Go Digital, Go Fuzzy

Jayaram K. Udupa[1] and George J. Grevera[2]

[1] Medical Image Processing Group,
Department of Radiology, University of Pennsylvania,
423 Guardian Drive 4th Floor Blockley Hall,
Philadelphia, PA 19104-6021
jay@mipg.upenn.edu
[2] Mathematics and Computer Science Department, BL 215,
St. Joseph's University,
5600 City Avenue,
Philadelphia, PA 19131
ggrevera@sju.edu

Abstract. In many application areas of imaging sciences, object information captured in multi-dimensional images needs to be extracted, visualized, manipulated, and analyzed. These four groups of operations have been (and are being) intensively investigated, developed, and applied in a variety of applications. In this paper, we put forth two main arguments: (1) Computers are digital, and most image acquisition and communication efforts at present are toward digital approaches. In the same vein, there are considerable advantages to taking an inherently digital approach to the above four groups of operations rather than using concepts based on continuous approximations. (2) Considering the fact that images are inherently fuzzy, to handle uncertainties and heterogeneity of object properties realistically, approaches based on fuzzy sets should be taken to the above four groups of operations. We give two examples in support of these arguments.

1 Introduction

In imaging sciences, particularly medical, there are many sources of multidimensional images [1]. For ease of further reference, we will refer to a multidimensional image simply as a *scene* and represent it by a pair $C = (\mathrm{C}, f)$, where C, the *scene domain*, is a multidimensional rectangular array of spatial elements (*spels* for short), and f, the *scene intensity,* is a function that assigns to every spel a vector whose component elements are from a set of integers. There is usually an *object* of study for which the scene is generated. This may be a physical object such as an organ/tissue component, a tumor/fracture/lesion/an abnormality, a prosthetic device, or a phantom. It may also be a conceptual object such as an isodose surface in a radiation treatment plan, an activity region highlighted by PET or functional MRI, or a mathematical phantom. The objects may be rigid, deformable, static, or dynamic. For example, the bones at a moving joint represent a dynamic, rigid object assembly. The heart muscle and the blood pool in the various chambers constitute a dynamic, deformable object assembly. We note that, most frequently, we are given one scene with scalar-valued intensities relating to an object of study. Often multiple scenes (each

S.K. Pal et al. (Eds.): PReMI 2005, LNCS 3776, pp. 137 – 146, 2005.
© Springer-Verlag Berlin Heidelberg 2005

with scalar- or vector-valued intensities) from multiple modalities such as CT, MRI, PET, CTA, MRA, and fMRI are also given. We will refer to any operation, which, for a given set of multimodality scenes of an object of study, produces information about the object of study, as *3D imaging*. The information may be qualitative - for example, a 3D rendition of a static object, an animation sequence of a dynamic object or an animation sequence of a static object corresponding to different viewpoints. It may be quantitative - for example, the volume of a static object, and the rate of change of volume of a dynamic, deformable object, the extent of motion of a rigid dynamic object. The purpose of 3D imaging is, therefore, given multiple scenes as input, to output qualitative/quantitative information about the object of study.

In this exposition, we will argue with strong evidence that, since the scenes are inherently digital, we should take entirely digital approaches to realize all 3D imaging operations. We also suggest that, since object information in scenes is always fuzzy, we should take approaches that are based on fuzzy set concepts. The need to fulfill these strategies opens numerous problems that are inherently digital in both hard and fuzzy settings. The reader is referred to [2] for a detailed account of 3D imaging operations and their medical applications.

3D imaging operations may be classified into four groups: *preprocessing, visualization, manipulation*, and *analysis*. All preprocessing operations aim at improving or extracting object information from the given scenes. The purpose of visualization operations is to assist humans in perceiving and comprehending the object structure/characteristics/function in two, three and higher dimensions. The manipulation operations allow humans to interactively alter the object structures (mimic surgery, for example). The analysis operations enable us to quantify object structural/morphological/functional information.

2 Go Digital, Go Fuzzy

Any scene of any body region exhibits the two following important characteristics of the objects contained in the body region.

Graded Composition: The spels in the same object region exhibit heterogeneity of scene intensity due to the heterogeneity of the object material, and noise, blurring, and background variation introduced by the imaging device. Even if a perfectly homogeneous object were to be imaged, its scene will exhibit graded composition.

Hanging-togetherness (Gestalt): In spite of the graded composition, knowledgeable humans do not have any difficulty in perceiving object regions as a whole (Gestalt). This is a fuzzy phenomenon and should be captured through a proper theoretical and computational framework.

There are no binary objects or acquired binary scenes. Measured data always have uncertainties. Additionally, scenes are inherently digital. As seen from the previous section, no matter what 3D imaging operation we consider, we cannot ignore these two fundamental facts – fuzziness in data and their digital nature.

We have to deal with essentially two types of data - scenes and structures. We argue that, instead of imposing some sort of a continuous model on the scene or the structure in a hard fashion, taking an inherently digital approach, preferably in a fuzzy setting, for all 3D imaging operations can lead to effective, efficient and practically

viable methods. Taking such an approach would require, in almost all cases, the development of the necessary mathematical theories and algorithms from scratch. We need appropriate theories and algorithms for topology, geometry and morphology, all in a fuzzy, digital setting, for dealing with the concept of a "structure" in scenes. Note that almost all the challenges we raised in the previous section relate to some form of object definition in scenes. Therefore, they all need the above mathematical and algorithmic developments. Additionally, since we deal with deformable objects (all soft-tissue organs, and often even bone, as in distraction osteogenesis – the process of enlarging or compressing bones through load applied over a long period of time), we also need fuzzy digital mechanics theories and algorithms to handle structure data realistically for the operations relating to registration, segmentation, manipulation and analysis. Because of the digital setting, almost all challenges raised previously lead to discrete problems. Since we are dealing with n-dimensional scenes, the mathematics and algorithms need to be developed for hard and fuzzy sets of spels defined in n-dimensions.

In the rest of this section, we shall give two examples of digital/fuzzy approaches that motivated us to the above argument. The first relates to modeling an object as a 3D surface and visualizing it. The second relates to the fuzzy topological concept of connectedness and its use in segmentation.

2.1 Polygonal Versus Digital Surfaces

In hard, boundary-based 3D scene segmentation (e.g., thresholding approaches), the output structure is often a 3D surface. The surface is represented usually either as a set of polygons (most commonly triangles [3]), or in a digital form [4] as a set of cubes or as a set of oriented faces of cubes. We shall describe in some detail how both the representation and rendering of such surfaces is vastly simpler and more efficient using digital approaches than using polygonal or other continuous approximations for them.

Let us consider the representation issues first. We shall subsequently consider the rendering issues. It is very reasonable to expect these surfaces to satisfy the following three properties since surfaces of real objects possess these properties. (1) The surface is connected. (2) The surface is oriented. This means that it has a well defined inside and outside. (3) The surface is closed; that is, it constitutes a Jordan boundary. The latter implies that the surface partitions the 3D space into an “interior” set and an “exterior” set such that any path leading from a point in the interior to a point in the exterior meets the surface.

The definitions of connectedness, orientedness, and Jordan property are much simpler, more natural and elegant in the digital setting using faces of cubes (or cubes) than using any continuous approximations such as representations via triangles. These global concepts can be arrived at using simple local concepts for digital surfaces. For example, orientedness can be defined by thinking of the faces to be oriented. That is, a face with a normal vector pointing from inside of the surface to its outside in the $-x$ direction is distinguished from a face at the same location with a face normal in exactly the opposite direction. Usually the surface normals at various points p on these surfaces (in both the polygonal and the digital case) are estimated independent of the geometry of the surface elements, for example, by the gradient at p of the

intensity function *f* of the original scene from which the surface was segmented. The gradient may also be estimated from other derived scenes such as a Gaussian smoothed version of the segmented binary scene. Since the surface normals dictate the shading in the rendering process, the surface elements are used here only as a geometric guide rather than as detailed shape descriptors of the surface. Therefore the digital nature of the geometry introducing "staircase" effects in renditions can be more or less completely eliminated. Since the surface elements in the digital case are simple *and* all of identical size and shape, they can be stored using clever data structures that are typically an order of magnitude more compact (Figure 1) than their polygonal counterparts [5]. Finally, there is a well-developed body of literature describing the theory of digital surfaces that naturally generalizes to *n*-dimensions for any *n*. Such a generalization is very difficult for polygonal representations.

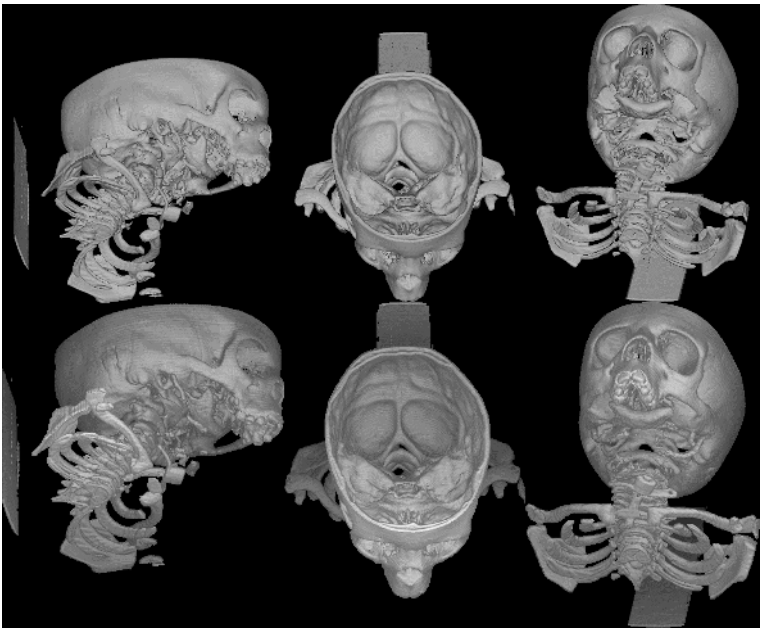

Fig. 1. Comparison of shell rendering and vs. triangulated surfaces. Top row: Triangulated surface via Marching Cubes/OpenGL requiring **3.4 sec on an SGI Reality Engine II**. Bottom row: Shell Rendering requiring **0.85 sec on a 300MHz Pentium**.

Let us now come to the rendering issues. Rendering of digital surfaces is considerably simpler and more efficient than that of polygonal surfaces, the main reason being the simplicity of the surface elements and of their spatial arrangement in the former case. There are mainly two computational steps in any rendering algorithm: hidden surface removal and shading. Both these steps can be considerably simplified exploiting the special geometry of the surface elements, reducing most expansive computations to table lookup operations. For example, the faces in a digital surface can be classified into six groups based on their face normals (corresponding to

the directions (+*x*, +*y*, +*z*, -x, -*y*, -*z*)). For any viewpoint, all faces from at least three of these groups are not visible and hence can be discarded without doing any computation per face. There are other properties that allow the rapid projection of discrete surfaces in a back-to-front or front-to-back order by simply accessing the surface elements in some combination of column, row, and slice order from a rectangular array. Triangular and other continuous approximations to surfaces simply do not possess such computationally attractive properties. It was shown in [5] based on 10 objects of various sizes and shapes that, digital surface rendering entirely in software on a 300 MHz Pentium PC can be done about 4-17 times faster than the rendering of the same objects, whose surfaces are represented by triangles, on a Silicon Graphics Reality Engine II hardware rendering engine for about the same quality of renditions. Note that the Reality Engine II workstation is vastly more expensive than the Pentium PC. Figure 1 contains an example of digital and polygonal surface rendering.

The design of rendering engines such as Reality Engine II was motivated by the need to visualize, manipulate, and analyze structure systems representing human-made objects such as automobiles and aircrafts. Early efforts in modeling in computer graphics took, for whatever reason, a continuous approach. Digital approaches to modeling have originated mostly in medical imaging [6] and have recently been applied also to the modeling of human-made objects [7] with equal effectiveness. Digital approaches are more appropriate for modeling natural objects (such as internal human organs) than continuous approaches. Natural objects tend to be more complex in shape and morphology than human-made objects. The applications of digital approaches to human-made objects [22] have demonstrated that the latter are equally appropriate even for human-made objects.

2.2 Traditional Volume Rendering Versus Digital Volume Rendering

Surface rendering as described in the previous section is distinguished from volume rendering in that geometric surface primitives such as triangles, rectangular faces of voxels, or geometric representations of the voxels themselves are employed as an intermediate representation in surface rendering [8]. Surface rendering typically employs a binary representation of a surface that is produced via some segmentation scheme. On the other hand, volume-rendering methods do not employ geometric primitives [9] and therefore do not need a binary or explicit hard representation of objects within the three-dimensional space. Although an explicit binary representation of the surface of objects is not required in volume rendering, classification techniques are employed [9] to assign object membership values to points in the three-dimensional space. This process is naturally viewed as the process of determining a "transfer function" that assigns to each point an opacity value based on the scene intensity at that point in the given three-dimensional scene. Given the fact that volume rendering employs classification techniques, one might argue that this is analogous to a segmentation scheme employed in surface rendering although done in a fuzzy manner (see [2], Chapter 1 for a detailed discussion of such matters). The main difference between surface and volume rendering lies in the number and types of elements that are projected onto the screen. Surface rendering methods project only those elements that comprise the surface. Volume rendering techniques, on the other

hand, project typically all of the voxels that are elements of the three-dimensional volume onto the screen. The transfer function is "consulted" as each voxel is projected to determine its contribution or influence on what appears on the screen. The transfer function (or lookup table) may indicate little or no contribution, a large contribution, or any amount in between. For a detailed comparison of a particular surface rendering method and a particular volume rendering method, one may refer to Udupa [10].

The general method of volume rendering was first proposed in Levoy [8], and Drebin [11]. The classification process associates a quadruple $\langle R(x), G(x), B(x), \alpha(x) \rangle$ with each scene intensity x occurring in C. R, G, B are the red, green, and blue components, respectively, and $\alpha \in [0,1]$ is referred to as opacity with $\alpha = 1$ corresponding to 100% opacity (opaque) and $\alpha = 0$ representing complete transparency. Rays are then cast through the classified volume orthogonal to the projection plane from every pixel in the projection plane. Values for discrete sample points along each ray are computed typically by trilinear interpolation, as the ray does not usually pass through voxel centers or grid points exactly. Compositing along each ray typically occurs in back-to-front order with respect to the screen or viewing direction. A variety of operators such as *over*, *attenuate*, *MIP* or *sum* (similar to radiographic projection) have been proposed. A specific hardware implementation of the above algorithm called "texture-mapped volume rendering" was first outlined in Cabral [12] using the SGI Reality Engine architecture. Briefly, the input scene, C, is loaded into texture memory (an area of dedicated, special purpose memory in the graphics hardware). Textures may be either one-, two-, or three-dimensional with two and especially three being germane to volume rendering.

We contrast traditional volume rendering with the same object-based technique, namely Shell Rendering [13, 14] that was compared with polygonal surface rendering in the previous section. Shell rendering is a flexible method that encompasses both surface rendering and volume rendering. A *shell* is a special data structure that contains voxels near the border of the object together with other items of information associated with each such voxel. At one extreme, the shell may be quite crisp (hard) in that it contains just the voxels on the object's surface. A crisp shell forms the basis for digital surface rendering. At the other extreme, the shell may be so thick as to contain all voxels in the foreground region of the entire scene. At this extreme, Shell Rendering becomes, in effect, a method of volume rendering. In between these two extremes, the shell becomes a fuzzy boundary between the object and the background.

In Shell Rendering, the information that is associated with each voxel includes the scene intensity gradient vector at the voxel, a code representing the scene intensity configuration of the voxels in the neighborhood, and the voxel coordinates. Shell rendering is done via voxel projection either in front-to-back or back-to-front order rather than by ray casting. Since all shell elements are identical in shape (a cuboid), many computations can be implemented as table lookup operations. These two factors contribute significantly to the speed of shell rendering over ray casting approaches.

Shell rendering is efficient because of the following factors (see Udupa [14] for details):

(1) The shell domain is typically a small subset of the entire scene domain.
(2) For every viewpoint, there exists an order of indexing of the columns, rows, and slices (one of 8 possible combinations) that projects the voxels in either front-to-back or back-to-front order.
(3) During front-to-back projection, an opacity-based stopping criterion implements early projection termination.
(4) Given a particular viewing direction, some elements of the shell domain may be discarded because they are completely occluded by opaque neighbors, as determined by the neighborhood configuration.
(5) Conversely, once a voxel with opaque neighbors behind it (with respect to the current viewing direction) is projected, projection along this ray can be terminated.

Shell rendering has been implemented in the freely available *3DVIEWNIX* software [15]. A tutorial regarding the specific steps in using the *3DVIEWNIX* software to perform shell rendering is also available [16].

Digital volume rendering via shell rendering in software was compared to traditional hardware assisted volume rendering. Input data consisted of four data sets: an MR head, a CT head, a CT knee, and MR angiography data. Shell rendering was executed on a 450 MHz Pentium PC. Hardware assisted volume rendering was executed on a 195 MHz SGI Octane, a dual 300 MHz processor SGI Octane, and a quad 194 MHz processor SGI Onyx 10000. Shell rendering was shown to perform at about the same speed as the 195 MHz Octane but slower than the other two SGI systems but did so by executing entirely in software on a much less expensive

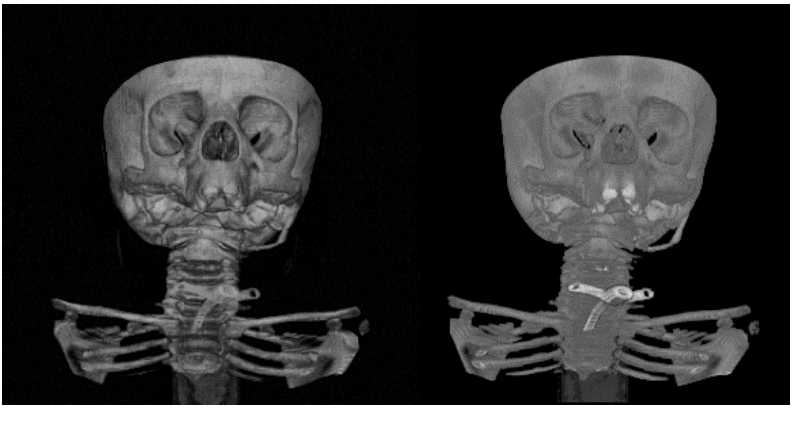

Fig. 2. Renditions of *192411* CT head data (left: shell rendering **requiring 0.9 to 2.9 sec, depending upon settings, on a 450MHz Pentium**; right: volume rendering requiring **0.8 to 1.7 sec on a dual processor 300MHz SGI Octane or a single processor 195MHz SGI Octane**, respectively

computing platform. Figure 2 contains sample shell and volume renderings. Note that traditional volume rendering in these experiments did not include a shading model. This contributed to its speed but did so at reduced image quality.

2.3 Fuzzy Connected Object Definition

Although much work has been done in digital geometry and topology based on hard sets, analogous work based on fuzzy sets is rare [17]. As pointed out earlier, the uncertainties about objects inherent in scenes must be retained as realistically as possible in all operations instead of making arbitrary hard decisions. Although many fuzzy strategies have been proposed particularly for image segmentation, none of them has considered the spatio-topological relationship among spels in a fuzzy setting to formulate the concept of hanging-togetherness. (We note that the notion of hard connectedness has been used extensively in the literature. But hard connectedness already assumes segmentation and removes the flexibility of utilizing the strength of connectedness in segmentation itself.) This is a vital piece of information that can greatly improve the immunity of object definition (segmentation) methods to noise, blurring and background variation.

Given the fuzzy nature of object information in scenes, the frameworks to handle fuzziness in scenes should address questions of the following form: How are objects to be mathematically defined in a fuzzy, digital setting taking into account the graded composition and hanging-togetherness of spels? How are fuzzy boundaries to be defined satisfying a Jordan boundary property? What are the appropriate algorithms to extract these entities from scenes in such a way as to satisfy the definitions? These questions are largely open. We will give one example below of the type of approaches that can be taken. This relates to fuzzy connected object definition [17]. This framework and the algorithms have now been applied extensively on 1000's of scenes in several routine clinical applications attesting to their strength, practical viability, and effectiveness.

We define a fuzzy *adjacency* relation α on spels independent of the scene intensities. The strength of this relation is in [0, 1] and is greater when the spels are spatially closer. The purpose of α is to capture the blurring property of the imaging device. We define another fuzzy relation κ, called *affinity*, on spels. The strength of this relation between any two spels c and d lies in [0, 1] and depends on α as well as on how similar are the scene intensities and other properties derived from scene intensities in the vicinity of c and d. Affinity is a local fuzzy relation. If c and d are far apart, their affinity is 0. Fuzzy *connectedness* is yet another fuzzy relation on spels, defined as follows. For any two spels c and d in the scene, consider all possible connecting paths between them. (A path is simply a sequence of spels such that the successive spels in the sequence are "nearby".) Every such path has a strength, which is the smallest affinity of successive pairwise spels along the path. The strength of fuzzy connectedness between c and d is the largest of the strength of all paths between c and d.

A fuzzy connected object of a certain strength θ is a pool O of spels together with an objectness value assigned to each spel in O. O is such that for any spels c and d in O, their strength of connectedness is at least θ, and for any spels e in O and g not in O, their strength is less than θ.

Although the computation of a fuzzy connected object in a given scene for a given κ and θ appears to be combinatorially explosive, the theory leads to solutions for this problem based on dynamic programming. In fact, fuzzy connected objects in 3D scenes (256×256×60) can be extracted at interactive speeds (a few seconds) on PCs such as a 400 MHz Pentium PC.

3 Concluding Remarks

In this article, we have first given an overview of the operations available for 3D imaging - a discipline wherein, given a set of multidimensional scenes, the aim is to extract, visualize, manipulate, and analyze object information captured in the scenes. We have also raised numerous challenges that are encountered in real applications. Computers are digital. Current attempts in image acquisition, storage, and communication are completely digital or are proceeding in that direction. We have presented an argument with evidences that there are considerable advantages in taking an inherently digital approach to realizing all 3D imaging operations. We have also argued that, since object information in scenes is fuzzy, the digital approaches should be developed in a fuzzy framework to handle the uncertainties realistically. This calls for the development of topology, geometry, morphology and mechanics, all in a fuzzy and digital setting, all of which are likely to have a significant impact on imaging sciences such as medical imaging and their applications.

Acknowledgments

The authors' research is supported by NIH grants NS37172 and EB004395. We are grateful to Mary A. Blue for typing the manuscript.

References

[1] Cho, Z.H., Jones J.P. and Singh, M.: *Foundations of Medical Imaging*, New York, New York: John Wiley & Sons, Inc. (1993).

[2] Udupa, J. and Herman, G. (eds.): *3D Imaging in Medicine*, 2nd Edition, Boca Raton, Florida: CRC Press (1999).

[3] Lorensen, W. and Cline, H.: "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics* 21(1987), 163-169.

[4] Udupa, J.K., Srihari, S.N. and Herman G.T.: "Boundary Detection in Multidimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-4 (1982), 41-50.

[5] Grevera, G.J. and Udupa, J.K.: "Order of Magnitude Faster Surface Rendering Via Software in a PC Than Using Dedicated Hardware," *SPIE Proceedings* 3658 (1999), 202-211.

[6] Herman, G. and Liu, H.: "Three-Dimensional Display of Human Organs from Computed Tomograms," *Computer Graphics and Image Processing* 9 (1979), 679-698.

[7] Kaufman, A.: “Efficient Algorithms for 3-D Scan Conversion of Parametric Curves, Surfaces, and Volumes,” *Computer Graphics* 21 (1987), 171-179.

[8] Levoy, M.: "Display of surfaces from volume data," *IEEE Computer Graphics and Applications* 8 (1988), no. 3, 29-37.

[9] Elvins, T.T.: "A Survey of Algorithms for Volume Visualization," *Computer Graphics* 26 (1992), no. 3, 194-201.

[10] Udupa, J.K., and Hung, H.-M.: "Surface Versus Volume Rendering: A Comparative Assessment," *IEEE Computer Soc. Proc. First Conference on Visualization in Biomedical Computing* (1990), 83-91.

[11] Drebin, R., Carpenter, L., and Hanrahan, P: "Volume rendering," *Computer Graphics* 22 (1988), 65-74.

[12] Cabral, B., Cam, N., and Foran, J.: "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Symposium on Volume Visualization* (1994), 91-98.

[13] Udupa, J.K. and Odhner, D.: "Fast Visualization, Manipulation, and Analysis of Binary Volumetric Objects," *IEEE Computer Graphics and Applications* 11 (1991), no. 6, 53-62.

[14] Udupa, J.K. and Odhner, D.: "Shell Rendering," *IEEE Computer Graphics and Applications* 13 (1993), no. 6, 58-67.

[15] Udupa, J.K., Goncalves, R.J., Iyer, K., Narendula, S., Odhner, D., Samarasekera, S., and Sharma, S.: "3DVIEWNIX: An open, transportable software system for the visualization and analysis of multidimensional, multimodality, multiparametric images," *SPIE Proc. 1897, Medical Imaging 1993:Image Capture, Formatting, and Display* (1993), 47-58.

[16] "Part B: Generate a Fuzzy Surface for Volume Rendering, Tutorial 2: How to create 3D objects," *http://www.mipg.upenn.edu/~Vnews/tutorial/tutorial2.html*, Medical Image Processing Group, Dept. of Radiology, University of Pennsylvania, Philadelphia, PA.

[17] Udupa, J.K. and Samarasekera, S.: "Fuzzy Connectedness and Object Definition: Theory, Algorithms, and Applications in Image Segmentation," *Graphical Models and Image Processing* 58 (1996), no. 3, 246-261.