ImmerVol: An Immersive Volume Visualization System

Naimul Mefraz Khan, Matthew Kyan and Ling Guan
Department of Electrical and Computer Engineering, Ryerson University
Email: {n77khan,mkyan,lguan}@ee.ryerson.ca

***Abstract*—Volume visualization is a popular technique for analyzing 3D datasets, especially in the medical domain. An immersive visual environment provides easier navigation through the rendered dataset. However, visualization is only one part of the problem. Finding an appropriate Transfer Function (TF) for mapping color and opacity values in Direct Volume Rendering (DVR) is difficult. This paper combines the benefits of the CAVE Automatic Virtual Environment with a novel approach towards TF generation for DVR, where the traditional low-level color and opacity parameter manipulations are eliminated. The TF generation process is hidden behind a Spherical Self Organizing Map (SSOM). The user interacts with the visual form of the SSOM lattice on a mobile device while viewing the corresponding rendering of the volume dataset in real time in the CAVE. The SSOM lattice is obtained through high-dimensional features extracted from the volume dataset. The color and opacity values of the TF are automatically generated based on the user's perception. Hence, the resulting TF can expose complex structures in the dataset within seconds, which the user can analyze easily and efficiently through complete immersion.**

I. Introduction

Volume visualization is an important technique to analyze complex data in many domains. Volumetric datasets are comprised of *voxels*, which are essentially 3D pixels. Due to the added dimension over traditional 2D images, volume visualization can be a complex and tedious process.

The most popular technique for volume visualization is Direct Volume Rendering (DVR), where the voxels are mapped to different color and opacity values to generate a visualization of the volume. The task of DVR can be divided into three general stages: 1) voxel clustering; where the voxels are grouped into different clusters based on their properties (i.e. features), 2) color and opacity assignment; where the voxel groups in the first stage are mapped to different color and opacity values and 2) visualization; where the color and opacity values generated in the second stage are used to render and display the volume data.

The mapping of voxels to color and opacity values is called a Transfer Function (TF). Existing TF design tools involve extensive user interaction with low level parameters on widgets representing voxel features (typically 1D or 2D histograms) to cluster the voxels (the first stage) [1], [2]. The user then explicitly assigns ad-hoc color and opacity values to different voxel clusters (second stage). The end users for visualization systems may not be experts in image processing. Hence it is difficult for them to interact with low level widgets. Also, these widgets represent the voxel feature space directly. Hence, they can only represent 2D, at most 3D features. For complex datasets, higher dimensional feature space is needed. Moreover, since the color and opacity value assignments are ad-hoc, the resulting render will largely depend on the artistic ability of the user. A technique is needed to automatically assign color and opacity values based on the user's perception.

The last stage (i.e. visualization) involves the display of the visual form of the dataset. Traditionally, the 3D data is displayed on a 2D display which lacks the realistic sense of depth and size. Lack of judgement in depth or size can result in erroneous interpretation by the end user [3]. Advanced 3D displays like the CAVE automatic virtual environment [4] can provide a sense of immersion and, in turns, can result in better interpretation and analysis of the data.

In this paper, we propose a complete volume visualization tool which addresses the aforementioned limitations. The primary contributions of the proposed *ImmerVol* system can be summarized as follows:

- To simplify the voxel clustering process so that the user does not have to manipulate low level parameters, we use the Self-Organizing Map (SOM) algorithm, an unsupervised clustering algorithm that maps continuous input feature patterns to a set of discrete output nodes. SOM provides a topology-preserving low-dimensional representation of the original feature space. Since the output nodes are arranged in a regular lattice structure, SOMs are easy to visualize and interact with. All the complexity of the training is hidden behind the simple color-code spherical lattice of the SOM in our system. The user simply selects or de-selects part of the map and the corresponding voxel groups are automatically rendered. There is no low level parameter tweaking necessary [1].
- The user also does not need to assign color and opacity values manually. The opacity and color values are automatically generated by combining different voxel group properties (spatial variance, gradient magnitude etc.) with aesthetically pleasing *harmonic colors* [6].
- The last stage is the visualization for the rendered volume. We want to use the CAVE environment to visualize the data to provide the user with a sense of immersion. However, the user also needs to control the visibility of the voxel groups through the SOM lattice as described above. Hence, we provide the user with a mobile device (a tablet), where the SOM lattice can be visualized. The user immerses him or herself in

[1]This part of the ImmerVol system is based on our basic SOM-based TF generation method presented in [5].

the CAVE while easily controlling the visibility of the voxels.

The combination of these techniques makes the whole visualization process becomes simpler and more robust.

The rest of the paper is organized as follows: Section II discusses some of the related works in volume visualization. Section III details our proposed method. Section IV provides results on some well-known datasets. Finally, Section V provides the conclusion.

II. Related Work

Traditionally, TFs are one-dimensional, where the color and opacity is manually assigned by the user through the 1D intensity histogram [1]. Some recent Multi-dimensional TFs make use of other features along with intensity values, such as gradient magnitude [1], [2], second derivative [7], spatial information [8] etc. Most of these methods make use of histograms, and represent the feature space at its original dimension. As a result, higher-dimensional features can not be used. These systems are also not adaptable to use of new features, which can be important if the volume visualization application needs to be domain independent. Also, the user has to tweak low-level parameters which is not desirable. Some recent methods have used different clustering algorithms on the histogram data to divide the voxels into different groups and assign TF properties accordingly [9], [10]. Even with these methods, the user has to play around with clustering parameters (e.g. number of clusters, cluster spread etc.). Also the color and opacity assignments are ad-hoc and depends entirely on the artistic ability of the user.

The concept of *harmonic colors* [6] can be used to automate the color assignment process. Harmonic colors are sets of colors whose inter-spacing on the color wheel provides a better visual experience than user-defined colors.

The CAVE environment has also been used for volume visualization previously. But most of the existing volume visualization tools in the CAVE only have the basic 1D histogram based TF editing, which can be difficult to manipulate in an immersive environment to achieve good results.A system of volume rendering in the CAVE was first proposed in [11]. This system had a primitive 1D TF editor, where the color and opacity values could be separately placed. A similar system was proposed in [12], where the focus was to separate the immersive environment from the rendering system to make it modular. An analysis of the effect of immersion on volume visualization was presented in [3]. There was no TF editing options in that system, as it was not the focus. Our proposed ImmerVol system provides a complete and robust volume visualization tool which provides a user with a friendly TF editing system based on clustering.

To provide a high-dimensional clustering of the voxel features and easy visualization of the cluster space, we make use of the Self-Organizing Map (SOM). It is an unsupervised clustering method [13] that clusters the data and projects data points onto a lower-dimensional space while preserving the input topology. The data points are projected onto a regular lattice during training, when the weights of the nodes in the lattice are updated. Different coloring methods are then applied on the lattice to color code it so that the cluster regions can be visualized. The topology-preserving clustering and easy visualization properties of the SOM makes it an attractive choice for TF generation.

Different lattice structures exist for the SOM, such as the 2D rectangular lattice, the cubic lattice etc [14]. But these structures have a restricted neighborhood problem and can result in unwanted clustering artifacts such as folding and twisting. To achieve best results, we make use of the Spherical SOM (SSOM) [14]. The spherical lattice has a regular structure where each node has exactly 6 neighbors. As a result, this structure offers minimum discontinuity and better cluster visualization [14].

III. Proposed Method

In this section, we describe the details of our proposed ImmerVol system. We will divide the system into the three stages of volume rendering as described before: 1) voxel clustering, 2)color and opacity assignment and 3) visualization.

A. Voxel Clustering

The first step for voxel clustering is feature extraction. Since we will be using SOM, the feature dimension is independent of the algorithm. Any feature suitable for a particular domain can be used in our system. To show the robustness of our system, we have used a 6-dimensional feature set with the following features for each voxel:

- The X,Y and Z coordinates,
- the intensity value,
- the 3D gradient magnitude. The 3D gradient magnitude for each voxel is defined by:
$$G = \sqrt{G_x^2 + G_y^2 + G_z^2}, \tag{1}$$
where G_x,G_y and G_z are the gradient values along X, Y and Z direction, respectively.
- The second derivative. This feature can be calculated from the Eigen values of the Hessian matrix [15]. The Hessian matrix can be defined as:
$$H = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}, \tag{2}$$
Where $I_{xx}, I_{xy}, \ldots, I_{zz}$ are the second partial derivatives. The sum of the three Eigenvalues of this matrix $\lambda_1 + \lambda_2 + \lambda_3$ (where λ_i represents one Eigenvalue) roughly approximates the information obtained from the second derivative of the intensity values [15]. This sum value is used as a feature in our experiments.

All the features are scaled to fall between the value of $\{0,1\}$.

After feature extraction, the Spherical SOM (SSOM) was trained. The training phase of the SSOM is similar to the classical SOM [13]. Let, the input space of N voxels be represented by $\mathscr{X} = \{x_i\}_{i=1}^{N}$. Let, the SSOM be represented by M nodes ($M << N$). Each node in the SSOM lattice has a corresponding weight vector w. All these weight vectors together represent our SSOM space $\mathscr{W} = \{w_i\}_{i=1}^{M}$. Each node also has a *neighborhood* associated with it. A neighborhood is a set of nodes consisted of the node itself and its neighbors. Let, the neighborhood set for node i be represented by Θ_i^r. Here, r represents the neighborhood spread, $r = 1, \ldots, R$. R is the maximum neighborhood radius, which is set to a value such that it covers half of the spherical space [14].

The input voxels are randomly introduced to the SSOM during training. For each voxel, a *Best Matching Unit (BMU)* among all the nodes is selected. BMU is the node which is closest to the input voxel in terms of Euclidean distance. The update step then takes place, where the weight vector of the BMU and it's neighboring nodes (Θ_{BMU}^r) are updated in a way so that they are pulled closer to the weight of the input voxel. After training, the SSOM weight vectors are arranged in such a way that represents the underlying distribution of the input data.

The step-by-step training algorithm is described below:

- **Initialization:** The weight vectors of the SSOM nodes are initialized first. Random values can be used for initialization, but as pointed out by Kohonen et al. in [13], random initialization will take more time to converge. We have followed the initialization method stated in [13] i.e. initialize the weight vectors with values that lie on the subspace spanned by the eigenvectors corresponding to the two largest principal components of the input data distribution. A count vector $\mathscr{C} = \{c_i\}_{i=1}^M$ [14] is used to keep track of the hits to each node. This vector is initialized to zero. This is used in the BMU selection step to prevent cluster under-utilization [14]. In other words, this parameter makes sure that there is a fair competition and one node does not win it all.
- **Training:** For each input voxel x, do the following:
  - **BMU Selection:** Calculate the distance of the voxel feature vector x with all nodes:
  $$e_i = (c_i + 1)\|x - w_i\|, \quad i = 1, \ldots, M. \tag{3}$$
  BMU is the node for which this distance is the smallest.
  - **Weight Update:** Update the weights for the BMU and it's neighboring nodes (defined by Θ_{BMU}^r) as follows:
  $$w = w + b(t) * h(s,r) * \|x - w\|, \tag{4}$$
  $$c_w = c_w + h(s,r), \tag{5}$$
  where $w \in \Theta_{BMU}^r$, $b(t) = \alpha e^{-\frac{t}{T}}$ and $h(s,r) = e^{-\frac{r^2}{s*R}}$.
  The functions $b(t)$ and $h(s,r)$ control the rate of learning and the neighborhood effect, respectively. $b(t)$ decreases in value as the iteration number $t = 1, 2, \ldots, T$ increases. It also depends on the learning rate α. $h(s,r)$ depends on the neighborhood size parameter s, which is user controlled. $h(s,r)$ is a Gaussian function. The further a neighboring node is from a BMU, the less it's weight will be affected. As discussed before, the count-dependent parameter c_w is increased here to prevent cluster under-utilization.
- Repeat the training steps for a pre-defined number of iterations (T).

The main control parameters in SSOM training are the learning rate α, the number of iterations T and the neighborhood size parameter s. In case of volume rendering, we are dealing with a huge number of voxels (e.g. for a volume of dimension $256X256X256$, there are over 16 million voxels). As we have found experimentally, for such high number of voxels, the SOM typically converges within only one iteration. The α is set to 0.1 in our experiments. The neighborhood size parameter is set to $s = 2$, which is determined through trial and error.

After training of the SSOM is completed, a color code the spherical lattice using the U-Matrix approach [14] is presented to the user, which provides a visual form of the distance between the weight vectors of the nodes. For each node, the average Euclidean distance of its weight vector with the weight vectors of all the immediate neighboring nodes is calculated. These distance measures are then mapped to a color-map for visualization purposes. In this way, a homogeneously colored region will represent a cluster, while the cluster boundaries will incur a change in coloring. An important point to note here is that since volume rendering is an entirely perceptual process, it is not important to strictly define how many clusters we have or whether the cluster boundaries are very well defined or not. The important point is to color the SSOM lattice in such a way that intuitively interacting with it will directly result in meaningful rendering. The user interacts with this lattice and groups some of the nodes together. Since the SOM has a large number of nodes (642 in our case) and the user is typically interested in only 2-3 groups of voxels, the user can select a number of nodes and define them as a group. The user gets immediate feedback with an intermediate rendering of the volume data to see which voxels correspond to the selected group of nodes. When the user is satisfied with the grouping, he or she can finalize a group and the final rendering based on our color and opacity assignment algorithm (explained below) is immediately shown. The user does not need to interact with any low level parameters, the only required operations are selection of the nodes to define groups. There are also options of turning a particular group on/off to better analyze the whole volume [2].

B. Color and Opacity Assignment

After the user defines the separate voxel groupings on the SSOM lattice, we generate the color and opacity values to create the final rendering. We apply the theory of color harmonization [6] for our color assignment. Harmonic colors define sets of colors that are aesthetically pleasing. A harmonic set is described by specifying the relative position of the colors on a color wheel rather than specific colors themselves. Based on this color wheel, several templates can be defined. Figure 1 shows template type T, which we have used in our method. This template can also be rotated at arbitrary degrees to generate a new set of colors.

We use this template to generate the hue values for *HSV* color space. We need separate hue value for each group of

[2]The reader is encouraged to watch the video demo at http://youtu.be/62lTPz3VTQ8 for a better understanding of the whole process.

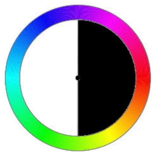

Fig. 1. Type T Hue Template

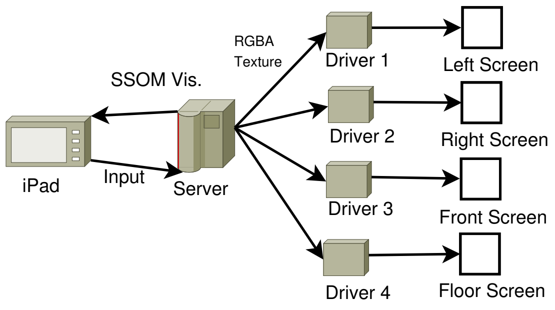

Fig. 2. Overall system architecture of ImmerVol

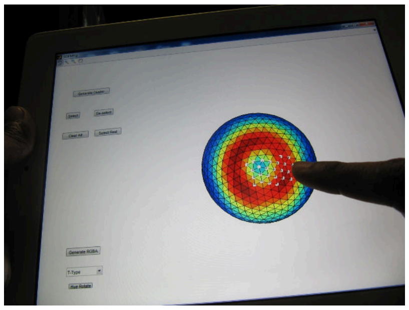

Fig. 3. The SSOM lattice on the iPad

voxels selected by the user. We equally space the required hue values along the available hue range so that we've a visually pleasing result as a combination of all of them. However, if we simply use one hue value for each user selected group, the details will not be revealed. For volume rendering, boundary enhancement is important to reveal the inner structures [1]. As a result, instead of using one hue value, we use a small range with one lower and one higher limit, ξ_l^i and ξ_h^i for each group. The hue value for each voxel of each group is calculated by as follows:

$$H_i^v = \xi_l^i + (\xi_h^i - \xi_l^i) * \digamma_v, \ \ i = 1, \ldots, Z, \tag{6}$$

where Z denotes the total number of groups selected by the user. $\digamma_v$ is the *Gradient Factor* and is defined as:

$$\digamma_v = \frac{G_v}{max_{v \in \mathscr{X}} G_v}. \tag{7}$$

Here, $\mathscr{X}$ denotes the group that the voxel v belongs to. G_v is the 3D gradient magnitude for each voxel (defined in Equation (1)). In this way, even for a single group, the hue values will have some variation based on the gradient value instead of having a single hue value assigned to all of them.

In the *HSV* color space, the S and V values are also important to define a color. The S and V values respectively determine the saturation and brightness of the color. We generate the S and V values based on the understanding of the user's perception [6]. Voxel groups that have a small spatial variance occupy a smaller viewing area compared to groups that have relatively larger spatial variance. We can intuitively assume that the groups with smaller spatial variance needs to have a more saturated color to highlight them properly [6]. Hence, we calculate the S value for each voxel group according to the following equation:

$$S_i = \frac{1}{(1+\sigma_i)}, \ \ i = 1, \ldots, Z. \tag{8}$$

Here, σ_i represents the spatial variance of each voxel group.

Similarly, we can assume that voxel groups closer to the center of the volume needs to be brighter so that they are not overshadowed by groups that are further from the center [6]. Hence, The V value for each voxel group is calculated as follows:

$$V_i = \frac{1}{(1+\mathbf{D}_i)}, \ \ i = 1, \ldots, Z. \tag{9}$$

Here, $\mathbf{D}_i$ denotes the distance of the centroid of the $i-$th group to the center of the volume [6]. By assigning the saturation and brightness values this way, we can assign more vivid colors to the voxel groups that are relatively more difficult to highlight.

The last parameter to define the full TF is opacity. Since voxel groups with smaller spatial variances are likely to be obstructed for proper viewing by groups with larger spatial variances, we calculate the opacity values based on spatial variances. We also want to emphasize the boundaries to reveal the detailed structures. Hence, our opacity values for each voxel are calculated using the following equation:

$$O_i^v = (1 - \frac{1}{\sigma_i}) * (1 + \digamma_v), \ \ i = 1, \ldots, Z. \tag{10}$$

Here, σ_i is the spatial variance of the $i-$th voxel group. $\digamma_v$ is the Gradient Factor as defined in Equation (7). Inspecting Equation (10), we can see that voxel groups with smaller spatial variances will be assigned higher opacity values. Moreover, all the opacity values will go through boundary enhancement by being multiplied by $(1 + \digamma_v)$.

After calculating the *HSV* and opacity values, we convert them into the final *RGBA* texture. The *HSV* triplets are converted into *RGB*, and the opacity value is appended to generate the *RGBA* quadruples. This is passed to the rendering stage to generate the final rendering.

C. Visualization in the CAVE

The final stage in ImmerVol is the visualization of the rendering in the CAVE. The CAVE used in this system is a 4-screen CAVE, left, right, front and floor screens. Each screen is operated by a driver, where each driver consists of a computer connected to a projector. Since our target is complete immersion where the user resides in the CAVE during the whole visualization experience, we will have to

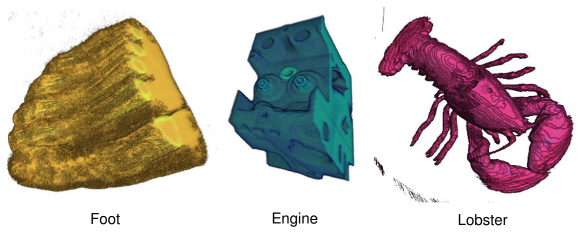

Fig. 4. Rendering of the three datasets without any voxel grouping

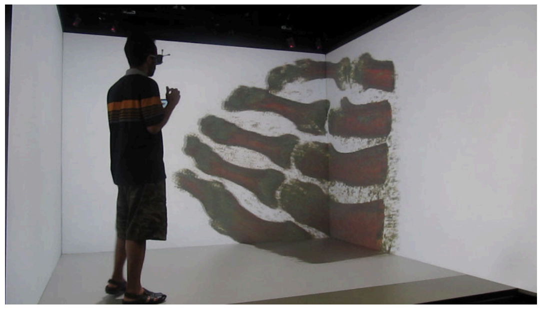

Fig. 5. Rendering of the Foot dataset on the CAVE

make the SSOM visualization part portable. Hence, we project the SSOM lattice on an iPad. Figure 3 shows the UI on the iPad screen. The server communicates with the iPad and listens for user selection of voxel groups. The server also sends the corresponding RGBA texture information to the drivers in real time. The drivers render the data correspondingly. The stitching and accurate placement of different screens together and the head tracking for the 3D glasses to provide an immersive experience is maintained by VR Juggler [16]. The rendering is done through OpenSceneGraph [17].

IV. Experimental Results

We provide the result of ImmerVol system on three datasets [18] : CT scans of a Foot, an Engine and a Lobster. Since volume datasets have a lot of vacant regions (air) around the object, we use a simple region growing technique [19] to segment the volume first. Alternatively, a simple thresholding with the threshold value close to zero could be used too.

Figure 4 shows the first renders of these datasets. These renders are achieved without any user selection performed on the corresponding SSOM lattices. As we can see, no structures are visible in the datasets without any processing.

Figure 5 shows the ImmerVol system in action, where the user has selected two voxel groups and the corresponding render is shown immediately. Since it is difficult to convey the effectiveness of the system without experiencing it immersively, we request the reader to watch the video demo at http://youtu.be/62lTPz3VTQ8. To show some results obtainable by the system, we include renderings achieved with simulation mode in VR Juggler in this paper. In simulation mode, everything else remains the same, except the output is redirected to a regular desktop monitor instead of the CAVE.

Figure 6 shows the final render results of different voxel groups defined by the user on the Foot dataset. The voxel groups can be seen on the spherical lattice, with different groups of nodes separately colored (white, black and red dots).

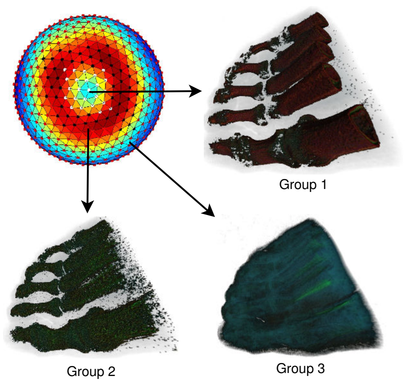

Fig. 6. Voxel groupings for the Foot dataset and render of the corresponding groups.

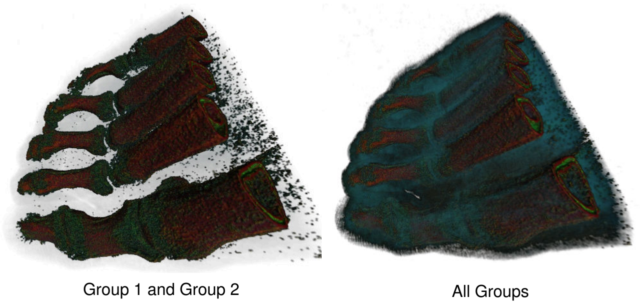

Fig. 7. Rendering of the Foot dataset with different groups activated

As can be seen, due to high-dimensional feature clustering, the structures in the dataset can be revealed quickly and efficiently. Group 1 (corresponding to the nodes colored in white) reveal the inner structures of the bones, while Group 2 (corresponding to the nodes colored in black) emphasize the joints among the bones. All the other voxels are assigned to Group 3 (corresponding to the nodes colored in red), which roughly represents the outer layer of the foot.

The effectiveness of our color and opacity assignment techniques can be seen in Figure 7, where we show the rendering results with 2 groups activated and all 3 groups activated. These results are achieved with type T template (Figure 1) starting at 0°. Since our opacity assignment depends on the spatial variance (Equation (10)), the inner groups of voxels are assigned higher opacity values, and can be seen clearly even with all groups activated.

Another interesting observation is the spread of the voxel groups selected by the user. In the map, the blue colors represent higher cluster densities, while shift towards red denotes the opposite. But as we stated before, volume rendering is a complex process and the principal quality assessment is how the user perceives the output. If we picked the voxel groups automatically based on cluster density, a desirable output will be difficult to achieve. But by leaving the final group definition as a user task, we provide enough flexibility. The only knowledge the user needs is the detection of a small cluster. Typically, a small cluster (blue patch) surrounded by a shift towards other clusters (red patches) suggest that a small group of voxels reside in this cluster which are different from other regions of the volume. So the user can work his/her way

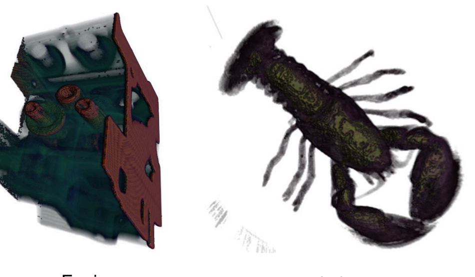

Fig. 8. Rendering of the Engine and Lobster datasets with voxel grouping

Dataset Name	Size	Training Time
Foot	256X256X256	442.5
Engine	256X256X256	369.7
Lobster	301X324X56	111.11

TABLE I. THE SIZE OF THE VOLUME DATASETS AND THE REQUIRED SSOM TRAINING TIMES (IN *seconds*).

out from there and define the voxel groups as he or she wants to see it.

Figure 8 show the results obtained for the Engine dataset and the Lobster dataset. Both were obtained with two voxel groups and type T template starting at 90° and 108°, respectively. As we can again see, with the ImmerVol system, the obtained results clearly reveal the inner structures, which are easily navigable with a total immersive experience in the CAVE with full control within the fingertips on an iPad.

The size of each dataset and training times are listed in Table I. Please note that the training of a SSOM has to be done only once for each dataset and does not effect the rendering process of our proposed method, which is required to be real-time. The perfect convergence of map is also not very critical; as long as we can have a color-coded SSOM where different cluster regions and the borders between them are visually distinguishable.

V. CONCLUSION

In this paper, we proposed an immersive volume visualization system ImmerVol, which uses the CAVE environment and a mobile device to provide the user with an easy and efficient Transfer Function (TF) design tool with immediate results. Spherical Self Organizing Map (SSOM) is used to cluster high-dimensional voxel features. The color coded SSOM lattice is presented to the user on an iPad, where the user does simple operations to provide required voxel groupings. These groupings are immediately translated to color and opacity mapping using algorithms tailored to suit the user's perception. Experimental results on 3 benchmark datasets show the effectiveness of the ImmerVol system.

REFERENCES

[1] G. Kindlmann and J. Durkin, "Semi-automatic generation of transfer functions for direct volume rendering," in *Proceedings of the 1998 IEEE symposium on Volume visualization*. IEEE Press, 1998, pp. 79–86.

[2] M. Selver, M. Alper, and C. Guzeli, "Semiautomatic transfer function initialization for abdominal visualization using self-generating hierarchical radial basis function networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 395–409, May 2009.

[3] B. Laha, K. Sensharma, J. D. Schiffbauer, , and D. A. Bowman, "Effects of immersion on visual analysis of volume data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 597–606, April 2012.

[4] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-screen projection-based virtual reality: the design and implementation of the cave," in *20th annual Conference on Computer Graphics and Interactive Techniques*. ACM, 1993, pp. 135–142.

[5] N. Khan, M. Kyan, and L. Guan, "Intuitive volume exploration through spherical self-organizing map," in *9th Workshop on Self-Organizing Maps*. Springer-Verlag, 2012, pp. 75–84.

[6] Y. Liu, C. Lisle, and J. Collins, "Quick2insight: A user-friendly framework for interactive rendering of biological image volumes," in *IEEE Symposium on Biological Data Visualization*. IEEE Press, 2011, pp. 1–8.

[7] J. Kniss, G. Kindlmann, and C. Hansen, "Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets," in *IEEE symposium on Volume visualization*. IEEE Press, 2001, pp. 255–262.

[8] C. Lundstrom, P. Ljung, and A. Ynnerman, "Local histograms for design of transfer functions in direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 1570–1579, May 2006.

[9] R. Maciejewski, I. Wu, W. Chen, , and D. Ebert, "Structuring feature space: A non-parametric method for volumetric transfer function generation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 1473–1480, 2009.

[10] B. Nguyen, W.-L. Tay, C.-K. Chui, and S.-H. Ong, "A clustering-based system to automate transfer function design for medical image visualization," *The Visual Computer*, vol. 28, pp. 181–191, 2012.

[11] J. Schulze-Dobold, U. Wossner, S. P. Walz, and U. Lang, "Volume rendering in a virtual environment," in *5th IPTW and Eurographics virtual environments*. Springer Verlag, 2001, pp. 187–198.

[12] R. Shen, P. Boulanger, and M. Noga, "Medvis: A real-time immersive visualization environment for the exploration of medical volumetric data," in *5th International Conference on BioMedical Visualization*. IEEE, 2008, pp. 63–68.

[13] T. Kohonen, Ed., *Self-organizing maps*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997.

[14] A. Sangole and A. Leontitsis, "Spherical self-organizing feature map: An introductory review," *International Journal of Bifurcation and Chaos*, vol. 16, pp. 3195–3206, 2006.

[15] J. Hladuvka, A. Konig, and E. Groller, "Exploiting eigenvalues of the hessian matrix for volume decimation," in *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*. Vaclav Skala, 2001, pp. 124–129.

[16] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira, "VR Juggler: A Virtual Platform for Virtual Reality Application Development," in *Proceedings of the Virtual Reality 2001 Conference (VR'01)*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 89–96.

[17] D. Burns and R. Osfield, "OpenSceneGraph. A: Introduction, B: Examples and Applications," in *Proceedings of the IEEE Virtual Reality 2004*. IEEE Computer Society, 2004, pp. 265–.

[18] S. Roettger, "The volume library," 2012, Ohm Hochschule Nurnberg. [Online]. Available: {http://www9.informatik.uni-erlangen.de/External/vollib/}

[19] W. K. Pratt, Ed., *Digital Image Processing*. New York, NY, USA: John Wiley and Sons, 2007.