On Spatial Rendering and User-centered Visualization in 3D

KYAN Matthew[1], SABER Yusuf [1]， JAIN Nazuk [1]， CANCILLA James [1], WANG Yongjin[2]
1. Department of Electrical and Computer Engineering, Ryerson University, Canada;
2. Department of Electrical and Computer Engineering, University of Toronto, Canada;
Corresponding author: mkyan@ee.ryerson.ca

Abstract

This paper presents a framework for the rapid isolation and representation of structures from spatially registered image stacks. The introduced framework enables dynamic visualization with next generation volumetric displays, in which image volumes are rendered in real 3D space. Two recent semi-supervised approaches to the problem of segmenting relevant structures are investigated and integrated to yield a generic interactive visualization pipeline. Information obtained from the segmentation process is used to guide layered object reconstruction via the SpatialGL SDK. The reconstructed original objects in 3D may then be selectively visualized on the Perspecta: a spatially-rendered display. Practical considerations, future improvements and applications are discussed.

1. Introduction

In the fields of medicine and biology, there is a growing trend in imaging technologies towards the direct extraction of 3D information regarding structures or specimens of interest. This is particularly evident in increasingly prevalent modalities such as Magnetic Resonance Imaging (MRI), Computed Tomography (CT), 3D Ultrasound, Confocal Laser Scanning Microscopy (CLSM), and variants thereof. It is natural then, that data acquired through such means be visualized and examined in a way that reflects the true spatial arrangement of such information. As such, there is an increasing desire to build more natural, volumetric representations that facilitate an observer's ability to analyze both surface and internal details of tissues, organs, cells and various other relevant structures being targeted by the imaging system [1].

Stereoscopic immersion has long been a technology of choice for conveying the sensation of depth and structure within dynamic 3D environments. Recently however, a new type of display technology has begun to emerge: spatially rendered (volumetric) displays [2-5]. Rather than relying on the illusion of 3D, where each eye is presented with two slightly offset viewpoints of the scene, volumetric displays render the scene in actual physical space. This, by its nature, permits the ability to view specimens from multiple perspectives, by multiple observers simultaneously, thereby facilitating the collaborative observation of data. The use of further tools to highlight or interact with data on such displays is also more intuitive, as conceivably, tools may be manipulated in the same space as the data (e.g. a haptic pen used as a virtual scalpel). In essence, such displays afford more direct, one-to-one correspondence between the specimen, display and observer.

Traditional approaches to the general problem of volumetric selection include transfer function selection, and manual or assisted segmentation tools. Transfer function selection proceeds under the assumption that distinct materials in the volume correspond to different value ranges in the volume data values. Specifically, transfer functions distribute color and opacity values across these data ranges, thereby selectively rendering or removing them from the visible volume. Often the above assumption is inadequate, and may result in a loss of information for objects that exhibit a complex range/pattern of data values within its voxels. Furthermore, the manipulation of transfer functions is akin to the manual selection of multiple threshold levels: involving much trial and error, and is generally considered counter-intuitive [6].

In general, prior to tasks such as reconstruction, visualization and even data interaction/manipulation, there is a need to be able to effectively segment regions or objects described within the data. In fact, segmentation is a necessary functional step in the process of rendering to a volumetric display, due to the fact that different objects are likely to overlap and occlude one another within the volume. Unlike mesh-based rendering of well-defined object surfaces in computer graphics applications, it is generally advantageous for volumes of interest to be viewed in unison, regardless of how they overlap or intersect.

Manual segmentation enables specific definition of materials/regions of interest, however they force the user to define the entire region at the voxel level. This process becomes extremely tedious and subjective.

978-1-4244-3930-03930-009/$25.00 ©2009 IEEE

Semi-supervised segmentation refers to that which is conducted with minimal user input to the algorithm. The basis of this approach is that it is relatively simple for a user to quickly feed small hints to the program regarding where the object of interest is likely to exist. The more impractical task of accurately delineating the entire region is then offloaded to the algorithm. A semi-supervised approach allows for a balance that leverages the users knowledge with computational resources, speed and efficiency.

Many semi-supervised approaches reported in the literature are based on the minimization of some cost or energy function relating to both boundary and/or enclosed regional properties [7-8]. Such approaches do not extend well to 3D. The level set methods [9] and normalized cuts [10] attempt to find an approximate solution that is often non-optimal. Other approaches have emphasized the importance of imposing hard constraints (or seed pixels that help to direct the segmentation process). Intelligent scissors [11] and live wire [12] require that the user indicate pixels through which the boundary must pass, attempting to infer a path that snaps to such seeds in a way that minimizes an energy function (relating to gradient information in the image). Intelligent Paint [13] uses a similar seeding strategy, where boundary paths are inferred via region growing, however is subject to leaking across weak or gradual gradients. Unlike 2D however, seeding voxels accurately along known boundaries in 3D becomes extremely difficult.

Inspired by the imposition of hard constraints that seed regions rather than boundaries, recent approaches seem to be more amenable to 3D segmentation. Approaches based on graph cuts have been proposed, however the more recent approach [14] based on application of min-cut/max-flow optimization demonstrates good performance for foreground/ background extraction. Random walker [15,16] is another recent technique is also computationally efficient, offers the guarantee of a unique solution, along with the ability to identify multiple regions simultaneously.

This paper presents a generic framework in which interactive segmentation algorithms may effectively function with volumetric displays. In this regard, both graph cut and random walker techniques are applied to the interactive visualization of image data on Actuality Medical's Perspecta display [2]. The primary objective is for the observer to be able to rapidly isolate part of an image (object of interest) so that it can be easily identified and viewed on the Perspecta. In doing so, a discussion of practical issues encountered in segmenting across volumes is presented.

The remainder of this paper is organized as follows. In section 2, we introduce the nature of different approaches to volumetric displays. Section 3 discusses the core background underpinning the semi-supervised segmentation methods explored in this work. Section 4 outlines the proposed framework, functional specifications, and practical considerations, followed in section 5 by preliminary results and discussion. Conclusion is provided in section 6.

2. Volumetric Displays

In earlier incarnations, the notion of a volumetric display was conceived as one that "*permits the generation, absorption, or scattering of visible radiation from a set of localized and specified regions within a physical volume*" [17]. More recently, the term volumetric display has generally implied any that achieves the 3D effect without the need for eyewear (auto-stereoscopic). Unlike traditional stereoscopic methods, or those utilizing parallax barriers, volumetric displays project with more consistent *vergence* and *accommodation* cues (the ability for our gaze to converge, and then focus on an object we are viewing) [3].

In this work, we restrict our attention to displays that are considered space filling. Within this class of displays, there are two major categories: *Swept-volume* displays, and *Static-volume* displays.

In swept-volume displays [2,18], a surface (plane/helix) translates or rotates through a volume, whilst projected optics reflect, or emit light from its location. By projecting a synchronized sequence of image sections as the volume is traversed by the surface, a continuous volume of light is perceived via persistence of vision (similar to our perception of moving images from a sequence of static frames in video).

In static-volume displays [19], light is selectively emitted from points in a volume of material in which there are no moving parts [3], for instance: a transparent crystal filling a volume that is doped with optically active ions of rare earths. Ions are excited by two intersecting IR-laser beams with different wavelengths to emit visible photons.

2.1. The Perspecta display

The Perspecta 1.9 display [2], developed by Actuality Medical (formerly Actuality Systems Inc.) was used in this work. Specifically, it belongs to the class of *swept-volume* volumetric displays. With a projection engine based on Texas Instruments Digital Light Processing (DLP) technology, a 3-chip configuration (one chip per color channel) is used to provide rapidly sequenced 2D projections onto the

rotating screen, to produce a 360º field of view image, at a resolution of 768×768×198 (slices). Each image is updated every 168μs.

2.2. The SpatialGL API

The SpatialGL API [5] is used to communicate with the Perspecta display, essentially acting as an extension of the OpenGL API. This set of functions allows for image data and texture information to be stored and eventually displayed. There are a number of features in SpatialGL that allow for some advanced image manipulation and reconstruction, however only a small subset of these features will be used in this work.

SpatialGL contains memory functions that allow large amounts of data to be stored in memory and reused with ease. These functions are important because storing an entire dataset of images can occupy an enormous amount of consecutive memory elements. As an example, imagine a dataset containing 100, 24-bit images, with each image having a size of 256 by 256. For a single image, the storage requirements would be 192KB. For 100 images, 19.2MB of consecutive memory elements is required. The color map feature used by SpatialGL allows for an object to be created and maps the individual pixel values to colors to be displayed. This function allows for a multidimensional array to be created that can hold individual color values (red, green, blue) as well as an intensity value. This information is then stored in memory using SpatialGL functions and is subsequently used internally by the Perspecta software to generate colors of varying intensities on the display.

3. User-centered Segmentation

Each of the two algorithms considered extends the notion of a connected graph representation of an image array. Voxels (3D version of pixels) are treated as nodes (V) in the graph, whilst edges (E) represent weighted connections that are indicative of some form of similarity measure between adjacent voxels. Minimal interaction is conducted by the user, and involves selection of a "seed" subset of voxels representative of known *foreground*, and a subset of "seed" voxels representative of known *background* (or in the case of the random walker approach, multiple sets of voxels representing different foreground regions). From here, each algorithm attempts to classify all remaining voxels based on the initial 'seed' voxels provided by the user. In the case of Graph-cut, the goal is to sever a cut along a set of edges within the overall graph to segregate the likely foreground regions from the background. In the Random Walker approach, a probabilistic representation attempts to infer which voxels are more likely to belong to which seeded labels.

3.1. Graph cut segmentation

In the Graph-cut approach, the analogy is that the edges between nodes represent capacities for a flow. Additional edges are added to connect two special terminal nodes to allow a flow that emanates from a *source S*, and drains through all edges in the graph to the *sink T*. Flow is systematically increased from the source, and along edges until it saturates the graph. Bottlenecks of flow given imposed constraints on capacities between nodes are indicative of natural discontinuities between regions, and thus define logical sites for a boundary.

3.1.1. Defining edge weights

Edge weights represent flow capacities and reflect the relative similarity between two connected nodes (voxels) in the graph. In effect, the more similar two nodes (p and q) are considered, the larger the capacity or tendency for flow to occur between the nodes.

Three types of edges are instantiated: edges connecting voxels directly to one another $\{p, q\}$, which follow traditional neighborhood connectivity within an image; edges $\{p, S\}$ connecting voxels directly to a terminal node S representing the source (foreground); and finally, edges $\{p, T\}$ connecting voxels directly to a second terminal node T representing the sink (background).

Given that A defines a segmentation solution of foreground A_f and background A_b labels, two types of constraints are used to define edge weightings:

1. Hard Constraints: initial seed voxels that are directly classified as foreground or background.
2. Soft Constraints: defined by a cost functional $E(A)=\lambda R(A)+B(A)$ that provides a weighted combination of how well a voxel fits into a known model for a region $R(A)$, along with a boundary term $B(A)$ based on the similarity between neighboring voxels.

Initially, prior to any seeding, each edge connecting a voxel p to its neighboring voxel q, is assigned a capacity solely according to the boundary term $B_{\{p,q\}}$. Thus large discontinuities between voxels will result in low capacities. Information supplied by the user via seeds are then embedded into the rest of the graph:

Edges connecting seeds to their opposing terminal nodes incur a zero capacity, thus prevent flow (e.g. zero flow from source S to a background seed). Edges connecting seeds to their corresponding terminal nodes

incur a maximal capacity K: a worst-case capacity based on the sum of all capacities in the edges linking voxels to one another (i.e. $\propto \sum B_{\{p,q\}}$).

The remaining unlabelled voxels p connected to the source S are assigned a regional penalty $R_p(A_b)$ based on how well they fit a model of the background (i.e. if they do not fit the background well, this penalty will be high and so too, will the capacity of the edge connected to the source). In the reverse case, edges linking unlabelled nodes to the sink T will reflect a penalty for assigning them to the foreground $R_p(A_f)$. This is summarized in Table 1 [14].

Table 1. Summary of costing for each type of edge in graph cut segmentation

Edge	Weight(cost)	For
{p,q}	$B_{\{p,q\}}$	*{p,q} Є non terminal edge*
{p,S}	$\lambda \cdot R_p(A_b)$	*p = unlabelled*
	K	*p ∈foreground*
	0	*p ∈background*
{p,T}	$\lambda \cdot R_p(A_f)$	*p = unlabelled*
	0	*p ∈foreground*
	K	*p ∈background*

3.1.2. Flow and cut procedures

Once the graph has been set up properly, the "cut procedure" is implemented to separate the object of interest from the background. The cut (represented by set C) is determined using the following procedure:

1. Go through every pixel p and break one terminal link (t-link) based on the cost (i.e. break the link with the lower eight between either the source and the pixel or the sink and the pixel). This results in the labeling of each pixel as A_f or A_b.
2. The boundary will be made up of pixels (i.e. $\{p,q\} \in C$) if and only if p and q are linked with t-links to different, separate terminals.
3. If the pixel is marked as a foreground seed by the user, (i.e. $p \in$ *foreground*), then the t-link of pixel p to A_b is broken (i.e. $\{p, T\} \in C$).
4. If the pixel is marked as a background seed by the user, (i.e. p ∈ *background*), then the t-link of pixel p to A_f is broken (i.e. $\{p, S\} \in C$).

Once this procedure has been implemented, the Min-Cut Max-Flow algorithm [14] is used to obtain the optimal cut. The main aim is to obtain a minimal cut ($C \in E$) such that the source and sink terminals get separated with the minimum cost (i.e. the elimination of the weakest links). This is equivalent to the maximum flow from the source terminal S to the sink terminal T such that all the bottlenecks in the system are eliminated.

3.2. Random walker segmentation

The statistical concept of a random walk (or drunkard's walk) is used to find the probability that each voxel in the image will randomly walk to each of the user-defined labels. This probability translates to a form of "membership" of each voxel to each label. The image is segmented then, by associating each pixel with the label to which it will most likely randomly walk to first (i.e. highest probability). Naturally, the ability to "walk" between two given nodes (voxels) is related to a form of impedance that is dictated by the image properties (discontinuities) between each node.

It has been shown that, to find these probabilities, one can imagine the image-graph as an electric circuit [16]. In this formulation, the probabilities in question become the potentials from each pixel to each label. Impeding characteristics may then be thought of as a resistance (or conductance, denoting similarity). One should note, however, that to format the image-graph as a probabilistic electric circuit, the label-nodes must be examined one-at-a-time, by grounding the label-nodes that are not being examined, and giving the label-node in question a potential of 1.

The potentials can then be found by simply applying Kirchhoff's Current Law (KCL) to every voxel in the image. This application results in a system of K linear equations (with unknowns relating to the set of unlabelled nodes). The system can then be solved using methods such as Gaussian elimination.

Random walker permits the use of multiple labels, and thus the delineation of multiple regions. This is achieved by repeating the above process for each label (i.e. each set of seeds corresponding to a new foreground object). In this way, a vector of potentials, each element corresponding to the probability of that voxel belonging to the corresponding label. Each voxel is ultimately assigned to the label corresponding to the maximum potential (probability) in the vector.

3.2.1. Defining edge weights – random walker

Defining edge weights in the random walker algorithm is somewhat less complicated than for graph cut. Firstly, we only need consider edges linking neighbors in the 3D image array according to some standard level of connectivity (as in Section 3.1.1).

Next, edge weights need only reflect the discontinuity (or the similarity) between voxels. Typically, the weight of an edge $w_{p,q}$ will be computed using the image properties from each of vertices p and q. Alternative image properties can be used to

represent the edge weight, and of course each different equation would represent some unique quality.

In keeping with the notion of similarity used in the graph cut algorithm, we use a standard Gaussian weighting on the difference in intensities between the two voxels, as defined in [15]:

$$w_{p,q} = \exp(- \beta (I_p - I_q)^2) \quad (1)$$

where $w_{p,q}$ is the edge weight, I_p and I_q are the image intensities at voxels p and q respectively, and β is a scaling factor. A similar scheme was used for weighting. Please see [15] for further details.

4. Proposed Visualization Framework

In this section, we outline the proposed operational framework for integrating both semi-supervised segmentation procedures into the visualization pipeline for the Perspecta display.

4.1. GUI

As this is an early prototype, only simple features were implemented, with an emphasis on enabling interactivity for the segmentation procedures. OpenCV was used to facilitate preprocessing tasks, graphical user interface (GUI) operations and future functionality. Some of the basic features offered by the GUI were:

- Browse and load support for common image file stacks
- Defaults segmentation tool to display central image slice from the volume
- Different colored markers to mark foreground and background seeds (user input)
- Option to choose between histogram equalization and/or contrast stretching before segmentation
- Option to start segmentation followed by display by pressing "r" on the image window, or skipping segmentation altogether by providing no seeds on the image (for direct display onto the Perspecta)
- Option to press "Escape" and start segmentation on another image data-set

A post-segmentation GUI was implemented to manipulate the 3D image displayed on the Perspecta, which included the following features:

- Changing the color of the foreground and background areas of the displayed image
- Application of a global opacity to foreground and background segmented regions
- Adjusting the orientation of the image

4.2. Framework

The proposed framework is summarized in Figure 1, where the user first loads in an arbitrary image stack via the GUI. A central slice is automatically presented to the user for seeding segmentation, however the user has the ability to browse slices if a more appropriate slice is desired (i.e. containing more information). The user then uses two markers to paint voxels from the chosen slice as either foreground or background seeds. These seeds then form a base seed set (considered a ground truth). Either of the two methods of segmentation can be forked at this point, however segmentation is restricted to sub blocks of 3 slices at a time, to reduce memory overhead. In this way, each segmentation will yield additional foreground/ background voxels, which are considered as estimates. These sets are sampled to generate new seeds from the periphery slices of the previous segmentation, and the block of three slices is shifted and a new segmentation performed. This may be done with arbitrary overlap. The process is repeated until the entire volume is segmented. The estimates from each slice are recombined to form the final segmentation labels. These are then used to re-map voxels onto a new color map where global opacity is controlled with a slider to fade in/out foreground/ background regions. Colors defining each may also be selected. Reconstruction and display of the image is updated to the Perspecta through SpatialGL

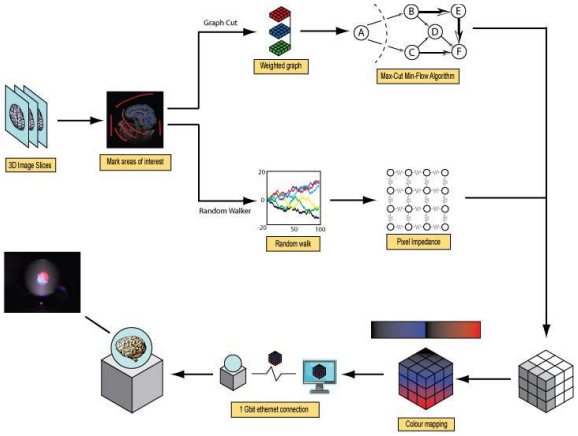


Figure 1. The proposed visualization framework

4.3. Spatial Rendering

Determining a coordinate system on the Perspecta is not a straight forward process. Since any images drawn on the Perspecta appear within a domed, circular area,

there are no actual x- and z-axes (width and depth). The SpatialGL API has defined a coordinate system, and thus it is possible to control where images are drawn. In order to draw images on the Perspecta, SpatialGL defines a 3-dimensional box where the voxels for an image will be drawn.

Once the box has been defined, an image can be drawn on the Perspecta. Voxels are drawn on the screen beginning at the origin, and then moving forward, then up and finally across. Defining what "forward" and "across" actually are can be confusing in an environment where the only definitive directions are up and down. However, because SpatialGL has defined a 3-dimensional box for the drawing area, width and depth can be arbitrarily assigned.

During the segmentation process, the segmented object of interest (foreground) pixel information is stored in a Boolean. For any given pixel, the value is set to "true" if the pixel is part of the segmentation and "false" if the pixel is part of the background. This Boolean array is then used as a mask to differentiate the segmented pixels from the other pixels to be displayed in different colors on the Perspecta.

In order for the image to be reconstructed on the Perspecta, all of the image data needs to be read into memory. This data is read in one pixel at a time so that it can be stored in the proper location of the one dimensional storage array. During this reading operation, the segmentation array is accessed to see if the current pixel is part of the segmented portion. In order to visually distinguish a background voxel from a segmented voxel, normalization is performed on the voxel value with respect to all voxels in the labeled region. For instance, in foreground/background cases, all of the voxels from the background are rescaled to have a value within the range from 0 to 127, and all of the voxels from the foreground are scaled to the range of 128 to 255. The two halves of the intensity range were then devoted to two separate color maps. This is a crude approach, ideally more aesthetic rendering techniques may be implemented to color voxels on an individual basis.

4.4. Practical Considerations

4.4.1. Memory Management - Graph Cut

To connect the image slices with each other, additional edges were added between the nodes of the individual slices. Each pixel in a 2D image slice is connected to three voxels in the 2D image slices above and below. Therefore, additional weighted edges/links (almost 2×n pixels number of links) were added for the interconnection of the three image slices leading to a 3D graph, to which terminal nodes and links were added based on the users-provided algorithm-estimated seeds.

A variable was introduced to keep track of the marked image slice (from the three images being segmented) and another variable to keep track of the image to be marked next (to facilitate segmentation of the next set of images).

The seed information for peripheral image blocks (indicated by the image to be marked next) were generated by randomly sub sampling 1/50 of the voxels estimated as foreground from the previously segmented image block sub sampling 1/100 voxels from the background estimates. The estimates were saved in text files to be read later as respective seeds for future segmentations.

4.4.2. Memory Management - Random Walker

As previously noted, KCL is applied to each node in the graph to achieve a system of K linear equations. If the image is of $M \times N$ dimensions, then $K=M \times N$, which means that the system will be of $K \times K$ dimensions. Considering the advancements in digital photography, image dimensions can normally be of enormous proportions such as 4096×4096, and standard HDTV dimensions span 1080×720 pixels. For an HDTV image, the system would contain (1080×720)×2 elements, which is 604,661,760,000 elements. Assuming that each element is 1 byte in size, this system requires roughly 70 GB of memory. Fortunately, the majority of the elements in this system are zeroes. This can be taken advantage of by using a sparse matrix, which only keeps account of the non-zero elements and their locations in the system. Each row in the system consists of a maximum of 5 elements. Therefore, the system reduces to, at most, 5×(1080×720), or 3,888,000 elements, which only requires roughly 474 KB. Although in general, random walker needs less memory to run, a similar block based strategy as that used for graph cut allows the segmentation to propagate throughout the volume.

5. Results and Discussion

A number of single channel image datasets were tested with the system: An MRI scan of the brain, a CT scan of the skull, and a confocal image dataset of plant cell chromosomes. In each case, segmentation was performed through the GUI interface in Section 4.1. A photo of the system is shown in Figure 2-a. Note that the seeding of slices at this point is conducted on a separate monitor. Ideally, given more appropriate interaction methods for volumetric

displays, this seeding can occur directly on the display itself.

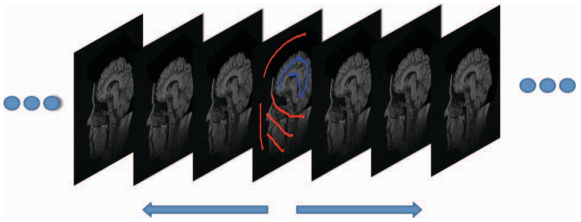
Figure 2-b shows the process of interactive seeding, where blue seeds represent foreground, and red seeds represent desired background. Segmentation for adjacent slices is shown in Figure 2-c. As can be seen, the complex brain structure was easily extracted with a small amount of user input. The segmented slices are represented with labeled masks for each object, and the background. Figure 3-a shows the final volumetric representation of the MRI dataset (with the brain displayed in red, and the remainder of the head in blue). The full 3D structure offers 360º viewing within the Perspecta display.

With the CT dataset, the skull was extracted and isolated from the peripheral tissues, displayed as a single object (Figure 3-b). In the final dataset, chromosomes were extracted from plant cells, and displayed in the context of surrounding cellular materials in Figure 3-c. Run-time per block was in the order of seconds. To propagate fully through the 100 slice, 256×256 MRI dataset took ~1 minute. Significant improvement speed is expected through a GPU implementation: currently under development. Future work will also include the implementation of more illustrative rendering motifs to better emphasize structures of interest rather than simple color mapped representations. Investigations are also being conducted regarding into more intuitive interaction methods for user seeding, through the use of accelerometer-based input and/or tracked gestures. It is anticipated that such interactivity, combined with speed of processing, can provide a more direct, one-to-one experience of exploring volumetric datasets.

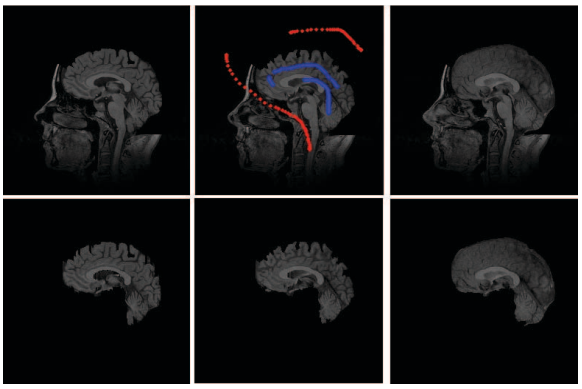
6. Conclusion

This paper introduces a systematic framework for spatial rendering and visualization in 3D. Two segmentation techniques are applied for object of interest extraction, and a spatially rendered display Perspecta is utilized for visualizing the reconstructed volumetric model. The effectiveness of the proposed method is demonstrated through real medical imaging data. It is anticipated that the employment of such techniques will facilitate visualization, information extraction and thus, our ultimate understanding of the complex environments and interactions that occur across a variety of application contexts.

(a)

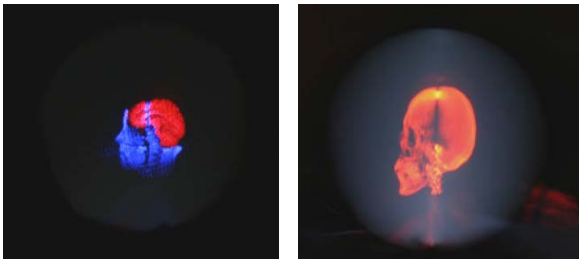


(b)

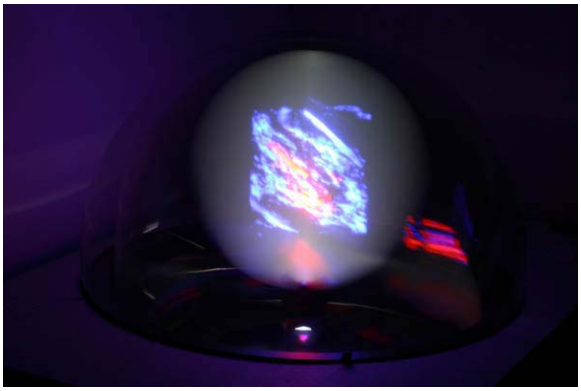


(c)

Figure 2: (a) System overview (b) Segmentation estimates propagate out from the originally marked slice in stages (blocks of three image slices are used for each segmentation) (c) Result of segmentation of slices adjacent to the originally seeded slice in the MRI set



(a) (b)



(c)

Figure 3: 3D visualization of (a) MRI data; (b) CT data; and (c) Confocal image of plant chromosomes.

7. References

[1] C.H. Lim, et al, "Interactive & Immersive VR Image Processing and Visualization", *9th Int. Conf. on Control, Automation, Robotics and Vision, ICARCV '06*, 5-8 Dec. 2006 pp.1-5.

[2] G.E. Favalora, et al, "100 Million-voxel volumetric display," in *Cockpit Displays IX: Displays for Defense Applications*, D.G. Hopper, Ed., Proceedings of SPIE vol. 4712, 2002, pp. 300-312.

[3] G.E. Favalora, "Volumetric 3D Displays and Application Infrastructure", *Computer*, vol.38, no.8, Aug. 2005, pp.37-44.

[4] B. Delaney, "Forget the funny glasses [autostereoscopic display systems]", *Computer Graphics and Applications, IEEE*, vol. 25, no. 3, May-June 2005, pp.14-19.

[5] W.-S. Chun et al., "Spatial 3D Infrastructure: Display-Independent Software Framework, High-Speed Rendering Electronics, and Several New Displays," *Proc. SPIE-IS&T Electronic Imaging*, vol. 5664, 2005, pp. 302-312.

[6] K.-L. Ma, "Machine Learning to Boost the Next Generation of Visualization Technology", *Computer Graphics and Applications, IEEE,* vol. 27, no. 5, Sept.-Oct. 2007, pp.6-9.

[7] I. H. Jermyn and H. Ishikawa, "Globally optimal regions and boundaries", *International Conference on Computer Vision*, vol. II, 1999, pp.904–910.

[8] L. D. Cohen. "On active contour models and balloons", *Computer Vision, Graphics, and Image Processing: Image Understanding*, vol. 53, no. 2, 1991, pp. 211–218.

[9] S. Osher and J. Sethian. "Fronts propagating with curvature dependent speed: algorithms based on the Hamilton-Jacobi formulation", *Journal of Computational Physics*, vol. 79, 1988, pp.12–49.

[10] J. Shi and J. Malik. "Normalized cuts and image segmentation", *In IEEE Conference on Computer Vision and Pattern Recognition*, 1997, pp. 731–737.

[11] E. N. Mortensen and W. A. Barrett, "Interactive segmentation with intelligent scissors", *Graphical Models and Image Processing*, vol. 60, 1998, pp.349–384.

[12] A. X. Falc˜ao, J. K. Udupa, S. Samarasekera, and S. Sharma, "User-steered image segmentation paradigms: Live wire and live lane", *Graphical Models and Image Processing*, 60:233–260, 1998.

[13] L. J. Reese, "Intelligent paint: Region-based interactive image segmentation", *Master's thesis, Brigham Young University*, 1999.

[14] Y. Boykov andM.-P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images," *International Conference on Computer Vision*, vol. I, 2001, pp.105–112.

[15] L. Grady, "Random Walks for Image Segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28, 2006, pp.1-17.

[16] L. Grady and G. Funka-Lea, "Multi-Label Image Segmentation for Medical Applications Based on Graph-Theoretic Electric Potentials", *Workshop on Computer Vision Approaches to Medical Image Analysis and Mathematical Methods in Biomedical Image Analysis*, 2004.

[17] B. Blundell and A. Schwarz, *Volumetric Three-Dimensional Display Systems*, John Wiley & Sons, 2000.

[18] A. Jones, I. McDowall, H. Yamada, M. Bolas, P. Debevec, "Rendering for an Interactive 360° Light Field Display", *ACM Trans. Graph.* 26, 3, Article 40, July 2007.

[19] K. Langhans, C. Guill, E. Rieper, K. Oltmann, D. Bahr, "SOLID FELIX: A Static Volume 3D-Laser Display", *Proc. SPIE*, vol. 5006, 161, 2003.