. Fig. 4 shows the rendering of a simple test environment consisting of two mutually perpendicular translucent polygons, and a polygonal sphere embedded within the volume. A total of 258 polygons is rendered together with the volume. The hybrid rendering takes approximately 0.6 second.

Fig. 4: A CT head interspersed with 2 perpendicular translucent polygons and a sphere

Polygonal planes can be used for interactive specification of cutting planes. As an example, a two-sided plane can be used. Each side of the plane corresponds to either an exterior or interior side, and can be colored differently. Voxels lying to the exterior side of the plane can be clipped out, while the rest of the volume maintained. Fig. 5 illustrates how such a visible cutting plane could be put into action.

Finally, Fig. 6 illustrates hybrid rendering with relatively more complex geometries. The head is resized to 256 * 256 * 168 using Gaussian resampling filter, rotate it 60 degrees about the Y-axis, and -15 degrees about the Z-axis, and rendered with a hat and a pair of sunglasses. A total of 496 polygons is rendered. Rendering takes approximately 2 seconds.

a)



b)

Fig. 5: a) A cutting plane placed across a CT head; b) CT head cropped as specified by cutting plane

Fig. 6: "Pretty" hybrid rendering

7.0 CONCLUSION

This paper presented a hybrid-rendering technique based on the shear-warp factorization. It outlined the approach for parallel projection, and dealt with the problem of arbitrary mixture of translucent and opaque polygons. The approach, termed zlist-buffering, is introduced in the context of volume rendering, and requires only a conceptually simple extension to the well-known z-buffering. Polygons are rendered prior to volume composition and warping, facilitating a modular implementation of the hybrid-rendering algorithm. Memory consumption is minimized by maintaining that collective opacity in each zlist does not exceed a maximal value. The algorithm has also been shown to work well with runlength-encoding of the volume and the intermediate image.

REFERENCES

[1] B. Cabral, N. Cam, J. Foran, *Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware*, ACM/IEEE Volume Visualization Symposium Proceedings, ACM Press, October 1994, pp. 91-97.

[2] L. Carpenter, *The A-buffer; an Antialiased Hidden Surface Method*, Computer Graphics (SIGGRAPH '84 Proceedings), Vol. 18, July 1984, pp 103-108.

[3] E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD Thesis, Report UTEC-CS-74-133, December 1974.

[4] D. Ebert, R. Yagel, J. Scott and Y. Kurzion, Volume Rendering Methods for Computational Fluid Dynamics Visualization, *Proceedings Visualization '94*, October 1994, pp. 232-240.

[5] J. D. Foley, A. Van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990.

[6] D. S. Goodsell, S. Mian, and A. J. Olson, Rendering of Volumetric Data in Molecular Systems, *Journal of Molecular Graphics*, 7(1), March 1989, pp. 41-47.

[7] S. J. Gortler, He, Li-Wei, M. F. Cohen, *Rendering Layered Depth Images*, Technical Report MSR-TR-97-09, Microsoft Research, March 1997.

[8] A. Kaufman, Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes, *Computer Graphics*, Vol. 21, No. 3, July 1987, pp. 171-179.

[9] A. Kaufman, *An Algorithms for 3D Scan-Conversion of Polygons*, Proceedings Eurographics, G. Marechal, Ed., North Holland, Amsterdam, August 1987, pp. 197-208.

[10] A. Kaufman, R. Yagel, and D. Cohen, Intermixing Surface and Volume Rendering, 3D Imaging, in, *Medicine: Algorithms, Systems, Applications*, K. H. Hoehne, H. Fuchs, S. M. Pizer, Eds., Springer-Verlag, Berlin, 1990, pp. 217-227.

[11] P. Lacroute, M. Levoy, *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*, Proc. SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994). Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, 1994, pp. 451-458.

[12] P. Lacroute, *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*, Ph.D. Thesis, Stanford University, 1994.

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

[14] M. Levoy, Display of Surfaces from Volume Data, *IEEE Computer Graphics and Applications* 8 (3), May 1988, pp. 29-37.

[15] M. Levoy, A Hybrid Ray Tracer for Rendering Polygons and Volume Data, *IEEE Computer Graphics and Applications* 10(3), March 1990, pp. 33-40.

[16] W. Lorensen, and H. Cline, Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Proceedings of *SIGGRAPH '87*, in *Computer Graphics* 21(4), July 1987, pp. 163-169.

[17] W. Lorensen, H. Cline, S. Ludke, C. R. Crawford and B. C. Teeter, Two Algorithms for 3-dimensional Reconstruction of Tomograms, *Medical Physics*, 15(3), May/June 1988, pp. 320-327.

[18] O'Rourke, J., *Computational Geometry in C*, Cambridge Press, 1998.

[19] L. Sobierajski, and A. Kaufman, Volumetric Ray Tracing, *ACM/IEEE Volume Visualization Symposium Proceedings*, ACM Press, October 1994, pp. 11-18.

[20] J. Wilhelms, and A. V. Gelder, A Coherent Projection Approach for Direct Volume Rendering, *Computer Graphics* 25, pp. 275-284, July 1991.

[21] The VolPack Volume Rendering Library, *http://www-graphics.stanford.edu/software/volpack/*.

[22] Programming with OpenGL: Advanced Rendering, *http://www.sgi.com/Technology/OpenGL/advanced97/notes/*.

BIOGRAPHY

M Nordin Zakaria is a lecturer at the Department of Multimedia, Universiti Putra Malaysia, where he teaches and supervises Computer Graphics courses and projects. Current research interests include 3D Computer Animation and Computer Games.

N. Selvanathan is a lecturer at the Faculty of Computer Science and Information Technology, Universiti Malaya. Current research interest is medical image processing.