

Volume visualization tools for medical applications in ubiquitous platforms

Ander Arbelaiz¹, Aitor Moreno¹ Luis Kabongo¹, and Alejandro García-Alonso²

¹ Vicomtech-IK4,
Paseo Mikeletegi, 57, 20009 Donostia / San Sebastián, Spain,
{aarbelaiz, amoreno, lkabongo}@vicomtech.org,

² University of the Basque Country,
Paseo Manuel de Lardizabal 1, 20018 Donostia / San Sebastián, Spain,
alex.galonso@ehu.es

Abstract. This paper presents three required functionality when volume datasets are aimed to be visualized in ubiquitous platforms: *i*) support of segmented volume datasets, *ii*) navigation inside the volume and *iii*) direct visualization of DICOM datasets. DICOM is the de-facto standard in the medical imaging field. The results shows that these functionalities can be achieved using the Volume Rendering component implemented in X3DOM in several web browsers in different platforms (from desktop computer to tablets and mobile phones).

Key words: Volume Rendering, Medical imaging, DICOM, Visualization, X3DOM, Ubiquitous platforms

1 Introduction

In the medical field, the volumetric datasets are key in several phases of the medical procedures. The CT or MRI scans produce a large amount of information of a patient, normally as a set of 2D slices. These slices can be visualized and analyzed one by one, but they provide more information if they are considered as a whole volumetric dataset. The introduction of volume rendering techniques in desktop platforms was a milestone in the medical field. New methodologies and algorithms were implemented to take into account the 3D information embedded in the set of 2D slices that compose the scans.

In the last years, the new mobile paradigm has changed the landscape of the information sharing and the visualization schemes. The graphical power of the mobile devices has made possible to render interactive 3D content (commercial games, etc.). It was a matter of time to get open standards to deal with the 3D content generation and visualization without worrying about the underlying platform (hardware or software). The X3D standard aim to accomplish this challenge and X3DOM provides a WebGL implementation of the standard.

The volume rendering nodes in X3D are specified, but they do not materialize the precise functionality that the users require to interact with the volumetric datasets.

This work presents three functionalities that have been detected as missing in the web-based volume rendering toolkits, including X3DOM. A solution has been devised for all the three functionalities and the results show that the new functionality works in different platforms.

The next section provides a summary of the bibliography and the related work. Sect. 3 presents the three functionalities to include in the volumetric visualization toolkits and how we have provided a solution for them. This paper concludes with the conclusions and future work.

2 Related work

This section is divided in two parts: the first one describes previous work and the second one describes the volume ray-casting algorithm implemented in WebGL.

2.1 Previous work

Volumetric visualization has been extensively studied over the years. Nowadays it is currently used on a variety of fields, but especially in medicine. Different volume visualization techniques can be found in the literature; our work focuses in ray-casting. Originally introduced by Kajiya and Herzen [5], ray-casting method was later translated to the GPU by Kruger and Westermann [6].

The growth of mobile devices and their rapid adoption on every days work has made them a target platform for computationally expensive applications like medical visualization. Rodríguez et al. [8] analyzed different volume rendering algorithms on mobile devices. More recently, Schiewe et al. [10] reviewed the status of volume rendering with the latest graphics APIs available on iOS devices.

WebGL-based rendering positions as the only choice to achieve a ubiquitous volumetric visualization. The first WebGL-based ray-casting method was introduced by Congote et al. [2]. In this direction, improvements have been published by Mobeen et al. [7]. Also, the volume size constraints of this approach have been addressed by Noguera and Jiménez [8]. The web browser is a common denominator in current devices; this has promoted initiatives to create web based medical oriented visualization tools. Between the current available scientific frameworks we should mention XTK [4], VJS [11], and X3DOM [1].

2.2 WebGL volume ray-casting

Our work is integrated in the X3DOM framework under the volume rendering component. Volumetric rendering is performed with a single-pass ray-casting algorithm. In the scene a cube is rendered and it will be used to place the actual volume data during the ray-casting. Using the programmable pipeline provided by the WebGL API, a single pair of vertices and fragment shaders is used. In the vertex shader, each vertex position of the cube is multiplied by the `Volume ModelView` matrix and the `Projection` matrix, transforming the vertices into clip space.

In the fragment shader, the ray traversal is actually computed. When the triangles of the cube are rasterized into fragments, the interpolated vertices of the unitary cube represent 3D texture coordinates. From the inverse `ModelView` matrix the camera position can be obtained. After, by subtracting the interpolated vertex position with the camera position the ray direction can be determined. Taking both the interpolated position as the ray origin and the ray direction, using a fixed length loop statement in the fragment shader the ray traversal is created. For an in depth description of the single-pass approach, we refer the reader to Mobeen et al. [7].

The ray traversal is discretized into a series of steps. At each step the position of the ray is used as a 3D texture coordinates to fetch the volume data. However, WebGL does not support 3D textures, as first stated by Congote et al. [2] this can be overcome using a texture atlas (*ImageTextureAtlas*): the cross sectional slices that compose the volume are tiled into a matrix configuration to compose a single 2D texture.

3 Medical Visualization

Volumetric data is often used in the medical field in a large variety of situations: from research and diagnosis to educational purposes. In terms of visualization interactivity and usability, mobile platforms should provide the same tools as their desktop counterpart.

In pursuit of a ubiquitous medical volumetric visualization, we have detected three features that we consider that a ubiquitous volume visualization toolkit must provide: *i*) the visualization of segmented data, *ii*) camera navigation inside the 3D volume and *iii*) the support of DICOM file format, a widely used medical data exchange format.

3.1 Segmented medical data

Volume visualization must be extended to provide the tools that the medical use cases require. Trained specialists in medical imaging segment regions of interest from the cross-sectional slices that compose a volume. The segmentation of the volume is important to focus the attention or to limit further work at the region of interest. Thus, any medical visualization tool must be capable of helping the user to discern the region of interest from the whole volume.

We have extended the initial ray-casting algorithm presented at Sect. 2.2 to support segmented visualization. In the ray-casting loop the ray traverses the volume accumulating color and opacity. The segmented data acts as an identifier on each voxel to allow the selection of an alternative accumulation algorithm at the region of interest. Using a distinctive color and opacity in sections where the ray traverses, allows standing out a region by visually enhancing the segmented area from the whole volume. The algorithm at Fig. 1 describes how the segmented data must be implemented inside the ray-casting loop.

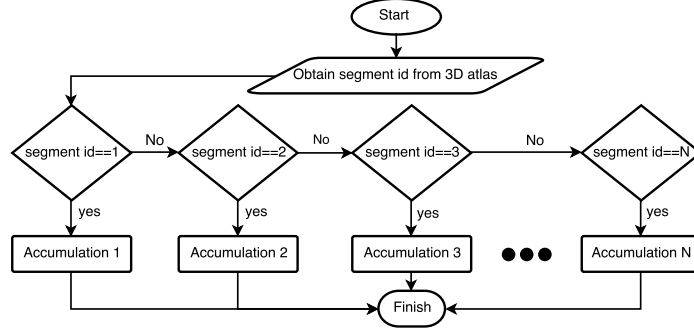


Fig. 1. Flowchart describing the segmentation algorithm in the ray traversal

As Fig. 1 shows, the segment data fetching is actually translated into a switch or selection statement that is performed in the ray-casting loop at a per step basis. From a performance point of view, this must be carefully considered as branching is not well supported in mobile devices. The type of variation in the accumulation process (see Fig. 1) will change depending of the use case, e.g., mapping colors with a transfer function (TF), modifying the opacity given the camera view and discarding the data. Fig. 2 shows two renders of datasets with segmented regions. Each one has been generated with different accumulation processes to make them distinguishable from the volume.

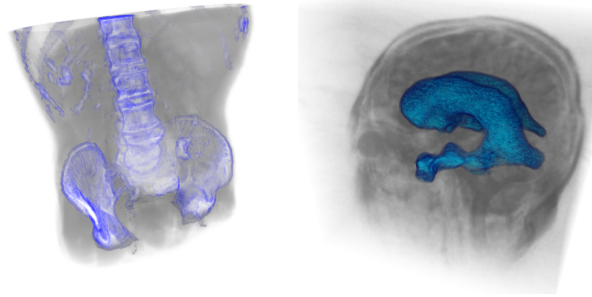


Fig. 2. Segmented data renderings of the aorta ($512 \times 512 \times 97$) and the *Head MRI* [12] ($256 \times 256 \times 124$) datasets. On the left, the edges of the segmented bones have been enhanced with color. On the right, the segmented ventricles had been colored with blue tones

We have seen that it is advisable to specify a user-defined argument to explicitly know the maximum number of segments beforehand. The maximum number of segments limits the branching statements and they are dynamically added at the fragment shader creation time. Otherwise, the number of branching statements will be unbounded and a big amount of unnecessary comparison operations will be performed with a potential impact on the GPU performance.

In our implementation, the segmented data is tiled into a matrix configuration with the same procedure that must be performed with the volume data (see Sect. 2.2). To gain performance the segmented data could be stored in the alpha channel of the original volume data texture atlas, however, it is better and easier to create a separate texture. In first place, communication between server and client should be considered. The segmentation can be processed automatically or manually in an off-line tool and be loaded when required.

In second place, the texture atlas resolution should be considered. Due to memory restrictions in GPU mobile devices compared to desktop computers, the segmented texture atlas can be stored with a smaller texture size than the original texture atlas, and the same position coordinates would be valid to fetch the data from both textures. This technique will imply a precision loss of the segmented region.

3.2 Inside exploration of volume data

Another visualization requirement in the medical environment is related to the interaction and immersive experience that volumetric data can provide. This can be conceptualized as the ability to explore the data from within the volume. An inside exploration allows the user to easily discern the internal composition of the volume rather than looking at the cross-sectional 2D images. In our implementation we have allowed the exploration of the volume by dynamically changing the initial position of the ray origin. Fig. 3 describes the algorithm used. In this case, the cube is of unitary size.

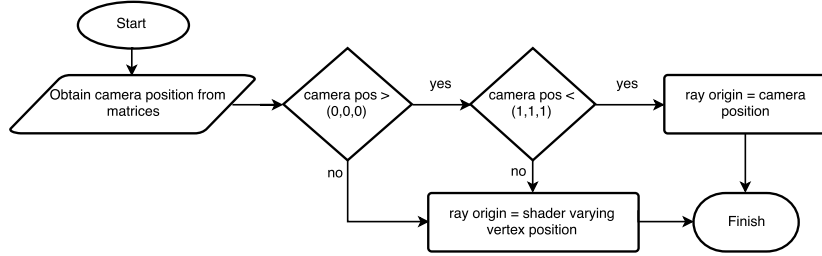


Fig. 3. Flowchart describing the ray origin computation in the single-pass ray-casting algorithm

First the camera position is obtained from the inverse **ModelView** matrix. Then using the maximum and minimum boundaries of the cube, it can be determined whether or not the camera is inside the volume. If the camera is outside the cube, the ray origin is assigned as the interpolated vertex position from the output of the vertex shader (**varying vertex** position), if not, the ray origin is the camera position. Fig. 4 shows an external and internal rendering of the *Head MRI* dataset [12]. Without changing the ray origin, the camera face of the cube will be clipped, making not possible an examination of the inside. Thus,

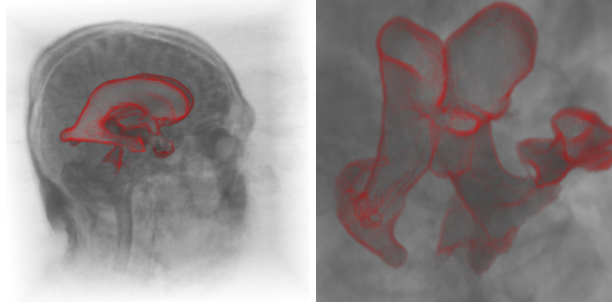


Fig. 4. Rendering of the *Head MRI* dataset ($256 \times 256 \times 124$) [12] enhancing a segmented area. On the left, the camera is outside the volume. On the right, the camera is inside the volume

back face culling must be disabled when internal exploration is required. When the camera is moved into the cube, the ray-casting direction for each ray is obtained by subtracting the interpolated back face vertex position with the camera position.

3.3 DICOM visualization

Medical imaging devices do not only produce the actual 2D set of slices. They are linked with a large amount of metadata with related information about the patient and other medical procedures. DICOM is the medical image standard to store and transfer all this information from and between imaging devices and medical image storage repositories. The wide utilization of DICOM by all manufacturers had a major impact on usability of the file format. Resulting sometimes in a variable set of tags to be read, interpreted and combined in order to achieve coherent restitution of the images for the final user.

Cornerstone JavaScript library [3] provides a set of functions to read and interact with the 2D set of slices stored in DICOM files and it relies on *dicom-Parser* to load DICOM tags, including pixel data.

The combination of *dicomParser* with the volume rendering nodes defined in a X3D scene provides a general and ubiquitous solution to the problem: the slices pixel data is extracted from the DICOM file and then, a texture atlas is created and linked to the required texture field of the *VolumeData* X3D node.

We have devised two ways to accomplish the information transfer from the DICOM file to the X3DOM framework. The first one is to use a `<canvas>` node defined inside the *ImageTextureAtlas* node and then, using JavaScript, fill the canvas with a texture atlas.

```
<ImageTextureAtlas id="atlas">
  <canvas id=voxelCanvas></canvas>
</ImageTextureAtlas>
```

The second integration method is to use the same auxiliary `<canvas>` node but defined out of the *ImageTextureAtlas* node. The *VolumeData* node uses an empty *ImageTextureAtlas* (no real URL is given). The JavaScript loader draws the atlas texture in this canvas as before, and then, the correct URL is provided as a *DataURL*, which means that the whole volumetric information is encoded and passed in the URL.

```
<ImageTextureAtlas id="atlas" url="data:"></ImageTextureAtlas>
document.getElementById("atlas").setAttribute(
    'url',
    document.getElementById("voxelCanvas").toDataURL()
);
```

4 Conclusions

This work has introduced three functionalities required by the users in the visualization and inspection of volumetric datasets, especially in the medical field. The utilization of segmented datasets has been made possible via the *Segmented-VolumeData* node of the X3D in combination with the X3DOM framework. The ability to visualize the volumetric information from inside the dataset enables novel capabilities like vessel navigation. Finally, DICOM files have been read and parsed in the web and different methods to pass the volumetric information to the interactive volume rendering X3D nodes have been explored. All these functionalities are currently available in the developer version of the X3DOM framework and they will be part of the next stable release.

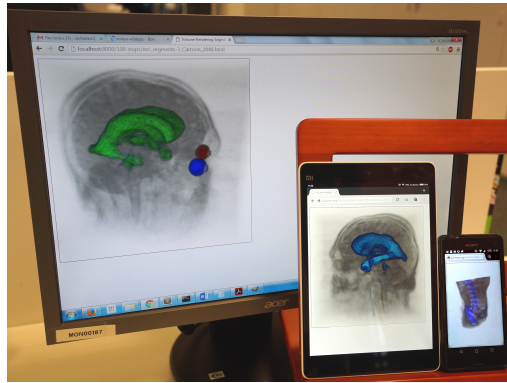


Fig. 5. Ubiquitous volume rendering with WebGL in multiple platforms using segmented medical datasets

The Fig. 5 shows how the presented functionality in this work can be used in the web browser in desktop and mobile platforms. Next steps will be oriented toward the further integration of DICOM in the web environment, as right now

only the scanned images are used in the visualization. More fields and metadata could be used to generate better immersive experiences in the volumetric data.

The utilization of modern VR devices like Oculus Rift, HTC Vive or Microsoft Hololens will increase the immersiveness in the information, but novel user interfaces should be provided. WebVR [13] proposes a preliminary solution to integrate VR devices into the Web and currently, there are sample demos of X3D scenes rendered in such devices. A responsive web application could provide a specific immersive visualization of volumetric datasets for VR devices or a mobile oriented visualization if a smartphone is detected.

References

1. Behr, J., Eschler, P., Jung, Y., & Zöllner, M. (2009, June).: X3DOM: a DOM-based HTML5/X3D integration model. In: Proceedings of the 14th International Conference on 3D Web Technology (pp. 127-135). ACM.
2. Congote J, Segura A, Kabongo L, Moreno A, Posada J, Ruiz O (2011).: Interactive Visualization of Volumetric Data with WebGL in Real-time. In: Proceedings of the 16th International Conference on 3D Web Technology, ACM, New York, NY, USA, Web3D 11, pp 137146
3. Cornerstone JavaScript library to display interactive medical images including but not limited to DICOM, <https://github.com/chafey/cornerstone>
4. Hähn D, Rannou N, Ahtam B, Grant PE, Pienaar R.: Neuroimaging in the Browser using the X Toolkit. Front. Neuroinform. Conference Abstract: 5th INCF Congress of Neuroinformatics., 2014
5. James T. Kajiya and Brian P Von Herzen. 1984.: Ray tracing volume densities. In: Proceedings of the 11th annual conference on Computer graphics and interactive techniques (SIGGRAPH '84), Hank Christiansen (Ed.). ACM, New York, NY, USA, 165-174.
6. Kruger, J.; Westermann, R.: Acceleration techniques for GPU-based volume rendering. In: Visualization, 2003. VIS 2003. IEEE , vol., no., pp.287-292, 24-24 Oct. 2003
7. Mobeen M, Feng L (2012).: High-Performance Volume Rendering on the Ubiquitous WebGL Platform. In: High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, pp 381388
8. Noguera, J., & Jiménez, J. (2012, June).: Visualization of very large 3D volumes on mobile devices and WebGL. In: 20th WSCG international conference on computer graphics, visualization and computer vision. WSCG.
9. Rodríguez M. B., Alcocer P. P. V.: Practical Volume Rendering in Mobile Devices. ISVC 7431, Chapter 67 (2012), 708718.
10. Schiewe, A and Anstoots, M and Krüger, J.: State of the Art in Mobile Volume Rendering on iOS Devices. In: Eurographics Conference on Visualization (EuroVis) - Short Papers. E. Bertini and J. Kennedy and E. Puppo (Ed.). The Eurographics Association. (2015)
11. VJS Medical Imaging Sugar for ThreeJS, <https://github.com/FNNDSC/vjs>
12. Volume Data obtained from, <http://www.volvis.org>
13. WebVR Bringing virtual reality to the web, <http://webvr.info/>